RWTH Aachen

Department of Computer Science
Technical Report

Parametrized Regular Infinite Games and Higher-Order Pushdown Strategies

Paul Hänsch, Michaela Slaats and Wolfgang Thomas

ISSN 0935–3232 · Aachener Informatik Berichte · AIB-2009-18

RWTH Aachen · Department of Computer Science · September 2009

Parametrized Regular Infinite Games and Higher-Order Pushdown Strategies

Paul Hänsch, Michaela Slaats[⋆], and Wolfgang Thomas

Lehrstuhl für Informatik 7
RWTH Aachen University, Germany
Email: 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).

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"

[⋆] Research supported by DFG Research Training Group AlgoSyn ("Algorithmic Synthesis of Reactive and Discrete-Continuous Systems") 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.

For instance, it is easy to see that even for a recursive set P, the exact analogue of the Büchi-Landweber Theorem fails. This is clear from the following example: Consider a non-recursive, recursively enumerable set S with enumeration $s_0, s_1, \ldots$, and let w be the ω-word $10^{s_0}10^{s_1}1\ldots$, which defines (the characteristic function χ_P of) a set P. Clearly P is recursive. We then have $n \in S$ iff the segment 10^n1 occurs in χ_P, i.e., iff there is a P-element such that its $(n+1)$-st successor is the next P-element. Thus S is reducible to the monadic second-order (even the first-order) theory of $(\mathbb{N}, +1, P)$ which must hence be undecidable. Now consider the winning condition $\varphi_n(X, Y)$ which does not depend on X and says: "Produce output 1^ω if the segment 10^n1 occurs in χ_P, otherwise 0^ω." Clearly for each φ_n there is a winning strategy for player 2, but it cannot be computed from φ_n.

In [13, 14], 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 [13, 14]. 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, 15]).

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

$$X(0), Y(0), X(1), Y(1), \dots$$

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 [13, 14].

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 [11]).

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.

$$\begin{aligned}
\delta(q_0, (0,0,0)) &= q_0\\
\delta(q_0, (0,0,1)) &= q_1\\
\delta(q_0, (0,1,0)) &= q_1\\
\delta(q_0, (0,1,1)) &= q_2\\
\delta(q_0, (1,0,0)) &= q_3\\
\delta(q_0, (1,0,1)) &= q_2\\
\delta(q_0, (1,1,0)) &= q_3\\
\delta(q_0, (1,1,1)) &= q_3
\end{aligned}$$

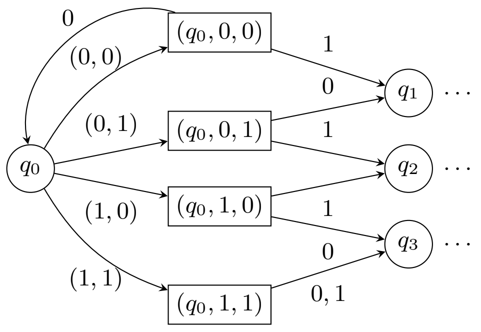

Fig. 1. Example for step 1 in the proof of Theorem 1. For the given transitions of δ the corresponding game graph $\mathcal{G}_\varphi$ is depicted.

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.*

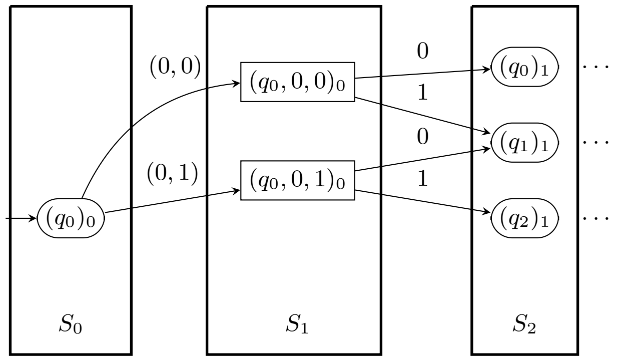

Fig. 2. Example for step 2 in the proof of Theorem 1. We assume $0 \notin P$ and show the unfolding of the game graph $\mathcal{G}_\varphi$ (of Figure 1) over time which results in the depicted game graph $\mathcal{G}'_\varphi$. Further more we have marked the slices S_0, S_1, S_2.

