Parametrized Regular Infinite Games and Higher-Order Pushdown Strategies

Paul Hänsch, Michaela Slaats*, and Wolfgang Thomas

Lehrstuhl Informatik 7, RWTH Aachen University, Aachen, Germany
paul.haensch@rwth-aachen.de
slaats@automata.rwth-aachen.de
thomas@informatik.rwth-aachen.de

Abstract. Given a set P of natural numbers, we consider infinite games where the winning condition is a regular ω-language parametrized by P. In this context, an ω-word, representing a play, has letters consisting of three components: The first is a bit indicating membership of the current position in P, and the other two components are the letters contributed by the two players. Extending recent work of Rabinovich we study here predicates P where the structure $(\mathbb{N}, +1, P)$ belongs to the pushdown hierarchy (or "Caucal hierarchy"). For such a predicate P where $(\mathbb{N}, +1, P)$ occurs in the k-th level of the hierarchy, we provide an effective determinacy result and show that winning strategies can be implemented by deterministic level-k pushdown automata.

1 Introduction

The starting point of this work is the Theorem of Büchi and Landweber [1]. This theorem gives a positive solution to "Church's Problem" on "regular" infinite games. In the simplest setting, we are dealing with a game where two players 1 and 2 choose bits in alternation, first player 1, then player 2, at each moment $i \in \mathbb{N}$. We call $X(i)$ the i-th bit chosen by player 1 and $Y(i)$ the i-th bit of player 2. A sequence $X(0), Y(0), X(1), Y(1), \ldots$ is a play of the game, and it defines (via the concept of characteristic function) two sets $X, Y \subseteq \mathbb{N}$. The winning condition of the game is a regular ω-language, presented in this paper by a monadic second-order formula $\varphi(X, Y)$ over the structure $(\mathbb{N}, +1)$. When a play (X, Y) satisfies this formula over the structure $(\mathbb{N}, +1)$ then player 2 wins the play, otherwise player 1 wins. In a standard way one now introduces the notion of strategy and winning strategy for the two players.

The Büchi-Landweber Theorem states that given a monadic second-order formula $\varphi(X, Y)$ as winning condition, the game associated with φ is determined (i.e., one of the two players has a winning strategy), one can decide who is the winner, and one can construct from φ a corresponding finite-state winning strategy (i.e., a strategy executable by a finite automaton with output).

* Research supported by DFG Research Training Group AlgoSyn ("Algorithmic Synthesis of Reactive and Discrete-Continuous Systems")

In the present paper we study a generalized setting in which a fixed set $P \subseteq \mathbb{N}$ is added as a "parameter". So one works in monadic second-order logic over a structure $(\mathbb{N}, +1, P)$ rather than $(\mathbb{N}, +1)$.

There are (at least) two motivations for this model: First, the resulting game can be viewed as an interaction between three agents. In addition to a standard scenario where a "controller" plays against a possibly hostile "environment" which is completely free in its choices, the predicate P represents now a "game context" that has dynamic behavior over time but is fixed and predictable. So one may call the games studied here *two-person games with context*. Secondly, the adjunction of a predicate P to the setting of the Büchi-Landweber Theorem gives a very natural step beyond the regular games, where new phenomena arise.

In [12, 13], Rabinovich showed that for recursive P, an analogue of the Büchi-Landweber Theorem holds *if the monadic second-order theory of* $(\mathbb{N}, +1, P)$ *is decidable*. In this case, determinacy holds again, the winner can be computed, and a recursive winning strategy (rather than a finite-state winning strategy) can be constructed from the winning condition.

The first aim of this paper is to develop a new presentation of Rabinovich's result which rests more on automata theoretic concepts than [12, 13]. While in that paper other sources are invoked for central details, we give a self-contained outline, using only standard facts.

Then we refine the claim on recursiveness of strategies in parametrized games, by providing – for a large class of sets P – a tight connection between the "complexity" of P and the complexity of winning strategies. Here we refer to those sets P such that the structure $(\mathbb{N}, +1, P)$ belongs to the "Caucal hierarchy" (of [6]). It is known that in this case the monadic second-order theory of $(\mathbb{N}, +1, P)$ is indeed decidable. A large class of interesting sets P is covered by the hierarchy, among them the powers k^n of a fixed number k, the powers n^k for fixed exponent k, and the set of factorial numbers $n!$. We show, using recent work of Carayol and Slaats [4], that for a set P such that $(\mathbb{N}, +1, P)$ belongs to the k-level of the hierarchy (short: "P is of level k"), a winning strategy (for the respective winner) can be guaranteed that also belongs to the k-th level. More precisely, we use the characterization of the levels of the Caucal hierarchy in terms of higher-order pushdown automata and show that for sets P of level k, winning strategies exist that are executable by deterministic level-k pushdown automata. This gives a substantial improvement over the general property of a strategy to be recursive (computable).

The last section offers a discussion and some open questions; e.g. on those predicates P where $(\mathbb{N}, +1, P)$ does not belong to the Caucal hierarchy but nevertheless the monadic second-order theory of this structure is decidable (for the latter class see e.g. [5, 14]).

