COMPUTABILITY OF DATA-WORD TRANSDUCTIONS OVER DIFFERENT DATA DOMAINS

LÉO EXIBARD, EMMANUEL FILIOT, NATHAN LHOTE, AND PIERRE-ALAIN REYNIER

Université Libre de Bruxelles, Brussels, Belgium and Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
e-mail address: leo.exibard@ulb.ac.be

Université Libre de Bruxelles, Brussels, Belgium

Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

ABSTRACT. In this paper, we investigate the problem of synthesizing computable functions of infinite words over an infinite alphabet (data ω-words). The notion of computability is defined through Turing machines with infinite inputs which can produce the corresponding infinite outputs in the limit. We use non-deterministic transducers equipped with registers, an extension of register automata with outputs, to describe specifications. Being non-deterministic, such transducers may not define functions but more generally relations of data ω-words. In order to increase the expressive power of these machines, we even allow guessing of arbitrary data values when updating their registers.

For functions over data ω-words, we identify a sufficient condition (the possibility of determining the next letter to be outputted, which we call next letter problem) under which computability (resp. uniform computability) and continuity (resp. uniform continuity) coincide.

We focus on two kinds of data domains: first, the general setting of oligomorphic data, which encompasses any data domain with equality, as well as the setting of rational numbers with linear order; and second, the set of natural numbers equipped with linear order. For both settings, we prove that functionality, *i.e.* determining whether the relation recognized by the transducer is actually a function, is decidable. We also show that the so-called next letter problem is decidable, yielding equivalence between (uniform) continuity and (uniform) computability. Last, we provide characterizations of (uniform) continuity, which allow us to prove that these notions, and thus also (uniform) computability, are decidable. We even show that all these decision problems are PSPACE-complete for $(\mathbb{N}, <)$ and for a large class of oligomorphic data domains, including for instance $(\mathbb{Q}, <)$.

CONTENTS

INTRODUCTION

Synthesis. Program synthesis aims at deriving, in an automatic way, a program that fulfils a given specification. It is very appealing when for instance the specification describes, in some abstract formalism (an automaton or ideally a logic), important properties that the program must satisfy. The synthesised program is then *correct-by-construction* with regard to those properties. It is particularly important and desirable for the design of safety-critical systems with hard dependability constraints, which are notoriously hard to design correctly. In their most general forms, synthesis problems have two parameters, a set of inputs In and a set of outputs Out, and relate two classes $\mathcal{S}$ and $\mathcal{I}$ of specifications and implementations respectively. A specification $S \in \mathcal{S}$ is a relation $S \subseteq \mathsf{In} \times \mathsf{Out}$ and an implementation $I \in \mathcal{I}$ is a function $I : \mathsf{In} \to \mathsf{Out}$. The $(\mathcal{S}, \mathcal{I})$-synthesis problem asks, given a (finite representation of a) specification $S \in \mathcal{S}$, whether there exists $I \in \mathcal{I}$ such that for all $u \in \mathsf{In}$, $(u, I(u)) \in S$. If such I exists, then the procedure must return a program implementing I. If all specifications

in $\mathcal{S}$ are *functional*, in the sense that they are the graphs of functions from In to Out, then the $(\mathcal{S}, \mathcal{I})$-synthesis is a membership problem: given $f \in \mathcal{S}$, does $f \in \mathcal{I}$ hold?

Automata-theoretic approach to synthesis. In this paper, we are interested in the automata-theoretic approach to synthesis, in the sense that specifications and implementations can be defined by automata, or by automata extended with outputs called *transducers*. In this approach, In and Out are sets of words over input and output alphabets Σ and Γ respectively. Perhaps the most well-known decidable instance of synthesis in this context is the celebrated result of Büchi and Landweber [JL69]: $\mathcal{S}$ is the class of ω-regular specifications, which relates infinite input words $i_0 i_1 \cdots \in \Sigma^\omega$ to infinite output words $o_0 o_1 \cdots \in \Gamma^\omega$ through ω-automata (e.g. deterministic parity automata), in the sense that the infinite convolution $i_0 o_0 i_1 o_1 \cdots \in (\Sigma\Gamma)^\omega$ must be accepted by an ω-automaton defining the specification. The class of implementations $\mathcal{I}$ is all the functions which can be defined by Mealy machines, or equivalently, deterministic *synchronous* transducers which, whenever they read some input $i \in \Sigma$, produce some output $o \in \Gamma$ and possibly change their own internal state. The seminal result of Büchi and Landweber has recently triggered a lot of research in reactive system synthesis and game theory, both on the theoretical and practical sides, see for instance [CHVB18]. We identify two important limitations to the now classical setting of ω-regular reactive synthesis:

(i) specifications and implementations are required to be synchronous, in the sense that a single output $o \in \Gamma$ must be produced for each input $i \in \Sigma$, and

(ii) the alphabets Σ and Γ are assumed to be finite.

Let us argue why we believe (i) and (ii) are indeed limitations. First of all, if a specification is not realizable by a synchronous transducer, then a classical synthesis algorithm stops with a negative answer. However, the specification could be realizable in a larger class of implementations $\mathcal{I}$. As an example, if S is the set of words $i_0 o_0 \ldots$ such that $o_\ell = i_{\ell+1}$, then S is not realizable synchronously because it is impossible to produce o_ℓ before knowing $i_{\ell+1}$. But this specification is realizable by a program which can delay its output production by one time unit. Enlarging the class of implementations can therefore allow to give finer answers to the synthesis problem in cases where the specification is not synchronously realizable. We refer to this type of relaxations as *asynchronous* implementations. An asynchronous implementation can be modelled in automata-theoretic terms as a transducer which, whenever it reads an input $i \in \Sigma$, produces none or several outputs, i.e. a finite word $u \in \Gamma^*$. Generalizations of reactive system synthesis to asynchronous implementations have been considered in [HKT12, FLZ11, WZ20]. In these works however, the specification is still synchronous, given by an automaton which strictly alternates between reading input and output symbols.

The synchronicity assumption made by classical reactive synthesis is motivated by the fact that such methods focus on the control of systems rather than on the data, in the sense that input symbols are Boolean signals issued by some environment, and output symbols are actions controlling the system in order to fulfil some correctness properties. From a data-processing perpective, this is a strong limitation. The synthesis of systems which need to process streams of data, like a monitoring system or a system which cleans noisy data coming from sensors, cannot be addressed using classical ω-regular synthesis. Therefore, one needs to extend specifications to asynchronous specifications, in the sense that the specifications must describe properties of executions which do not strictly alternate

between inputs and outputs. Already on finite words however, the synthesis problem of asynchronous specifications by asynchronous implementations, both defined by transducers, is undecidable in general [CL14], and decidable only in some restricted cases [FJLW16]. The second limitation (ii) is addressed in the next paragraph.

From finite to infinite alphabets. To address the synthesis of systems where *data* are taken into account, one also needs to extend synthesis methods to handle infinite alphabets. As an example, in a system scheduling processes, the data are process ids. In a stream processing system, data can be temperature or pressure measurements for example. Not only one needs synthesis methods able to handle infinite alphabets of data, but where those data can be compared through some predicates, like equality or a linear order. Recent works have considered the synthesis of (synchronous) reactive systems processing *data words* whose data can be compared for equality [KMB18, ESKG14, KK19, EFR19] as well as comparison with a linear order on the data [EFK21]. To handle data words, just as automata have been extended to *register automata*, transducers have been extended to *register transducers*. Such transducers are equipped with a finite set of registers in which they can store data and with which they can compare data for equality, inequality or in general any predicate, depending on the considered data domain. When a register transducer reads a data, it can compare it to the data stored in its registers, assign it to some register, and output the content of none or several registers, i.e., a finite word v of register contents. To have more expressive power, we also allow transducers to guess an arbitrary data and assign it to some register. This feature, called data guessing, is arguably a more robust notion of non-determinism notion for machines with registers and was introduced to enhance register automata [KZ10]. We denote by NRT the class of non-deterministic register transducers. As an example, consider the (partial[1]) data word function g which takes as input any data word of the form $u = su_1su_2\dots \mathbb{N}^\omega$, s occurs infinitely many times in u, and $u_i \in (\mathbb{N} \setminus \{s\})^+$ for all $i \geq 1$. Now, for all $i \geq 1$, denote by $|u_i|$ the length of u_i and by d_i the last data occurring in u_i. The function g is then defined as $g(u) = d_1^{|u_1|}sd_2^{|u_2|}s\dots$. This function can be defined by the NRT of Figure 1. Note that without the guessing feature, this function could not be defined by any NRT.

Thanks to the non-determinism of NRT, in general and unlike the previous example, their might be several accepting runs for the same input data word, each of them producing a possibly different output data word. Thus, NRT can be used to define binary relations of data ω-words, and hence specifications. In the works [KMB18, ESKG14, KK19, EFR19, EFK21] already mentioned, NRT have been used as a description of specifications, however they are assumed to be synchronous and without guessing.

Objective: synthesis of computable data word functions. In this paper, our goal is to define a synthesis setting where both limitations (i) and (ii) are lifted. In particular, specifications are assumed to be given by (asynchronous) non-deterministic register transducers equipped with a Büchi condition (called NRT). To retain decidability, we however make some hypothesis: specifications are assumed to be functional, i.e., they already define a function from input data ω-words to output data ω-words. While this a strong hypothesis, it is motivated

---

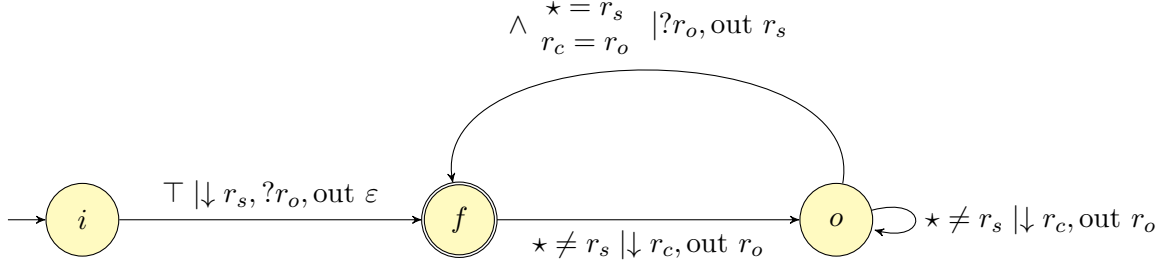[1]In this paper, data word functions can be partial by default and therefore we do not explicitly mention it int the sequel

Figure 1: An NRT defining the data word function g, equipped with a Büchi condition. The current data denoted by $\star$ is tested with respect to the content of the registers on the left of the bar $|$. On the right of the bar, there are instructions such as assigning an arbitrary data to r (notation $?r$), outputting the content of a register or nothing (out r), or assigning the current data to some register ($\downarrow r$). The Büchi condition makes sure that the first data, initially stored in r_s during the first transition, occurs infinitely many times. The register r_c stores the last data that has been read. r_o is meant to store the last data d_i of an input chunk u_i. It has to be guessed whenever a new chunk u_i is starting to be read, and on reading again r_s, the automaton checks that the guess was right by evaluating whether $r_c = r_o$ (at that moment, r_c contains d_i).

by two facts. First, the synthesis problem of asynchronous implementations from asynchronous specifications given by (non-functional) NRT is undecidable in general, already in the finite alphabet case [CL14]. Second, functional NRT define uncomputable functions in general, and therefore they cannot be used as machines that compute the function they specify. Since those functions are defined over infinite inputs, let us make clear what we mean by computable functions. A (partial) function f of data ω-words is computable if there exists a Turing machine M that has an infinite input $x \in \mathrm{dom}(f)$, and produces longer and longer prefixes of the output $f(x)$ as it reads longer and longer prefixes of the input x. Therefore, such a machine produces the output $f(x)$ in the limit. As an example, the function g previously defined is computable. A Turing machine computing it simply has to wait until it sees the last data d_i of a chunk u_i (which necessarily happens after a finite amount of time), compute the length ℓ_i of u_i and once it sees d_i, output $d_i^{\ell_i}$ at once. However, consider the extension f to any input data word defined as follows: $f(u) = g(u)$ if u is in the domain of g, and otherwise $f(u) = s^\omega$ where s is the first data of u. Such function is not computable. For instance, on input $x = sd^\omega$ (where $d \neq s$ are arbitrary data), we have $f(sd^\omega) = s^\omega$, as x is not in the domain of g. Yet, on any finite prefix $\alpha_k = sd^k$ of sd^ω, any hypothetical machine computing f cannot output anything. Indeed, there exists a continuation of α_k which is in the domain of g, and for which f produces a word which starts with a different data than $f(\alpha_k d^\omega)$: it suffices to take the continuation $(sd)^\omega$, as we have $f(\alpha_k (sd)^\omega) = g(\alpha_k (sd)^\omega) = d^k (sd)^\omega$.

In this paper, our goal is therefore to study the following synthesis problem: given a functional NRT defining a function f of data ω-words, generate a Turing machine which computes f if one exists. In other words, one wants to decide whether f is computable, and if it is, to synthesize an algorithm which computes it.

Contributions. Register transducers can be parameterized by the set of data from which the ω-data words are built, along with the set of predicates which can be used to test those data. We distinguish a large class of data sets for which we obtain decidability results for the later problem, namely the class of oligomorphic data sets [BKL14]. Briefly, oligomorphic data sets are countable sets D equipped with a finite set of predicates which satisfies that for all n, D^n can be partitioned into finitely many equivalence classes by identifying tuples which are equal up to automorphisms (predicate-preserving bijections). For example, any set equipped with equality is oligomorphic, such as $(\mathbb{N}, \{=\})$, $(\mathbb{Q}, \{<\})$ is oligomorphic while $(\mathbb{N}, \{<\})$ is not. However $(\mathbb{N}, \{<\})$ is an interesting data set in and of itself. We also investigate NRT over such data set, using the fact that it is a substructure of $(\mathbb{Q}, \{<\})$ which is oligormorphic. Our detailed contributions are the following:

(1) We first establish a general correspondence between computability and the classical mathematical notion of continuity (for the Cantor distance) for functions of data ω-words (Theorems 2.14 and 2.15). This correspondence holds under a general assumption, namely the decidability of what we called the *next-letter problem*, which in short asks that the next data which can be safely outputted knowing only a finite prefix of the input data ω-word is computable, if it exists. We also show similar correspondences for more constrained computability and continuity notions, namely Cauchy, uniform and m-uniform computability and continuity. In these correspondences, the construction of a Turing machine computing the function is effective.

(2) We consider a general computability assumption for oligomorphic data sets, namely that they have decidable first-order satisfiability problem [Boj19]. We call such data sets *decidable*. We then show that functions defined by NRT over decidable oligomorphic data sets and over $(\mathbb{N}, \{<\})$, have decidable next-letter problem. As a consequence (Theorems 3.11 and 4.22), we obtain that a function of data ω-words definable by an NRT over decidable oligomorphic data sets and over $(\mathbb{N}, \{<\})$, is computable iff it is continuous (and likewise for all computability and continuity notions we introduce). This is a useful mathematical characterization of computability, which we use to obtain our main result.

(3) As explained before, an NRT may not define a function in general but a relation, due to non-determinism. Functionality is a semantical, and not syntactical, notion. We nevertheless show that checking whether an NRT defines a function is decidable for decidable oligomorphic data sets (Theorem 3.12). This problem is called the *functionality problem* and is a prerequisite to our study of computability, as we assume specifications to be functional. We establish PSpace-completeness of the functionality problem for NRT over $(\mathbb{N}, \{<\})$ (Corollary 4.20) and for oligomorphic data sets (Theorem 3.12) under some additional assumptions on the data set that we call polynomial decidability. In short, it is required that the data set has PSpace-decidable first-order satisfiability problem.

(4) Finally, we show (again Theorem 3.12) that continuity of functions defined by NRT over decidable (resp. polynomially decidable) oligomorphic data sets is decidable (resp. PSpace-c). We also obtain PSpace-completeness in the non-oligomorphic case $(\mathbb{N}, \{<\})$ (Theorem 4.28). These results also hold for the stronger notion of uniform continuity (see also Theorem 4.24). As a result of the correspondence between computability and continuity, we also obtain that computability and uniform computability are decidable for functions defined by NRT over decidable oligomorphic data sets, and PSpace-c for

polynomially decidable oligomorphic data sets as well as $(\mathbb{N}, \{<\})$. This is our main result and it answers positively our initial synthesis motivation.

The proof techniques we use have in common the following structure: first, we characterize non functionality and non continuity by structural patterns on NRT and establish small witness properties for the existence these patterns. Then, based on the small witness properties, we show how to decide whether given an NRT, such patterns are matched or not. While the proofs have some similarities between the oligomorphic case, the case $(\mathbb{N}, \{<\})$ and the functionality and continuity problems, there are subtle technical differences which make them hard to factorize with reasonable amount of additional notations and theoretical assumptions.

Related Work. We have already mentioned works related to the synthesis problem. We now give references to results on computability and continuity. The notion of continuity with regards to Cantor distance is not new, and for rational functions over finite alphabets, it was already known to be decidable [Pri02]. The approach of Prieur is to reduce continuity to functionality by defining from a transducer T a transducer realizing its topological closure. We were able to extend this approach to almost all the cases we considered, except for transducers over $(\mathbb{N}, \{<\})$ with guessing allowed, so we chose a different proof strategy. The connection between continuity and computability for functions of ω-words over a finite alphabet has recently been investigated in [DFKL20] for one-way and two-way transducers. Our results lift the case of one-way transducers from [DFKL20] to data words. Our results were partially published in conference proceedings [EFR20]. In this later publication, only the case of data sets equipped with the equality predicate was considered. We now consider oligomorphic data sets (which generalise the latter case), the data set $(\mathbb{N}, \{<\})$ and new computability notions. Despite the fact that our results are more general, this generalisation also allows to extract the essential arguments needed to prove this kind of results. Moreover, compared to [EFR20], we add here the possibility for the register transducer to make non-deterministic register assignment (data guessing), which strictly increases their expressive power.

1. Data alphabet, languages and transducers

1.1. **Data as logical structures.** Let Σ be a finite signature with relation and constant symbols. Let $\mathbb{D} = (D, \Sigma^{\mathbb{D}})$ be a logical structure over Σ with a countably infinite domain D and an interpretation of each symbol of Σ. Note that we often identify $\mathbb{D}$ and D when the structure considered is clear, from context.

An *automorphism* of a structure $\mathbb{D}$ is a bijection $\mu : D \to D$ which preserves the constants and the predicates of $\mathbb{D}$: for any constant c in D, $\mu(c) = c$ and for any relation of Σ, $R \subseteq D^r$, we have $\forall \bar{x},\ R(\bar{x}) \Rightarrow R(\mu(\bar{x}))$, where μ is naturally extended to D^r by applying it pointwise. We denote by $\mathrm{Aut}(\mathbb{D})$ the set of automorphisms of $\mathbb{D}$. Let $\bar{x} \in \mathbb{D}^d$, the set $\{\mu(\bar{x}) \mid \mu \in \mathrm{Aut}(\mathbb{D})\}$ is called the *orbit* of $\bar{x}$ under the action of $\mathrm{Aut}(\mathbb{D})$.

