Playing Games in the Baire Space

Benedikt Brütsch   Wolfgang Thomas
RWTH Aachen University
bruetsch@automata.rwth-aachen.de   thomas@automata.rwth-aachen.de

We solve a generalized version of Church's Synthesis Problem where a play is given by a sequence of natural numbers rather than a sequence of bits; so a play is an element of the Baire space rather than of the Cantor space. Two players Input and Output choose natural numbers in alternation to generate a play. We present a natural model of automata ("$\mathbb{N}$-memory automata") equipped with the parity acceptance condition, and we introduce also the corresponding model of "$\mathbb{N}$-memory transducers". We show that solvability of games specified by $\mathbb{N}$-memory automata (i.e., existence of a winning strategy for player Output) is decidable, and that in this case an $\mathbb{N}$-memory transducer can be constructed that implements a winning strategy for player Output.

1 Introduction

The algorithmic theory of infinite games was started in 1957 when Church formulated his "synthesis problem". This problem asked for presentation of a transformation $\alpha \mapsto \beta$ of ω-sequences over a finite alphabet Σ, computable letter-to-letter and satisfying a logical specification $R(\alpha,\beta)$. If we write $\alpha = \alpha(0)\alpha(1)\ldots, \beta = \beta(0)\beta(1)\ldots$ and set $\alpha\hat{}\beta = \alpha(0)\beta(0)\alpha(1)\beta(1)\ldots$, the specification R can be captured by the ω-language

$$L = \{\alpha\hat{}\beta \in \Sigma^\omega \mid R(\alpha,\beta)\}$$

Church's synthesis roblem asks: Given an ω-language L defined in a "logistic system", is there a letter-to-letter transformation $\alpha \mapsto \beta$ in the format of some kind of circuit such that $\alpha\hat{}\beta \in L$ for each $\alpha \in \Sigma^\omega$?

In descriptive set theory a related question had been studied in game-theoretic terminology, regarding games between two players called here Input and Output, where L serves as the winning condition for Output. A play $\alpha(0)\beta(0)\alpha(1)\beta(1)\ldots$ is won by player Output if it belongs to L, and a transformation as mentioned above is then a winning strategy for Output. These "Gale-Stewart games" [12] were studied in descriptive set theory focussing on the problem of determinacy (whether one of the two players has a winning strategy). A major result in this theory says that if the set L is Borel then the associated Gale-Stewart game, which we denote by $\Gamma(L)$, is determined [15, 16]. Church's Problem posed a sharpened question, namely, given a finite description of L, to determine who wins and to exhibit a concrete presentation of a winning strategy for the winner.

This problem was solved by Büchi and Landweber [4] in the following strong sense: If L is a regular ω-language (presented, e.g., by a deterministic Muller automaton), then the winner of the game $\Gamma(L)$ can be computed, and a winning strategy can be presented in the format of a finite-state machine (a Mealy automaton).

This fundamental result has been extended in many ways, among them into the framework of infinite-state systems. For example, Walukiewicz [18] showed the analogue of the Büchi-Landweber Theorem for pushdown systems. It is remarkable, however, that a different kind of "infinite extension" of the Büchi-Landweber Theorem has not been addressed in the literature, namely the case where the input alphabet over which ω-sequences are formed is infinite. Taking the typical case of a finite alphabet to be

T. Brihaye, B. Delahaye, L. Jezequel, N. Markey, J. Srba (Eds.):
Cassting Workshop on Games for the Synthesis of Complex Systems
and 3rd International Workshop on Synthesis of Complex Parameters
(Cassting'16/SynCoP'16).
EPTCS 220, 2016, pp. 13–25, doi:10.4204/EPTCS.220.2

$2 = \{0, 1\}$ and the typical case of an infinite alphabet to be $\mathbb{N}$ (we will focus solely on the alphabet $\mathbb{N}$ in this paper), we are no more dealing with sequences from $2^{\mathbb{N}}$ but from $\mathbb{N}^{\mathbb{N}}$.

In set-theoretic topology (and in descriptive set theory) this is the step from the Cantor space $2^{\mathbb{N}}$ to the Baire space $\mathbb{N}^{\mathbb{N}}$. The topological classification theory of sets $L \subseteq \mathbb{N}^{\mathbb{N}}$ is developed in very close analogy to that of sets $L \subseteq 2^{\mathbb{N}}$ (cf. [15, 16]), and determinacy of Borel games then holds for $\mathbb{N}^{\mathbb{N}}$ as it does for $2^{\mathbb{N}}$. A small difference occurs in the representation of projective sets: In the Baire space, these can be described as projections of closed sets, whereas in the Cantor space one has to resort to projections of G_δ-sets.

In automata theory, however, the step to infinite alphabets is highly non-trival. It requires automata that work over infinite alphabets, in particular $\mathbb{N}$. Several proposals exist to introduce finite-state devices that can process finite or infinite words over an alphabet such as $\mathbb{N}$. Let us recall some of them.

A straightforward approach is to code a number m by a word over $\{0, 1\}$, such as 0^m or 1^m or the binary expansion of m. Then we consider Banach-Mazur games in which a move by a player consists of a choice of a sequence of letters (from a finite alphabet) rather than single letters (for a recent reference on Banch-Mazur games see, e.g., [13]). So a play $m_0 m_1 m_2 \ldots$ may be coded by the bit sequence $0^{m_0+1} 1^{m_1+1} 0^{m_2+1} \ldots$ in which a player contributes a word of 0^+ or 1^+. A disadvantage of this approach is the fact that finite-state devices cannot check simple properties of (ω-)words, for instance the equality of successive numbers of a play.

As models of automata working directly on infinite alphabets we mention the register automata of Kaminski and Francez [14], the data automata of Bojanczyk et al. [2], and the register automata over data words of [9] that allow equality tests between letters (e.g., natural numbers) that occur in a word. Taking $\mathbb{N}$ as the alphabet, a weakness of these models is their inability to check the order between successive letters or just the condition that a letter m is followed by $m + 1$ or by $m - 1$.