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 [17], in automata theoretic terminology. It involves the well-known merging relation that was already used by McNaughton [12] 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 $n_1, \dots, n_m$, the number $n \in \{n_1, \dots, n_m\}$, 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 $p_0, p_1, \dots$ 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 parameters $n_1, \dots, n_m, n, q_f$. For the original specification φ let p_φ be the corresponding tuple $(n_1, \dots, n_m, n, q_f)$ 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 [16]). 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 ([18]). 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 on higher-order pushdown stacks and systems. For this we first introduce the higher-order pushdown stacks and operations to manipulate them. Afterwards we define higher-order pushdown systems and introduce a concept of "regularity" for sets of higher-order stacks. Then we define deterministic higher-order pushdown sequence generators which produce sets $P \subseteq \mathbb{N}$ as output. Finally we give the definition of higher-order pushdown parity games and recall a result of [4] that we need.

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 $k \geq 1$.

Example 1. Consider for example the level-2 stack $s_2 = [[abb]_1[abc]_1[acc]_1]_2$. It consists of three level-1 stacks and we use the convention that $s_1 = [acc]_1$ is the topmost level-1 stack of s_2 and c is the topmost stack symbol of s_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:

$$push_x([s_0,\dots,s_n]_1) = [s_0,\dots,s_n,x]_1,$$
$$pop_x([s_0,\dots,s_n]_1) \;\; = \begin{cases} [s_0,\dots,s_{n-1}]_1 & \text{if } s_n = x \\ \text{non defined} & \text{otherwise.} \end{cases}$$

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:

$$copy_k([s_0,\dots,s_n]_{k+1}) \qquad = [s_0,\dots,s_n,s_n]_{k+1},$$
$$\overline{copy}_k([s_0,\dots,s_{n-1},s_n]_{k+1}) = \begin{cases} [s_0,\dots,s_{n-1}]_{k+1} & \text{if } s_{n-1} = s_n \\ \text{non defined} & \text{otherwise.} \end{cases}$$

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 $\psi([s_0,\dots,s_n]_\ell) = [s_0,\dots,\psi(s_n)]_\ell$. We now define inductively the set of all operations of some level $k+1$ over some stack alphabet Γ by:

$$\mathrm{Ops}_1 = \{push_x, pop_x \mid x \in \Gamma\} \cup \{T_{[\;]_1}\},$$
$$\mathrm{Ops}_{k+1} = \mathrm{Ops}_k \cup \{copy_k, \overline{copy}_k, T_{[\;]_{k+1}}\}.$$

Moreover, we denote by Ops_k^* the monoid for the compositions of partial functions generated by Ops_k.

Example 2. Take the stack $s_2 = [[abb]_1[abc]_1[acc]_1]_2$ and the operation sequence $\rho = pop_c pop_c push_b push_c \overline{copy_1}$. If we apply σ to s_2 we proceed as follows:

$$\begin{aligned} pop_c pop_c push_b push_c \overline{copy_1}\ ([[abb]_1[abc]_1[acc]_1]_2) &= \\ pop_c push_b push_c \overline{copy_1}\ ([[abb]_1[abc]_1[ac]_1]_2) &= \\ push_b push_c \overline{copy_1}\ ([[abb]_1[abc]_1[a]_1]_2) &= \\ push_c \overline{copy_1}\ ([[abb]_1[abc]_1[ab]_1]_2) &= \\ \overline{copy_1}\ ([[abb]_1[abc]_1[abc]_1]_2) &= \\ [[abb]_1[abc]_1]_2 & \end{aligned}$$

Definition 1. *A higher-order pushdown system $\mathcal{A}$ of level k (k-HOPDS for short) is defined as a tuple $(Q, \Sigma, \Gamma, \Delta)$ 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 $(p,\alpha,\gamma,q) \in \Delta$ such that $s' = \gamma(s)$.

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 $O \subseteq \mathrm{Ops}_k^*$ 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 $\mathrm{OReg}_k(\Gamma)$ for the regular sets of stacks of level k.

Example 3. To illustrate the here defined term of regularity for sets of higher-order pushdown stacks we will consider some example. So take as set of stacks of level 2 the set:

$$\{[[a^n][a^{n-1}]\dots[a][\ ]]_2 \mid n \geq 1\}.$$

It can be generated by the following regular operation sequence which is applied to the empty level-2 stack:

$$push_a^+(copy_1pop_a)^+T_{[\ ]_1}([\ ]_2).$$

The sequence $push_a^3(copy_1pop_a)^3T_{[\ ]_1}$ applied to $[\ ]_2$ generates for example the stack $[[aaa][aa][a][\ ]]_2$.