We will be interested in structures that have a lot of symmetry. For instance the structures $(\mathbb{N}, \{0, =\})$, $(\mathbb{Z}, \{<\})$ and $(\mathbb{Q}, \{<\})$ fall under our study as well as more sophisticated structures like $(1(0 + 1)^*, \otimes)$ where $\otimes$ is the bitwise xor operation. Other structures like $(\mathbb{Z}, \{+\})$ will not have enough internal symmetry to be captured by our results.

Definition 1.1. A logical structure $\mathbb{D}$ is *oligomorphic* if for any natural number n the set $\mathbb{D}^n$ has finitely many orbits under the action of $\mathrm{Aut}(\mathbb{D})$.

Example 1.2. Oligomorphic structures can be thought of as "almost finite". Consider $(\mathbb{N}, \{=\})$, then $\mathbb{N}^2$ only has two orbits: the diagonal $\{(x,x) \mid x \in \mathbb{N}\}$ and its complement $\{(x,y) \in \mathbb{N}^2 \mid x \neq y\}$. In fact $(\mathbb{N}, \{=\})$ is oligomorphic, since the orbit of an element of $\mathbb{N}^n$ is entirely determined by which coordinates are equal to each other. Similarly, one can see that the dense linear order $(\mathbb{Q}, \{<\})$ is oligomorphic.

The automorphism group of $(\mathbb{Z}, \{<\})$ consists of all translations. This means that $\mathbb{Z}$ only has one orbit. However, $\mathbb{Z}^2$ has an infinite number of orbits since the difference between two numbers is preserved by translation. Hence $(\mathbb{Z}, \{<\})$ is not oligormorphic. However, the fact that $(\mathbb{Z}, \{<\})$ is a substructure of $(\mathbb{Q}, \{<\})$ will allow us to extend our results to this structure, with some additional work. For more details on oligomorphic structures see [Boj19, Chap. 3].

Let G be a group acting on both X, Y, then a function $f : X \to Y$ is called *equivariant* if for all $x \in X, \mu \in G$ we have $f(\mu(x)) = \mu(f(x))$.

1.2. **Words and data words.** For a (possibly infinite) set A, we denote by A^* (resp. A^ω) the set of finite (resp. infinite) words over this alphabet, and we let $A^\infty = A^* \cup A^\omega$. For a word $u = u_1 \ldots u_n$, we denote $|u| = n$ its length, and, by convention, for $x \in A^\omega, |x| = \infty$. The empty word is denoted ε. For $1 \leq i \leq j \leq |w|$, we let $w[i{:}j] = w_i w_{i+1} \ldots w_j$ and $w[i] = w[i{:}i]$ the ith letter of u. For $u, v \in A^\infty$, we say that u is a prefix of v, written $u \leq v$, if there exists $w \in A^\infty$ such that $v = uw$. In this case, we define $u^{-1}v = w$. For $u, v \in A^\infty$, we say that u and v *match* if either $u \leq v$ or $v \leq u$, which we denote by $u \parallel v$, and we say that they *mismatch*, written $u \nparallel v$, otherwise. Finally, for $u, v \in A^\infty$, we denote by $u \wedge v$ their longest common prefix, i.e. the longest word $w \in A^\infty$ such that $w \leq u$ and $w \leq v$.

Let $\mathbb{D}$ be a logical structure. A word over $\mathbb{D}$ is called a $\mathbb{D}$-*data word* (or just *data word*). Note that $\mathrm{Aut}(\mathbb{D})$ naturally acts on $\mathbb{D}^\infty$.

1.3. **Functions and relations.** A (binary) relation between sets X and Y is a subset $R \subseteq X \times Y$. We denote its domain $\mathrm{dom}(R) = \{x \in X \mid \exists y \in Y, (x, y) \in R\}$. It is *functional* if for all $x \in \mathrm{dom}(R)$, there exists exactly one $y \in Y$ such that $(x, y) \in R$. Then, we can also represent it as the function $f_R : \mathrm{dom}(R) \to Y$ such that for all $x \in \mathrm{dom}(R)$, $f(x) = y$ such that $y \in Y$ (we know that such y is unique). f_R can also be seen as a *partial* function $f_R : X \to Y$.

Convention 1.3. In this paper, unless otherwise stated, functions of data words are assumed to be partial, and we denote by $\mathrm{dom}(f)$ the domain of any (partial) function f.

1.4. **Register transducers.** Let $\mathbb{D}$ be a logical structure, and let R be a finite set of variables. We define R-*tests* by the following grammar:

$$\phi ::= P(\bar{t}) \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg \phi$$

where P is a symbol of arity k in the signature of $\mathbb{D}$ and $\bar{t}$ a k-tuple of terms. We denote by $\mathsf{Test}(R)$ the set of R-tests. Terms are defined by either a constant of $\mathbb{D}$ or a variable of R. In other words R-tests are exactly the quantifier-free formulas over the signature of $\mathbb{D}$ using variables in R.

Remark 1.4. We choose tests to be be quantifier-free formulas. However we could have chosen existential first-order formulas without affecting our results. Note that we choose this formalism just for simplicity's sake, and that it does not make any difference for structures which admit quantifier elimination such as $(\mathbb{N}, \{=\})$ or $(\mathbb{Q}, \{<\})$.

A *non-deterministic register transducer* (NRT for short) over $\mathbb{D}$ is a tuple $(Q, R, \Delta, q_0, \overline{c_0}, F)$. Where Q is a finite set of states, R is a finite set of registers, $q_0 \in Q$, $\overline{c_0} \in \Sigma^R$ is a vector of constant symbols, $F \subseteq Q$, and Δ is a finite subset of

$$\underbrace{Q}_{\text{current state}} \times \underbrace{\mathsf{Test}(R \uplus \{\mathsf{input}\})}_{\text{current registers + input data}} \times \underbrace{Q}_{\text{target state}} \times \underbrace{\{\mathsf{keep}, \mathsf{set}, \mathsf{guess}\}^R}_{\text{register operations}} \times \underbrace{R^*}_{\text{output word}}$$

A *non-guessing* transducer (NGRT) has a transition function which is included in $Q \times \mathsf{Test}(R \uplus \{\mathsf{input}\}) \times Q \times \{\mathsf{keep}, \mathsf{set}\}^R \times R^*$. Finally, a *deterministic* transducer (DRT) satisfies an even stronger condition: its transition relation is a function of type $Q \times \mathsf{Test}(R \uplus \{\mathsf{input}\}) \to Q \times \{\mathsf{keep}, \mathsf{set}\}^R \times R^*$.

Remark 1.5. Note that in the definition of a transducer we require that $\mathbb{D}$ contains at least one constant symbol. This is needed for annoying technical reasons, namely in order to initialize registers to some value.

However it is not too damaging since, given a Σ-structure $\mathbb{D}$ of domain D, one can always consider the $\Sigma \uplus \{c_0\}$-structure $\mathbb{D}_\perp$ with domain $D \uplus \{\perp\}$, which is just the structure $\mathbb{D}$ with the extra constant symbol being interpreted as the new element $\perp$, the other relations and constants are unchanged, except naturally for the equality relation which is extended to include $(\perp, \perp)$.

For simplicity's sake we will sometimes talk about structures without mentioning any constant, implicitly stating that we extend the structure to include $\perp$. Also note that this operation of adding a fresh constant does not affect oligomorphicity.

Let T be an NRT given as above. A *configuration* C of T is given by a pair $(q, \bar{d})$ where $q \in Q$ is a state and $d \in \mathbb{D}^R$ is a tuple of data values, hence the group $\mathrm{Aut}(\mathbb{D})$ naturally acts on the configurations of T by not touching the states and acting on the content of the registers pointwise. The *initial configuration* is the pair $C_0 = (q_0, \bar{d}_0)$ with $\bar{d}_0 = \overline{c_0}^{\mathbb{D}}$ being the interpretation of the constants in $\mathbb{D}$. A configuration is called *final* if the state component is in F. Let $C_1 = (q_1, \bar{d}_1), C_2 = (q_2, \bar{d}_2)$ be two configurations, let $d \in \mathbb{D}$ and let $t = (q_1, \phi, q_2, \mathsf{update}, v) \in \Delta$. We say that C_2 is a *successor configuration* of C_1 by reading d through t and producing $w \in \mathbb{D}^{|v|}$ if the following hold:

- for all $r \in R$, if $\mathsf{update}(r) = \mathsf{keep}$, then $\bar{d}_2(r) = \bar{d}_1(r)$
- for all $r \in R$, if $\mathsf{update}(r) = \mathsf{set}$, then $\bar{d}_2(r) = d$
- $w(i) = \bar{d}_2(v(i))$ for all $i \in \{1, \ldots, |v|\}$

Moreover, we write $C_1 \xrightarrow{d, \phi, \mathsf{update}|w}_T C_2$ to indicate that fact. Often we don't mention T (when clear from context), nor ϕ and update, and we simply write $C_1 \xrightarrow{d|w} C_2$. Given a sequence of successor configurations, called a *run*, $\rho = C_1 \xrightarrow{d_1|w_1} C_2 \xrightarrow{d_2|w_2} C_3 \ldots C_n \xrightarrow{d_n|w_n} C_{n+1}$, we write $C_1 \xrightarrow{d_1 d_2 \cdots d_n | w_1 w_2 \cdots w_n} C_{n+1}$. We sometimes even don't write the output $C \xrightarrow{u} C'$ stating that there is a sequence of transitions reading u going from C to C'.

Let $\rho = C_1 \xrightarrow{d_1|v_1} C_2 \ldots C_n \xrightarrow{d_n|v_n} C_{n+1} \ldots$ denote a possibly infinite run. If $C_1 = C_0$, then ρ is called *initial*. If an infinite number of configurations of ρ are final, we say that ρ is *final*. A run which is both initial and final is *accepting*. We say that the run ρ is over the input word $x = d_1 \ldots d_n \ldots$ and produces $w = v_1 \ldots v_n \ldots$ in the output. Then the semantics of T is defined as $[\![T]\!] = \{(x, w) \mid \rho$ is over x, produces y and is accepting$\} \subseteq \mathbb{D}^\omega \times \mathbb{D}^\infty$. An NRT is called *functional* if $[\![T]\!]$ is a (partial) function. Note that in the following we will mainly consider transducers that only produce ω-words. Restricting the accepting runs of a transducer to runs producing infinite outputs is a Büchi condition and can easily be done by adding one bit of information to states.

2. Continuity and computability

2.1. Continuity notions.
We equip the set A^∞ with the usual distance: for $u, v \in A^\infty$, $\|u, v\| = 0$ if $u = v$ and $\|u, v\| = 2^{-|u \wedge v|}$ otherwise. A sequence of (finite or infinite) words $(w_n)_{n \in \mathbb{N}}$ converges to some word w if for all $\epsilon > 0$, there exists $N \geq 0$ such that for all $n \geq N$, $\|w_n, w\| \leq \epsilon$. Given a language $L \subseteq A^\infty$, we denote by $\bar{L}$ its topological closure, *i.e.* the set of words which can be approached arbitrarily close by words of L.

Remark 2.1. Whether the alphabet A is finite or infinite substantially modifies the properties of the metric space A^∞. Indeed when A is finite this space is compact, but it is not when A is infinite.

2.1.1. *Continuity.*

Definition 2.2 (Continuity). A function $f : A^\omega \to B^\omega$ is *continuous* at x if (equivalently):
 (a) for all sequences of words $(x_n)_{n \in \mathbb{N}}$ converging towards $x \in \mathrm{dom}(f)$, where for all $i \in \mathbb{N}$, $x_i \in \mathrm{dom}(f)$, we have that $(f(x_n))_{n \in \mathbb{N}}$ converges towards $f(x)$.
 (b) $\forall i \geq 0, \; \exists j, \; \forall y \in \mathrm{dom}(f), \; |x \wedge y| \geq j \Rightarrow |f(x) \wedge f(y)| \geq i$
The function f is called *continuous* if it is continuous at each $x \in \mathrm{dom}(f)$.

2.1.2. *Cauchy continuity.*
A Cauchy continuous function maps any Cauchy sequence to a Cauchy sequence. One interesting property of Cauchy continuous functions is that they always admit a (unique) continuous extension to the completion of their domain. Since we deal with A^∞ which is complete, the completion of the domain of a function f, denoted $\overline{\mathrm{dom}(f)}$, is simply its closure.

Definition 2.3 (Cauchy continuity). A function $f : A^\omega \to A^\omega$ is *Cauchy continuous* if the image of a Cauchy sequence in $\mathrm{dom}(f)$ is a Cauchy sequence.

Remark 2.4. Any Cauchy continuous function f can be continuously extended over $\overline{\mathrm{dom}(f)}$ in a unique way, which we denote by $\bar{f}$.

2.1.3. *Uniform continuity.*

Definition 2.5 (Uniform continuity). A function $f : A^\omega \to A^\omega$ is *uniformly continuous* if there exists a mapping $m : \mathbb{N} \to \mathbb{N}$ such that:

$$\forall i \geq 0, \forall x, y \in \mathrm{dom}(f), |x \wedge y| \geq m(i) \Rightarrow |f(x) \wedge f(y)| \geq i$$

Such a function m is called a *modulus of continuity*[2] for f. We also say that f is *m-continuous*. Finally, a functional NRT T is *uniformly continuous* when $[\![T]\!]$ is uniformly continuous.

Remark 2.6. In the case of a finite alphabet, and in general for compact spaces, Cauchy continuity is equivalent to uniform continuity, but for infinite alphabets this does not hold anymore. Consider the following function f computable by a DRT over the data alphabet $(\mathbb{N}, \{0, <\})$ and defined by $u0x \mapsto x$, for $u0 \in \mathbb{N}^*$ being a strictly decreasing sequence. Then this function is not uniformly continuous, since two words may be arbitrarily close yet have very different images. However one can check that the image of a Cauchy sequence is indeed Cauchy: let $(x_n)_{n \in \mathbb{N}}$ be a Cauchy sequence in the domain of f. Let us assume without loss of generality that all the x_n's begin with the same letter $i \in \mathbb{N}$. Then, after reading at most $i + 1$ symbols of one of the x_n's, the DRT outputs something. Let $j \in \mathbb{N}$ and let N be such that for all $m, n \geq N$ we have $|x_m \wedge x_n| \geq i + j + 1$. Thus we have $|f(x_m) \wedge f(x_n)| \geq j$, which means that $(f(x_n))_{n \in \mathbb{N}}$ is Cauchy.

We've seen in the previous remark that Cauchy continuity and uniform continuity don't coincide over infinite alphabets. However when dealing with oligomorphic structures we recover some form of compactness, that is compactness of $\mathbb{D}^\infty / \mathrm{Aut}(\mathbb{D})$, which ensures that the two notions do coincide in this case.

Proposition 2.7. *Let $\mathbb{D}$ be an oligomorphic structure and let $f : \mathbb{D}^\omega \to \mathbb{D}^\omega$ be an equivariant function. Then f is uniformly continuous if and only if it is Cauchy continuous.*

Proof. It is clear that a uniformly continuous function is in particular Cauchy continuous. Let $\mathbb{D}$ be an oligomorphic structure and let $f : \mathbb{D}^\omega \to \mathbb{D}^\omega$ be an equivariant function. Let us assume that f is not uniformly continuous. This means that there exists $i \in \mathbb{N}$ and a sequence $(x_n, y_n)_{n \in \mathbb{N}}$ such that for all n, $|x_n \wedge y_n| \geq n$ and $|f(x_n) \wedge f(y_n)| \leq i$. Let us consider the sequence $([x_n], [y_n])_{n \in \mathbb{N}}$ of pairs of elements in $\mathbb{D}^\omega / \mathrm{Aut}(\mathbb{D})$, *i.e.* words are seen up to automorphism. Since $\mathbb{D}^\omega / \mathrm{Aut}(\mathbb{D})$ is compact, we can extract a subsequence (which we also call $([x_n], [y_n])_{n \in \mathbb{N}}$ for convenience) and which is convergent. This means that there are morphisms $(\mu_n)_{n \in \mathbb{N}}$ such that the sequence $(\mu_n(x_n))_{n \in \mathbb{N}}$ (and thus $(\mu_n(y_n))_{n \in \mathbb{N}}$) converges. Hence by interleaving $(\mu_n(x_n))_{n \in \mathbb{N}}$ and $(\mu_n(y_n))_{n \in \mathbb{N}}$, we obtain a converging sequence whose image is divergent, which means that f is not Cauchy continuous. □

Remark 2.8. In order to refine uniform continuity one can study m-continuity for particular kinds of functions m. For instance for $m : i \mapsto i + b$, m-continuous functions are exactly 2^b-Lipschitz continuous functions. Similarly, for $m : i \mapsto ai + b$, m-continuous function are exactly the $\frac{1}{a}$-Hölder continuous functions.

Note that while these notions are interesting in and of themselves, they are very sensitive to the metric that is being used. For instance the metric $d(x, y) = \frac{1}{|x \wedge y|}$ while defining the same topology over words, yields different notions of Lipschitz and Hölder continuity.

---

[2]The usual notion of modulus of continuity is defined with respect to distance, but here we choose to define it with respect to longest common prefixes, for convenience. Given m a modulus of continuity in our setting we can define $\omega : x \mapsto 2^{-m\left(\left\lceil \log_2\left(\frac{1}{x}\right)\right\rceil\right)}$ and recover the usual notion.

2.2. **Computability notions.** Let $\mathbb{D}$ be a data set. In order to reason with computability, we assume in the sequel that the countable set of data values $\mathbb{D}$ we are dealing with has an effective representation, meaning that each element can be represented in a finite way. For instance, this is the case when $\mathbb{D} = \mathbb{N}$. Moreover, we assume that checking if a tuple of values belongs to some relation of $\mathbb{D}$ is decidable. We say that the structure $\mathbb{D}$ is *representable*. Formally, a structure is representable if there exists a finite alphabet A and an injective function $\mathsf{enc} : \mathbb{D} \to A^*$ such that the sets $\{\mathsf{enc}(d) \mid d \in \mathbb{D}\}$, $\{\mathsf{enc}(c) \mid c$ is a constant of $\mathbb{D}\}$ and $\{\mathsf{enc}(d_1)\sharp \cdots \sharp\mathsf{enc}(d_k) \mid (d_1, \ldots, d_k) \in R\}$ are decidable for all predicates R of $\mathbb{D}$ and $\sharp \notin A$. Any infinite word $d_1 d_2 \cdots \in \mathbb{D}^\omega$ can be encoded as the ω-word $\mathsf{enc}(d_1)\sharp\mathsf{enc}(d_2)\sharp \cdots \in (A^*\sharp)^\omega$.

We now define how a Turing machine can compute a function from $\mathbb{D}^\omega$ to $\mathbb{D}^\omega$. We consider deterministic Turing machines whose cells can contain a letter from $A \cup \{\sharp\}$ or a letter from a finite working alphabet. They have three tapes: a read-only one-way input tape on alphabet $A \cup \{\sharp\}$ (containing an encoding of an infinite input data word), a two-way working tape, and a write-only one-way output tape on alphabet $A \cup \{\sharp\}$ (on which they write the encoding of the infinite output data word). Since we always work modulo encoding, for the sake of simplicity, from now on and in the rest of the paper, we assume that each cell of the Turing machine, on the input and output tapes, contain a data $d \in \mathbb{D}$, while cells of the working tape are assumed to contain either a data $d \in \mathbb{D}$ or a letter from the working alphabet. So, instead of saying the input contains the encoding of a data word x, we just say that it contains the input data word x. We discuss in Remark 2.13 how the computability notions we introduce hereafter are sensitive to encodings.