In the present paper, we work with automata which can check such relations of "incremental change" between letters from $\mathbb{N}$, called progressive grid walking automata (PGAs) and introduced recently in [8]. They cover all properties of the Banach-Mazur coding of sequences over $\mathbb{N}$, and they allow to check the relation between successive letters (as far as expressible in monadic second-order logic MSO over $(\mathbb{N}, +1, 0)$). The idea is to code a letter by a column labelling of a labelled two-dimensional grid. A word $m_1 \ldots m_\ell$ is coded by a grid with ℓ column ω-words, where the value m_i is coded by the ω-word $\#1^{m_i}\perp^\omega$ as shown in Figure 1. For ω-words this grid is also right-infinite.

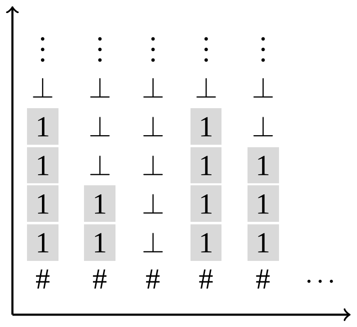

$$\begin{array}{cccccc}
\vdots & \vdots & \vdots & \vdots & \vdots & \\
\perp & \perp & \perp & \perp & \perp & \\
1 & \perp & \perp & 1 & \perp & \\
1 & \perp & \perp & 1 & 1 & \\
1 & 1 & \perp & 1 & 1 & \\
1 & 1 & \perp & 1 & 1 & \\
\# & \# & \# & \# & \# & \cdots
\end{array}$$

Figure 1: Grid representation of the sequence of letters $42043\ldots$

A progressive grid automaton is a three-way automaton that walks through such a grid from left to right, scanning a column in two-way mode and moving from a column to the next by a step to the right. The latter feature allows to preserve a natural number value (the value k if the step to the right occurs at height k of a column). This feature amounts to a memory for values in $\mathbb{N}$. For example the value of an input m_i may be handed from the i-th column to the next, by stepping out of the column at height m_i, in order to check, for example, that $m_{i+1} = m_i + 1$. In the present paper we introduce a slightly stronger variant of PGAs, called $\mathbb{N}$-*memory automata*, and we also define a corresponding model of transducer. These automata use tokens of three kinds, "memory token", "memory update token", and (for transducers) "output token" to indicate values of $\mathbb{N}$. A column is scanned again in two-way mode, but starting from the bottom, using the memory token that is located somewhere on the column (namely at the position where the memory update token was placed in the previous column), and in the current column the memory update token is then placed for handing a value k of $\mathbb{N}$ over to the next column (where the memory token will be on this position k). The transducer's output token is used to specify a value from $\mathbb{N}$ as the result of a computation.

Our main result will be an analogue of the Büchi-Landweber Theorem for $\mathbb{N}$-memory automata: *Given a Baire space game $\Gamma(L)$ where $L \subseteq \mathbb{N}^\omega$ is defined by an $\mathbb{N}$-memory automaton with parity acceptance condition one can decide who wins and construct an $\mathbb{N}$-memory transducer that executes a winning strategy for the winner.*

The remainder of the paper is structured as follows: In the subsequent section we present some prerequisites on MSO-logic. In Section 3 we introduce $\mathbb{N}$-memory automata, first as acceptors of ω-sequences over $\mathbb{N}$, and then as transducers. In Section 4 we state and prove the main result. In the conclusion we address some perspectives and open problems.

2 Prerequisites on MSO-Logic

We assume that the reader is familiar with the basics on MSO-logic (as presented, e.g., in [17]). We recall known results to be used in later sections.

It will be convenient to work with relational structures only. So we consider the structure $\mathcal{N} = (\mathbb{N}, Succ)$ with the successor relation *Succ* over $\mathbb{N}$ rather than the structure $(\mathbb{N}, +1, 0)$.

The MSO-theory of $\mathcal{N}$ is the set of all MSO-sentences that are true in $\mathcal{N}$. From [3] we know that this theory is decidable.

We use two transfer results on preservation of the decidability of MSO-theories. The first refers to "MSO-interpretations". A structure $\mathcal{A} = (A, R^A)$, say with just one binary relation $R^A \subseteq A \times A$ is MSO-interpretable in a structure $\mathcal{B}$ (possibly with different signature) if MSO-formulas $\varphi(x), \psi(x,y)$ exist that describe the structure $\mathcal{A}$ in $\mathcal{B}$, in the sense that the elements satisfying $\varphi(x)$ in $\mathcal{B}$ provide a copy of the domain A in B, and the pairs (a,b) satisfying $\psi(x,y)$ in $\mathcal{B}$ give a copy of R^A over the copy of A. The following is well-known (see, e.g., [10]):

Proposition 1. *If the structure $\mathcal{A}$ is MSO-interpretable in the structure $\mathcal{B}$ and the MSO-theory of $\mathcal{B}$ is decidable, then so is the MSO-theory of $\mathcal{A}$.*

The second model transformation is the step from a structure $\mathcal{A}$ to a product $[1,\dots,k] \times \mathcal{A}$, the k-fold copy of $\mathcal{A}$. Let us consider just the case where $\mathcal{A} = \mathcal{N} = (\mathbb{N}, Succ)$. The domain of this product is the set $\{1,\dots,k\} \times \mathbb{N}$, and we have the following relations:

- $SUCC = \{((r,n),(r,n+1)) \mid r \in [1,\dots,k],\ n \in \mathbb{N}\}$
- $P_r = \{(r,n) \mid n \in \mathbb{N}\}$ (fixing membership in the r-th copy of $\mathcal{N}$)
- $SameNumber = \{((r,n),(s,n)) \mid r,s \in [1,\dots,k],\ n \in \mathbb{N}\}$

In the general case of a relational structure $\mathcal{A}$, the first item is applied to all relations that are present in $\mathcal{A}$. It is easy to show (see, e.g., [1]) that the MSO-theory of $[1,\dots,k] \times \mathcal{A}$ is decidable if the MSO-theory of $\mathcal{A}$ is decidable. We need here only the case $\mathcal{A} = \mathcal{N}$:

Proposition 2. *For any k, the MSO-theory of* $[1,\dots k]\times\mathcal{N}$ *is decidable.*

In “monadic second-order transductions” as developed by Courcelle (see [7]) the operations of MSO-interpretations and of k-fold copying are combined into one.