In the subsequent definition of a pushdown automaton that produces 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 $\mathcal{R} = \{T_1, \dots, T_n\}$ 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 $(q_0, [\ ]_k) \xrightarrow{a_0} (q_1, s_1) \xrightarrow{\alpha_1} (q_2, s_2) \xrightarrow{\alpha_2} (q_3, s_3) \xrightarrow{\alpha_3} \dots$ such that α is obtained from $\alpha_0\alpha_1\alpha_2\alpha_3\dots$ 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.*

Proof. $\Rightarrow$: In particular we have to show that if $(\mathbb{N}, +1.P)$ is in the k-th level of the Caucal hierarchy then there exists a HOPDSG $\mathcal{A}_P$ of level k such that for each $i \in \mathbb{N}$ the i-th bit produced in the run of $\mathcal{A}_P$ is the same as $\chi_P(i)$.

Let $(\mathbb{N}, +1, P)$ be in the k-th level of the Caucal hierarchy and let $\mathcal{G}_P$ be the graph corresponding to $(\mathbb{N}, +1, P)$. As $P \subseteq \mathbb{N}$ the graph $\mathcal{G}_P = (V, E, P)$ can be considered as a "simple" infinite path with $V = \mathbb{N}$ and $(i, j) \in E$ iff $j = i + 1$ $\forall i, j \in \mathbb{N}$. To conserve the parameter in the graph introduce the $\{0, 1\}$-labeling of the edges where $(i, j) \in E_0$ if $i \notin P$ and $(i, j) \in E_1$ if $i \in P$ for all $(i, j) \in E$.

Then by [3] we know that there exists a higher-order pushdown automaton $\mathcal{A}$ of level k such that $\mathcal{G}_P$ is the configuration graph of $\mathcal{A}_P$. Remark that they show the proof for higher-order pushdown automata which use the unconditional pop but in fact the proof works by first using the here used higher-order pushdown automata with equality pop and then showing that the configurations graphs of this two automata models are equivalent. So the proof can be used here directly.

Assume $\mathcal{A}_P$ is the HOPDA constructed by [3]. Then in the configuration graph of $\mathcal{A}_P$ exists an infinite path labeled by χ_P. Usually the HOPDA are non-deterministic, i.e. for a configuration (q, s) there could be different transitions $(q, b_0, \gamma_0, q_0), \ldots, (q, b_n, \gamma_n, q_n)$ with $n \geq 0$ which are applicable. In the HOPDSG we add the tests to guarantee that for each configuration there is only one transition applicable. For this define for each transition the regular set of stacks at which they are used in the configuration graph of $\mathcal{A}_P$ to follow the infinite χ_P-path. Then we add those regular sets of stacks as tests to the respectively transitions in the HOPDSG.

Those tests may now not fulfill the conditions we have proposed on restricted tests. For the condition that no output of $0, 1$ appears in the transitions with tests we introduce two dummy transitions, i.e. replace (q, b, γ, T, q') by $(q, \varepsilon, \gamma, T, p)$ and $(p, b, \tau, \emptyset, q')$.

The second condition of the test says that by the tests just a look into the future is allowed which means that the operations which follow the application of the by the tests allowed transitions are applicable. For this condition it has figured out that the automata can be transformed such that the test are only needed for the case that in the future is a $\overline{copy_k}$ operation for which the topmost but one stack has to be rebuild. For this case the test which are needed follow exactly the definition of restricted tests (as they are only defined for this case).
$\Leftarrow$: We have to show that if a k-HOPDSG defines parameter P, i.e. produces a sequence χ_P then is the structure $(\mathbb{N}, +1, P)$ is in the k-th level of the Caucal hierarchy.

Let $\mathcal{A}$ be a higher-order pushdown sequence generator of level k with restricted tests which generates a predicate P. We delete the tests from $\mathcal{A}$ and get so instead of only an infinite path a (slightly larger) configuration graph which still includes P as one path. In particular by the definition of the restricted tests all infinite paths in the resulting configuration graph of the system without tests which are not the "original path" of χ_P are finally (i.e. from the moment they leave the "original path" onwards) only labeled by ε. This holds as well for the finite paths that abort from the path χ_P.