2 Parametrized Regular Games and Their Solution

We use standard terminology as introduced, e.g., in [9]. By a regular game we mean an infinite two-player game in the sense of Gale and Stewart [8] where the winning condition is given by a regular ω-language. Both players, called 1 and 2, pick in each move an element from a finite alphabet; for notational simplicity we assume here that the alphabet is $\{0, 1\}$ for each of the players. (All definitions and results of this paper extend in a straightforward way to the case of arbitrary finite alphabets.) A play is a sequence $X(0), Y(0), X(1), Y(1), \ldots$ where $X(i)$ is supplied by player 1 and $Y(i)$ by player 2. As formalism to express winning conditions we use formulas $\varphi(X, Y)$ of monadic second-order logic (MSO-logic) over $(\mathbb{N}, +1)$; it is known that MSO-logic allows to define precisely the regular ω-languages. So we speak of a regular game. (We use here freely the correspondence between a set P of natural numbers and its characteristic bit sequence χ_P.) When a set P (and a corresponding constant again denoted P) is added, we refer to the structure $(\mathbb{N}, +1, P)$, denote the winning condition sometimes as $\varphi(P, X, Y)$, and speak of a *regular P-game*. A play of this game may be viewed as an ω word over the alphabet $\{0, 1\}^3$:

Predicate	0	1	1	0	1	...	$= P$
Player 1	0	1	0	1	1	...	$= X$
Player 2	1	0	1	0	1	...	$= Y$

The aim of this section is a new shape of proof for the following result of Rabinovich [12, 13].

Theorem 1. *Regular P-games are determined, and if the MSO-theory of the structure $(\mathbb{N}, +1, P)$ is decidable then the winner can be computed and a recursive winning strategy can be constructed from the winning condition.*

For the proof we use three fundamental results summarized in the following proposition (for details and definitions see [9]):

Proposition 1. (Known Facts)

(a) Each MSO-formula can be transformed into an equivalent (deterministic) parity automaton.

(b) The MSO theory of $(\mathbb{N}, +1, P)$ is decidable iff the following decision problem Aut_P is decidable.
Aut_P: Given a parity (or Büchi) automaton $\mathcal{A}$, does $\mathcal{A}$ accept χ_P?

(c) Parity games (even over infinite game arenas) are determined, and the winner has a positional winning strategy.

Proof. (of Theorem 1.) We present here a detailed sketch (a full account appears in [10]).

Step 1. Given $\varphi(P, X, Y)$ with fixed interpretation of P, we start with a parity automaton $\mathcal{A}_\varphi$, say with state set Q and $n = |Q|$, that is equivalent to $\varphi(Z, X, Y)$ (i.e. it has arbitrary ω-words over $\{0, 1\}^3$ as inputs). First we transform $\mathcal{A}_\varphi$ into a game arena. This just means to split a transition from state p via a triple (b, c, d) of bits into a state q into two transitions: The first takes the "context bit" b and the choice c of player 1 into account and leads from state p to an intermediate state (p, b, c). In this state the bit d supplied by player 2 is processed, and state q is reached. In the states p, Player 1 moves (first in the role of the "context player", then by his own bit) and in the states (p, b, c) player 2 moves. We obtain a finite game arena G_φ. The parity acceptance condition of the automaton is turned into a winning condition over G_φ; the coloring of vertices is inherited from the coloring of the states of the automaton.

Step 2. In a second step we transform G_φ into an infinite game arena which takes into account the fixed choice of the set P. For this we parametrize the vertices p and (p, b, c) by natural numbers, calling the new vertices $(p)_i$, respectively $(p, b, c)_i$. The initial vertex is now $(q_0)_0$. From $(p)_i$ we have an edge to $(p, b, c)_i$ iff in G_φ there is an edge from p to (p, b, c) *and* the bit b indicates correctly whether $i \in P$ or not (being 1 in the first and 0 in the second case). From $(p, b, c)_i$ we have an edge to $(q)_{i+1}$ iff in G_φ there is an edge from (p, b, c) to q. The color of $(p)_i$ is that of p, similarly for $(p, b, c)_i$. Call the resulting game graph G'_φ. Now we have:

Player 2 wins the regular P-game defined by φ iff Player 2 wins the parity game over G'_φ from its initial vertex.

Note that the game graph G'_φ is acyclic and structured into slices $S_0, S_1, \ldots$, each of which contains only a bounded number of vertices. For $k = 2i$, the slice S_k contains up to n $(= |Q|)$ vertices $(p)_i$, and for $k = 2i+1$, the slice S_k contains up to $2n$ vertices $(p, b, c)_i$ (note that b is fixed for given i). In order to have the same time scale in the characteristic sequence χ_P and the sequence of slices, we group the slices into a sequence of pairs $(S_0, S_1), (S_2, S_3), \ldots$ and code this sequence – and hence G'_φ – by an ω-word over an appropriate alphabet Σ. Let us denote by α_φ the ω-word coding G'_φ.

Finally, we note that the transformation of this step can be implemented by a finite automaton, uniformly in P:

Lemma 1. *Given a finite game arena G_φ, there is a finite-state transducer (in the format of a Mealy automaton) which transforms the characteristic sequence of a set P into the corresponding sequence α_φ.*

Step 3. By the memoryless determinacy of parity games, one of the two players has a memoryless winning strategy in G'_φ. From this we obtain the determinacy claim of the Theorem. We now deal with the effectiveness claims and by symmetry focus on player 2 alone. A memoryless strategy for player 2 is a function that maps each vertex $(p, b, c)_i$ to some state $(q)_{i+1}$, i.e. for each i we apply a map from a set with at most $2n$ elements to a set with at most n elements. Let Γ be the finite set of these maps. A memoryless strategy of player 2 is thus coded by an ω-word $\gamma = \gamma(0)\gamma(1)\ldots$ over Γ where $\gamma(i)$ is the map applied at moment i by player 2. It is a straightforward exercise to set up a deterministic parity automaton $\mathcal{T}_\varphi$ that runs on input words over $\Sigma \times \Gamma$ and checks for an ω-word $\alpha_\varphi \circ \gamma := (\alpha_\varphi(0), \gamma(0)),\ (\alpha_\varphi(1), \gamma(1)), \ldots$ whether γ represents a winning strategy in the parity game coded by α_φ.