A third result needed in a later section is concerned with parity games over infinite game arenas. We refer to [17] for background. We consider a parity game graph as a structure $G=(V,P_0,P_1,E,C_1,\dots,C_r)$ where V is the (at most countable) set of vertices, P_0 and P_1 are unary predicates defining the partition of V into the vertices of player 0 and player 1, respectively (we use these names rather than Input and Output to be in accordance with the literature on parity games), E is the edge relation, and the C_i define a partition of V where $v\in C_i$ means that vertex v carries color (or priority) i.

It is well-known that the parity game over G is determined with positional winning strategies [11].

In [19] it is shown that the winning regions of the two players are MSO-definable (by MSO-formulas $\varphi_j(x)$, for $j=0,1$). Moreover, as we shall see, under certain conditions the standard proof of positional determinacy yields an extension of this result, namely that for each player there is an MSO-definable winning strategy on the respective winning region. The definition of a winning strategy is given by a formula $\psi_j(x,y)$ for the respective player j, such that for each u in the winning region of player j, there is exactly one vertex v such that (u,v) satisfies $\psi_j(x,y)$, where v is the choice determined by the considered winning strategy of player j, from u. In order to guarantee this definability, we proceed in two steps: We show (in a later section) that for reachability games over the game arenas considered here, MSO-definable winning strategies exist, and then lift this result to parity games. (In a reachability game, a play is won by Output if it reaches a given MSO-defined target set at some point.) We settle the second step in the following poposition.

Proposition 3. *Let G be a parity game graph over V with MSO-definable sets of priorities. The winning regions of the two players of the associated parity game are MSO-definable. Moreover, if in each reachability game over G with MSO-definable target set there is an MSO-definable positional winning strategy of the winner on his/her winning region, then this also holds for the considered parity game over G .*

The first part of the claim is shown in [19]. For the second, we proceed by induction on the number of colors (priorities) of the game graph under consideration, following the standard determinacy proof as given in [17]. If there is one color r only, the claim is trivial (since player 0 wins by any choice if r is even, and player 1 wins by any choice if r is odd). Assume now we have formulas $\varphi_i^k(x)$ defining the winning region W_i^k of player i in a game with k colors, and formulas $\psi_i^k(x,y)$ defining a winning strategy of player i over W_i^k with k colors. Let us treat the case where $k+1$ is even (the other works by exchanging the players). Using the formula φ_0^{k+1} defining W_0^{k+1} in the considered game with $k+1$ colors (known from [19]) we obtain $\psi_0^{k+1}(x,y)$ as follows. The winning region W_0^{k+1} is composed of the attractor $A=A_0(C_{k+1}\cap W_0^{k+1})$ of player 0 and the complement of this set in W_0^{k+1}. This complement does not contain vertices with color $k+1$, so we have a formula $\varphi_0^k(x,y)$ defining a winning strategy for player 0 on this set. For the attractor, we note that it is MSO-definable by a formula $\varphi_A(x)$ saying “x is in all sets X containing $C_{k+1}\cap W_0^{k+1}$ and satisfying the following closure properties”:

$$\forall z\Big[\Big(z\in P_0\wedge\exists z'\big(E(z,z')\wedge X(z')\big)\to z\in X\Big)\wedge\Big(z\in P_1\wedge\forall z'\big(E(z,z')\to X(z')\big)\to z\in X\Big)\Big]$$

We now invoke the assumption on the MSO-definability of the winning strategy of Output in the reachability game with target set $C_{k+1}\cap W_0^{k+1}$ over his winning region (which is A), say by the formula $\psi(x,y)$. We thus obtain an MSO-definable winning strategy of Output over W_0^{k+1} by a formula saying

$$x\in W_0^{k+1}\setminus A\to\varphi_0^k(x,y)\quad\wedge\quad x\in A\to\psi(x,y)$$

3 Automata Models for Sequences of Natural Numbers

3.1 $\mathbb{N}$-Memory Automata

In this section, we introduce $\mathbb{N}$-*memory automata*, which work on sequences of natural numbers. Such a sequence $\alpha = a_0a_1a_2\ldots \in \mathbb{N}^\omega$ is represented by a labeled grid as illustrated in Figure 1, where each number a_i is represented by a column: In the ith column of the grid, the first a_i nodes, starting from the bottom, are labeled with 1 and the remaining nodes are labeled with $\perp$. For technical reasons, a node labeled with # is added at the bottom of every column.

Formally, the *grid representation* of a sequence $\alpha = a_0a_1a_2\ldots \in \mathbb{N}^\omega$ is a function $g_\alpha \colon \mathbb{N} \times \mathbb{N} \to \{1, \perp, \#\}$ labeling the positions of the grid with

$$g_\alpha(i,j) = \begin{cases} \# & \text{if } i = 0, \\ 1 & \text{if } 1 \le i \le a_j, \\ \perp & \text{if } i > a_j. \end{cases}$$

An $\mathbb{N}$-memory automaton can traverse a grid representation by moving up and down within the current column or switching to the next column (but not the previous one). It has a finite set of states, but is additionally equipped with a *memory token*, which marks a row of the grid, and a *memory update token*, which can be placed by the automaton at the end of processing a column and which determines the position of the memory token on the next column.

For a formal definition, let $D = \{\uparrow, \downarrow, \to, \diamond\}$ be the set of possible actions of the automaton (move up, move down, switch to next column, place memory update token).

Definition 1 ($\mathbb{N}$-Memory Automaton)**.** *An* $\mathbb{N}$-memory parity automaton *is a tuple $\mathscr{A} = (Q, q_0, \Delta, c)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, $\Delta \subseteq Q \times \{1, \perp, \#\} \times \{0,1\}^2 \times Q \times D$ is the transition relation with $\Delta \cap \Big(Q \times \{\#\} \times \{0,1\}^2 \times Q \times \{\downarrow\}\Big) = \emptyset$, and $c \colon Q \to \{0, \ldots, m\}$ is a function assigning priorities to the states.*

We call $\mathscr{A}$ deterministic if for all $\big(p, a, (b_1, b_2)\big) \in Q \times \{1, \perp, \#\} \times \{0,1\}^2$, there is exactly one pair $(p, d) \in Q \times D$ such that $\big(p, a, (b_1, b_2), p, d\big) \in \Delta$.