By deleting the tests we get an usual HOPDA $\mathcal{B}$ of level k with equality *pop*. Wöhrle proved in [19](p.74) that for a HOPDA $\mathcal{B}$ of level k with equality *pop* there exists a HOPDA $\mathcal{C}$ of level k with unconditional *pop* such that they generate the same graph. From [3] we know if a graph is the configuration graph of a HOPDA of level k then it is also in the k-th level of the Caucal hierarchy. Lets call the graph we get by this $\mathcal{G}_P$. To receive really the structure $(\mathbb{N}, +1, P)$ we have to add in the MSO-interpretation of $\mathcal{G}_P$ the following conditions:

– As vertices we take only these nodes which have an outgoing 0 or 1 edge.
– For the edge relation we take the ε-closure to connect these vertices.
– As parmeter P we take those vertices that have an outgoing 1 edge.

So we have that $(\mathbb{N}, +1, P)$ is in the k-th level of the Caucal hierarchy.

Example 4. 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:

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

A concrete higher-order pushdown sequence generator $\mathcal{A}_P$ for the powers of 2 can be defined as follows with $\mathcal{A}_P = (Q, \{0, 1, \varepsilon\}, \{0, 1, 2\}, q_0, \Delta)$, where $Q = \{q_i \mid i \in [0, 25]\}$ and Δ is given in Table 1. The set of resticted tests is also given in Table 1.

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 ...

Start of computation:	
$(q_0, 0, \tau, \emptyset, q_1)$	output for $\chi_P(0)$
$(q_1, 1, \tau, \emptyset, q_2)$	output for $\chi_P(1)$
$(q_2, 1, push_0, \emptyset, q_3)$	output for $\chi_P(2)$
$(q_3, 0, \tau, \emptyset, q_4)$	output for $\chi_P(3)$
$(q_4, 1, push_0, \emptyset, q_5)$	output of 1 for position 2^i $i \geq 2$
Marking of first level 1 stack by pushing 2:	
$(q_5, \varepsilon, push_2, \emptyset, q_6)$	
$(q_6, \varepsilon, copy_1, \emptyset, q_7)$	
$(q_7, \varepsilon, pop_2, \emptyset, q_8)$	
Building tower of 0 in stack:	
$(q_8, \varepsilon, copy_1, \emptyset, q_9)$	
$(q_9, \varepsilon, pop_0, \emptyset, q_8)$	case: if topmost stack is not empty
$(q_9, \varepsilon, T_{[\,]_1}, \emptyset, q_{10})$	case: if topmost stack is empty
$(q_{10}, \varepsilon, \overline{copy_1}, \emptyset, q_{11})$	rebuilding the right stack
$(q_{11}, \varepsilon, push_0, \emptyset, q_{12})$	
$(q_{12}, \varepsilon, \overline{copy_1}, \emptyset, q_{13})$	
Start search from top:	
$(q_{13}, 0, pop_0, \emptyset, q_{14})$	replace 0 by 1, output 0
$(q_{14}, \varepsilon, push_1, \emptyset, q_{15})$	
Search for 0 downwards:	
$(q_{15}, \varepsilon, pop_1, \emptyset, q_{16})$	preparation for deletion of topmost stack
$(q_{16}, \varepsilon, push_0, \emptyset, q_{17})$	
$(q_{17}, \varepsilon, push_0, T_{push_0\overline{copy_1}}, q_{18})$	case: in stack below is in front a 0
$(q_{17}, \varepsilon, push_1, T_{push_1\overline{copy_1}}, q_{19})$	case: in stack below is in front a 1
$(q_{17}, \varepsilon, push_2, T_{push_2\overline{copy_1}}, q_{24})$	case: in stack below is in front a 2
Case: 0 in front of stack below, 0 to 1:	
$(q_{18}, \varepsilon, \overline{copy_1}, \emptyset, q_{20})$	
$(q_{20}, 0, pop_0, \emptyset, q_{21})$	replace 0 by 1, output 0
$(q_{21}, \varepsilon, push_1, \emptyset, q_{22})$	
$(q_{22}, \varepsilon, copy_1, \emptyset, q_{23})$	rebuilding of tower of 0's
$(q_{23}, \varepsilon, pop_1, \emptyset, q_8)$	
Case: 1 in front of stack below, down:	
$(q_{19}, \varepsilon, \overline{copy_1}, \emptyset, q_{15})$	
Case: 2 in front of stack below, start:	
$(q_{24}, \varepsilon, \overline{copy_1}, \emptyset, q_{25})$	
$(q_{25}, \varepsilon, pop_2, \emptyset, q_4)$	