Step 4. Invoking the transducer of Lemma 1 of Step 2, we can transform $\mathcal{T}_\varphi$ into an automaton $\mathcal{T}'_\varphi$ that runs over the input alphabet $\{0, 1\} \times \Gamma$ rather than $\Sigma \times \Gamma$. On an input $\chi_P \circ \gamma$, $\mathcal{T}'_\varphi$ computes, using the transducer, the sequence α_φ from χ_P and on $\alpha_\varphi \circ \gamma$ simultaneously simulates $\mathcal{T}_\varphi$. We call $\mathcal{T}'_\varphi$ a "winning strategy *tester*" for φ. Now we have: $\mathcal{T}'_\varphi$ accepts $\chi_P \circ \gamma$ iff γ represents a winning strategy of player 2 in the P-game with winning condition φ.

Step 5. The strategy tester of Step 4 will now be transformed into a nondeterministic "winning strategy *guesser*" $\mathcal{S}_\varphi$ that runs over the input alphabet $\{0, 1\}$ only. On the input word χ_P, this automaton guesses a sequence $\gamma \in \Gamma^\omega$ and on $\chi_P \circ \gamma$ works like $\mathcal{T}'_\varphi$. It is obtained in the format of a nondeterministic parity automaton. For convenience we assume it converted into a nondeterministic Büchi automaton $\mathcal{B}_\varphi$.

Proposition 2. *The Büchi automaton $\mathcal{B}_\varphi$ accepts the characteristic sequence of a set P iff player 2 has a winning strategy in the regular P-game with winning condition φ.*

This shows the first effectiveness claim of the Theorem: If the MSO-theory of $(\mathbb{N}, +1, P)$ is decidable, one can decide whether player 2 wins the regular P-game with winning condition φ. It just suffices to apply item (b) of Proposition 1 (of known facts) above.

Step 6. Finally, given that player 2 wins the regular P-game with winning condition φ, we have to construct a recursive strategy for him. In view of Step 5 it suffices to construct effectively an accepting run of $\mathcal{B}_\varphi$ from the assumption that such a run exists. (We use here the fact that the strategy can be extracted from an accepting run of the Büchi automaton.) In terms of MSO-logic, this amounts to the proof of a Selection Lemma: *Assume that the MSO-theory of* $(\mathbb{N}, +1, P)$ *is decidable. If* $(\mathbb{N}, +1) \models \exists Z \psi(P, Z)$ *then a satisfying recursive set Z can be constructed (i.e. a procedure that decides for each i whether $i \in Z$).*

We give a proof, following an argument of Siefkes [16], in automata theoretic terminology. It involves the well-known merging relation that was already used by McNaughton [11] in his proof of determinization of Büchi automata.

Let $\mathcal{B}$ be a Büchi automaton with state set S. We call two words $\mathcal{B}$-equivalent (short $u \sim_\mathcal{B} v$) if for each pair s, s' of states, $\mathcal{B}$ can reach s' from s via u iff this is the case for v.

Denote by $P[i, j]$ the segment $\chi_P(i) \dots \chi_P(j)$ of the characteristic sequence of P. Call two positions i, j *mergable* if there is a $k > i, j$ such that $P[i, k] \sim_\mathcal{B} P[j, k]$. This is an equivalence relation over $\mathbb{N}$ of finite index. We can compute a representative for each merge-equivalence class. For this, one uses the MSO-theory of $(\mathbb{N}, +1, P)$ repeatedly as "oracle", also in order to determine that enough representatives, say $n_1, \dots, n_m$, have been computed. (Just observe that i and j merge iff $(\mathbb{N}, +1, P)$ satisfies the sentence expressing $\exists z P[i, z] \sim_\mathcal{B} P[j, z]$. We know that all representatives occur up to position k by checking truth of the sentence expressing "$\forall x > k \exists y \leq k : x, y$ merge".)

Again using the MSO-theory of $(\mathbb{N}, +1, P)$ as oracle, we pick a representative n from $n_1, \dots, n_m$ with the following property: There is a $\mathcal{B}$-run ρ_{acc} on χ_P that visits a certain fixed final state q_f at infinitely many times k that merge with n. (It is clear how to express this property of n.) Note that such q_f and n can be found by a finite search process, due to the finite index of the merging relation and the assumption that an accepting run exists.