Consider such a Turing machine M and some input data word $x \in \mathbb{D}^\omega$. For any integer $k \in \mathbb{N}$, we let $M(x, k)$ denote the finite output data word written by M on its output tape after reaching cell number k of the input tape (assuming it does). Observe that as the output tape is write-only, the sequence of data words $(M(x, k))_{k \geq 0}$ is non-decreasing, and thus we denote by $M(x)$ the limit content of the output tape.

Definition 2.9 (Computability)**.** Let $\mathbb{D}$ be a representable data domain. A data word function $f : \mathbb{D}^\omega \to \mathbb{D}^\omega$ is *computable* if there exists a deterministic multi-tape machine M such that for all $x \in \mathrm{dom}(f)$, $M(x) = f(x)$. We say that M *computes* f.

Definition 2.10 (Cauchy computability)**.** Let $\mathbb{D}$ be a representable data domain. A data word function $f : \mathbb{D}^\omega \to \mathbb{D}^\omega$ is *Cauchy computable* if there exists a deterministic multi-tape machine M computing f such that for all x in the topological closure $\overline{\mathrm{dom}(f)}$ of $\mathrm{dom}(f)$, the sequence $(M(x, k))_{k \geq 0}$ converges to an infinite word. In other words a Cauchy computable function is a function which admits a continuous extension to the closure of its domain and which is computable. We say that M *Cauchy computes* f.

Definition 2.11 (Uniform computability)**.** Let $\mathbb{D}$ be a representable data domain. A data word function $f : \mathbb{D}^\omega \to \mathbb{D}^\omega$ is *uniformly computable* if there exists a deterministic multi-tape machine M and a computable mapping $m : \mathbb{N} \to \mathbb{N}$ such that M computes f and for all $i \geq 0$ and $x \in \mathrm{dom}(f)$, $|M(x, m(i))| \geq i$. Such a function m is called a *modulus of computability* for f. In that case f is called *m-computable*. We say that M *uniformly computes* f, and also that M *m-computes* f.

Example 2.12. The function g defined in the Introduction (p. 4), for the data domain of integers, is computable. Remind that it is defined on all input words of the form $x = su_1 d_1 su_2 d_2 s \ldots$ such that s occurs infinitely often, and for all i, s does not occur in $u_i d_i$, by

$g(x) = d_1^{|u_1|+1} s d_2^{|u_2|+1} s \dots$. A Turing machine just needs to read the input up to d_1, then output d_1 exactly $|u_1| + 1$ times, and so on for the other pieces of inputs.

In Remark 2.6, the function f is not uniformly continuous but Cauchy continuous. It is actually not uniformly computable but Cauchy computable. As a matter of fact, all the computability notions we define here entails the respective continuity notions defined before. We make this formal in Section 2.3.

Remark 2.13 (Robustness to encoding). When actually representing words over an infinite alphabet, it is not realistic to assume that one letter takes a constant amount of space and can be read in a constant amount of time. Then, which of the many notions introduced above are sensitive to encoding and which ones are more robust?

Let $\mathbb{D}$ be a representable structure, and let $\mathsf{enc} : \mathbb{D} \to A^*$ be its encoding function. Let $f : \mathbb{D}^\omega \to \mathbb{D}^\omega$ be a function and let $f_{\mathsf{enc}} : (A \uplus \sharp)^\omega \to (A^* \uplus \sharp)^\omega$ be defined as $\mathsf{enc}_\sharp \circ f \circ \mathsf{enc}_\sharp^{-1}$, where $\mathsf{enc}_\sharp : d_1 \cdots d_n \mapsto \mathsf{enc}(d_1) \sharp \cdots \sharp \mathsf{enc}(d_n)$. Continuity and computability are robust enough so that f is continuous (resp. computable) if and only if f_{dec} is. Cauchy continuity and computability also fall under this category. In contrast, uniform continuity and uniform computability are very sensitive to encoding. As an example, the function which maps a word to the second letter in the word is *never* uniformly continuous, since the encoding of the first letter may be arbitrarily long. Nevertheless, uniform computability is still relevant notion, as it provides guarantees on the maximal number of input data which need to be read to produce a given number of output data, even though the encoding of those data can be arbitrarily large.

2.3. Computability versus continuity.
In this section, we show that all the computability notions (computability, Cauchy continuity, ...) imply their respective continuity notions. We then give general conditions under which the converse also holds.

2.3.1. *From computability to continuity.*

Theorem 2.14. *Let $f : \mathbb{D}^\omega \to \mathbb{D}^\omega$ and let $m : \mathbb{N} \to \mathbb{N}$, the following implications hold:*
- *f is computable $\Rightarrow$ f is continuous*
- *f is Cauchy computable $\Rightarrow$ f is Cauchy continuous*
- *f is uniformly computable $\Rightarrow$ f is uniformly continuous*
- *f is m-computable $\Rightarrow$ f is m-continuous*

Proof. Assume that f is computable by a deterministic multi-tape Turing machine M. Let x be in the topological closure of $\mathrm{dom}(f)$ and $(x_n)_n$ be a sequence in $\mathrm{dom}(f)$ converging to x. We show that $(f(x_n))_n$ converges to $M(x)$ if $M(x)$ is infinite. For all $k \geq 0$, let p_k the prefix of x of length k. Since $\lim_{n\infty} x_n = x$, for all k, there exists n_k such that for all $m \geq n_k$, $p_k \leq x_m$. As M is a deterministic machine, it implies that $M(x, k) \leq f(x_m)$. So, for all k, $M(x, k) \leq f(x_m)$ for all but finitely many m. It follows that $(f(x_m))_m$ converges to $M(x) = \lim_{k\infty} M(x, k)$ if $M(x)$ is an infinite word. It is the case for all $x \in \mathrm{dom}(f)$ since f is computable, entailing continuity of f. If additionally f is Cauchy computable, then it is also the case for all $x \in \overline{\mathrm{dom}(f)}$, entailing Cauchy continuity of f.

It remains to show the fourth statement, which entails the third. So, let us assume that f is m-computable by some machine M. We show it is m-continuous. Let $i \geq 0$, $x, y \in \mathrm{dom}(f)$ such that $|x \wedge y| \geq m(i)$. We must show that $|f(x) \wedge f(y)| \geq i$. We have

$M(x, m(i)) = M(y, m(i))$ because M is deterministic and $|x \wedge y| \geq m(i)$. By definition of m-computability, we also have $|M(x, m(i))| \geq i$. Since $M(x, m(i)) \leq f(x)$ and $M(y, m(i)) \leq f(y)$, we get $|f(x) \wedge f(y)| \geq i$, concluding the proof. $\qquad \square$

2.3.2. *From continuity to computability.* While, as we have seen in the last section, computability of a function f implies its continuity, and respectively for all the notions of computability and continuity we consider in this paper, the converse may not hold in general. We give sufficient conditions under which the converse holds for f, namely when it has a computable next-letter problem. This problem asks, given as input two finite words $u, v \in \mathcal{D}^*$, to output, if it exists, a data $d \in \mathbb{D}$ such that for all $y \in \mathbb{D}^\omega$ such that $uy \in \mathrm{dom}(f)$, we have $vd \leq f(uy)$. Because of the universal quantification on y, note that d is unique if it exists. Formally:

Next-Letter Problem for f:

Input	$u, v \in \mathbb{D}^*$
Output	$\begin{cases} d \in \mathbb{D} & \text{if for all } y \in \mathbb{D}^\omega \text{ s.t. } uy \in \mathrm{dom}(f), vd \leq f(uy) \\ \texttt{none} & \text{otherwise} \end{cases}$

Theorem 2.15. *Let $f : \mathbb{D}^\omega \to \mathbb{D}^\omega$ be a function with a computable next-letter problem and let $m : \mathbb{N} \to \mathbb{N}$, the following implications hold:*

- *f is continuous $\Rightarrow$ f is computable*
- *f is Cauchy continuous $\Rightarrow$ f is Cauchy computable*
- *f is uniformly continuous $\Rightarrow$ f is uniformly computable*
- *f is m-continuous $\Rightarrow$ f is m-computable*

Proof. Let us assume the existence of a procedure $\mathsf{Next}_f(u, v)$ which computes the next-letter problem. To show the four statements, we show the existence of a deterministic Turing machine M_f common to the four statements, in the sense that M_f computes f if f is continuous, respectively Cauchy computes f is f is Cauchy continuous, etc. The Turing machine M_f is best presented in pseudo-code as follows.

Algorithm 1: Turing machine M_f defined in pseudo-code.

Data: $x \in \mathbb{D}^\omega$
1 $v := \epsilon$;
2 **for** $i = 0$ **to** $+\infty$ **do**
3 $\quad$ $d := \mathsf{Next}_f(x[:i], v)$;
4 $\quad$ **while** $d \neq \textit{none}$ **do**
5 $\quad\quad$ output d;   // write d on the output tape
6 $\quad\quad$ $v := vd$;
7 $\quad\quad$ $d := \mathsf{Next}_f(x[:i], v)$;
8 $\quad$ **end**
$\quad$ /* end while loop when $d = \mathrm{none}$ */
9 **end**

Now, we show that if f is continuous, then M_f computes f, i.e. $M_f(x) = f(x)$ for all $x \in \mathrm{dom}(f)$. First, for all $i \geq 0$, let $v_i = d_{i,1} d_{i,2} \ldots$ be the sequence of data outputted at line 5 at the ith iteration of the for loop. Note that the ith iteration may not exist when the test at line 4 is forever true. In that case, we set $v_i = \epsilon$. By definition of $M_f(x, i)$,

we have[3] $M_f(x, i) = v_0 v_1 \ldots v_i$ for all $i \geq 0$. Moreover, by definition of the next-letter problem, we also have $M_f(x, i) \leq f(x)$. Now, by definition, $M_f(x) = v_0 v_1 v_2 \ldots$. So, if it is infinite, then $M_f(x) = f(x)$. It remains to show that it is indeed true when f is continuous. Suppose it is not the case. Then there exists i_0 such that for all $i \geq i_0$ the call $\mathsf{Next}_f(x[:i], v_0 \ldots v_{i_0})$ returns $\mathtt{none}$ (assume i_0 is the smallest index having this property). Let d such that $v_0 \ldots v_{i_0} d < f(x)$. Then, for all $i \geq i_0$, there exists $\alpha_i \in \mathcal{D}^\omega$ and $d' \neq d$ such that $x[:i]\alpha_i \in \mathrm{dom}(f)$ and $v_0 \ldots v_{i_0} d' < f(x[:i]\alpha_i)$. Clearly, the sequence $(f(x[:i]\alpha_i))_i$, if it converges, does not converge to $f(x)$. Since $(x[:i]\alpha_i)_i$ converges to x, this contradicts the continuity of f.

Consider now the case where f is Cauchy-continuous. It implies that f admits a unique continuous extension $\overline{f}$ to $\overline{\mathrm{dom}(f)}$ (see Remark 2.4). By the first statement we just proved, $M_{\overline{f}}$ computes $\overline{f}$, and by definition of Cauchy-computability, we conclude that M_f Cauchy-computes f.

We finally prove the fourth statement, which implies the third. Suppose that f is m-continuous for some modulus of continuity m. We show that M_f m-computes f. We already proved that M_f computes f (as f is continuous). Let $i \geq 0$ and $x \in \mathrm{dom}(f)$. It remains to show that $|M_f(x, m(i))| \geq i$. Let $j = m(i)$. Since the algorithm M_f calls $\mathsf{Next}_f(x[:j], \cdot)$ until it returns $\mathtt{none}$, we get that $v_0 v_1 \ldots v_j$ is the longest output which can be safely output given only $x[:j]$, i.e. $v_0 \ldots v_j = \bigwedge \{f(x[:j]\alpha) \mid x[:j]\alpha \in \mathrm{dom}(f)\}$. If v_j is infinite, then $|M_f(x, j)| = +\infty$ and we are done. Suppose v_j is finite. Then, there exists $\alpha \in \mathcal{D}^\omega$ such that $x[:j]\alpha \in \mathrm{dom}(f)$ and $f(x) \wedge f(x[:j]\alpha) = v_0 v_1 \ldots v_j = M_f(x, j)$. Since $|x \wedge x[:j]\alpha| \geq j = m(i)$, by m-continuity of f, we get that $|M_f(x, j)| \geq i$, concluding the proof. $\square$

3. Oligomorphic data

In this section we consider a structure $\mathbb{D}$ which is oligomorphic. We will show that in this case one can decide, under reasonable computability assumptions, all the notions of continuity introduced in the previous section, as well as compute the next-letter problem. The first step is to prove characterizations of these properties, and then show that the characterizations are decidable. Let $\mathcal{R} : \mathbb{N} \to \mathbb{N}$ denote the *Ryll-Nardzewski function* of $\mathbb{D}$ which maps k to the number of orbits of k-tuples of data values, which is finite thanks to oligomorphism.

3.1. Characterizing functionality and continuity.
The main goal of this section is to give for NRT characterizations of functionality, continuity and uniform continuity, that consist in small witnesses. These small witnesses are obtained by pumping arguments that rely on the fact that there is only a finite number of configuration orbits.

We start by defining *loop removal*, a tool which will prove useful throughout this section. The main idea is that although no actual loop over the same configuration can be guaranteed over infinite runs, the oligomorphicity property guarantees that a configuration *orbit* will be repeated over long enough runs. A *loop* is a run of the shape $C \xrightarrow{u|v} D \xrightarrow{w|z} \mu(D)$ such that $\mu(C) = C$. The shorter run $C \xrightarrow{\mu(u)|\mu(v)} \mu(D)$ is thus called the run obtained after *removing the loop*.

---

[3]For any ω-word α, we let $\alpha.\epsilon = \alpha$.

Proposition 3.1 (Small run witness). *let T be an* NRT *with k registers and state space Q, and let $C \xrightarrow{u|v} D$ be a run. Then there exists u', v' with $|u'| \leq |Q| \cdot \mathcal{R}(2k)$ such that $C \xrightarrow{u'|v'} D$.*

Proof. The idea behind this proposition is simple: any large enough run must contain a loop and can thus be shortened. Let $u = a_1 \cdots a_n$ with a run $C_1 \xrightarrow{a_1|v_1} C_2 \xrightarrow{a_2|v_2} C_3 \cdots C_n$. Let us assume that $n \geq |Q|\mathcal{R}(2k)$, we want to obtain a shorter run from C_1 to C_n. Let us consider the orbits of the pairs $(C_1, C_1), (C_1, C_2), \ldots, (C_1, C_n)$. Since $n \leq |Q|\mathcal{R}(2k)$, there must be two pairs in the same orbit, *i.e.* there must be two indices $1 \leq i < j \leq n$ and some automorphism μ such that $\mu(C_1, C_i) = (C_1, C_j)$. Hence we obtain the run $\mu(C_1) \xrightarrow{\mu(a_1 \cdots a_{i-1}|v_1 \cdots v_{i-1})} \mu(C_i) = C_1 \xrightarrow{\mu(a_1 \cdots a_{i-1}|v_1 \cdots v_{i-1})} C_j$ which is strictly shorter. $\square$

3.1.1. *Characterization of non-emptiness.* Let an NRA (non deterministic register automaton) be simply an NRT without any outputs. We give a characterization of non-emptiness in terms of small witnesses.

Proposition 3.2. *Let A be an* NRA, *the following are equivalent:*

(1) *A recognizes at least a word*
(2) *there exist $C_0 \xrightarrow{u} C \xrightarrow{v} \mu(C)$ with C being a final configuration, and $\mu \in \text{Aut}(\mathbb{D})$*
(3) *there exist $C_0 \xrightarrow{u} C \xrightarrow{v} \mu(C)$ with $|u|, |v| \leq |Q| \cdot \mathcal{R}(2k)$, C being a final configuration, and $\mu \in \text{Aut}(\mathbb{D})$*

Proof. Let us assume (1) and let x be a word accepted by A. Since $\mathbb{D}$ is oligomorphic, there is only a finite number of orbits of configurations. Thus an accepting run of A over x must go through some accepting configuration orbit infinitely often, and in particular at least twice. Hence (2) holds.

Let us assume that (2) holds. We obtain (3) by using loop removal from Proposition 3.1.

Let us assume (3), then the word $uv\mu(v)\mu^2(v)\cdots$ is accepted by A. $\square$

3.1.2. *Characterizing functionality.* Characterizing functionality is slightly more complicated since we have to care about outputs. Moreover, we need to exhibit patterns involving two runs which makes pumping more involved.

We start with a useful yet quite technical lemma, which will allow us to remove loops while preserving mismatches.

Lemma 3.3 (Small mismatch witness). *There exists a polynomial $P(x, y, z)$ such that the following hold.*

Let T be an NRT, *with k registers, a state space $|Q|$ and a maximal output length L. Let $C_1 \xrightarrow{u|u_1} D_1$ and $C_2 \xrightarrow{u|u_2} D_2$ be runs such that $u_1 \nparallel u_2$. Then there exists u' of length less than $P(\mathcal{R}(4k), |Q|, L)$ and u'_1, u'_2, so that $C_1 \xrightarrow{u'|u'_1} D_1$, $C_2 \xrightarrow{u'|u'_2} D_2$ with $u'_1 \nparallel u'_2$.*

Proof. Let $\rho_1 = C_1 \xrightarrow{u|u_1} D_1$ and $\rho_2 = C_2 \xrightarrow{u|u_2} D_2$ be runs such that $u_1 \nparallel u_2$. We assume that $|u| > P(\mathcal{R}(2k), |Q|, L)$, and we want to show that we can obtain a strictly smaller word with the desired property. Our goal is thus to remove synchronous loops in ρ_1, ρ_2 while preserving the mismatch. A synchronous loop is defined as a loop over the same input

in the product transducer $T \times T$. Let $u_1 = \alpha a_1 \beta_1$, $u_2 = \alpha a_2 \beta_2$ with $a_1 \neq a_2$. We will only consider removing loops that do *not* contain the transitions producing the mismatching letters a_1 or a_2.

We call the *effect* of a loop on ρ_1, the length of the output factor removed from α, in particular the effect is 0 if the factor removed is in β_1 (and symmetrically for ρ_2). Removing loops that have the same effect on ρ_1 and ρ_2 preserves the mismatch.

The only case remaining is when any loop has different effects on ρ_1 and ρ_2. The first step is to show that removing a loop with different effects and which cancels a mismatch ensures a very strong periodicity property on the outputs.