where the tests are defined by the following regular expressions over $\mathrm{Ops}_2(\Gamma)$:

$$\begin{aligned}
T_{push_0\overline{copy_1}} &:= \mathrm{Ops}_2^*\ push_0\ \overline{copy_1}\ copy_1\ pop_0\ \ ([\,]_2)\\
T_{push_1\overline{copy_1}} &:= \mathrm{Ops}_2^*\ push_1\ \overline{copy_1}\ copy_1\ pop_1\ \ ([\,]_2)\\
T_{push_2\overline{copy_1}} &:= \mathrm{Ops}_2^*\ push_2\ \overline{copy_1}\ copy_1\ pop_2\ \ ([\,]_2)
\end{aligned}$$

Table 1. Definition of the transition relation Δ and the set of restricted tests $\mathcal{R} = \{T_{push_0\overline{copy_1}}, T_{push_1\overline{copy_1}}, T_{push_2\overline{copy_1}}\}$ of the higher-order pushdown sequence generator $\mathcal{A}_P$ which produces the sequence of the powers of 2 for Example 4.

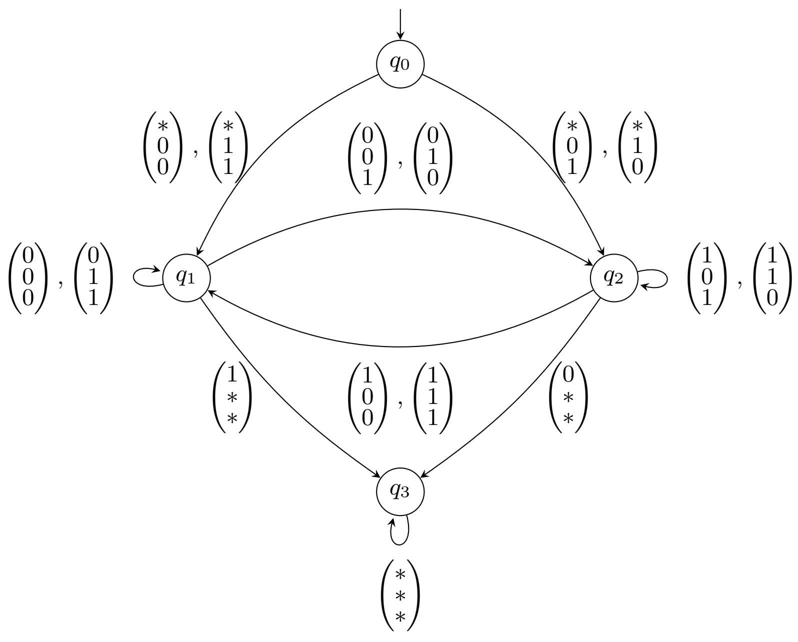
q_0
$\begin{pmatrix}*\\0\\0\end{pmatrix}, \begin{pmatrix}*\\1\\1\end{pmatrix}$
$\begin{pmatrix}0\\0\\1\end{pmatrix}, \begin{pmatrix}0\\1\\0\end{pmatrix}$
$\begin{pmatrix}*\\0\\1\end{pmatrix}, \begin{pmatrix}*\\1\\0\end{pmatrix}$
$\begin{pmatrix}0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\1\end{pmatrix}$
q_1
q_2
$\begin{pmatrix}1\\0\\1\end{pmatrix}, \begin{pmatrix}1\\1\\0\end{pmatrix}$
$\begin{pmatrix}1\\*\\*\end{pmatrix}$
$\begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}1\\1\\1\end{pmatrix}$
$\begin{pmatrix}0\\*\\*\end{pmatrix}$
q_3
$\begin{pmatrix}*\\*\\*\end{pmatrix}$

Fig. 3. Deterministic parity automaton with coloring $p_0 = p_1 = p_2 = 0$, $p_3 = 1$ for the definition of the winning condition of Example 4. The $*$ in the labeling of the transitions means either 0 or 1 possible.