Using q_f and n, we construct effectively a run ρ of $\mathcal{B}$ on χ_P visiting q_f infinitely often, thus accepting χ_P. We start out by looking for a position p_0 which merges with n *and* such that q_f is reachable from the initial state q_0 of $\mathcal{B}$ via $P[0, p_0 - 1]$ using a finite run ρ_0. Such p_0 exists by assumption on n. The run ρ_0 will be an initial segment of the desired accepting run ρ. For some $k_0 > n, p_0$ we know $P[n, k_0] \sim_{\mathcal{B}} P[p_0, k_0]$. Hence ρ_0 can be extended such that at position k_0 the same state as that of ρ_{acc} is reached. We can now pick $p_1 > k_0$ such that p_1 merges again with n *and* such that q_f is reachable from q_0 via $P[0, p_1 - 1]$, by a finite run which is an extension of ρ_0. Call this finite run ρ_1. Continuing in this way by successive finite extensions each of which is computable and ends by a final state, we construct the accepting run ρ as desired. By the choice of $n_1, \ldots, n_m$, the number $n \in \{n_1, \ldots, n_m\}$, and the state q_f, we can check for the merge equivalence between n and candidate numbers c by an effective procedure; note that for sufficiently high k we always find that $P[c, k] \sim_{\mathcal{B}} P[n_i, k]$ for some i. So the sequence of numbers $p_0, p_1, \ldots$ is computable if P is recursive. For arbitrary P the sequence is recursive in P.

On the other hand, it is to be noted that the *construction* of this strategy (which is recursive in P) involves an unbounded number of queries to the MSO-theory of $(\mathbb{N}, +1, P)$. These queries are needed for the computation of the above-mentioned paramaters $n_1, \ldots, n_m, n, q_f$. For the original specification φ let p_φ be the corresponding tuple $(n_1, \ldots, n_m, n, q_f)$ of parameters. The function $F : \varphi \mapsto p_\varphi$ captures the complexity of the synthesis problem for the set P; this function (or rather its graph considered as a set S) is Turing-reducible to the MSO-theory of $(\mathbb{N}, +1, P)$. We do not know whether this reducibility relation can be sharpened to tt-reducibility (truth-table reducibility; see [15]). It is known that in general the MSO-theory of $(\mathbb{N}, +1, P)$ is tt-reducible (but not btt-reducible) to the second jump P'' of P ([17]). So for the set S coding the construction of winning strategies we have

$$S \leq_{\mathrm{T}} \text{MSO-theory of } (\mathbb{N}, +1, P) \leq_{\mathrm{tt}} P''.$$

3 Background on Higher-order Pushdown Automata

In the next two sections we consider sets P such that the structure $(\mathbb{N}, +1, P)$ belongs to the Caucal hierarchy. Caucal introduced in [6] a large class of infinite graphs which can be generated starting from finite trees and graphs applying MSO-interpretations and unfoldings in alternation. The resulting hierarchy is a very rich collection of models each of them having a decidable MSO-theory. In [3] Carayol and Wöhrle showed that the graphs of the Caucal hierarchy coincide with the transition graphs of higher-order pushdown automata. We will develop here a representation of the parameter sets P by higher-order pushdown automata. For this we define a new type of deterministic higher-order pushdown automaton that produces an infinite 0-1-sequence (and hence a set P) as output.

We start with some background definitions, in three stages: We introduce higher-order pushdown systems, higher-order pushdown generators (of sets P), and higher-order pushdown games.

A level-1 stack over a finite alphabet Γ can be seen as a word of Γ^*; the empty stack (written $[\,]_1$) corresponds just to ε. A *level-*$(k{+}1)$ *stack* for $k \geq 1$ is a non-empty sequence of level-k stacks. The empty stack of level $k+1$ is the level-$(k{+}1)$ stack containing only the empty stack of level k and is written $[\,]_{k+1}$. The set of all stacks of some level is written $Stacks_1(\Gamma) := \Gamma^*$ for level 1 and $Stacks_{k+1}(\Gamma) := (Stacks_k(\Gamma))^+$ for level $k \geq 1$.

We define the following partial functions on higher-order stacks called *operations*. On level 1 we have as operations for each symbol $x \in \Gamma$ the operations $push_x$ and pop_x. They are respectively defined on level-1 stacks by $push_x([s_0,\ldots,s_n]_1) = [s_0,\ldots,s_n,x]_1)$ and $pop_x([s_0,\ldots,s_n,x]_1) = [s_0,\ldots,s_n]_1$. For each level $k{+}1 \geq 2$, we consider the level-$(k{+}1)$ operation $copy_k$ which adds a copy of the top-most level-k stack on top of the existing level-k-stacks. We also allow the symmetric operation $\overline{copy}_k$ which removes the top-most level-k stack if it is equal to its predecessor level-k-stack. Formally, these operations are defined on level-$(k{+}1)$ stacks by $copy_k([s_0,\ldots,s_n]_{k+1}) = [s_0,\ldots,s_n,s_n]_{k+1}$ and $\overline{copy}_k([s_0,\ldots,s_n,s_n]_{k+1}) = [s_0,\ldots,s_n]_{k+1}$. In addition, for each level k, we define a level-k operation written $T_{[\,]_k}$ allowing to test emptiness at level k. Formally $T_{[\,]_k}(s)$ is equal to s if $s = [\,]_k$ and is undefined otherwise.
An operation ψ of level k is extended to stacks of level $\ell > k$ using the definition $\psi([s_0,\ldots,s_n]_\ell) = [s_0,\ldots,\psi(s_n)]_\ell$. We now define inductively $Ops_1 = \{push_x, pop_x \,|\, x \in \Gamma\} \cup \{T_{[\,]_1}\}$ and $Ops_{k+1} = Ops_k \cup \{copy_k, \overline{copy}_k, T_{[\,]_{k+1}}\}$. Moreover, we denote by Ops_k^* the monoid for the compositions of partial functions generated by Ops_k.