Let us consider the first synchronous loop occuring in ρ_1, ρ_2 which does not contain the transitions producing the mismatching outputs. From the previous cases we can assume that this loop has a non-zero effect on ρ_1 (without loss of generality). Let us assume that the loop occurs after the transition producing the mismatching output in ρ_2. This means that there are no loops occuring before the transition producing a_2, and thus $|\alpha| < L|Q|^2 \cdot \mathcal{R}(4k)$ and that all loops have a null effect on ρ_2. From that we can deduce that there are fewer than $L|Q|^2 \cdot \mathcal{R}(4k)$ loops in total since any loop has to have a non null effect on ρ_1. Thus we can deduce that $|u| \leq (L|Q|^2 \cdot \mathcal{R}(4k))(|Q|^2 \cdot \mathcal{R}(4k))$, which yields a contradiction.

The only case which remains is that all loops occur before the transitions producing the mismatching outputs. We can assume, for u large enough, that there are at least two loops. Let $u_1 = \alpha_1 \beta_1 \beta_1' \gamma_1 \delta_1 \delta_1' \zeta_1 a_1 \eta_1$ and $u_2 = \alpha_2 \beta_2 \beta_2' \gamma_2 \delta_2 \delta_2' \zeta_2 a_2 \eta_2$ with $\alpha_1 \beta_1 \beta_1' \gamma_1 \delta_1 \delta_1' \zeta_1 = \alpha_2 \beta_2 \beta_2' \gamma_2 \delta_2 \delta_2' \zeta_2 = u_1 \wedge u_2$, where $\beta_1 \beta_1', \beta_2 \beta_2'$ denote the outputs produced by the first occurring loop and $\delta_1 \delta_1', \delta_2 \delta_2'$ correspond to the outputs of the second loop. What we mean by that is that removing the first loop gives the outputs $\alpha_1 \lambda(\beta_1) \gamma_1 \delta_1 \delta_1' \zeta_1 a_1 \eta_1$ and $\alpha_2 \lambda(\beta_2) \gamma_2 \delta_2 \delta_2' \zeta_2 a_2 \eta_2$; while removing the second loop gives the outputs $\alpha_1 \beta_1 \beta_1' \gamma_1 \mu(\delta_1) \zeta_1 a_1 \eta_1$ and $\alpha_2 \beta_2 \beta_2' \gamma_2 \mu(\delta_2) \zeta_2 a_2 \eta_2$. Without loss of generality, we assume that $d = |\beta_1'| - |\beta_2'| > 0$. After removing the first loop we obtain $u_1' = \alpha_1 \lambda(\beta_1) \gamma_1 \delta_1 \delta_1' \zeta_1 a_1 \eta_1$ and $u_2' = \alpha_2 \lambda(\beta_2) \gamma_2 \delta_2 \delta_2' \zeta_2 a_2 \eta_2$, and we assume that $u_1' \parallel u_2'$, otherwise we are done. We define $u_1 = \alpha \zeta a_1 \eta_1$ and $u_2 = \alpha \zeta a_2 \eta_2$ where α is the longest word bewteen $\alpha_1 \beta_1 \beta_1' \gamma_1 \delta_1 \delta_1'$ and $\alpha_2 \beta_2 \beta_2' \gamma_2 \delta_2 \delta_2'$, so that ζ is a suffix of both ζ_1 and ζ_2.

Let us assume that $|\zeta| < d$. This means that $|u_1 \wedge u_2| \leq 2L|Q|^2 \cdot \mathcal{R}(4k)$. Since any loop has to produce at least one letter in u_1 or u_2, u must have fewer than $4L|Q|^2 \cdot \mathcal{R}(4k)$ double loops. Thus we get that $|u| < (4L|Q|^2 \cdot \mathcal{R}(4k))(|Q|^2 \cdot \mathcal{R}(4k))$ which gives a contradiction. Thus we can safely assume that $d \leq |\zeta|$. Let v denote the length d suffix of ζ, then we have that $\alpha_1 \lambda(\beta_1) \gamma_1 \delta_1 \delta_1' \zeta_1 v = \alpha_2 \lambda(\beta_2) \gamma_2 \delta_2 \delta_2' \zeta_2$. Thus we have that ζ ends in v^2, assuming that it is long enough. Repeating this we obtain that ζ is in the language $v' v^*$ for some v' suffix of v. Let t be the primitive root of v then we can even say that ζ is in $t' t^*$ for some t' suffix of t. So we obtain that $\zeta a_1 \parallel t' t^\omega$ and $\zeta \parallel t' t^\omega$ and $\zeta a_2 \nparallel t' t^\omega$. Let us consider now removing the second loop corresponding to outputs δ_1', δ_2'. Using the same arguments we get that $|\beta_1'| - |\beta_2'|$ and $|\delta_1'| - |\delta_2'|$ are both multiples of $|t|$. Note moreover that we cannot have $|\delta_1| - |\delta_2| < 0$, otherwise we would conclude that $\zeta a_1 \nparallel t' t^\omega$. Hence we obtain that all loops must have a larger effect on ρ_1 than on ρ_2 and that the effect difference is a multiple of $|t|$. Now that we have established that strong periodicity property of ζa_1, the main idea is the following: remove all loops while preserving a ζ larger than the maximal effect of a loop but still smaller than *e.g.* twice that. Once this is done, repump just enough loops so that the first word is larger than the second word. This is sure to cause a mismatch since $x\zeta \nparallel y\zeta a_2$, for any x, y satisfying $0 < |x| - |y| < |\zeta|$. $\qquad \square$

Proposition 3.4 (Functionality). *There exists a polynomial $P(x, y, z)$ such that the following holds.*

Let $R \subseteq \mathbb{D}^\omega \times \mathbb{D}^\omega$ be given by an NRT *T with k registers, a state space Q and a maximum output length L. The following are equivalent:*

(1) *R is not functional*

(2) *there exist $C_0 \xrightarrow{u|u_1} C_1 \xrightarrow{v|v_1} D_1 \xrightarrow{w|w_1} \mu(C_1)$, $C_0 \xrightarrow{u|u_2} C_2 \xrightarrow{v|v_2} D_2 \xrightarrow{w|w_2} \mu(C_2)$ with C_1, D_2 final, $\mu \in \mathrm{Aut}(\mathbb{D})$ such that $u_1 \nparallel u_2$*

(3) *there exist $C_0 \xrightarrow{u|u_1} C_1 \xrightarrow{v|v_1} D_1 \xrightarrow{w|w_1} \mu(C_1)$, $C_0 \xrightarrow{u|u_2} C_2 \xrightarrow{v|v_2} D_2 \xrightarrow{w|w_2} \mu(C_2)$ with C_1, D_2 final and $|u|, |v|, |w| \leq P(\mathcal{R}(k), |Q|, L)$, $\mu \in \mathrm{Aut}(\mathbb{D})$ such that $u_1 \nparallel u_2$*

Proof. Let us assume that (1) holds, meaning that T has two accepting runs ρ_1, ρ_2 over some word $x \in \mathbb{D}^\omega$ which produce different outputs. Let $C_0, C_1, C_2 \dots$ denote the configurations of ρ_1 and $C_0, C_1', C_2' \dots$ the ones of ρ_2. Let us consider the orbits of pairs of configurations C_i, C_i'. We know that there is a finite number of such orbits. We also know that an infinite number of such pairs is accepting in the first component and an infinite number is accepting in the second component. Thus we can see (ρ_1, ρ_2) as a sequence of the following shape, with C and D' final:

$$(C_0, C_0) \xrightarrow{u_0} (C, C') \xrightarrow{v_0} (D, D') \xrightarrow{u_1} \mu_1(C, C') \xrightarrow{v_1} \nu_1(D, D') \xrightarrow{u_2} \cdots$$

Since the outputs are different and infinite then they mismatch at some position i. Then, there exists n such that $u = u_0 v_0 \cdots u_n v_n$ has produced at least i symbols, both for ρ_1 and ρ_2. Hence we have shown that $C_0 \xrightarrow{u|\alpha_1} \mu_n(C) \xrightarrow{u_{n+1}|\beta_1} \nu_{n+1}(D) \xrightarrow{v_{n+1}|\gamma_1} \mu_{n+1}(C)$, $C_0 \xrightarrow{u|\alpha_2} \mu_n(C') \xrightarrow{u_{n+1}|\beta_2} \nu_{n+1}(D') \xrightarrow{v_{n+1}|\gamma_2} \mu_{n+1}(C')$, and by assumption $|\alpha_1|, |\alpha_2| \geq i$, and thus $\alpha_1 \nparallel \alpha_2$. Hence we have shown that (2) holds.

Let us assume that (2) holds: *i.e.* we have $C_0 \xrightarrow{u|u_1} C_1 \xrightarrow{v|v_1} D_1 \xrightarrow{w|w_1} \mu(C_1)$, $C_0 \xrightarrow{u|u_2} C_2 \xrightarrow{v|v_2} D_2 \xrightarrow{w|w_2} \mu(C_2)$ with C_1, D_2 final, $\mu \in \mathrm{Aut}(\mathbb{D})$ such that $u_1 \nparallel u_2$. We use Lemma 3.3 (small mismatch witness) to obtain that $C_0 \xrightarrow{u'|u_1'} C_1$, $C_0 \xrightarrow{u'|u_2'} C_2$ with $u_1' \nparallel u_2'$ and $|u'|$ small. We get $C_0 \xrightarrow{u'|u_1'} C_1 \xrightarrow{v|v_1} D_1 \xrightarrow{w|w_1} \mu(C_1)$, $C_0 \xrightarrow{u'|u_2'} C_2 \xrightarrow{v|v_2} D_2 \xrightarrow{w|w_2} \mu(C_2)$. We now only have to use some loop removal on v and w, just as in Proposition 3.2, in order to obtain words smaller than $|Q|^2 \mathcal{R}(4k)$.

Showing that (3) implies (1) is the easiest part since the pattern clearly causes non-functionality over the word $uvw\mu(vw)\mu^2(vw)\cdots$. $\qquad\square$

3.1.3. *Characterizing continuity.* Here we characterize continuity and uniform continuity using patterns similar to the one of functionality. Before doing so we introduce two properties of configuration. A configuration is *co-reachable* if there is a final run from it.

For this we define a notion of *critical pattern*.

Definition 3.5. Let T be an NRT. We associate to it the set of critical patterns, denoted $\mathsf{Critical}_T(u, v, w, z, C_1, \mu(C_1), D_1, C_2, \mu(C_2), D_2)$ (T is omitted when clear from context), given by the runs $C_0 \xrightarrow{u|u_1} C_1 \xrightarrow{v|v_1} \mu(C_1) \xrightarrow{w|w_1} D_1$, $C_0 \xrightarrow{u|u_2} C_2 \xrightarrow{v|v_2} \mu(C_2) \xrightarrow{z|w_2} D_2$ so that D_1, D_2 are co-reachable and one of the following holds:

a) $u_1 \nparallel u_2$, or

b) $v_i = \epsilon$ and $u_j \not\Vert u_i w_i$ for $\{i, j\} = \{1, 2\}$, or

c) $v_1 = v_2 = \epsilon$ and $u_1 w_1 \not\Vert u_2 w_2$

Before characterizing continuity and uniform continuity, we show a small critical pattern property.

Claim 3.6 (Small critical pattern)**.** There exists a polynomial $P'(x, y, z)$ such that the following holds.

Let T be an NRT, with k registers, a state space $|Q|$ and a maximal output length L. Let $C_0 \xrightarrow{u|u_1} C_1 \xrightarrow{v|v_1} \mu(C_1) \xrightarrow{w|w_1} D_1$, $C_0 \xrightarrow{u|u_2} C_2 \xrightarrow{v|v_2} \mu(C_2) \xrightarrow{z|w_2} D_2$ be a critical pattern. Then there exists u', v', w', z' of length less than $P'(\mathcal{R}(4k), |Q|, L)$ and $u'_1, v'_1, w'_1, u'_2, v'_2, w'_2$, so that $C_0 \xrightarrow{u'|u'_1} C_1 \xrightarrow{v'|v'_1} \mu(C_1) \xrightarrow{w'|w'_1} D_1$, $C_0 \xrightarrow{u'|u'_2} C_2 \xrightarrow{v'|v'_2} \mu(C_2) \xrightarrow{z'|w'_2} D_2$ is a critical pattern.

Proof. We want to remove loops in u, v, w, z without affecting the mismatches. The idea is to see such a critical pattern as a pair of runs which mismatch and leverage Lemma 3.3. In order to make sure that the loops which are removed do not interfere with the intermediate configurations, we consider runs which are colored. We consider runs which start with red configurations, then the middle parts $C_1 \xrightarrow{v|v_1} \mu(C_1)$ and $C_2 \xrightarrow{v|v_2} \mu(C_2)$ are colored in black and the final parts are colored in crimson again. Using Lemma 3.3, we obtain runs smaller than $P(\mathcal{R}(4k), 3|Q|, L)$. Since the loops have to be removed in the monochromatic parts, we obtain the desired result. $\qquad\square$

Let us give a characterization of continuity and uniform continuity for functions given by an NRT.

Proposition 3.7 (Continuity/uniform continuity)**.** *There exists a polynomial $P'(x, y, z)$ such that the following holds. Let $f : \mathbb{D}^\omega \to \mathbb{D}^\omega$ be given by an NRT T with k registers, a state space Q and a maximum output length L. The following are equivalent:*

(1) *f is not uniformly continuous (resp. continuous)*

(2) *there exists a critical pattern in $\mathsf{Critical}(u, v, w, z, C_1, \mu(C_1), D_1, C_2, \mu(C_2), D_2)$ (resp. with C_1 final).*

(3) *there exists a critical pattern in $\mathsf{Critical}(u, v, w, z, C_1, \mu(C_1), D_1, C_2, \mu(C_2), D_2)$ such that $|u|, |v|, |w|, |z| \le P'(\mathcal{R}(4k), |Q|, L)$ (resp. with C_1 final).*

Proof. Let us assume that (1) holds, meaning that f is not uniformly continuous (resp. not continuous) at some point $x \in \mathbb{D}^\omega$. This means that there exists i such that for any n, there are two accepting runs ρ_1, ρ_2 over x_1, x_2 and producing y_1, y_2 respectively such that $|x \wedge x_1 \wedge x_2| > n$ and $|y_1 \wedge y_2| < i - 1$. Moreover, if f is not continuous, we can even assume that $x_1 = x$ and $\rho_1 = \rho$, some accepting run over x. Let us consider some $n > 2i \cdot |Q|^2 \cdot \mathcal{R}(2k)$. Let $u = x \wedge x_1 \wedge x_2$, since u is large enough we have that some pair of configurations in ρ_1, ρ_2 has to repeat at least $2i$ times, up to automorphism. If f is not continuous, we choose n large enough so that a final configuration appears at least $2i \cdot |Q|^2 \cdot \mathcal{R}(2k)$ times in the first n transitions of ρ. That way we can ensure that some pair of configuration in ρ, ρ_2 repeats at least $2i$ times, up to automorphism, with the configuration of ρ being accepting.

Thus we obtain two sequences: $C_0 \xrightarrow{u_0|v_0} C \xrightarrow{u_1|v_1} \mu_1(C) \cdots \mu_{2i-1}(C) \xrightarrow{u_{2i}|v_{2i}} \mu_{2i}(C)$ and $C_0 \xrightarrow{u_0|w_0} D \xrightarrow{u_1|w_1} \mu_1(D) \cdots \mu_{2i-1}(D) \xrightarrow{u_{2i}|w_{2i}} \mu_{2i}(D)$. Moreover, let $x_1 = u x'_1$, $x_2 = u x'_2$, let $y_1 = v_0 \cdots v_{2i} y'_1$ and $y_2 = w_0 \cdots w_{2i} y'_2$. We do a case analysis; note that the cases are not necessarily mutually exclusive.

Case 1) let us first assume that there is some index $j \in \{1, \ldots, 2i\}$ so that $\mu_j(C) \xrightarrow{u_j|\epsilon}$ $\mu_j(C)$ and $\mu_{j-1}(D) \xrightarrow{u_i|\epsilon} \mu_j(D)$. Since the outputs y_1, y_2 mismatch, we have some prefixes of ρ_1, ρ_2 of the shape $C_0 \xrightarrow{u_0 \cdots u_{j-1}|v_0 \cdots v_{j-1}} \mu_j(C) \xrightarrow{u_j|\epsilon} \mu_j(C) \xrightarrow{x_1''|y_1''} E$ and $C_0 \xrightarrow{u_0 \cdots u_{j-1}|w_0 \cdots w_{j-1}}$ $\mu_j(D) \xrightarrow{u_j|\epsilon} \mu_j(D) \xrightarrow{x_2''|y_2''} F$ with $v_0 \cdots v_{j-1}y_1'' \not\parallel w_0 \cdots w_{j-1}y_2''$. Hence we obtain a critical pattern of shape c).

Case 2) we assume that there are at least i indices $j \in \{1, \ldots, 2i\}$ such that $v_j \neq \epsilon$ and at least i indices such that $w_j \neq \epsilon$. Then we have that $v_0 \cdots v_{2i} \not\parallel w_0 \cdots w_{2i}$ and thus we get a critical pattern of shape a).

Case 3), let us assume (without loss of generality) that there strictly fewer than i indices such that $w_j \neq \epsilon$. This means that ther must be at least i indices such that $v_j \neq \epsilon$, otherwise we refer back to case 1). Let us consider a prefix of ρ_2 $C_0 \xrightarrow{u_0 \cdots u_{2i}|w_0 \cdots w_{2i}} \mu_{2i}D \xrightarrow{x_2''|y_2''}$ such that $v_0 \cdots v_{2i} \not\parallel w_0 \cdots w_{2i}y_2''$. Let j be such that $w_j = \epsilon$, then we add the loop corresponding to index j after the configuration $(\mu_{2i}(C), \mu_{2i}(D))$. Doing this may modify the mismatching letter of y_2'' as to cancelling the mismatch, however if it is the case, we can simply add the loop twice, which guarantees that the mismatch is preserved. Thus we obtain a critical pattern of shape b).

If we assume that (2) holds then Claim 3.6 gives us (3).

Let us assume that (3) holds. Then f is discontinuous at $x = uv\mu(v)\mu^2(v) \cdots \in \overline{\mathrm{dom}(f)}$ and is thus not uniformly continuous. Moreover, if C_1 is final we have that f is discontinuous at $x \in \mathrm{dom}(f)$, and hence f is not continuous. □

3.2. Deciding functionality, continuity and computability.
We use a key property of oligomorphic structures, namely that orbits can be defined by first order formulas.

Let $\mathbb{D} = (D, \Sigma^{\mathbb{D}})$ be an oligomorphic structure. We denote by $\mathsf{FO}[\Sigma]$ the set of first-order formulas over signature Σ, and just FO if Σ is clear from the context.

Proposition 3.8 ([Boj19, Lemma 4.11]). *Let $\mathbb{D}$ be an oligomorphic structure and let k be a natural number. Any orbit of an element of $\mathbb{D}^k$ is first-order definable.*

We say that $\mathbb{D}$ is *decidable* if its Ryll-Nardzewski function is computable and $\mathsf{FO}[\Sigma]$ has decidable satisfiability problem over $\mathbb{D}$. Moreover, we say that $\mathbb{D}$ is *polynomially decidable* if an orbit of $\mathbb{D}^k$ can be expressed by an FO formula of polynomial size in k and the FO satisfiability problem is decidable in PSpace. One (but not us) could easily define a similar notion of exponentially decidable, or f-decidable for some fixed complexity function f.