The deterministic parity word automaton over $\{0,1\}^3$ which defines this winning condition is given in Figure 3.
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 the 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 $Q = Q_0 \uplus Q_1$ 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.*

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

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.*

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 initial 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 defined by:

$$\Omega((q_{\mathcal{P}},q_{\mathcal{C}},x,(b_0,b_1,b_2))) = \Omega_{\mathcal{C}}(q_{\mathcal{C}}) \text{ for } x \in \{\Phi_{\mathcal{C}},\Phi_1,\Phi_2\}$$
$$\Omega((q_{\mathcal{P}},q_{\mathcal{C}},\Phi_{\mathcal{P}},(b_0,b_1,b_2))) = (2\cdot n) \text{ where } n \text{ is the maximal color in } \Omega_{\mathcal{C}}.$$

The state partitioning is defined by:

$$Q_1 = Q_{\mathcal{P}} \times Q_{\mathcal{C}} \times \{\Phi_1,\Phi_{\mathcal{P}},\Phi_{\mathcal{C}}\} \times \{0,1\}^3$$
$$Q_2 = Q_{\mathcal{P}} \times Q_{\mathcal{C}} \times \{\Phi_2\} \times \{0,1\}^3.$$

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.

Now we will proof in detail that this construction is correct. For this it remains to show that:

Player 2 wins the regular P-game with winning condition defined by $\mathcal{C}$ if and only if the higher-order pushdown parity game $\mathcal{G}_P$ is won by player 2 from the initial configuration.

⇒: Assume player 2 wins the regular P-game.
For the parameter P with $\chi_P = p_0p_1p_2\ldots$ let $\rho^{\mathcal{P}}$ be the run of $\mathcal{P}$, i.e. $\rho^{\mathcal{P}}(0) = (q_0^{\mathcal{P}},[\;]_k)$ and $\rho^{\mathcal{P}}(i) \xrightarrow[\mathcal{P}]{a_i^j} \rho^{\mathcal{P}}(i+1)$ for all $i \geq 0$ where $j \in \mathbb{N} \cup \{\bullet\}$ and $j = \ell$ if $a_i^j = p_\ell$ for all $\ell \in \mathbb{N}$ and $j = \bullet$ if $a_i^j \notin \{0,1\}$.

If player 2 solves the regular P-game player 2 can choose for every possible input of player 1 a correct output such that the winning condition is fulfilled. In particular for all $X = x_0x_1\ldots \in \{0,1\}^\omega$ chosen by player 1, player 2 can construct $Y = y_0y_1\ldots \in \{0,1\}^\omega$ such that $(P,X,Y) \in (\{0,1\}^3)^\omega$ is accepted by $\mathcal{C}$. For some X and Y chosen by player 1 respectively player 2, let $\rho^{\mathcal{C}}$ be the accepting run of $\mathcal{C}$ on (P,X,Y) with $\rho^{\mathcal{C}}(0) = q_0^{\mathcal{C}}$ and $\delta_{\mathcal{C}}(\rho^{\mathcal{C}}(i),(p_i,x_i,y_i)) = \rho^{\mathcal{C}}(i+1)$ for all $i \geq 0$.

We have to show that in the HOPDPG $\mathcal{G}_P$ which is constructed as described above and where player 1 and player 2 choose the bits according to player 1 and player 2 of the regular P-game, player 2 wins this play starting from the initial configuration $((q_0^{\mathcal{P}},q_0^{\mathcal{C}},\Phi_{\mathcal{P}},(0,0,0)),[\;]_k)$.

Due to the definition of the game from $((q_0^{\mathcal{P}},q_0^{\mathcal{C}},\Phi_{\mathcal{P}},(0,0,0)),[\;]_k)$ player 1 has to compute the first bit of the parameter P. We have to distinguish here several cases. Let in $\rho^{\mathcal{P}}$ [2] for $i \in \mathbb{N}$ a_i^0 be the first time when a bit is produced in the

[2] We note for all $i \in \mathbb{N}$, $\rho^{\mathcal{P}}(i) = (q_i, s_i)$.

run $\rho^{\mathcal{P}}$, i.e. for $\rho^{\mathcal{P}}(0) \xrightarrow[\mathcal{P}]{a_0^{j_0}} \rho^{\mathcal{P}}(1) \xrightarrow[\mathcal{P}]{a_1^{j_1}} \dots \xrightarrow[\mathcal{P}]{a_i^{j_i}} \rho^{\mathcal{P}}(i+1)$ we have for all $0 \leq \ell < i$ $a_\ell^{j_\ell} \notin \{0,1\}$ $(j_\ell = \bullet)$ and $a_i^{j_i} \in \{0,1\}$ $(j_i = 0)$.