A *configuration* of $\mathscr{A}$ is a tuple $(q, h, v, i, j) \in Q \times \mathbb{N}^4$, where q is the current state, h is the horizontal position of the automaton (i.e., the current column), v is the vertical position (within the current column), and i, j are the current positions of the memory token and the memory update token, respectively, on the current column. (If the memory update token is not yet placed, we assume position 0 for it by default.)

A run of $\mathscr{A}$ on a sequence $\alpha \in \mathbb{N}^\omega$ is an infinite sequence $\pi = c_0c_1c_2\ldots$ such that $c_0 = (q_0, 0, 0, 0, 0)$ is the initial configuration and for every pair of consecutive configurations $c_\ell = (q_\ell, h_\ell, v_\ell, i_\ell, j_\ell)$, $c_{\ell+1} = (q_{\ell+1}, h_{\ell+1}, v_{\ell+1}, i_{\ell+1}, j_{\ell+1})$, one of the following holds:

- $\big(q, g_\alpha(v_\ell, h_\ell), (b_1, b_2), q_{\ell+1}, \uparrow\big) \in \Delta$, and $v_{\ell+1} = v_\ell + 1$, $h_{\ell+1} = h_\ell$, $i_{\ell+1} = i_\ell$, $j_{\ell+1} = j_\ell$, or
- $\big(q, g_\alpha(v_\ell, h_\ell), (b_1, b_2), q_{\ell+1}, \downarrow\big) \in \Delta$, and $v_{\ell+1} = v_\ell - 1$, $h_{\ell+1} = h_\ell$, $i_{\ell+1} = i_\ell$, $j_{\ell+1} = j_\ell$, or
- $\big(q, g_\alpha(v_\ell, h_\ell), (b_1, b_2), q_{\ell+1}, \to\big) \in \Delta$, and $v_{\ell+1} = v_\ell$, $h_{\ell+1} = h_\ell + 1$, $i_{\ell+1} = j_\ell$, $j_{\ell+1} = 0$, or
- $\big(q, g_\alpha(v_\ell, h_\ell), (b_1, b_2), q_{\ell+1}, \diamond\big) \in \Delta$, and $v_{\ell+1} = v_\ell$, $h_{\ell+1} = h_\ell$, $i_{\ell+1} = i_\ell$, $j_{\ell+1} = v_\ell$,

where $\quad b_1 = \begin{cases} 1 & \text{if } i_\ell = v_\ell, \\ 0 & \text{otherwise} \end{cases} \quad$ and $\quad b_2 = \begin{cases} 1 & \text{if } j_\ell = v_\ell, \\ 0 & \text{otherwise} \end{cases}$

indicate whether the memory token and the memory update token, respectively, are at the current vertical position v_ℓ.

A run π is accepting if $max\Big(Inf\big(c(\pi)\big)\Big)$ is even. A sequence $\alpha \in \mathbb{N}^\omega$ is accepted by $\mathscr{A}$ if the run of $\mathscr{A}$ on its grid representation g_α is accepting.

For example, the singleton language $\{1234\dots\}$ is recognized by a deterministic $\mathbb{N}$-memory automaton that works as follows: It checks that there is exactly one 1 in the first column, and moves the memory update token to the row 1. After switching to the next column, the memory token now marks row 1. The automaton goes up to that row and checks that there is exactly one 1 above that position. Then it moves the memory update token to the position above the memory token and switches to the next column, and so on. The states assumed in this process have color 2; once the checking process fails, color 1 is assumed. A variation of this idea shows the recognizability of the ω-language $\mathbb{N}^*1\mathbb{N}^*2\mathbb{N}^*3\mathbb{N}^*\dots$.

The language $\{\alpha \in \mathbb{N}^\omega \mid \alpha \text{ is unbounded}\}$ is recognized by a deterministic $\mathbb{N}$-memory automaton that moves the memory update token to the position of the topmost 1 of the current column if that position is higher than the current position of the memory token. After any move of the memory update token, the automaton goes to a state with the even priority 2, otherwise to a state with priority 1.

We give three further examples of languages recognized by $\mathbb{N}$-memory automata (without proof):

1. $\{m_0m_1m_2\dots \mid m_{i+1} = m_i + 1 \text{ or } m_{i+1} = m_i - 1\}$
2. $\{m_0m_1m_2\dots \mid m_{i+1} \text{ even iff } m_i \text{ odd}\}$
3. $\{m_0m_1m_2\dots \mid m_{2i+2} = m_{2i} + 1 \text{ if } m_{2i+1} \text{ even}$
   $m_{2i+2} = m_{2i} - 1 \text{ if } m_{2i+1} \text{ odd } \}$

Thus, $\mathbb{N}$-memory automata can recognize some interesting ω-languages over $\mathbb{N}$. The ability to compare successive (and also "distant") letters and to define properties of unboundedness seems to be a feature that is missing in known models of automata over the alphabet $\mathbb{N}$. Let us note that in the context of temporal logic, a related idea appears in [5]; however there equality and incremental change of values from $\mathbb{N}$ is restriced to occurrences within a bounded (time-)interval – so a language such as $\mathbb{N}^*1\mathbb{N}^*2\mathbb{N}^*3\mathbb{N}^*\dots$ (as mentioned above) is not covered.

An alternative version of $\mathbb{N}$-memory automata can be defined by abstracting from the steps within one column and representing the steps from one column to the next one by MSO-formulas. In this description we use the product structure $Q \times \mathscr{N}$ with domain $Q \times \mathbb{N}$ as defined in Section 2.

An automaton with this logical specification of the transitions, which we call *MSO $\mathbb{N}$-memory automaton*, is of the form $\mathscr{A} = (Q, q_0, \big(\varphi((p,x), y, (q,z))\big)_{p,q\in Q}, c)$. (This notation indicates a formula $\varphi(r,s,t)$ with $P_p(r)$ and $P_q(s)$.) The following condition should be satisfied: Starting in state p with memory token on position i, after processing the input number m, the automaton will reach state q with memory token on the new position j iff $Q \times \mathscr{N} \models \varphi[(p,i), m, (q,j)]$. [1] We call such a step of the automaton a *macro transition*.