Roughly speaking the main automata problems which we will consider (emptiness, functionality, continuity, *etc*) will be PSpace (resp. decidable) whenever the structure $\mathbb{D}$ is polynomially decidable (resp. decidable).

3.2.1. *Computing the next letter.* In this section, we show how to compute the next letter problem for NRT over a decidable representable oligomorphic structure $\mathbb{D}$. By Theorems 2.15 and 2.14, this entails that continuity and computability coincide for functions defined by transducers over decidable representable oligomorphic structures, as stated in Theorem 3.11 below.

Before tackling the next letter problem, we consider the emptiness problem of register automata.

Theorem 3.9. *Let $\mathbb{D}$ be a decidable (resp. polynomially decidable) oligomorphic structure. The emptiness problem for* NRA *is decidable (resp. in* PSPACE*).*

Proof. We show the result in case of a polynomially decidable structure, the more general case can be obtained by forgetting about complexity. Let $\mathbb{D}$ be a polynomially decidable oligomorphic structure and let T be an NRA with k registers and state space Q. Since $\mathbb{D}$ is polynomially decidable, any orbit of $\mathbb{D}^k$ can be represented by a formula of size polynomial in k. This means that $\mathcal{R}(k)$ is exponential. Using the non-emptiness characterization from Proposition 3.2, we only need to find a run of length polynomial in Q and $\mathcal{R}(k)$. The idea is to use a counter bounded by this polynomial, using space in $\log(|Q|\mathcal{R}(k))$, and execute an NPSPACE algorithm which will guess a run of the automaton and update the type of configurations in space polynomial in k and $\log(|Q|)$. Simulating the run goes like this: first the type of the configuration is initialized to $q_0, (d_0, \ldots, d_0)$. Then a new type $\phi(y_1, \ldots, y_k)$ is guessed, as well as a transition which we see as given by a state q and a formula $\psi(x_1, \ldots, x_k, y_1, \ldots, y_k)$. We check that the transition is valid by deciding the satisfiability of $\exists y_1, \ldots, y_k\ \psi(d_0, \ldots, d_0, y_1, \ldots, y_k) \wedge \phi(y_1, \ldots, y_k)$ which can be done in PSPACE, by assumption. We thus move to the new configuration type given by q, ϕ and we continue the simulation. At some point when q is final, we keep the configuration type in memory and guess that we will see it again. $\qquad\square$

Lemma 3.10. *Let $f : \mathbb{D}^\omega \to \mathbb{D}^\omega$ be a function defined by an* NRT *over a decidable representable oligomorphic structure $\mathbb{D}$. Then, its next letter problem is computable.*

Proof. In the next letter problem we get as input two words $u, v \in \mathbb{D}^*$. Our first goal is to decide if there exists $d \in D$ such that $f(u\mathbb{D}^\omega) \subseteq vd\mathbb{D}^\omega$. In other words we want to know if there exist two runs $C_0 \xrightarrow{u|u_1} C_1 \xrightarrow{w|w_1} D_1$, $C_0 \xrightarrow{u|u_2} C_2 \xrightarrow{w|w_2} D_2$ such that $|u_1 w_1|, |u_2 w_2| > |v|$ and either $|u_1 w_1 \wedge v| < |v|$ or $|u_1 w_1 \wedge u_2 w_2| \leq |v|$. The existence of such runs only depends on the type of u, v, hence we can define a an automaton which simulates T and starts by reading some input of type of u and checks whether there is a mismatch occurring before the $|v|$ outputs. Thus we reduce the non-existence of a next letter to the emptiness of an automaton, which is decidable.

Once we know that such a next letter exists, we only have to simulate any run of T over u, and see what the $|v| + 1^{\text{th}}$ output is (note that we can avoid ϵ-producing loops). To be able to simulate T over u, we use the fact that $\mathbb{D}$ is representable and decidable. For every transition that we want to take, from decidability we can check whether the transition is possible. Then, once we know the transition is possible, we can enumerate the representations of elements of $\mathbb{D}$ and check that they satisfy the transition formula. $\qquad\square$

As a direct corollary of Lemma 3.10, Theorem 2.15 and Theorem 2.14, we obtain:

Theorem 3.11. *Let $f : \mathbb{D}^\omega \to \mathbb{D}^\omega$ be a function defined by an* NRT *over a decidable oligomorphic structure $\mathbb{D}$, and let $m : \mathbb{N} \to \mathbb{N}$ be a total function. Then,*
(1) *f is computable iff f is continuous*
(2) *f is uniformly computable iff f is uniformly continuous*
(3) *f is m-computable iff f is m-continuous*

3.2.2. *Deciding functionality, continuity and computability.*

Theorem 3.12. *Given a decidable (resp. polynomially decidable) oligomorphic structure $\mathbb{D}$ functionality, continuity and uniform continuity are decidable (resp.* PSpace-*c) for functions given by* NRT. *As a consequence, if $\mathbb{D}$ is representable, then computability and uniformity computability are decidable (resp.* PSpace-*c).*

Proof. The proofs are very similar, whether we consider functionality, continuity or uniform continuity. Let us show the result for functionality. Moreover we assume that $\mathbb{D}$ is polynomially decidable, the argument in the more general case can easily be obtained just by forgetting about complexity.

Let us consider an NRT T with k register, state space Q and maximum output length L. We want to show that we can decide the existence of two runs with a mismatch. From the characterization given in Proposition 3.4, we know that the pattern we are looking for is small. We consider a counter bounded by the value $3P(\mathcal{R}(4k), |Q|, L)$, which can be represented using polynomial space because $\mathcal{R}(4k)$ is exponential in k ($\mathbb{D}$ is polynomially decidable). Our goal is to simulate T and exhibit a pattern of length bounded by that counter. As we have seen before, we can easily simulate runs of T in PSpace. The additional difficulty here is that at some point we have to check that two output positions mismatch. We use two additional counters which will ensure that the two mismatching outputs correspond to the same position. Let us now describe how the algorithm goes, in a high-level manner. We start by initializing two runs in parallel, as well as our counters. We keep in memory the $2k$-type of the two configurations, which can be done in polynomial space since $\mathbb{D}$ is polynomially decidable. We keep guessing in parallel two transitions for our runs and updating the $2k$-type using the fact that satisfiability of FO is in PSpace. Every time a run outputs some letter, its counter is incremented. At some point we may guess that we output the mismatching value in one of the runs, in which case we stop the counter corresponding to that run. We crucially also need to be able to check later that the value output mismatches. In order to do this we keep in memory a $2k+1$-type, always keeping the value which we output. At some point we output the second mismatching position, we check that the counters coincide and that the outputs are indeed different, which is given by the $2k+1$-type. In parallel, we also have to check that we reach some final configurations and that some configuration repeats. To do this we need to keep one or two additional $2k$-type in memory, which again can easily be done in PSpace.

The approach for continuity and uniform continuity is exactly the same except that the patterns of Proposition 3.7 are slightly more involved. Moreover we also need to decide on the fly that the configurations reached are co-reachable. This can be done exactly like for non-emptiness of automata in PSpace.

Finally, the PSpace lower bound is obtained by reducing the problems to emptiness of register automata over $(\mathbb{N}, \{=\})$, which is PSpace-c [DL09]. Since the data domains are countable, they can always simulate $(\mathbb{N}, \{=\})$, and the proofs of [EFR20] can easily be adapted. □

Theorem 3.13. *For relations given by* NRT *over* $(\mathbb{Q}, \{<\})$ *deciding functionality, continuity/computability and uniform continuity/uniform computability are* PSpace-*complete.*

Proof. We only need to argue that $(\mathbb{Q}, \{<\})$ is representable and polynomially decidable. Clearly rational numbers are representable. Moreover, since $(\mathbb{Q}, \{<\})$ is homogeneous, it admits quantifier elimination, which means that any k-type can be defined by a formula

polynomial in k (actually linear). Indeed a type is just given by the linear order between the free variables. Moreover, satisfiability of first-order logic over $(\mathbb{Q}, \{<\})$ is in PSPACE.   □

4. A NON OLIGOMORPHIC CASE: $(\mathbb{N}, \{<, 0\})$

We now turn to the study of the case of natural numbers equipped with the usual order. Such domain is not oligomorphic (cf Example 1.2), so there might not exist loops in the transducer, as there are infinitely many orbits of configurations.

Notation 4.1. For simplicity, in the rest of this section, $(\mathbb{N}, \{<, 0\})$ and $(\mathbb{Q}_+, \{<, 0\})$ (where $\mathbb{Q}_+$ is the set of rational numbers which are greater than or equal to 0) are respectively denoted $\mathbb{N}$ and $\mathbb{Q}_+$.

We thus need to study more precisely which paths are iterable, i.e. can be repeated unboundedly or infinitely many times. We show that such property only depends on the relative order between the registers, i.e. on the type of the configurations, seen as belonging to $\mathbb{Q}_+$, where the type of a configuration is an FO-formula describing its orbit (cf Proposition 3.8 and Section 4.2). More generally, the fact that $\mathbb{N}$ is a subdomain of $\mathbb{Q}_+$, along with the property that any finite run in $\mathbb{Q}_+$ corresponds to a run in $\mathbb{N}$ (by multiplying all involved data by the product of their denominator), allows us to provide a characterisation of continuity and uniform continuity which yields a PSPACE decision procedure for those properties.

4.1. On loop iteration.

Example 4.2.
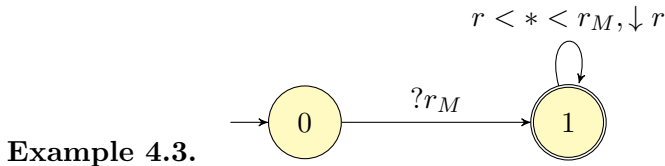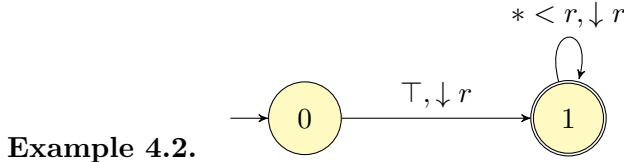
Example 4.3.

The NRA of Example 4.2 is non-empty in $\mathbb{Q}_+$, since it accepts e.g. the word $1 \frac{1}{2} \ldots \frac{1}{n} \ldots$. However, it is empty in $\mathbb{N}$. Indeed, any data word compatible with its only infinite run necessarily forms an infinite descending chain, which is impossible in $\mathbb{N}$. Similarly, in Example 4.3, the NRA initially guesses some upper bound B which it stores in r_M, and then asks to see an infinite increasing chain which is bounded from above by B. This is possible in $\mathbb{Q}_+$, but not in $\mathbb{N}$.

That is why we need to study more closely what makes a given path ω-*iterable*, i.e. that can be taken an infinite number of times. To characterise continuity, we will also need the weaker notion of *iterable* path, i.e. of a path that can be taken arbitarily many times over finite inputs which are increasing for the prefix order. For instance, the loop in Example 4.2 is not iterable: the first letter in the input sets a bound on the number of times it can be

taken. The loop in Example 4.3 is iterable: it suffices to guess bigger and bigger values of the initial upper bound. However, there can be no infinite run which contains infinitely many occurrences of such loop, as the value that is initially guessed for a given run sets a bound on the number of times the loop can be taken, so it is not ω-iterable.

We show that the notions of iterability and ω-iterability are both characterised by properties on the order between registers of a pair of configurations, which can be summed up into a type, hence opening the way to deciding such properties.

4.2. $\mathbb{Q}$-**types.** In our study, the relative order between registers plays a key role. Such information is summed up by the type of the configuration, interpreted as a configuration in $\mathbb{Q}_+$.

Since we will need to manipulate different types of copies of some set of registers, we adopt the following convention:

Convention 4.4. In the following, we assume that for a set of registers R, R_1 and R_2 are two disjoint copies of R, whose elements are respectively r_1 and r_2 for $r \in R$. Similarly, R' is a primed copy of R, whose elements are r' for $r \in R$. Note that the two can be combined to get R'_1, R'_2. Note also that primes and indices are also used as usual notations, but no ambiguity should arise.

Definition 4.5. For a configuration $C : R \to \mathbb{N}$, we define $\tau(C)$ as $\tau_{\mathbb{Q}_+}(C)$ the type of the configuration in $\mathbb{Q}_+$, i.e. an FO-formula describing its orbit (such FO-formula exists by Proposition 3.8 since $\mathbb{Q}_+$ is oligomorphic). Note that such type can be represented e.g. by the formula $p = q \wedge \bigwedge_{\bowtie \in \{<,>,=\}} \bigwedge_{r,r' \in R | C(r) \bowtie C(r')} r \bowtie r' \wedge \bigwedge_{r \in R | C(r) = 0} r = 0$. Thus, the type specifies the current state, and summarises the information of the order between registers, as well as whether they are equal to 0 or not.

We will also need to have access to the relative order between registers of two configurations. Thus, for two configurations $C_1, C_2 : R \to \mathbb{N}$, we define $\sigma(C_1, C_2) = \tau_{\mathbb{Q}_+}(C_1 \uplus C'_2)$, where $C_1 \uplus C'_2$ is the disjoint union of C_1 and of a primed copy of C_2, so that the registers of C_2 can be distinguished from those of C_1. We then have, for all registers $r, s \in R$ and all relations $\bowtie \in \{<, >, =\}$ that $\sigma(C_1, C_2) \Rightarrow r \bowtie s'$ if and only if $C_1(r) \bowtie C_2(s)$.

Remark 4.6. Recall that by definition of an orbit, we have that for any configurations C and C' such that $\tau(C) = \tau(C')$, there exists a morphism $\mu \in \text{Aut}(\mathbb{Q}_+)$ such that $\mu(C) = C'$.

The core property is the following:

Property 4.7. Let R be a set of registers, and let σ be a $\mathbb{Q}$-type defined over $R \uplus R'$, where R' is a primed copy of R. We say that σ has the property $\star$ for the set of registers $X \subseteq R$ if:

- for all $r \in X$, $\sigma \Rightarrow r \leq r'$
- for all $r, s \in X$, if $\sigma \Rightarrow s = s'$ and $\sigma \Rightarrow r \leq s$, then $\sigma \Rightarrow r = r'$

By extension, for two configurations C and C' over R, we say that C and C' have the property $\star$ for the set of registers $X \subseteq R$ if $\sigma(C, C')$ has the $\star$ property for X.

Finally, when $X = R$, we simply state that σ has the $\star$ property.

Such property ensures, for the considered subset of registers, that they cannot induce infinite descending chains nor infinite bounded increasing chains, if a run loops over configurations whose pairwise type is σ.

4.3. **Relations between machines over $\mathbb{N}$ and over $\mathbb{Q}_+$.** There is a tight relation between machines operating over $\mathbb{N}$ and over $\mathbb{Q}_+$. First, since $\mathbb{N}$ is a subdomain of $\mathbb{Q}_+$, runs in $\mathbb{N}$ are also runs in $\mathbb{Q}_+$. Over finite runs, by multiplying all data by the product of their denominators, we can get the converse property.

Proposition 4.8. *Let $X \subset_f \mathbb{Q}_+$ be a finite subset of $\mathbb{Q}_+$. There exists an automorphism $\lambda \in \mathrm{Aut}(\mathbb{Q}_+)$ such that $\lambda(X) \subset \mathbb{N}$, $\lambda(\mathbb{N}) \subseteq \mathbb{N}$ and λ is non-contracting, i.e. for all $x, y \in \mathbb{Q}_+$, $|\lambda(x) - \lambda(y)| \geq |x - y|$.*

Proof. By writing $X = \left\{ \frac{p_1}{q_1}, \ldots, \frac{p_n}{q_n} \right\}$, let $K = \prod_i q_i$. Then $\lambda : d \mapsto Kd$ is an automorphism satisfying the required properties. $\qquad\square$

We then get the following:

Proposition 4.9. *Let A be a NRA, and let ν and ν' be $\mathbb{Q}$-types.*
If there exists two configurations $C, C' : R \to \mathbb{Q}_+$ such that $\tau(C) = \nu$, $\tau(C') = \nu'$ and a data word $v \in \mathbb{Q}_+^*$ such that $C \xrightarrow{u} C'$, then there also exists two configurations D, D' and a data word w which satisfy the same properties, and which belong to $\mathbb{N}$, i.e. such that $D, D' : R \to \mathbb{N}$ and $w \in \mathbb{N}^*$.

Proof. Let A be a NRA, and assume that $C \xrightarrow{u} C'$. By applying Proposition 4.8 to $X = C(R) \cup C'(R) \cup \mathsf{data}(\mathsf{u})$, we get that $\lambda(C) \xrightarrow{\lambda(u)} \lambda(C')$ is also a run of A, and $D = \lambda(C)$, $D' = \lambda(C')$ and $w = \lambda(u)$ satisfy the required properties. $\qquad\square$

Remark 4.10. As a corollary, we obtain that for any NRA A over finite words, $L_{\mathbb{N}}(A) \neq \varnothing$ if and only if $L_{\mathbb{Q}_+}(A) \neq \varnothing$.

Note that such property does not hold over infinite runs, as witnessed by Examples 4.2 and 4.3. The property $\star$ ensures that a loop can be iterated in $\mathbb{N}$, as shown in the next key proposition:

Proposition 4.11. *Let T be an NRT and assume that $B \xrightarrow{u} B'$ following some sequence of transitions π, where $\tau(B) = \tau(B')$, $u \in \mathbb{Q}_+^*$ and B and B' have the property $\star$. Then there exists an infinite run $D \xrightarrow{x}$ over the sequence of transitions π^ω, with $x \in \mathbb{N}^\omega$ and $\tau(D) = \tau(B)$.*

Before showing this proposition, let us introduce some intermediate notions:

Definition 4.12. Let C, C' be two configurations over $\mathbb{Q}_+$ such that $\tau(C) = \tau(C')$. We say that C' is *wider* than C whenever for any $r, s \in R$, we have:
- $|C'(s) - C'(r)| \geq |C(s) - C(r)|$
- $C'(r) \geq C(r)$

Note that the second item of the definition is required to ensure that the interval between $C'(r)$ and 0 is also wider than the interval between $C(r)$ and 0.

Proposition 4.13. *Let σ be a type over $R \uplus R'$ such that $\sigma_{|R} = \sigma_{|R'}$. If σ has the $\star$ property, then there exist two configurations C and C' in $\mathbb{N}$ such that $\sigma(C, C') = \sigma$ and such that C' is wider than C.*

Proof. First, assume that there exists some register $r_0 \in R$ such that $\sigma \Rightarrow r = 0$ (thus $\sigma \Rightarrow r' = 0$, since we assumed that $\sigma_{|R} = \sigma_{|R'}$). If this is not the case, consider instead the type $\sigma_0 = \sigma \wedge r_0 = 0 \wedge r_0' = 0$ over $R \uplus \{r_0\}$. Indeed, the reader can check that if C and C'

are two configurations in $\mathbb{N}$ such that $\sigma(C, C') = \sigma_0$ and C' is wider than C', then $C_{|R}$ and $C'_{|R}$ are two configurations in $\mathbb{N}$ such that $\sigma(C, C') = \sigma$ and $C'_{|R}$ is wider than $C_{|R}$.