Definition 1. *A higher-order pushdown system $\mathcal{A}$ of level k (k-HOPDS for short) is defined as a tuple (Q,Σ,Γ,Δ) where Q is the finite set of states, Σ is the input alphabet, Γ is the stack symbol alphabet and $\Delta \subseteq Q \times \Sigma \times Ops_k \times Q$ is the transition relation.*

A configuration is a pair $(p,s) \in Q \times Stacks_k(\Gamma)$. We write $(p,s) \xrightarrow{\alpha} (q,s')$ if there exists a transition $(p,\alpha,\rho,q) \in \Delta$ such that $s' = \rho(s)$.

Now we introduce a notion of regularity for sets of higher-order pushdown stacks which relies on the construction of the stacks by operations. We need "regular" sets of stacks for a new type of tests in deterministic higher-order pushdown automata. This format will be appropriate for the generation of 0-1-sequences (i.e., predicates $P \subseteq \mathbb{N}$).
The notion of *regularity* for (symmetric) operations was introduced independently in [2] and [7]. Observe that from a given level-k-stack a word from Ops_k^* yields a new stack, and a language $O \subseteq Ops_k^*$ a set of stacks. A set of level-k stacks is *regular* if it can be obtained by applying a regular subset of Ops_k^* to the empty level-k stack $[\,]_k$. We write $OReg_k(\Gamma)$ for the regular sets of stacks of level k.

In the subsequent definition of pushdown automata that produce a 0-1-sequence as output, we refer to a finite family $\mathcal{R}$ of regular sets of stacks. The output alphabet is $\Sigma = \{0,1,\varepsilon\}$; ε serves as a formal output token for the transitions that do not produce either 0 or 1. By τ we shall denote the identity function on Ops_k, i.e. $\tau(s) = s$ for all $s \in Stacks_k(\Gamma)$.

Definition 2. *A* higher-order pushdown sequence generator of level k *(short: k-HOPDSG) is a deterministic higher-order pushdown automaton $\mathcal{A}$ of level k with tests in a finite set $\mathcal{R}$ of subsets of $Stacks_k(\Gamma)$ which is given by the tuple $(Q, \Sigma, \Gamma, q_0, \Delta)$ where Q is a finite set of states, $\Sigma = \{0, 1, \varepsilon\}$ is the output alphabet, Γ is the stack alphabet, $q_0 \in Q$ is the initial state, and $\Delta \subseteq Q \times \Sigma \times Ops_k \times \mathcal{R} \times Q$ is the transition relation. The set of tests is defined by $\mathcal{R} = \{T_1, \ldots, T_n\}$ with $T_i \in OReg_k(\Gamma)$ for all $i \in [1, n]$.*

A configuration of $\mathcal{A}$ is again a tuple in $Q \times Stacks_k(\Gamma)$ and the initial configuration is $(q_0, [\ ]_k)$. We write $(p, s) \xrightarrow{\alpha} (q, s')$ if there exists a transition $(p, \alpha, \gamma, T, q) \in \Delta$, such that $s' = \gamma(s)$ and $s \in T$.
The automaton is *deterministic* if for every configuration (q, s) there is at most one transition $(q, \alpha, \gamma, T, p)$ in Δ which can be applied.
An ω-word $\alpha \in \{0, 1\}^\omega$ is *defined* by the automaton $\mathcal{A}$ if there exists an infinite run $(q_0, [\ ]_k) \xrightarrow{\alpha_0} (q_1, s_1) \xrightarrow{\alpha_1} (q_2, s_2) \xrightarrow{\alpha_2} (q_3, s_3) \xrightarrow{\alpha_3} \ldots$ such that α is obtained from $\alpha_0\alpha_1\alpha_2\alpha_3 \ldots$ by deleting all occurrences of ε. (Of course, an automaton may produce just a finite word. We focus on the infinite words generated by HOPDSG's.)

The "regular tests" in our level k-HOPDSG's are introduced to obtain a model of computation that is deterministic and generates precisely the sets P such that $(\mathbb{N}, +1, P)$ is in the Caucal hierarchy. Determinism is needed for our game-theoretic context. The automata in the literature have less powerful tests but are non-deterministic. In our model we can restrict to apply a "test" which checks if the operations that follow the current transition can indeed be applied to the current stack. We shall use the tests only in transitions with output ε and then speak of *restricted tests.*

Definition 3. *A set $P \subseteq \mathbb{N}$ is* level-k-definable *if there is a higher-order pushdown sequence generator $\mathcal{A}$ of level k with restriced tests that defines P.*

Theorem 2. *A structure $(\mathbb{N}, +1, P)$ is in the k-th level of the Caucal hierarchy iff P is level-k-definable.*