If the MSO-$\mathbb{N}$-memory automaton is deterministic (as in the present paper), then for every (p,i) and m, there is exactly one (q,j) such that $Q \times \mathscr{N} \models \varphi[(p,i), m, (q,j)]$.

Proposition 4. *For every $\mathbb{N}$-memory automaton, an equivalent MSO-$\mathbb{N}$-memory automaton can be constructed.*

The proof is straightforward but tedious regarding the details. The idea is to describe the segments of a computation on a given column letter from one placement of the memory update token to the next. This computation segment can visit a given position of the given column only $\leq |Q|$ times; otherwise a repetition of configurations occurs and the computation does not terminate. Hence such a run segment can be described by an existential MSO-formula with $|Q|^2$ existential set quantifiers. The processing of

[1] Strictly speaking, m is not an element of $Q \times \mathbb{N}$; by abuse of notation we write m to denote the element (q_0, m).

a column is a sequence of such computation segments, ending at the point where the automaton switches to the next column, so it is captured by the transitive closure of the segment computations. It is easy to express this invoking the definability of transitive closure in MSO.

Furthermore, let us list some properties (not needed below, however) that are proved similarly to [8].[2]

Remark 1. 1. *The emptiness problem for $\mathbb{N}$-memory automata over words from $\mathbb{N}^*$ and the emptiness problem for $\mathbb{N}$-memory parity automata over $\mathbb{N}^\omega$ are decidable.*

2. *This fails when the automata are equipped with two memory tokens (and memory update tokens).*

3.2 $\mathbb{N}$-Memory Transducers

We use deterministic $\mathbb{N}$-memory automata to represent winning conditions in Gale-Stewart games in the Baire space. To represent strategies in such games, we introduce $\mathbb{N}$-memory transducers, which are defined in close analogy to $\mathbb{N}$-memory automata, with two modifications: Firstly, there is no priority function as used for the parity acceptance condition (since we are not dealing with infinite runs). Secondly, there is an additional token, the *output token*; used to indicate a natural number that is produced as output after reading a word of natural numbers as given input sequence.

Thus, we define an extended set of actions $\widehat{D} = \{\uparrow, \downarrow, \rightarrow, \diamond, \Box\}$, and the transition relation is now of the form $\Delta \subseteq Q \times \{1, \bot, \#\} \times \{0,1\}^3 \times Q \times \widehat{D}$. In a transition of the form $(p, a, (b_1, b_2, b_3), q, \Box)$, the output token is placed at the current vertical position.

We will only be interested in deterministic $\mathbb{N}$-memory transducers, where for all $\big(p, a, (b_1, b_2, b_3)\big) \in Q \times \{1, \bot, \#\} \times \{0,1\}^3$, there is exactly one pair $(p, d) \in Q \times \widehat{D}$ such that $\big(p, a, (b_1, b_2, b_3), p, d\big) \in \Delta$.

An $\mathbb{N}$-memory transducer works like an $\mathbb{N}$-memory automaton, but it distinguishes between input and output columns. After processing a given input column, it switches to an output column, which is unlabeled except for the tokens (initially just the memory token). The position of the output token upon moving to the next column then indicates the output number at that point.

By processing input and output columns in alternation, the transducer produces an output sequence $\beta = e_0 e_1 e_2 e_3 \ldots \in \mathbb{N}^\omega$ for a given input sequence $\alpha = a_0 a_1 a_2 a_3 \ldots \in \mathbb{N}^\omega$, yielding the play $a_0 e_0 a_1 e_1 a_3 \ldots$.

4 Solving Games in the Baire Space

Our aim here is to prove the following result:

Theorem 1. *For a Baire space game $\Gamma(L)$ where $L \subseteq \mathbb{N}^\mathbb{N}$ is defined by a deterministic $\mathbb{N}$-memory parity automaton $\mathscr{A}$, one can*

- *decide who wins $\Gamma(L)$, and*
- *construct a winning strategy for the winner realized by an $\mathbb{N}$-memory transducer.*

In order to show the theorem, we proceed in two steps, following a pattern as known from the classical solution of Church's Problem in the Cantor space.

1. Convert the automaton into a parity game with designated start vertex.
   (In contrast to the classical setting, the game arena will be infinite here.)
2. Solve the parity game (finding the winner and computing a memoryless winning strategy).

[2] A more detailed study of $\mathbb{N}$-memory automata – including a systematic analysis of closure properties and the inequivalence between the deterministic and the non-deterministic model – is the subject of a forthcoming paper by P. Landwehr and the authors.

In the first subsection we deal with the first step and the decision about the winner, in the subsequent subsection we present the construction of the desired transducer.

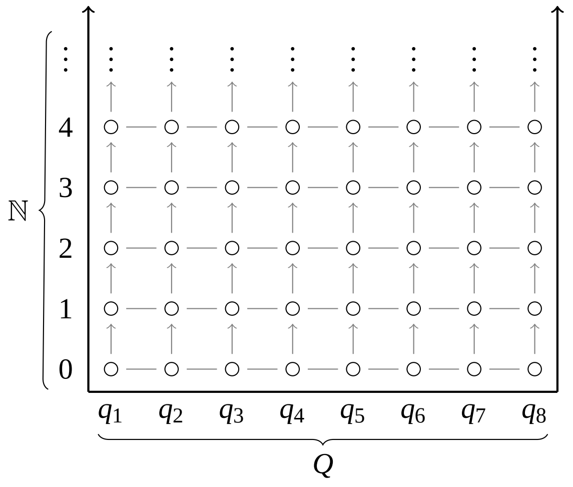
4.1 Deciding the Winner

To transform the given deterministic $\mathbb{N}$-memory automaton $\mathscr{A}$, recognizing $L \in \mathbb{N}^{\mathbb{N}}$, into a game arena, we first construct an equivalent MSO-$\mathbb{N}$-memory automaton $\mathscr{A}'$. We assume that the state set Q can be partitioned into sets Q_0 and Q_1 such that all macro transitions from Q_0 lead to Q_1 and vice versa. This can always be achieved using two copies of the original state set.