Now, for a pair of configurations (C, C'), define its shrinking intervals: $S(C, C') = \{(r, s) \mid |C'(s) - C'(r)| < |C(s) - C(r)|\}$, and say that (C, C') has $k = |S(C, C')|$ shrinking intervals. We need to show that given a pair of configurations B and B' such that $\sigma(B, B') = \sigma$, if they have $k > 0$ intervals that shrink, we can exhibit a pair of configurations C and C' such that $\sigma(C, C') = \sigma$ and which has $l < k$ intervals that shrink.

Thus, let B and B' be two configurations such that $\sigma(B, B') = \sigma$ and which have $k > 0$ shrinking intervals. Then, let $(r, s) \in S(B, B')$; w.l.o.g. assume that $B'(s) \geq B'(r)$. As B and B' have the same type, we get $B(s) \geq B(r)$. Moreover, as $(r, s) \in S$, $B(s) \neq B(r)$, so $B(s) > B(r)$, which implies $B'(s) > B'(r)$. Finally, we have that $B'(s) > B(s)$. Indeed, since σ has the $\star$ property, we have that $B'(s) \geq B(s)$, and moreover if we had $B'(s) = B(s)$, we would get that $B(r) = B'(r)$ as $B(r) \leq B(s)$, which would mean that $(r, s) \notin S(B, B')$.

Finally, let $M = \max \left(\{B(t) \mid t \in R, B(t) < B'(s)\} \cup \{B'(t) \mid t \in R, B'(t) < B'(s)\} \right)$ be the maximum value seen in B and B' which is lower than $B'(s)$. Note that we have $M \geq B(r)$, $M \geq B'(r)$ and $M \geq B(s)$.

Now, let $c = \left(B(s) - B(r)\right) - \left(B'(s) - B'(r)\right)$. As $(r, s) \in S$, $c > 0$. Then, consider the morphism $\mu \in \mathrm{Aut}(\mathbb{Q}_+)$ defined as $\begin{cases} x \in [0; M] \mapsto x \\ x \in]M; B'(s)] \mapsto M + \frac{B'(s) + c - M}{B'(s) - M}(x - M) \\ x \in]B'(s); +\infty[\mapsto x + c \end{cases}$

It can be checked that for all $x, y \in \mathbb{Q}_+$, $|\mu(y) - \mu(x)| \geq |y - x|$, so $|S(\mu(B), \mu(B'))| \leq |S(B, B')|$. Now, we have that $(r, s) \notin S(\mu(B), \mu(B'))$. Indeed, $\mu(B(s)) = B(s)$, $\mu(B(r)) = B(r)$ and $\mu(B'(r)) = B'(r)$ since $B(s), B(r), B'(r) \in [0; M]$. Finally, $\mu(B'(s)) = B'(s) + c$. Overall, we get that $\left(\mu(B(s)) - \mu(B(r))\right) - \left(\mu(B'(s)) - \mu(B'(r))\right) = \left(B(s) - B(r)\right) - \left(B'(s) + c - B'(r)\right) = c - c = 0$, which means that $|B'(s) - B'(r)| = |B(s) - B(r)|$, so $(r, s) \notin S(\mu(B), \mu(B'))$: $|S(\mu(B), \mu(B'))| < |S(B, B')|$.

Since $\mu \in \mathrm{Aut}(\mathbb{Q}_+)$, we get that $(\mu(B), \mu(B'))$ is such that $\sigma(\mu(B), \mu(B')) = \sigma$, so we exhibited a pair of configurations which has $l < k$ intervals that shrink.

By iteratively applying this process to (B, B') until no shrinking intervals remain, we get a pair (C, C') which is such that $\sigma(C, C') = \sigma$ and C' is wider than C. Now, by multiplying all data in C and C' by the product of their denominators, we get two configurations D and D' which are in $\mathbb{N}$ such that $\sigma(D, D') = \sigma$ and D' is wider than D. $\qquad\square$

We finally need the following technical result:

Proposition 4.14. *Let C and C' be two configurations in $\mathbb{N}$ such that C' is wider than C. Then, there exists some morphism $\mu \in \mathrm{Aut}(\mathbb{Q}_+)$ such that $C' = \mu(C)$ and $\mu(\mathbb{N}) \subseteq \mathbb{N}$.*

Proof. Let $\{a_0, \dots, a_k\} = C(R) \cup \{0\}$ and $\{a'_0, \dots, a'_k\} = C(R) \cup \{0\}$, with $0 = a_0 < \cdots < a_k$ and $a'_0 < \cdots < a'_k$. Note that since $\tau(C) = \tau(C')$, we indeed have that $|C(R) \cup \{0\}| = |C'(R) \cup \{0\}|$; moreover, since C' is wider than C, we have that for all $0 \leq i < k$, $a'_{i+1} - a'_i > a_{i+1} - a_i$. Consider the following easy lemma (the proof is in Appendix A for completeness):

Lemma 4.15. *Let $a, b, c, d \in \mathbb{N}$ be such that $a < b$, $c < d$ and $d - c \geq b - a$. Then, there exists a function $f : [a; b] \to [c; d]$ which is increasing and bijective, and such that $f([a; b] \cap \mathbb{N}) \subseteq \mathbb{N}$.*

Then, apply it to each interval $[a_i; a_{i+1}]$ to get a family of increasing and bijective functions $(\mu_i)_{0 \leq i < k}$ which are such that $\mu_i([a_i; a_{i+1}]) = [a'_i; a'_{i+1}]$ and $\mu([a_i; a_{i+1}] \cap \mathbb{N}) \subseteq \mathbb{N}$. Then, let $\mu_k : x \in [a_k; +\infty[\mapsto x + a'_k - a_k$. We get that $\mu = \cup_{0 \leq i \leq k} \mu_i \in \mathrm{Aut}(\mathbb{Q}_+)$ is such that $C' = \mu(C)$ and satisfies $\mu(\mathbb{N}) \subseteq \mathbb{N}$. $\qquad\square$

We are now ready to prove Proposition 4.11:

Proof of Proposition 4.11. Let T be an NRT and assume that $B \xrightarrow[\pi]{u} B'$ following some sequence of transitions π, where $\tau(B) = \tau(B')$, $u \in \mathbb{Q}_+^*$ and B and B' have the property $\star$.

Let $\sigma = \sigma(B, B')$. By Proposition 4.13, we know that there exist two configurations C and C' in $\mathbb{N}$ such that $\sigma(C, C') = \sigma$ and C' is wider than C. Let $\nu \in \mathrm{Aut}(\mathbb{Q}_+)$ be some morphism such that $\nu(B) = C$ and $\nu(B') = C'$ (such morphism exists since $\sigma(B, B') = \sigma(C, C')$). Then, we have that $C \xrightarrow[\pi]{\nu(u)} C'$. By multiplying all involved data by their common denominator (cf Propostion 4.8), we get two configurations D and D', along with some data word v, all belonging to $\mathbb{N}$, such that $D \xrightarrow[\pi]{v} D'$, and such that D' is wider than D. By Proposition 4.14, there exists a morphism $\mu \in \mathrm{Aut}(\mathbb{Q}_+)$ such that $\mu(D) = D'$ and $\mu(\mathbb{N}) \subseteq \mathbb{N}$. Thus, by letting $x = v\mu(v)\mu^2(v)\ldots$, we get that $D \xrightarrow[\pi]{v} \mu(D) \xrightarrow[\pi]{\mu(v)} \mu^2(D)\ldots$ is a run over $x \in \mathbb{N}^\omega$ over the sequence of transitions π^ω, and $\tau(D) = \tau(B)$. $\qquad\square$

A last important property is the following:

Proposition 4.16. *Let A be a NRA, and assume that $C \xrightarrow{u} C'$ and that $C'' \xrightarrow{v}$, where $\tau(C') = \tau(C'')$, $u \in \mathbb{Q}_+^*$ and $v \in \mathbb{N}^\omega$. Then there exist $w \in \mathbb{N}^*$, $x \in \mathbb{N}^\omega$ and two configurations $D, D' : R \to \mathbb{N}$ such that $D \xrightarrow{w} D' \xrightarrow{x}$, $\tau(C) = \tau(D)$ and $\tau(C') = \tau(D')$.*

Proof. Since $\tau(C') = \tau(C'')$, there exists some $\mu \in \mathrm{Aut}(\mathbb{Q}_+)$ such that $\mu(C') = C''$, thus $\mu(C) \xrightarrow{\mu(u)} C''$. Now, by applying Proposition 4.8 to $X = (\mu(C))(R) \cup C''(R) \cup \mathsf{data}(\mu(u))$, we get that $\lambda(\mu(C)) \xrightarrow{\lambda(\mu(u))} \lambda(C'') \xrightarrow{\lambda(v)}$ satisfies the required property. $\qquad\square$

4.4. Emptiness of automata.

Proposition 4.17 (Non-emptiness). *Let A be an NRA over $\mathbb{N}^\omega$. The following are equivalent:*

(1) $L(A)$ *is non-empty*

(2) *there exist two runs whose input words belong to $\mathbb{Q}_+^*$, which are as follows:*
   (a) $C_0 \xrightarrow{u} C$
   (b) $D \xrightarrow{v} D'$ *with $\tau(D) = \tau(D') = \tau(C)$, D is a final configuration, and $\sigma = \sigma(D, D')$ satisfies $\star$ property.*

Proof. Assume (2) holds. The result easily follows from Propositions 4.9, 4.11 and 4.16.

Assume now that $L(A)$ is non-empty. Let $\rho = C_0 \xrightarrow{d_0} C_1 \xrightarrow{d_1} C_2 \ldots$ be an accepting run over input $x = d_0 d_1 \ldots$ in A (where $C_0 = (q_0, \bar{0})$). For each $i \geq 0$, let $\nu_i = \tau(C_i)$. As ρ is acccepting and there are only finitely many types, we get that there exists some type ν such that the state is accepting and $(q_i, \nu_i) = (q, \nu)$ for infinitely many $i \in \mathbb{N}$. Let $(C_j)_{j \in \mathbb{N}}$ be an infinite subsequence of C_i such that for all j, $\tau(C_j) = \nu$. Now, colour the set of unordered pairs as follows: $c(\{\tau(C_j), \tau(C_k)\}) = \sigma_{jk}(C_j, C_k)$ (where we assume w.l.o.g. that $j < k$)

By Ramsey's theorem, there is an infinite subset such that all pairs have the same colour σ. Let $(C_k)_{k \in \mathbb{N}}$ be an infinite subsequence such that for all $j < k$, $\sigma(C_j, C_k) = \sigma$. Now, assume that σ breaks the $\star$ property. There are two cases:

- There exists some r such that $\sigma \Rightarrow r > r'$. Then, it means that for all $j < k$, $C_j(r) > C_k(r)$. In particular, this means that $C_0(r) > C_1(r) > \cdots > C_n(r) \ldots$, which yields an infinite descending chain in $\mathbb{N}$, and leads to a contradiction.
- There exists some s such that $\sigma \Rightarrow s = s'$ and some r which satisfies $\sigma \Rightarrow r < s$ and $\sigma \not\Rightarrow r = r'$. If $\sigma \Rightarrow r > r'$, we are back to the first case. Otherwise, it means $\sigma \Rightarrow r < r'$. Then, on the one hand, $C_0(r) < C_1(r) < \cdots < C_n(r) < \ldots$. On the other hand, $C_0(s) = C_1(s) = \cdots = C_n(s) = \ldots$. But we also have that for all $k \in \mathbb{N}, C_k(r) < C_k(s) = C_0(s)$. Overall, we get an infinite increasing chain which is bounded from above by $C_0(s)$, which again leads to a contradiction.

Thus, σ satisfies the $\star$ property. So, this is in particular the case for some pair of configurations $C = D = C_k$ and $D' = C_{k'}$ for some $k < k'$ taken from the last extracted subsequence. Such configurations are such that (recall that the C_k are configurations of an accepting run over some input, which is in particular initial):

(a) $C_0 \xrightarrow{u} C$
(b) $D \xrightarrow{v} D'$.

Moreover, $\tau(D) = \tau(D') = \tau(C)$ and D is final, by definition of $(C_k)_{k \in \mathbb{N}}$.   □

Corollary 4.18. *Emptiness for* NRA *over* $\mathbb{N}^\omega$ *is decidable in* PSPACE.

Proof. The algorithm is similar to the one for deciding non-emptiness for NRA over oligomorphic domains. Indeed, the sought witness lies in $\mathbb{Q}_+$, which is oligomorphic; it suffices to additionally check that the pairwise type of D and D' satisfies the star property. Thus, the algorithm initially guesses $\tau(C)$ and σ. Then, checking that there indeed exists a configuration whose type is $\tau(C)$ and that can be reached from C_0 (item 2a of Proposition 4.17) can be done in the same way as for Theorem 3.12, by simulating symbolically (i.e. over $\mathbb{Q}$-types) a run of the automaton. Now, for item 2b, the algorithm again symbolically simulates a run from D, by keeping track of the type of the current configuration $\tau(D')$, and additionally of the pairwise type $\sigma(D, D')$. Since $\sigma(D, D')$ is a $\mathbb{Q}$-type over $2|R|$ registers, it can be stored in polynomial space; moreover, given a transition test ϕ, it can also be updated in polynomial space.   □

4.5. **Functionality.** Following the study of the relationships between $\mathbb{N}$ and $\mathbb{Q}$, we are now ready to provide a characterization of non functionality over $\mathbb{N}$. Intuitively, it amounts to finding two pairs of runs whose inputs are in $\mathbb{Q}$: first, a prefix witnessing a mismatch, and second, an accepting loop satisfying the $\star$ property to ensure its iterability over $\mathbb{N}$.

Proposition 4.19 (Functionality). *Let* $R \subseteq \mathbb{N}^\omega \times \mathbb{N}^\omega$ *be given by an* NRT T. *The following are equivalent:*

(1) *R is not functional*
(2) *there exist two pairs of runs whose input words belong to $\mathbb{Q}_+^*$, which are as follows:*

   (a) $C_0 \xrightarrow{u|u_1} C_1$ *and* $C_0 \xrightarrow{u|u_2} C_2$ *with* $u_1 \not\parallel u_2$,
   (b) $D_1 \xrightarrow{v} D_1'$ *and* $D_2 \xrightarrow{v} D_2'$ *with* $\tau(D_1 \uplus D_2) = \tau(D_1' \uplus D_2) = \tau(C_1 \uplus C_2)$, *both runs visit a final state of* T, *and* $\sigma_i = \tau((D_1 \uplus D_2) \uplus (D_1' \uplus D_2'))$ *satisfies property* $\star$.

Proof. First, assume that (2) holds. By applying Proposition 4.11 to the product transducer $T \times T$, there exists two configurations E_1 and E_2 and an infinite data word $x \in \mathbb{N}^\omega$ such that $E_1 \xrightarrow{x|y_1}$ and $E_2 \xrightarrow{x|y_2}$. Moreover, both runs are accepting as each finite run is required to visit a final state of T.

Now, since $\tau(E_1 \uplus E_2) = \tau(D_1 \uplus D_2) = \tau(C_1 \uplus C_2)$, we can apply Proposition 4.16 to get two runs $C_0 \xrightarrow{u'|u_1'} F_1 \xrightarrow{x'|y_1'}$ and $C_0 \xrightarrow{u'|u_2'} F_2 \xrightarrow{x'|y_2'}$. Moreover, since morphisms preserve mismatches, we know that $u_1' \nmid u_2'$. Thus, we obtained a witness of non-functionality, since we have $(u'x, u_1'y_1'), (u'x, u_2'y_2') \in T$, with $u_1'y_1' \neq u_2'y_2'$ since $u_1' \nmid u_2'$.

We now assume that R is not functional. By definition, and as we assume R only contains infinite output words, there are two runs on some input word x whose outputs mismatch. By splitting both runs after the first mismatch, we get two runs $C_0 \xrightarrow{t|t_1} B_1 \xrightarrow{w|y}$ and $C_0 \xrightarrow{t|t_2} B_2 \xrightarrow{w|z}$ with $t_1 \nmid t_2$ (note that we do not necessarily have that $y = z$). Now, from the product transducer $T \times T$, one can define an NRA A with registers $R \uplus R'$ recognising the language $L(A) = \left\{ w' \mid E_1 \xrightarrow{w'|y'}_T, E_2 \xrightarrow{w'|z'}_T \text{ and } \tau(E_1 \uplus E_2) = \tau(B_1 \uplus B_2) \right\}$ which starts by guessing some configuration $E_1 \uplus E_2$ and checks that $\tau(E_1 \uplus E_2) = \tau(B_1 \uplus B_2)$, then simulates $T \times T$. Such language is non-empty, since it at least contains w. By Proposition 4.17, we get that there exists runs whose input words belong to $\mathbb{Q}_+^*$ which are $C_0 \uplus C_0 \xrightarrow{u'} C_1 \uplus C_2$ and $D_1 \uplus D_2 \xrightarrow{v} D_1' \uplus D_2'$ with $\tau(D_1 \uplus D_2) = \tau(D_1' \uplus D_2') = \tau(C_1 \uplus C_2)$, D_1 and D_1' final and $\sigma(D_1 \uplus D_2, D_1' \uplus D_2')$ satisfies the $\star$ property, which immediately yields item 2b.

Since $C_0 \uplus C_0 \xrightarrow{u'} C_1 \uplus C_2$, by definition of the considered NRA, we have that $E_1 \xrightarrow{u'|u_1'}_T C_1$ and $E_2 \xrightarrow{u'|u_2'}_T C_2$, with $\tau(E_1 \uplus E_2) = \tau(B_1 \uplus B_2)$. Thus, by applying a morphism μ such that $\mu(B_1) = E_1$, $\mu(B_2) = E_2$, such runs can be glued with the runs $C_0 \xrightarrow{t|t_1} B_1$ and $C_0 \xrightarrow{t|t_2} B_2$ to yield two runs $C_0 \xrightarrow{u|u_1} C_1$ and $C_0 \xrightarrow{u|u_2} C_2$, with $u = \mu(t)u'$ and $u_1 = \mu(t_1)u_1'$, $u_2 = \mu(t_2)u_2'$, with $u_1 \nmid u_2$ since $\mu(t_1) \nmid \mu(t_2)$ (recall that morphisms preserve mismatches, since they are bijective), so we get item 2a. $\qquad\square$

Corollary 4.20. *Functionality for relations over $\mathbb{N}^\omega \times \mathbb{N}^\omega$ given by* NRT *is decidable in* PSPACE.

Proof. By Lemma 3.3, if item 2a holds, then we can assume that the length of u is bounded by $P(\mathcal{R}(4k), |Q|, L)$, where $\mathcal{R}$ denotes the Ryll-Nardzewski function of $\mathbb{Q}_+$, which is exponential. Thus, the existence of u can be checked with a counter that is polynomially bounded. Then, item 2b can be checked in polynomial space, since it reduces to checking emptiness of the NRA A that we described in the above proof. $\qquad\square$

4.6. **Next-letter problem.** We now show that for any function definable by an NRT over $\mathbb{N}$, the next-letter problem is computable.