As an example for the application of sequence generators, let us describe the idea for a level-2 higher-order pushdown sequence generator defining the set $P = \{2^i \mid i \in \mathbb{N}\}$ of the powers of 2. Note that after output 1 at position 2^i, the next output 1 occurs 2^i steps later at position 2^{i+1}. The idea for the automaton is to remember in its first level 1 stack the current i by the stack content 0^i. Above this bottom-line the automaton can build a tower of i stacks with the contents $0^{i-1}, 0^{i-2}, \ldots, 0$. We can now allow the top symbols of these i stacks to be 0 or 1; so the sequence of $b_1 \ldots b_i$ of top symbols is a binary number (the leading bit corresponds to the bottom stack) which we use to "count" in binary up to 1^i, where of course many steps are needed to proceed from one binary number to the next. When such a new binary number is reached the automaton outputs a 0 (otherwise ε). More precisely, the automaton deletes the stacks with top symbol 1 until it reaches a stack with top symbol 0; it turns it into 1 and goes up again building towers of 0 of decreasing length as at the start:

$$\begin{bmatrix} \mathbf{0} & & & \\ 0 & \mathbf{1} & & \\ 0 & 0 & \mathbf{1} & \\ 0 & 0 & 0 & \mathbf{0} \end{bmatrix} \Rightarrow \begin{bmatrix} \mathbf{1} & & & \\ 0 & \mathbf{1} & & \\ 0 & 0 & \mathbf{1} & \\ 0 & 0 & 0 & \mathbf{0} \end{bmatrix} \Rightarrow \begin{bmatrix} & & & \\ & & & \\ & & & \\ 0 & 0 & 0 & \mathbf{1} \end{bmatrix} \Rightarrow \begin{bmatrix} \mathbf{0} & & & \\ 0 & \mathbf{0} & & \\ 0 & 0 & \mathbf{0} & \\ 0 & 0 & 0 & \mathbf{1} \end{bmatrix}$$

Let us continue the example and discuss a regular P-game for $P = \{2^i \mid i \in \mathbb{N}\}$. The winning condition requires that player 2 copies the bits played by player 1 except for the moments $i-1$ where $i \in P$; in these moments the converse bit is required. An example play won by player 2 could be:

Set P	0 1 1 0 1 0 0 0 1 0 . . .
Player 1	0 1 0 0 1 0 0 1 1 1 . . .
Player 2	1 0 0 1 1 0 0 0 1 1 . . .

It is easy to see that a finite-state winning strategy does not suffice for player 2 to win this game; no finite memory suffices to determine the moments $i-1$ with $i \in P$. On the other hand, if player 2 has the computational means of a HOPDSG that defines P, he can detect the critical moments without using a look-ahead.

We return to the preparations of main result. In the following we introduce parity games played on the configuration graph of a higher-order pushdown system and a result we need for our main theorem.

Definition 4. *A* higher-order pushdown parity game $\mathcal{G}$ of level k *(k-HOPDPG) is given by a k-HOPDS $P = (Q, \Sigma, \Gamma, \Delta)$, a partition of the states $Q_0 \uplus Q_1$ and a coloring mapping $\Omega_P : Q \to \mathbb{N}$. The induced game arena is (V_0, V_1, E, Ω) where: $V_0 = Q_0 \times \mathrm{Stacks}_k(\Gamma)$, $V_1 = Q_1 \times \mathrm{Stacks}_k(\Gamma)$, E is the Σ-labeled transition relation of P and Ω is defined for $(p, s) \in Q \times \mathrm{Stacks}_k(\Gamma)$ by $\Omega(p, s) := \Omega_P(p)$.*

Theorem 3 **([4]).** *Given a pushdown parity game of level k, we can construct in* k-EXPTIME *reduced level-k automata*[1] *describing the winning region, respectively a global positional winning strategy for each of the two players.*

4 Regular *P*-Games with *P* in the Pushdown Hierarchy

We now want to show that a regular P-game where P is defined by a higher-order pushdown sequence generator of level k with restricted tests can be solved in k-EXPTIME, and that the winner has a winning strategy which is executable by a level-k pushdown automaton.

Theorem 4. *Let $P \subseteq \mathbb{N}$ be defined by a higher-order pushdown sequence generator $\mathcal{P}$ of level k with restricted tests. The regular P-game where the winning condition is given by a deterministic parity word automaton $\mathcal{C}$ over $\{0,1\}^3$ is (determined and) solvable: It can be decided who wins the game and for the winner one can construct a level-k HOPDA that computes a winning strategy.*

[1] The reduced level-k automata are finite automata running over Ops_k and accepting regular sets of stacks, i.e. sets in $OReg_k(\Gamma)$. See [4] for more details.

In the proof, we first treat solvability and the format of the winning strategy; the statement on complexity is shown afterwards.

Proof. Let $\mathcal{P} = (Q_{\mathcal{P}}, \Sigma_{\mathcal{P}}, \Gamma_{\mathcal{P}}, q_0^{\mathcal{P}}, \Delta_{\mathcal{P}})$ be a k-HOPDSG with restricted tests defining P, and let $\mathcal{C} = (Q_{\mathcal{C}}, \Sigma_{\mathcal{C}}, q_0^{\mathcal{C}}, \delta_{\mathcal{C}}, \Omega_{\mathcal{C}})$ be a parity word automaton over the alphabet $\Sigma_{\mathcal{C}} = \{0,1\}^3$ defining the winning condition.