Now we construct a game arena $G_{\mathscr{A}}$ with domain $Q \times \mathbb{N}$, the relations as defined in Section 2 for $Q \times \mathscr{N}$, and the additional edges

$$(p,i) \xrightarrow{m} (q,j)$$

according to the macro transitions of $\mathscr{A}'$. In the following, we call a tuple (p,i) as it occurs here a "configuration".



Lemma 1. *$G_{\mathscr{A}}$ is MSO-interpretable in $Q \times \mathscr{N}$.*

The proof is straightforward by describing the edge relations of $G_{\mathscr{A}}$ in $Q \times \mathscr{N}$. Thus we obtain the following proposition.

Proposition 5. *The MSO-theory of $G_{\mathscr{A}}$ is decidable.*

We now can decide the winner of $\Gamma(L)$. For this we use the first claim of Proposition 3 (Section 2): Describe the initial vertex $(q_0,0)$ of $G_{\mathscr{A}}$ by a formula $\psi_{\text{init}}(x)$, and let $\varphi_{\text{Out}}(x)$ be a formula defining the winning region of Player Output. We check whether

$$G_{\mathscr{A}} \models \exists x\big(\psi_{\text{init}}(x) \wedge \varphi_{\text{Out}}(x)\big)$$

4.2 Constructing a Transducer

We treat here the case that the winner is Player Output. We first want to apply Proposition 3 (Section 2). So we have to show that the assumption of Proposition 3 regarding reachability games holds for the games considered here, namely that an MSO-definable winning strategy for Output (over his winning region) exists for a reachability game over $G_{\mathscr{A}}$ with MSO-definable target set. Then, applying Proposition 3, we know that in the parity game over $G_{\mathscr{A}}$, an MSO-definable winning strategy exists for Player Output on his winning region. In the second step we use this fact to obtain the desired transducer.

4.2.1 MSO-Definable Winning Strategies in Reachability Games

We show the following, referring to the game arena $G_{\mathscr{A}}$ introduced above.

Proposition 6. *In every reachability game over $G_{\mathscr{A}}$ with an MSO-definable target set F, Player Output has an MSO-definable positional winning strategy on his winning region.*

We show this claim by a transformation of the reachability game over $G_{\mathscr{A}}$ with target set F into a pushdown reachability game $\mathscr{P}$ over an extended domain $P \times G_{\mathscr{A}}$ for some finite P. A transition $(p,i) \to (q,j)$ in $G_{\mathscr{A}}$ (via some input number m) will be dissolved into a sequence of steps over the pushdown arena $\mathscr{P}$, proceeding from stack content $\#1^i$ to stack content $\#1^j$ in steps each of which changes the stack only by 1. Some complications arise from the fact that a transition from (p,i) to (q,j) depends on an input value m from the infinite domain $\mathbb{N}$. As we shall see, we can handle this using finite information about m when the target value j is "near" to 0 or i; otherwise the target value j will be "near" to m, and the stack will be changed accordingly.

As a preparation we need an obvious fact on the behaviour of the deterministic automaton $\mathscr{A}$:

Lemma 2. *There is a bound B such that from configuration (p,i) with input m, the automaton $\mathscr{A}$ will reach an exit configuration (q,j) where the distance of j to 0 or i or m is bounded by B.*

The lemma is clear by the fact that between the marked positions $0,i,m$ in a column the automaton $\mathscr{A}$ is processing one-letter input words. On such words of sufficiently large length B, the automaton $\mathscr{A}$ will assume a periodic behaviour and hence would violate the condition that a unique value j is reached upon termination.

According to the lemma, the configuration upon leaving a column can be represented by a tuple $(q,t,k) \in Q \times \{\text{“0”},\text{“I”},\text{“M”}\} \times \{-B,\dots,B\}$, which we call an *exit combination*. For example, the tuple $(q,\text{“I”},2)$ would indicate that the column is left in state q with the memory update token on position $i+2$. Note that the set E of exit combinations is finite.

A second remark refers to the periodic behaviour of the deterministic automaton $\mathscr{A}$ on words over a singleton alphabet. Such word segments occur between the positions 0, the memory token position i, and the input position m. The states assumed by $\mathscr{A}$ occur periodically. There is a finite prefix length ℓ_0 and a period length ℓ (which can be taken as $|Q|!$) such that given any starting state p at position i, the state of $\mathscr{A}$ at position $i-k$ or $i+k$ is fixed by the number in $[0,\ell_0+\ell]$ which is identical to k when $k \leq \ell_0$ or otherwise in $[\ell_0+1,\ell_0+\ell]$ and with same remainder modulo ℓ as k. Call this number the "(ℓ_0,ℓ)-status of k" (or just *status* of k).

Note that for any $p \in Q$, $i,m \in \mathbb{N}$, the corresponding exit combination is determined by the status of i, the status of m, the status of $|i-m|$, and whether $i<m$ (we refer to the last three items as the *relative status* of m with respect to i). Writing S for the set of individual status informations, and $0,1$ for the information whether $i<m$ or not, we obtain a finite (and effectively computable) relation $R \subseteq S^3 \times \{0,1\} \times E$ consisting of those tuples where the last component is determined by the first four components.

We now give a sketch of the proof of Proposition 6. We define a pushdown arena $\mathscr{P}$ where, intuitively, the height of the stack indicates the current position of the memory token. The control states of the pushdown system indicate the current state p of $\mathscr{A}$ and also the status of the current stack height.

Consider a configuration of the pushdown system where the state of $\mathscr{A}$ is p and the height of the stack, representing the memory token position, is i (and its status is stored in the control state). The current player, say Output, can now choose a tuple $r \in R$ where the first component of r is the status of i. This amounts to a decision about the number m that Player Output wants to play in the original game: it fixes the relative status of m with respect to i (and thus the exit combination representing the behavior of $\mathscr{A}$ on a column of height m).