Lemma 4.21. *Let $f : \mathbb{N}^\omega \to \mathbb{N}^\omega$ be a function defined by an* NRT *over* $\mathbb{N}$. *Then, its next-letter problem is computable.*

Proof. The algorithm is in two phases:

(1) decide whether there exists a next letter
(2) if there exists a next letter, compute it

Remind that as input to the next-letter problem, there are two finite data words $u, v \in \mathbb{N}^*$ and the goal is to decide whether there exists some $d \in \mathbb{N}$ such that for all $uy \in \mathrm{dom}(f)$, $v \leq f(uy)$ implies $vd \leq f(uy)$. Let us assume that f is defined by some NRT $T = (Q, R, \Delta, q_0, \overline{c_0}, F)$. Let us explain how to algorithmically realize the two latter steps.

1. To decide the existence of such a d, we reduce the problem to a functionality problem for an NRT T_{uv}. First, let us define the convolution $x_1 \otimes x_2$ of two data words $x_1 = d_1 d_2 \ldots, x_2 = d_1' d_2' \cdots \in \mathbb{N}^\omega$ as the data word $d_1 d_1' d_2 d_2' \ldots$. The intuitive idea is to construct T_{uv} in such a way that it defines the relation $R_{uv} = R_{uv}^1 \cup R_{uv}^2$ defined by $R_{uv}^i = \{(ux_1 \otimes ux_2, vd^\omega) \mid ux_1, ux_2 \in \mathrm{dom}(f), vd \preceq f(ux_i)\}$. It is not difficult to see that R_{uv} is a function iff the next-letter has a positive answer for u and v. Once T_{uv} is constructed, checking its functionality is possible thanks to Corollary 4.20. Let us now explain how to construct T_{uv}. It is the union of two transducers T_{uv}^1 and T_{uv}^2 defining R_{uv}^1 and R_{uv}^2 respectively. The constructions are similar for both: T_{uv}^1 ignores the even position of the input word after u has been read while T_{uv}^2 ignores the odd position after u has been read, but they otherwise are defined in the same way. Let us explain how to construct T_{uv}^1. First, T_{uv}^1 makes sure that its input $\alpha \otimes \beta$ is such that $u \leq \alpha$ and $u \leq \beta$, or equivalently that $\alpha \otimes \beta = (u \otimes u)(x \otimes y)$ for some x, y. This is possible by hardcoding u in the transitions of T. Likewise, T_{uv}^1 also makes sure that v is a prefix of $f(ux)$. It is also possible by hardcoding in T the output v (to check for instance that a run of T output the ith data d_i of v, it suffices when T outputs a register r on its transitions, to add the test $r = d_i$). So, T_{uv}^1 simulates a run of T on ux (the odd positions of $\alpha \otimes \beta$) and a run of T on uy (the even positions of $\alpha \otimes \beta$). Once v has been consumed by the simulated run on ux, the first time this simulated run outputs something, the data is kept in a special register and outputted all the time by T_{uv}^1.

2. If we know that there exists a next letter d, the only thing that remains to be done is to compute it. To do so, we can again construct a transducer T_{uv}' which simulates T but makes sure to accept only input words that start with u, accepted by runs which outputs words starting with v. This can again be done by hardcoding u and v in T. Then, to compute a data d, it suffices to execute T_{uv}' by computing, at any point i, all the configurations reached by T_{uv}', and by keeping only those which are co-reachable. Testing whether a configuration is co-reachable can be done by testing the non-emptiness of an NRA starting in this initial configuration. Doing so, the algorithm computes all prefixes of accepting runs. Eventually, since there exists a next letter d, one of the run will output it. □

As a direct corollary of Lemma 4.21, Theorem 2.15 and Theorem 2.14, we obtain:

Theorem 4.22. *Let* $f : \mathbb{N}^\omega \to \mathbb{N}^\omega$ *be a function defined by an* NRT *over* $\mathbb{N}$*, and let* $m : \mathbb{N} \to \mathbb{N}$ *be a total function. Then,*

(1) *f is computable iff f is continuous*
(2) *f is uniformly computable iff f is uniformly continuous*
(3) *f is m-computable iff f is m-continuous*

4.7. **Uniform continuity.** We now turn to uniform continuity over $\mathbb{N}$, and show that it is enough to consider uniform continuity over $\mathbb{Q}$ and restrict our attention to configurations that are co-reachable w.r.t. data words over $\mathbb{N}$.

Given a configuration $\mathbb{Q}$-type τ, we say that it is co-reachable in $\mathbb{N}$ if there is an actual configuration C of type τ which is co-reachable.

Proposition 4.23. *Let T be an NRT over $(\mathbb{N}, \{<\})$ and $f = \llbracket T \rrbracket$. The following are equivalent:*

- *f is uniformly continuous,*
- *There is a critical pattern with D_1, D_2 co-reachable in $\mathbb{N}$.*

Proof. Let T' denote the same transducer as T except that 1) it is over $\mathbb{Q}$ and 2) it is restricted to configurations which are co-reachable in $\mathbb{N}$.

We claim that T realizes a uniformly continuous function if and only if T' does. From Proposition 3.7, this is enough to show the expected result.

Any run over T is in particular a run over T', hence if T is not uniformly continuous then, in particular, T' is not either. Conversely, let us assume that T' is not uniformly continuous. According to Proposition 3.7, this means that we can exhibit a critical pattern $C_0 \xrightarrow{u|u_1} C_1 \xrightarrow{v|v_1} \mu(C_1) \xrightarrow{w|w_1} D_1, C_0 \xrightarrow{u|u_2} C_2 \xrightarrow{v|v_2} \mu(C_2) \xrightarrow{z|w_2} D_2$ such that D_1, D_2 are co-reachable. By definition we have that D_1, D_2 are co-reachable in $\mathbb{N}$. Let i denote the mismatching position in the pattern. Let $n > 0$ we want to exhibit two inputs that have a common prefix of length at least n but outputs that have a longest common prefix smaller than i. We consider the two runs $C_0 \xrightarrow{u|u_1} C_1 \xrightarrow{v|v_1} \mu(C_1) \cdots \mu^n(C_1) \xrightarrow{\mu^n(v)|\mu^n(v_1)} \mu^{n+1}(C_1) \xrightarrow{\mu^n(w)|\mu^n(w_1)} \mu^n(D_1)$ and $C_0 \xrightarrow{u|u_2} C_2 \xrightarrow{v|v_2} \mu(C_2) \cdots \mu^n(C_2) \xrightarrow{\mu^n(v)|\mu^n(v_2)} \mu^{n+1}(C_2) \xrightarrow{\mu^n(w)|\mu^n(w_1)} \mu^n(D_2)$. Note that it could be that μ^n cancels the mismatch, in which case we can consider μ^{n+1}. Since $\mu^n(D_1)$ and $\mu^n(D_2)$ are co-reachable, we can use Proposition 4.16 and we obtain the result. $\qquad\square$

As a corollary, we obtain:

Theorem 4.24. *Let $f : \mathbb{N}^\omega \to \mathbb{N}^\omega$ be a function defined by an NRT over $\mathbb{N}$. Deciding its uniform continuity is PSPACE-c.*

Proof. We are left to show that uniform continuity of T' is in PSPACE. This problem is in PSPACE from Theorem 3.12. We have to be careful however, since computing T' explicitly may be too costly. The trick is that checking if a configuration is co-reachable can be done on the fly in PSPACE, using Proposition 4.17 and Corollary 4.18.

The complexity lower bound can again be obtained by reducing the problem to emptiness of register automata over $(\mathbb{N}, \{=\})$, which is PSPACE-c [DL09]. $\qquad\square$

4.8. **Continuity.** We end our study with the property of continuity of NRT over $\mathbb{N}$.

Assumption 4.25. *To simplify the following statement, we assume that transducers are equipped with a distinguished register r_m which along the run stores the maximal data seen in the input so far.*

Proposition 4.26 (Continuity). *Let $f : \mathbb{N}^\omega \to \mathbb{N}^\omega$ be given by an NRT T over $\mathbb{N}$. The following are equivalent:*

(1) *f is not continuous*

(2) *there exists a critical pattern in* $\mathsf{Critical}(u, v, w, z, C_1, C_2, C_1', C_2', D_1, D_2)$ *with* $u, v, w, z \in \mathbb{Q}_+^*$, *where* C_1 *is final,* D_1 *and* D_2 *are co-reachable in* $\mathbb{N}$ *and such that* $\sigma = \tau((C_1 \uplus C_2) \uplus (C_1' \uplus C_2'))$ *satisfies the* $\star$ *property for* $X = R_1 \cup R_2^l$, *where* $R_2^l = \{r_2 \in R_2 \mid \exists r_1' \in R_1', \sigma \Rightarrow r_2 \leq r_1'\}$.

Proof of (1) $\Rightarrow$ (2). Assume first that f is not continuous. Let $x \in \mathbb{N}^\omega$, and let $(x_n)_{n \in \mathbb{N}}$ be such that $\forall n \in \mathbb{N}, x_n \in \mathrm{dom}(f)$ $x_n \xrightarrow[n\infty]{} x$ but $f(x_n) \not\xrightarrow[n\infty]{} f(x)$. Up to extracting subsequences, we can assume w.l.o.g. that there exists some $k \in \mathbb{N}$ such that for all $n \in \mathbb{N}$, $|f(x) \wedge f(x_n)| \leq k$. We denote by ρ a run of T over x yielding output $f(x)$, and by $(\rho_n)_{n \in \mathbb{N}}$ a sequence of runs of T such that ρ_n yields output $f(x_n)$ over input $f(x)$.

First, since the set Δ of transitions of T is finite, Δ^ω is compact (cf Remark 2.1), so $(\rho_n)_{n \in \mathbb{N}}$ admits a converging subsequence, so we can assume w.l.o.g. that $(x_n)_{n \in \mathbb{N}}$ is such that $(\rho_n)_{n \in \mathbb{N}}$ converges to some infinite sequence of transitions π.

Remark 4.27. Note however that π is not necessarily a run over some concrete data word: since transducers are allowed to guess arbitrary data, we do not have the property that after reading input u, the configuration of the transducer takes its values in $\mathsf{data}(u) \cup \{0\}$, which would allow to build a concrete run by extracting subsequences of runs whose configurations coincide on longer and longer prefixes. If we disallow guessing, such property is restored, which greatly simplifies the proof. Indeed, then, we can require that σ has the $\star$ property, without allowing some wild registers to go astray (namely those that are above r_k', which necessarily correspond to data that have been guessed, otherwise their content is at most equal to the one of r_m), and having two runs that can be instantiated by actual data words allow to extract σ with a Ramsey argument which is similar to the one used for emptiness (see Proposition 4.17).

Now, let $(E_i)_{i \in \mathbb{N}}$ be the sequence of configurations of ρ, and, for each ρ_n, let $(C_{n,i})_{i \in \mathbb{N}}$ be its corresponding sequence of configurations. Then, for all $0 \leq i < j$, let $\tau_{n,i} = \tau(C_{n,i})$ and $\sigma_{n,i,j} = \sigma(E_i \uplus C_{n,i}, E_j \uplus C_{n,j})$. Since the $\tau_{n,i}$ and $\sigma_{n,i,j}$ take their values in a finite set, by using compactness arguments, we can extract two subsequences $(\tau_{n,i}')_{n,i \in \mathbb{N}}$ and $(\sigma_{n,i,j}')_{n \in \mathbb{N}, 0 \leq i < j}$ which respectively converge to $(\tau_i')_{i \in \mathbb{N}}$ and to a family $(\sigma_{i,j})_{0 \leq i < j}$. Formally, we require that for all $M \geq 0$, there exists some $N \geq 0$ such that for all $n \geq N$, we have that $\tau_{n,i}' = \tau_i'$ and $\sigma_{n,i,j}' = \sigma_{i,j}$ for all $0 \leq i < j \leq M$ (note that for types $\tau_{n,i}'$, this corresponds to convergence for infinite words). By first restricting to final configurations in ρ (we know there are infinitely many since ρ is accepting), we can assume that all E_i are final. We further restrict to E_i which all have the same type ν (there is at least one type which repeats infinitely often). To avoid cluttering the notations, we name again $(\rho_n)_{n \in \mathbb{N}}$ the corresponding subsequence of runs, $(\tau_{n,i})_{n,i \in \mathbb{N}}$ the corresponding types, and $(\sigma_{n,i,j})_{n,i,j}$ their associated family of pairwise types. Finally, by applying Ramsey's theorem to $(\tau_i)_{i \in \mathbb{N}}$ and $(\sigma_{i,j})_{0 \leq i < j}$, we know that there exists some type τ, some pairwise type σ and some infinite subset $I \subseteq \mathbb{N}$ such that for all $i, j \in I$ such that $i < j$, $\tau_i = \tau$ and $\sigma_{i,j} = \sigma$. For simplicity, we reindex the $(E_j)_{j \in I}$ and the $(C_{n,j})_{j \in I}$ over $\mathbb{N}$, that we again name $(E_j)_{j \in \mathbb{N}}$ and $(C_{n,i})_{i \in \mathbb{N}}$.

Now, assume by contradiction that σ does not have the property $\star$ for $X = R_1 \cup R_2^l$. There are two cases, that we treat iteratively:

- There exists some $r \in X$ such that $\sigma \Rightarrow r > r'$. There are two subcases:
  - If there exists such $r \in R_1$, then we get that for all $0 \leq i < j$, $E_i(r) > E_j(r)$, which immediately yields an infinite descending chain in $\mathbb{N}$, and hence a contradiction.

– If there exists such $r \in R_2^l$, then let r_M' be such that $\sigma \Rightarrow r \leq r_M'$. Since $r_M \in R_1$, we know that $\sigma \Rightarrow r_M \leq r_M'$, otherwise we are back to the previous case. Then, let $N \in \mathbb{N}$ be such that $\sigma_{N,i,j} = \sigma$ for all $0 \leq i < j \leq B = E_1(r_M') + 1$. Such N necessarily exists since we took the $(\rho_n)_{n \in \mathbb{N}}$ such that the $(\sigma_{n,i,j})_{0 \leq i < j}$ converges. Thus, $E_1(r_M') \geq C_{N,0}(r) > C_{N,1}(r) > \ldots C_{N,B}(r)$, which again yields a contradiction.

• Assume now that there exists $r, s \in X$ such that $\sigma \Rightarrow s = s'$, $\sigma \Rightarrow r \leq s$ but $\sigma \Rightarrow r < r'$. There are again two subcases:

– If $s \in R^1$, then we know that for all $i \in \mathbb{N}$, $E_i(s) = E_0(s)$. Now, if $r \in R^1$, then we get that $E_0(r) < E_1(r) < \ldots$ but for all $i \in \mathbb{N}$, $E_i(r) < E_i(s)$, which leads to a contradiction. If $r \in R_2^l$, then let N be such that $\sigma_{N,i,j} = \sigma$ for all $0 \leq i < j \leq B = E_0(s) + 1$. We then get a strictly increasing chain $C_{N,0}(r) < C_{N,1}(r) < \cdots < C_{N,B}(r)$ of length B which is bounded from above by $B - 1$, which does not exist in $\mathbb{N}$.

– If $s \in R_2^l$ then let $r_M' \in R_1$ be such that $\sigma \Rightarrow s \leq r_M'$. Then, let $B = E_1(r_M') + 1$, and let N be such that $\sigma_{N,i,j} = \sigma$ for all $0 \leq i < j \leq B$. If $r \in R^1$ then we have that $E_0(r) < E_1(d) < \cdots < E_B(r) \leq E_B(s) = E_0(s) \leq E_1(r_M')$; similarly if $r \in R_2^l$ we get $C_{N,0}(r) < C_{N,1}(r) < \cdots < C_{N,B}(r) \leq C_{N,B}(s) = C_{N,0}(s) \leq E_1(r_M')$. In both cases, this leads to a contradiction.

As a consequence, σ indeed has the property $\star$ for $X = R_1 \cup R_2^l$.

Now, it remains to exhibit a critical pattern from ρ and the (last) extracted subsequence $(\rho_n)_{n \in \mathbb{N}}$ and the corresponding inputs $(x_n)_{n \in \mathbb{N}}$. Let $k \in \mathbb{N}$ be such that for all $n \in \mathbb{N}$, $|f(x) \wedge f(x_n)| \leq k$. On the one hand, we have $\rho = E_0 \xrightarrow{u_0 | v_0} E_1 \xrightarrow{u_1 | v_1} E_2 \ldots$, where $x = u_0 u_1 \ldots$ and such that all $(E_i)_{i > 0}$ are final and they all have the same type. On the other hand, each ρ_n can be written $\rho_n = C_{n,0} \xrightarrow{u_0^n | v_0^n} C_{n,1} \xrightarrow{u_1^n | v_1^n} C_{n,2} \ldots$, where for all $j \geq 0, |u_j^n| = |u_j|$ (the configurations $C_{n,i}$ are synchronised with those of E_i, as we always extracted subsequences in a synchronous way) and $u_0^n u_1^n \cdots = x_n$. Now, let M be such that $\forall n \geq M$, we have $\forall 0 \leq i < j \leq B + 3, \sigma_{n,i,j} = \sigma$, where $B = 2k$. Finally, take some $l \geq M$ such that $|x_l \wedge x| \geq |u_0 u_1 \ldots u_{B+1}|$. There are two cases:

• $v_0 \ldots v_B \nparallel v_0^l \ldots v_B^l$. Then, we exhibited a critical parttern with two runs $E_0 \xrightarrow{u_0 \ldots u_B | v_0 \ldots v_B}$ $E_{B+1} \xrightarrow{u_{B+1} | v_{B+1}} E_{B+2} \xrightarrow{u_{B+2} | v_{B+2}} E_{B+3}$ along with $C_{l,0} \xrightarrow{u_0^l \ldots u_B^l | v_0^l \ldots v_B^l} C_{l,B+1} \xrightarrow{u_{B+1}^l | v_{B+1}^l}$ $C_{l,B+2} \xrightarrow{u_{B+2}^l | v_{B+2}^l} C_{l,B+3}$. First, the outputs indeed mismatch, by hypothesis, so we are in case a) of the definition of a critical pattern (see Definition 3.5).

• $v_0 \ldots v_B \parallel v_0^l \ldots v_B^l$. Then, since $v_0 \ldots v_B \leq f(x)$ and $v_0^l \ldots v_B^l \leq f(x_l)$, and since $|f(x) \wedge f(x_l)| \leq k$, we get that $|v_0 \ldots v_B| \leq k$ or $|v_0^l \ldots v_B^l| \leq k$. Now, there are again two cases:

– There exists some $i \leq B$ such that $v_i = v_i^l = \varepsilon$. Then, we exhibited a critical pattern with two runs $E_0 \xrightarrow{u_0 \ldots u_{i-1} | v_0 \ldots v_{i-1}} E_i \xrightarrow{u_i | \varepsilon} E_{i+1} \xrightarrow{u_{i+1} \ldots u_m | v_{i+1} \ldots v_m} E_{m+1}$ and $C_{l,0} \xrightarrow{u_0^l \ldots u_{i-1}^l | v_0^l \ldots v_{i-1}^l} C_{l,i} \xrightarrow{u_i^l | \varepsilon} C_{l,i+1} \xrightarrow{u_{i+1}^l \ldots u_m^l | v_{i+1}^l \ldots v_m^l} C_{l,m+1}$, where m is such that $v_0^l \ldots v_m^l \nparallel v_0 \ldots v_m$ (such m exists since $f(x_n) \nparallel f(x)$). We are then in case c) of the definition of a critical pattern (see Definition 3.5).