We construct a higher-order pushdown parity game (HOPDPG) $\mathcal{G}_P$, defined by the HOPDS $\mathcal{P}_{\mathcal{G}} = (Q, \Sigma_{\mathcal{P}}, \Gamma_{\mathcal{P}}, q_0, \Delta)$, the state partition Q_1, Q_2 and the coloring Ω, simulating the game between player 1 and player 2 with the external parameter P. The idea is that in $\mathcal{G}_P$ we compute with the help of $\mathcal{P}$, i.e. the level-k stack, the next bit of the sequence χ_P, then let first player 1 choose a bit then player 2. These three bits we store in the state of the current vertex and then compute by $\mathcal{C}$ the color of its vertex. (For this we give $\mathcal{C}$ those three bits as input.) The parity game $\mathcal{G}_P$ is then won by player 2 iff the given regular P-game is won by player 2. Using this allows us to invoke Theorem 3 to solve the game $\mathcal{G}_P$ and compute a winning strategy.
The HOPDS $\mathcal{P}_{\mathcal{G}}$ works repeatedly in four phases, indicated by the symbols of the alphabet $\Phi := \{\Phi_P, \Phi_1, \Phi_2, \Phi_C\}$. The symbol Φ_P indicates that the next bit of χ_P is computed by $\mathcal{P}$, the symbol Φ_i that player i chooses a bit, and the symbol Φ_C that the next state of $\mathcal{C}$ is computed by evaluating the chosen bits.
The HOPDS $\mathcal{P}_{\mathcal{G}}$ has the state set $Q = Q_{\mathcal{P}} \times Q_{\mathcal{C}} \times \Phi \times \{0,1\}^3$ where for a state $(q_{\mathcal{P}}, q_{\mathcal{C}}, x, (b_0, b_1, b_2)) \in Q$ we have that $q_{\mathcal{P}}$ resp. $q_{\mathcal{C}}$ is the current state in $\mathcal{P}$ resp. $\mathcal{C}$. Furthermore by the third component we know in which phase of a move we are, and by (b_0, b_1, b_2) we memorize the current bits of χ_P and the last bits chosen by player 1 and player 2. The start state is $q_0 = (q_0^{\mathcal{P}}, q_0^{\mathcal{C}}, \Phi_P, (0,0,0))$. The transitions Δ are the following. (Note that the bits b'_0, b'_1, b'_2 are the current choices for χ_P, player 1, respectively player 2.)

- for $(q_{\mathcal{P}}, \varepsilon, \gamma, T, q'_{\mathcal{P}}) \in \Delta_{\mathcal{P}}$:
  $((q_{\mathcal{P}}, q_{\mathcal{C}}, \Phi_{\mathcal{P}}, (b_0, b_1, b_2)), \varepsilon, \gamma, (q'_{\mathcal{P}}, q_{\mathcal{C}}, \Phi_{\mathcal{P}}, (b_0, b_1, b_2)))$
- for $b'_0 \in \{0,1\}, (q_{\mathcal{P}}, b'_0, \gamma, T, q'_{\mathcal{P}}) \in \Delta_{\mathcal{P}}$:
  $((q_{\mathcal{P}}, q_{\mathcal{C}}, \Phi_{\mathcal{P}}, (b_0, b_1, b_2)), b'_0, \gamma, (q'_{\mathcal{P}}, q_{\mathcal{C}}, \Phi_1, (b'_0, b_1, b_2)))$
- for $b'_1 \in \{0,1\}$: $((q_{\mathcal{P}}, q_{\mathcal{C}}, \Phi_1, (b_0, b_1, b_2)), b'_1, \tau, (q_{\mathcal{P}}, q_{\mathcal{C}}, \Phi_2, (b_0, b'_1, b_2)))$
- for $b'_2 \in \{0,1\}$: $((q_{\mathcal{P}}, q_{\mathcal{C}}, \Phi_2, (b_0, b_1, b_2)), b'_2, \tau, (q_{\mathcal{P}}, q_{\mathcal{C}}, \Phi_{\mathcal{C}}, (b_0, b_1, b'_2)))$
- for $\delta_{\mathcal{C}}(q_{\mathcal{C}}, (b_0, b_1, b_2)) = q'_{\mathcal{C}}$:
  $((q_{\mathcal{P}}, q_{\mathcal{C}}, \Phi_{\mathcal{C}}, (b_0, b_1, b_2)), \varepsilon, \tau, (q_{\mathcal{P}}, q'_{\mathcal{C}}, \Phi_{\mathcal{P}}, (b_0, b_1, b_2)))$

The coloring is given by $\Omega((q_{\mathcal{P}}, q_{\mathcal{C}}, x, (b_0, b_1, b_2))) = \Omega_{\mathcal{C}}(q_{\mathcal{C}})$ for $x \in \{\Phi_{\mathcal{C}}, \Phi_1, \Phi_2\}$ and $\Omega((q_{\mathcal{P}}, q_{\mathcal{C}}, \Phi_{\mathcal{P}}, (b_0, b_1, b_2))) = (2 \cdot n)$ where n is the maximal color in $\Omega_{\mathcal{C}}$.
The state partitioning is defined by $Q_1 = Q_{\mathcal{P}} \times Q_{\mathcal{C}} \times \{\Phi_1, \Phi_{\mathcal{P}}, \Phi_{\mathcal{C}}\} \times \{0,1\}^3$ and $Q_2 = Q_{\mathcal{P}} \times Q_{\mathcal{C}} \times \{\Phi_2\} \times \{0,1\}^3$.

The restricted tests which are used in the computation of χ_P, i.e. in the transitions of $\mathcal{P}$ to make the HOPDSG deterministic, are omitted in the game. This can be done because of their special form. Note that in the game $\mathcal{G}_P$ the computation of P is not completely deterministic because we attribute to player 1 the choice of bits for the sequence χ_P. If player 1 chooses such a bit incorrectly, however, then either the current stack operation or one of the subsequent ones will be undefined or he would get stuck in the computation of a later χ_P-bit (here we use the resticted tests). In the case that in a Φ_1-state some operation is not defined on the current stack, player 1 loses immediately; in the second case he will lose because the only color which is seen infinitely often in the game will be even; note that we colored the $\Phi_{\mathcal{P}}$-vertices by an even number that cannot be surpassed; so player 2 wins in this case.