In the following steps of the pushdown game, Player Output will modify the stack content to represent the new memory token position j according to the exit combination e that is determined by his choice of r. If e is of the form $(q, \text{“0”}, k)$, he can empty the stack and then increase its height to k. For a combination $e = (q, \text{“I”}, k)$, the height of the stack (currently representing i) is increased/decreased by k. If e is of the form $(q, \text{“M”}, k)$, the player can either increase of decrease the height of the stack step by step. While the stack is modified, the relative status of the current stack height with respect to i is tracked in the control state of the pushdown system. Whenever the current height of the stack is a number m with the previously chosen relative status (given by r), the player can finally increase/decrease the height by k, which determines the new memory token position.

Now we can apply the fact that attractor strategies in pushdown reachability games are definable by finite automata (see [6]) – and hence in MSO-logic. [3]

Proposition 7. *Positional winning strategies in pushdown reachability games with MSO-definable target set can be implemented by deterministic finite automata reading a given pushdown configuration and yielding as output the pushdown rule to be applied next.*

In this result, the choice of the next move is fixed by the name h of the pushdown rule to be applied. In MSO-logic, we obtain thus formulas $\psi_h(x)$ that are true if for position $x = (p, i)$ the rule to be applied is h. It is easy to transform these MSO-formulas into a single MSO-formula $\chi(x, y)$ which fixes y as the element reached from x by applying the unique rule h where $\psi_h(x)$ is true.

In the last step, we have to combine the finitely many steps of a player in $\mathscr{P}$ forming altoghether a macro transition of $\mathscr{A}$ into a single step, and we have to transfer the MSO-definability of the strategy from the arena $P \times G_{\mathscr{A}}$ (i.e., $P \times Q \times \mathscr{N}$) of the pushdown game to the structure $Q \times \mathscr{N}$.

To combine the intermediate steps forming a macro transition, we apply the (MSO-definable) transitive closure to the strategy formula $\chi(x, y)$ for the player under consideration, with the requirement that an exit configuration is finally reached, yielding another MSO-formula $\chi'(x, y)$.

To obtain an MSO-definable strategy over the original arena $Q \times \mathscr{N}$, it suffices to note that the finitely many tuples of $S^3 \times \{0, 1\} \times E$ can be coded in a finite label alphabet and that the status information of numbers is definable in MSO-logic.

4.2.2 From MSO-Definability of Strategies to Transducers

Proposition 8. *Given an MSO-definable winning strategy of Player Output in the parity game on $G_{\mathscr{A}}$, there is an $\mathbb{N}$-memory transducer realizing a winning strategy in $\Gamma(L(\mathscr{A}))$.*

Assume Player Output wins $\Gamma(L(\mathscr{A}))$. By Proposition 3, we have an MSO-formula $\varphi(x, y)$ defining a winning strategy on his winning region W_{Out} of $G_{\mathscr{A}}$. For the construction of the transducer, we will use the following lemma.

Lemma 3. *For a given MSO-formula $\varphi(x, y)$ over $Q \times \mathscr{N}$ and given $p, q \in Q$, we can construct a deterministic finite automaton $\mathscr{C}^{\varphi}_{pq}$, whose input is a column (i.e., a word) that is unlabeled except for tokens at positions i, j (memory token and memory update token), that terminates and that accepts iff $Q \times \mathscr{N} \models \varphi[(p, i), (q, j)]$.*

[3] In [6], also parity games are mentioned; for easier presentation we consider reachability games and apply Proposition 3 for the step to parity games.

This automaton is obtained as follows: For a formula $\varphi(x,y)$ over $Q \times \mathscr{N}$, we can construct corresponding formulas $\varphi'_{pq}(x',y')$ over $\mathscr{N}$ such that $\mathscr{N} \models \varphi'_{pq}[i,j]$ iff $Q \times \mathscr{N} \models \varphi[(p,i),(q,j)]$. To obtain such a formula, each second-order variable X in φ is replaced by a $|Q|$-tuple of second-order variables $(X_q)_{q\in Q}$ (see [1]).

Then the resulting MSO-formula can be translated into an equivalent Büchi automaton, which in turn can be converted into an NFA that accepts or rejects immediately after the last of the two tokens in the column has been read, depending on whether the Büchi automaton can reach an accepting loop on the unlabeled rest of the column. This NFA can be determinized, yielding the desired automaton $\mathscr{C}^{\varphi}_{pq}$.

Using Lemma 3, we can now construct the transducer as claimed in Proposition 8. Note that the formula $\varphi(x,y)$ defining a winning strategy fixes a unique update for a configuration (p,i) to a configuration (q,j). For the output of the transducer we have to find a number m such that $(p,i) \xrightarrow{m} (q,j)$ is a possible transition in the game graph $G_{\mathscr{A}}$. The transducer will go through the possible values of m, by placing the output token successively on position $0,1,2,\ldots$. In each case, say with the output token on position m, it works like $\mathscr{A}$ to find from start configuration (p,i) the new configuration (q,j). Now $\mathscr{C}^{\varphi}_{pq}$ is used to check whether the move to (q,j) is in accordance with the winning strategy. If this is the case, the current value of m is the desired output.

In more detail: Assume that the transducer has processed an input column and has just switched to the subsequent output column, in state p and with the memory token at position i. Starting with the output token on position 0, the transducer now proceeds as follows: It simulates the $\mathbb{N}$-memory automaton $\mathscr{A}$ (including the placements of the memory update token) on the column $\#1^m\bot^\omega$, where m (initially $m=0$) is the current position of the output token.

At some point, $\mathscr{A}$ would switch to the next column. Let j be the position of the memory update token and let q be the state of $\mathscr{A}$ at that point. The transducer now invokes the automaton $\mathscr{C}^{\varphi}_{pq}$ described in Lemma 3 to check whether (q,j) is the correct target position according to the strategy given by $\varphi(x,y)$. If this is the case (i.e., $\mathscr{C}^{\varphi}_{pq}$ accepts) then the transducer terminates processing the current column (and moves to the next input column). Otherwise, it moves the output token one position upwards and repeats the steps above. At some point, the correct target configuration (q,j) will be found, so the transducer will eventually produce the desired output number.

5 Summary and Perspectives

We have introduced $\mathbb{N}$-memory automata as a natural model of automata over the infinite alphabet $\mathbb{N}$, and in this framework we have obtained an algorithmic solution of Church's synthesis problem. It seems to be the first algorithmic solvability result on games in the Baire space.