– Otherwise, there necessarily exists some $i \leq B$ such that $v_i = \varepsilon$ or $v_i^l = \varepsilon$ since $|v_0 \ldots v_B| \leq k$ or $|v_0^l \ldots v_B^l| \leq k$. We assume that $v_i = \varepsilon$, the reasoning is symmetric if instead $v_i^l = \varepsilon$. Necessarily, $v_0^l \ldots v_{i-1}^l \nparallel f(x)$, otherwise we can find some i such that both $v_i = v_i^l$ and we are back to the previous case. Then, we exhibited a critical

pattern with two runs $E_0 \xrightarrow{u_0...u_{i-1}|v_0...v_{i-1}} E_i \xrightarrow{u_i|\varepsilon} E_{i+1} \xrightarrow{u_{i+1}...u_m|v_{i+1}...v_m} E_{m+1}$ and $C_{l,0} \xrightarrow{u_0^l...u_{i-1}^l|v_0^l...v_{i-1}^l} C_{l,i} \xrightarrow{u_i^l|v_i^l} C_{l,i+1} \xrightarrow{u_{i+1}^l...u_m^l|v_{i+1}^l...v_m^l} C_{l,m+1}$, where m is such that $v_0^l \ldots v_m^l \nparallel v_0 \ldots v_m$ (such m exists since we assumed that $v_0^l \ldots v_{i-1}^l \nparallel f(x)$). We are then in case b) of the definition of a critical pattern (see Definition 3.5).

Finally, in all the considered cases, the last configuration $D_1 = E_j$ of the first run of the critical pattern (the one which is a prefix of ρ) is final, since we extracted the (E_i) so that we only kept the final ones. Also, both $D_1 = E_j$ and $D_2 = C_{l,j}$ are co-reachable in $\mathbb{N}$ since they are configurations of accepting runs in $\mathbb{N}$, which concludes the proof of $(1) \Rightarrow (2)$. $\square$

Proof of $(2) \Rightarrow (1)$. We assume now that (2) holds and show that f is not continuous. Intuitively, following notations of the critical pattern, we will prove that f is not continuous at some input $x = u'v'\mu(v')\mu(v')\ldots$, where u', v' elements of $\mathbb{N}^+$, images of u, v by some automorphism.

Thus, let us consider two runs $C_0 \xrightarrow{u|u_1} C_1 \xrightarrow{v|v_1} \mu(C_1)$ with C_1 final, and $C_0 \xrightarrow{u|u_2} C_2 \xrightarrow{v|v_2} \mu(C_2) \xrightarrow{z|w_2} D_2$ with D_2 co-reachable in $\mathbb{N}$. Two cases can occur: either $u_1 \nparallel u_2$, or $v_2 = \epsilon$ and $u_1 \nparallel u_2 w_2$. As in the proof in the oligomorphic setting, we want to show that it is possible to build from the first run an infinite run looping forever along (some renaming of) v, and from the second run a family of infinite runs, looping more and more. While this is directly true in some oligomorphic data domain, as we saw before, iteration in $\mathbb{N}$ is more tricky.

These runs can be seen as finite runs in the transducer $T \times T$, with twice as many registers as T, which we denote by R_1 for the first copy, and R_2 for the second. By assumption, $\sigma = \tau((C_1 \uplus C_2) \uplus (C_1' \uplus C_2'))$ satisfies the $\star$ property for $X = R_1 \cup R_2^l$. If we had that σ satisfies the $\star$ property for $R_1 \cup R_2$, then we could immediately deduce by Proposition 4.11 that the two loops on v could be iterated infinitely many times. However, the weaker hypothesis we have will only allow us to show that the loop from C_1 can indeed by ω-iterated, while the loop from C_2 can be iterated as many times as we want, but finitely many times. To this end, we have to take care of registers in $R_2 \setminus R_2^l$ to show that these runs indeed exist.

As we assumed that there is a register r_m storing the maximal data appearing in the input word, the definition of the set R_2^l ensures that every register not in R_2^l has its value coming from a guess along the run $C_0 \xrightarrow{u} C_2$. This is directly linked with the technical difficulty inherent to the presence of guessing. In particular, this allows us to consider different runs, in which the input data word is not modified, but the guessed values are.

As C_2 and $\mu(C_2)$ have the same type, and registers not in R_2^l have "large" values, their behaviour along the cycle (w.r.t. types) $C_2 \xrightarrow{v} \mu(C_2)$ is very restricted: they can only increase or decrease at each step. In particular, if one starts from some configuration C_2 in which values of registers not in R_2^l are very far apart, this will allow to iterate the cycle several times. More precisely, for any integer n, we can compute an integer n' and values for the guesses of registers not in R_2^l which are pairwise n' far apart, ensuring that the cycle can be repeated at least n times.

Thus, the previous reasoning allows us to replicate the proof of Proposition 4.11 to show that there exist a word $v' \in \mathbb{N}^*$ and an automorphism α preserving $\mathbb{N}$, such that:

- $C_1' \xrightarrow{x|y}$, with $\tau(C_1') = \tau(C_1)$ and $x = v'\alpha(v')\alpha^2(v')\ldots$

- there exists C_2' and, for all n, some $C_2'(n)$ with $C_2'(n)_{|R_2^l} = C_{2|R_2^l}'$, $\tau(C_2'(n)) = \tau(C_2)$, and

$$C_2'(n) \xrightarrow{v'} \alpha(C_2'(n)) \ldots \xrightarrow{\alpha^n(v')} \alpha^{n+1}(C_2'(n))$$

Using previous remark on guessed values, together with Proposition 4.16 applied on $C_0 \xrightarrow{u} C_1$ and $C_0 \xrightarrow{u} C_2$, and Proposition 4.9 applied on $\mu(C_2) \xrightarrow{z|w_2} D_2 \xrightarrow{z_l}$, we end up with:

- a run $C_0 \xrightarrow{u'|u_1'} E_1 \xrightarrow{x'|y'}$, with $x' = v''\alpha(v'')\alpha^2(v'') \ldots$
- for every n, there is a run $C_0 \xrightarrow{u'|u_2'} E_2 \xrightarrow{v''} \alpha(E_2) \ldots \xrightarrow{\alpha^n(v'')} \alpha^{n+1}(E_2) \xrightarrow{z'|w_2'} D_2' \xrightarrow{z_l'}$
- $u_1' \nparallel u_2'$ or $u_1' \nparallel u_2'w_2'$

Hence, this proves the non functionality as we have a sequence of runs whose inputs converge towards $u'x'$ but whose outputs mismatch. □

Theorem 4.28. *Continuity of* NRT *on* $\mathbb{N}$ *is* PSPACE-*c.*

Proof. Small critical pattern property (Claim 3.6) on oligomorphic data can easily be adapted to $\mathbb{N}$. Indeed, the only difference lies in the loops on v, but is is important to notice that loop removal used to reduce length of runs preserves the extremal configurations (see Proposition 3.1), hence it preserves the type of the global run, here the type $\sigma = \tau((C_1 \uplus C_2) \uplus (C_1' \uplus C_2'))$.

Then, one can proceed similarly by guessing such a small critical pattern, and checking mismatches with counters. Note that we have to verify in addition that D_2 is co-reachable in $\mathbb{N}$, but again, as already detailed in previous proofs, this property can be decided on-the-fly.

Finally, the PSPACE lower bound can again be obtained by a reduction to the emptiness problem of register automata over $(\mathbb{N}, \{=\})$, which is PSPACE-c [DL09]. □

4.9. **Transfer result.** We have extended our result to one non-oligomorphic structure, namely $(\mathbb{N}, \{<, 0\})$, which is a substructure of $(\mathbb{Q}, \{<, 0\})$. If we want to extend the result to other substructures of $(\mathbb{Q}, \{<\})$, say $(\mathbb{Z}, \{<\})$, we could do the same work again and study the properties of iterable loops in $\mathbb{Z}$. However, a simpler way is to observe that $(\mathbb{Z}, \{<\})$ can be simulated by $\mathbb{N}$, using two copies of the structure. We show a quite simple yet powerful result: given a structure $\mathbb{D}$, if the structure $\mathbb{D}'$ can be defined as a first-order interpretation over $\mathbb{D}$ then the problems of emptiness, functionality, continuity, *etc* reduce to the same problems over $\mathbb{D}$.

A *first-order interpretation* of dimension l with signature Γ over a structure $(\mathbb{D}, \Sigma)$ is given by FO[Σ] formulas: a formula $\phi_{\text{domain}}(x_1, \ldots, x_l)$, for each constant symbol c of Γ a formula $\phi_{\mathsf{c}}(x_1, \ldots, x_l)$ and for each relation symbol R of Γ of arity r a formula $\phi_R(x_1^1, \ldots, x_l^1, \ldots, x_1^r, \ldots, x_l^r)$. The structure $\mathbb{D}'$ is defined by the following: a domain $D' = \{(d_1, \ldots, d_l) | \mathbb{D} \models \phi_{\text{domain}}(d_1, \ldots, d_l)\}$; an interpretation $(d_1, \ldots, d_l) \in D'$ for each constant symbol c such that $\mathbb{D} \models \phi_{\mathsf{c}}(d_1, \ldots, d_l)$ (we assume that there is a unique possible tuple satisfying the formula, which can be syntactically ensured); an interpretation $R^{\mathbb{D}} = \{(x_1^1, \ldots, x_l^1, \ldots, x_1^r, \ldots, x_l^r) | \mathbb{D} \models \phi_R(x_1^1, \ldots, x_l^1, \ldots, x_1^r, \ldots, x_l^r)\}$ for each relation symbol R.

Theorem 4.29. *Let* $\mathbb{D}'$ *be a first-order interpretation over* $\mathbb{D}$. *Let* P *denote a decision problem among non-emptiness, functionality, continuity, Cauchy continuity or uniform continuity. There is a* PSPACE *reduction from* P *over* $\mathbb{D}'$ *to* P *over* $\mathbb{D}$.

Proof. Let $R \subseteq \mathbb{D}'^\omega \times \mathbb{D}'^\omega$ be given by an NRT T. If we assume that $\mathbb{D}'$ is an l-dimension interpretation of $\mathbb{D}$, then we can view R as a relation $P \subseteq (\mathbb{D}^l)^\omega \times (\mathbb{D}^l)^\omega$. Note that P is empty (resp. functional, continuous, Cauchy continuous, uniformly continuous) if and only if R is.

Moreover, since $\mathbb{D}'$ is an interpretation, one can construct an NRT S which realizes P. It uses l registers for every register of T plus $l-1$ registers to store the input. In its first $l-1$ transitions it just stores the input and every l transition, it simulates a transition of T, just by substituting the formulas of the interpretation for the predicates. As usual, we do not construct S explicitly, but we are able to simulate it using only polynomial space. □

As a direct corollary we can, in particular, transfer our result over $(\mathbb{N}, \{<, 0\})$ to $(\mathbb{Z}, \{<, 0\})$:

Corollary 4.30. *The problems of non-emptiness, functionality, continuity, Cauchy continuity, uniform continuity over $(\mathbb{Z}, \{<, 0\})$ are in* PSPACE.

Proof. From Theorem 4.29, using the fact that $(\mathbb{Z}, \{<, 0\})$ is given by the following two dimensional interpretation over $(\mathbb{N}, \{<, 0\})$: $\phi_{\text{domain}} := (x = 0) \lor (y = 0)$, $\phi_0 := (x = y = 0)$, $\phi_< := (x_1 = x_2 = 0 \land (y_1 < y_2)) \lor (y_1 = y_2 = 0 \land (x_1 > x_2)) \lor (x_2 = y_1 = 0 \land \neg(x_1 = y_2 = 0))$. □

Remark 4.31. Of course our transfer result applies to many other substructures of $(\mathbb{Q}, \{<, 0\})$, such as the ordinals $\omega + \omega$, $\omega \times \omega$, *etc.*

FUTURE WORK

We have given tight complexity results in a family of oligomorphic structures, namely polynomially decidable ones. An example of an exponentially decidable oligomorphic structure is the one of finite bitvectors with bitwise xor operation. Representing the type of k elements may require exponential sized formulas (for example stating that a family of k elements is free, *i.e.* any non-trivial sum is non-zero). The same kind of proof would give ExpSpace algorithms over this particular data set (for the transducer problems we have been studying). One could try to classify the different structures based on the complexity of solving these kinds of problems.

We have been able to show decidability of several transducer problems over the data set $\mathbb{N}$ with the linear order. This was done using two properties: 1) that $\mathbb{N}$ is a substructure of $\mathbb{Q}$ and 2) that we were able to characterize the iterable loops in $\mathbb{N}$. Moreover we can transfer the result to other substructures of $\mathbb{Q}$, *e.g.* $\mathbb{Z}$. One possible extension would be to investigate data sets which have a tree-like structure, *e.g.* the infinite binary tree. There exists a tree-like oligomorphic structure, of which the binary tree is a substructure. Studying the iterable loops of the binary tree may yield a similar result as in the linear order case. Of course this does not have to be restricted to tree-like structures, but may be applied to any substructure of an oligomorphic structure for which we are able characterize iterable loops.

There are several classical ways of extending synthesis results, for instance considering larger classes of specifications, larger classes of implementations or both. In particular, an interesting direction is to consider non-functional specifications. As mentioned in Introduction however, and as a motivation for studying the functional case, enlarging both (non-functional) specifications and implementations to an asynchronous setting leads

to undecidability. Indeed, already in the finite alphabet setting, the synthesis problem of deterministic transducers over ω-words from specifications given by non-deterministic transducers is undecidable [CL14]. A simple adaptation of the proof of [CL14] allows to show that in this finite alphabet setting, enlarging the class of implementations to any computable function also yields an undecidable synthesis problem. An interesting case however is yet unexplored already in the finite alphabet setting: given a synchronous specifications, as an ω-automaton, is to possible to synthesise a computable function realizing it? In [HKT12, FLZ11], this question has been shown to be decidable for specifications with *total* input domain (any input word has at least one correct output by the specification). More precisely, it is shown that realizability by a continuous function is decidable, but it turns out that the synthesised function is definable by a deterministic transducer (hence computable). When the domain of the specification is partial, the situation changes drastically: deterministic transducers may not suffice to realize a specification realizable by a computable function. This can be seen by considering the (partial) function g of Introduction, seen as a specification and casted to a finite alphabet $\{a, b, c\}$: it is not computable by a deterministic transducer, since it requires an unbounded amount of memory to compute this function.

References

[BKL14] Mikolaj Bojanczyk, Bartek Klin, and Slawomir Lasota. Automata theory in nominal sets. *Log. Methods Comput. Sci.*, 10(3), 2014.

[Boj19] Mikołaj Bojańczyk. *Atom Book*. 2019.

[CHVB18] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, 2018.

[CL14] Arnaud Carayol and Christof Löding. Uniformization in Automata Theory. In *Proceedings of the 14th Congress of Logic, Methodology and Philosophy of Science Nancy, July 19-26, 2011*, pages 153–178. London: College Publications, 2014.

[DFKL20] Vrunda Dave, Emmanuel Filiot, Shankara Narayanan Krishna, and Nathan Lhote. Synthesis of computable regular functions of infinite words. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[DL09] Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009.

[EFK21] Léo Exibard, Emmanuel Filiot, and Ayrat Khalimov. Church synthesis on register automata over infinite ordered data domains. *STACS*, 2021.

[EFR19] Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier. Synthesis of data word transducers. In *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands.*, pages 24:1–24:15, 2019.

[EFR20] Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier. On computability of data word functions defined by transducers. In Jean Goubault-Larrecq and Barbara König, editors, *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, volume 12077 of *Lecture Notes in Computer Science*, pages 217–236. Springer, 2020.

[ESKG14] Rüdiger Ehlers, Sanjit A. Seshia, and Hadas Kress-Gazit. Synthesis with identifiers. In *Proceedings of the 15th International Conference on Verification, Model Checking, and Abstract Interpretation - Volume 8318*, VMCAI 2014, pages 415–433, 2014.

[FJLW16] Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. On equivalence and uniformisation problems for finite transducers. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, Rome, Italy*, pages 125:1–125:14, 2016.

[FLZ11]   Wladimir Fridman, Christof Löding, and Martin Zimmermann. Degrees of lookahead in context-free infinite games. In Marc Bezem, editor, *Computer Science Logic, 25th International Workshop / 20th Annual Conference of the EACSL, CSL 2011, September 12-15, 2011, Bergen, Norway, Proceedings*, volume 12 of *LIPIcs*, pages 264–276. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.

[HKT12]   Michael Holtmann, Lukasz Kaiser, and Wolfgang Thomas. Degrees of lookahead in regular infinite games. *Logical Methods in Computer Science*, 8(3), 2012.

[JL69]   J.R. Büchi and L.H. Landweber. Solving sequential conditions finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

[KK19]   Ayrat Khalimov and Orna Kupferman. Register-bounded synthesis. In *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands.*, pages 25:1–25:16, 2019.

[KMB18]   Ayrat Khalimov, Benedikt Maderbacher, and Roderick Bloem. Bounded synthesis of register transducers. In *Automated Technology for Verification and Analysis, 16th International Symposium, ATVA 2018, Los Angeles, October 7-10, 2018. Proceedings*, 2018.

[KZ10]   Michael Kaminski and Daniel Zeitlin. Finite-memory automata with non-deterministic reassignment. *Int. J. Found. Comput. Sci.*, 21(5):741–760, 2010.

[Pri02]   Christophe Prieur. How to decide continuity of rational functions on infinite words. *Theor. Comput. Sci.*, 276(1-2):445–447, 2002.

[WZ20]   Sarah Winter and Martin Zimmermann. Finite-state strategies in delay games. *Inf. Comput.*, 272:104500, 2020.

Appendix A. Proof of Lemma 4.15

Lemma A.1. *Let $a, b, c, d \in \mathbb{N}$ be such that $a < b$, $c < d$ and $d - c \geq b - a$. Then, there exists a function $f : [a; b] \to [c; d]$ which is increasing and bijective, and such that $f([a; b] \cap \mathbb{N}) \subseteq \mathbb{N}$.*

Proof. There are two cases:

- If $d - c = b - a$, then take $f : x \to x + c - a$.
- Otherwise, define f as: for all $x \in [a; b - 1]$ (note that such interval can be empty), let $f(x) = x + c - a$ as above. Then, for $x \in [b - 1; b]$, let $e = c + (b - 1 - a)$ and $f(x) = d - (b - x)(d - e)$. We have $f(b - 1) = e$, which is consistent with the definition of f on $[a; b - 1]$, and $f(b) = d$, and f is moreover increasing and bijective. Finally, $f(b - 1) \in \mathbb{N}$ and $f(b) \in \mathbb{N}$. Overall, f satisfies the required properties. $\qquad\square$