The idea for the construction of the strategy automaton for the player winning the game is similar as above. Assume player 2 wins the game $\mathcal{G}_P$. Then by Theorem 3 we get two regular sets of level-k stacks, say S_0 and S_1 where S_0 contains all configurations[2] where player 2 should take 0 as output and S_1 those where output 1 should be taken.

The strategy automaton is constructed similarly as the automaton $\mathcal{P}_{\mathcal{G}}$ except that in the transitions with Φ_2 we add as tests once S_0 and and once S_1, which ensure that player 2 takes the right transition. These tests are also used for the output function which outputs the corresponding bit for player 2. — If player 1 wins the game the construction is analogous.

Proposition 3. *The computation of the winner and the winning strategy in Theorem 4 is done in k-exponential time.*

Proof. By Theorem 3 we have a k-EXPTIME procedure to compute the winner of the game $\mathcal{G}_P$ and the positional winning strategy for the player winning $\mathcal{G}_P$. As the construction of $\mathcal{G}_P$ is polynomial in the size of the automata $\mathcal{P}$ and $\mathcal{C}$ we have altogether again an algorithm running in k-exponential time to compute the winner of the regular P-game as well as the desired winning strategy automaton.

5 Conclusion

The purpose of the present paper was twofold: First we developed a streamlined proof of a result of Rabinovich [12, 13] on regular P-games, using automata theoretic concepts and ideas that go back to Siefkes [16]. The result says that for recursive P, regular P-games can be solved effectively when the MSO-theory of $(\mathbb{N}, +1, P)$ is decidable, and that in this case also a recursive winning strategy for the winner can be constructed.

In the second part of the paper, we considered predicates that can be generated by higher-order pushdown automata (covering a large class of interesting examples) and showed that for such predicates P, regular P-games can be solved with strategies that are again computable by such automata. In this context, we mention some questions.

In natural examples, mentioned e.g. at the end of Section 3, the reference to P in the winning condition involves just a bounded look-ahead on P. In our approach a look-ahead is made superfluous by a corresponding computation from the past, which involves a big overhead. Strategies (maybe even finite-state

[2] The state of the configuration (p, s) is stored in the set by pushing it onto the topmost stack, i.e. we have $push_p(s) \in S_0$.

strategies) with bounded look-ahead on P seem to be a natural class, and the range of their applicability should be investigated.

A related question is to decide when a regular P-game where $(\mathbb{N}, +1, P)$ is in the Caucal hierarchy can be solved with finite-state winning strategies.

Finally, one can aim at finding more general frameworks than the Caucal hierarchy as considered here, and develop corresponding more general types of winning strategies (that are more restricted than the recursive strategies).

References

1. R. Büchi and L. Landweber. Solving sequential conditions by finite state strategies. *Transactions of the AMS*, 138(27):295 – 311, 1969.
2. A. Carayol. Regular sets of higher-order pushdown stacks. In *Proc. MFCS*, volume 3618 of *LNCS*, pages 168–179, Heidelberg, 2005. Springer.
3. A. Carayol and S. Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *Proc. FSTTCS*, volume 2914 of *LNCS*, pages 112–123, Heidelberg, 2003. Springer.
4. Arnaud Carayol and Michaela Slaats. Positional strategies for higher-order pushdown parity games. In *Proc. MFCS*, volume 5162 of *LNCS*, pages 217–228, Heidelberg, 2008. Springer.
5. O. Carton and W. Thomas. The monadic theory of morphic infinite words and generalizations. *Inf. Comput.*, 176(1):51–65, 2002.
6. D. Caucal. On infinite graphs having a decidable monadic theory. In *Proc. MFCS*, volume 2420 of *LNCS*, pages 165–176, Heidelberg, 2002. Springer.
7. S. Fratani. *Automates à piles de piles . . . de piles.* PhD thesis, Université Bordeaux 1, 2005.
8. D. Gale and F. M. Stewart. Infinite games with perfect information. In *Contributions to the Theory of Games.* Princeton University Press, 1953.
9. E. Grädel, W. Thomas, and T. Wilke, editors. *Automata Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *LNCS*. Springer, Heidelberg, 2002.
10. P. Hänsch. Infinite games with parameters. Master's thesis, Lehrstuhl für Informatik 7, RWTH Aachen, Aachen, Germany, 2009.
11. R. McNaughton. Testing and generating infinite sequences by a finite automaton. In *Inform. Contr.*, volume 9, pages 521–530, 1966.
12. A. Rabinovich. Church synthesis problem with parameters. In *Proc. CSL*, volume 4207 of *LNCS*, pages 546–561, Heidelberg, 2006. Springer.
13. A. Rabinovich. Church synthesis problem with parameters. *Logical Methods in Computer Science*, 3(4:9):1–24, 2007.
14. A. Rabinovich and W. Thomas. Decidable theories of the ordering of natural number with unary predicates. In *Proc. CSL*, volume 4207 of *LNCS*, pages 562 – 574, Heidelberg, 2006. Springer.
15. H. Rogers. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill, New York, 1967.
16. D. Siefkes. The recursive sets in certain monadic second order fragments of arithmetic. *Archiv math. Logik*, 17:71–80, 1975.
17. W. Thomas. The theory of successor with an extra predicate. *Math. Ann.*, 237:121–132, 1978.