Let us address some open issues:

1. Find a more direct construction for the decision of the winner and the winning strategy. We have invoked decidability results on MSO-theories.

2. Related to the first issue, a complexity analysis should be supplied – this is missing in the present paper.

3. One may wonder whether a logical framework of game specifications can be developed, replacing the presentation in terms of $\mathbb{N}$-memory automata. This, however, seems difficult, since the class of ω-languages recognized by $\mathbb{N}$-memory automata has only poor logical closure properties (for instance, already closure under intersection fails).

4. How can one strengthen the model of $\mathbb{N}$-memory automaton, still keeping decidability results as needed to obtain an algorithmic solution of Church's synthesis problem?
5. Replace plays over $\mathbb{N}$ by plays over Σ^* for finite Σ.
6. A related problem is to find such results relying on decidability of the MSO-theory of the infinite binary tree rather than of $(\mathbb{N}, \mathit{Succc})$.

References

[1] Achim Blumensath, Thomas Colcombet & Christof Löding (2008): *Logical theories and compatible operations*. In Jörg Flum, Erich Grädel & Thomas Wilke, editors: *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas], Texts in Logic and Games* 2, Amsterdam University Press, pp. 73–106.

[2] Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick & Luc Segoufin (2011): *Two-variable logic on data words*. *ACM Transactions on Computational Logic* 12(4), p. 27, doi:10.1145/1970398.1970403.

[3] J. Richard Büchi (1966): *On a Decision Method in Restricted Second Order Arithmetic*. In Ernest Nagel, Patrick Suppes & Alfred Tarski, editors: *Proceedings of the 1960 International Congress on Logic, Methodology and Philosophy of Science*, *Studies in Logic and the Foundations of Mathematics* 44, Elsevier, pp. 1–11, doi:10.1016/S0049-237X(09)70564-6.

[4] J. Richard Büchi & Lawrence H. Landweber (1969): *Solving Sequential Conditions by Finite-State Strategies*. *Transactions of the American Mathematical Society* 138, pp. 295–311, doi:10.2307/1994916.

[5] Claudia Carapelle, Shiguang Feng, Alexander Kartzow & Markus Lohrey (2015): *Satisfiability of ECTL* with Tree Constraints*. In Lev D. Beklemishev & Daniil V. Musatov, editors: *Computer Science - Theory and Applications - 10th International Computer Science Symposium in Russia, CSR 2015, Listvyanka, Russia, July 13-17, 2015, Proceedings*, *Lecture Notes in Computer Science* 9139, Springer, pp. 94–108, doi:10.1007/978-3-319-20297-6_7.

[6] Arnaud Carayol & Matthew Hague (2014): *Regular Strategies in Pushdown Reachability Games*. In Joël Ouaknine, Igor Potapov & James Worrell, editors: *Reachability Problems - 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings*, *Lecture Notes in Computer Science* 8762, Springer, pp. 58–71, doi:10.1007/978-3-319-11439-2_5.

[7] Bruno Courcelle & Joost Engelfriet (2012): *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*. *Encyclopedia of Mathematics and its Applications* 138, Cambridge University Press, doi:10.1017/CBO9780511977619.

[8] Christopher Czyba, Christopher Spinrath & Wolfgang Thomas (2015): *Finite Automata Over Infinite Alphabets: Two Models with Transitions for Local Change*. In Igor Potapov, editor: *Developments in Language Theory - 19th International Conference, DLT 2015, Liverpool, UK, July 27-30, 2015, Proceedings.*, *Lecture Notes in Computer Science* 9168, Springer, pp. 203–214, doi:10.1007/978-3-319-21500-6_16.

[9] Stéphane Demri & Ranko Lazic (2009): *LTL with the freeze quantifier and register automata*. *ACM Trans. Comput. Log.* 10(3), doi:10.1145/1507244.1507246.

[10] Heinz-Dieter Ebbinghaus, Jörg Flum & Wolfgang Thomas (1994): *Mathematical logic (2. ed.)*. Undergraduate Texts in Mathematics, Springer, doi:10.1007/978-1-4757-2355-7.

[11] E. Allen Emerson & Charanjit S. Jutla (1991): *Tree Automata, Mu-Calculus and Determinacy (Extended Abstract)*. In: *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, IEEE Computer Society, pp. 368–377, doi:10.1109/SFCS.1991.185392.

[12] D. Gale & F.M. Stewart (1953): *Infinite games with perfect information*. In: *Contributions to the Theory of Games*, Ann. Math. Studies, Princeton Univ. Press, Princeton, N.J., pp. 245–266, doi:10.1515/9781400881970-014.

[13] Erich Grädel & Simon Leßenich (2012): *Banach-Mazur Games with Simple Winning Strategies*. In Patrick Cégielski & Arnaud Durand, editors: *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, *LIPIcs* 16, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 305–319, doi:10.4230/LIPIcs.CSL.2012.305.

[14] Michael Kaminski & Nissim Francez (1994): *Finite-Memory Automata*. *Theoretical Computer Science* 134(2), pp. 329–363, doi:10.1016/0304-3975(94)90242-9.

[15] Alexander S. Kechris (1995): *Classical Descriptive Set Theory*. *Graduate Texts in Mathematics* 156, Springer New York, New York, NY, doi:10.1007/978-1-4612-4190-4.

[16] Yiannis N. Moschovakis (2009): *Descriptive set theory*. 155, American Mathematical Soc., doi:10.1090/surv/155.

[17] Wolfgang Thomas (1997): *Languages, Automata, and Logic*. In Grzegorz Rozenberg & Arto Salomaa, editors: *Handbook of Formal Languages*, Springer Berlin Heidelberg, pp. 389–455, doi:10.1007/978-3-642-59126-6_7.

[18] Igor Walukiewicz (2001): *Pushdown Processes: Games and Model-Checking*. *Information and Computation* 164(2), pp. 234–263, doi:10.1006/inco.2000.2894.

[19] Igor Walukiewicz (2002): *Monadic second-order logic on tree-like structures*. *Theoretical Computer Science* 275(1-2), pp. 311–346, doi:10.1016/S0304-3975(01)00185-2.