1. In the transitions applied in $\rho^{\mathcal{P}}$ to get from $\rho^{\mathcal{P}}(0)$ to $\rho^{\mathcal{P}}(i+1)$ we do not need a test. In this case player 1 has no choice and follows in the play just $\rho^{\mathcal{P}}$ until p_0 is delivered. In the play we end in $((q_{i+1}^{\mathcal{P}}, q_0^{\mathcal{C}}, \Phi_1, (p_0, 0, 0)), s_{i+1})$.
2. In the transitions applied in $\rho^{\mathcal{P}}$ to get from $\rho^{\mathcal{P}}(0)$ to $\rho^{\mathcal{P}}(i+1)$ there is at least one restricted test used. In this case we have deleted the tests in the construction of the game. So player 1 can choose between some edges for every step where such a test is deleted. Again we have two cases:
   (a) Player 1 chooses the correct edges and follows $\rho^{\mathcal{P}}$, then see case 1.
   (b) Player 1 chooses somewhere a wrong edge. Then by definition of $\mathcal{P}$ player 1 either ends up after a few steps in a deadlock and so player 2 wins, or he will never reach a point where he can apply an edge which is labeled by 0 or 1 and gets stuck in the "$\mathcal{P}$ choice phase" in the play. Due to the coloring which assigns to all those configurations in the "$\mathcal{P}$ choice phase" an even number, player 2 will win, too.

Now we consider the case that we have reached $((q_{i+1}^{\mathcal{P}}, q_0^{\mathcal{C}}, \Phi_1, (p_0, 0, 0)), s_{i+1})$.

There are only two possibilities for player 1 to go on, i.e. either to choose 0 or 1 as first bit. By assumption player 1 chooses the bit x_0 and we get to the configuration $((q_{i+1}^{\mathcal{P}}, q_0^{\mathcal{C}}, \Phi_2, (p_0, x_0, 0)), s_{i+1})$. From there player 2 chooses according to player 2 (of the regular P-game) the bit y_0 and we get to configuration $((q_{i+1}^{\mathcal{P}}, q_0^{\mathcal{C}}, \Phi_{\mathcal{C}}, (p_0, x_0, y_0)), s_{i+1})$. In this configuration again player 1 decides how to go on but he has by definition of the game, respectively because $\mathcal{C}$ is a deterministic parity automaton, only one choice namely to go to $((q_{i+1}^{\mathcal{P}}, \rho^{\mathcal{C}}(1), \Phi_{\mathcal{P}}, (p_0, x_0, y_0)), s_{i+1})$, as $\delta_{\mathcal{C}}(q_0^{\mathcal{C}}, (p_0, x_0, y_0)) = \rho^{\mathcal{C}}(1)$.

If we now assume that further in the play player 1 chooses in the Φ_1-vertices the bits according to X and in the $\Phi_{\mathcal{P}}$-vertices the "correct" edges and player 2 chooses in the Φ_2-vertices the bits according to Y we get the following play with the start like described above and:

$$\begin{array}{ll}
((q_j^{\mathcal{P}}, \rho^{\mathcal{C}}(i), \Phi_{\mathcal{P}}, (p_{i-1}, x_{i-1}, y_{i-1})), s_j) & \xrightarrow{a_{j+1}^{\bullet}} \dots \xrightarrow{a_{j+(k-2)}^{\bullet}} \xrightarrow{a_{j+(k-1)}^{i}} \\
((q_{j+k}^{\mathcal{P}}, \rho^{\mathcal{C}}(i), \Phi_1, (p_i, x_{i-1}, y_{i-1})), s_{j+k}) & \xrightarrow{x_i} \\
((q_{j+k}^{\mathcal{P}}, \rho^{\mathcal{C}}(i), \Phi_2, (p_i, x_i, y_{i-1})), s_{j+k}) & \xrightarrow{y_i} \\
((q_{j+k}^{\mathcal{P}}, \rho^{\mathcal{C}}(i), \Phi_{\mathcal{C}}, (p_i, x_i, y_i)), s_{j+k}) & \xrightarrow{\varepsilon} \\
((q_{j+k}^{\mathcal{P}}, \rho^{\mathcal{C}}(i+1), \Phi_{\mathcal{P}}, (p_i, x_i, y_i)), s_{j+k}) &
\end{array}$$

for all $i > 0$, $j, k, l \geq 0$ holds $a_{j+(k-1)}^{i} = p_i$ and $a_l^{\bullet} = \varepsilon$.

It remains to show that player 2 wins this play. This follows as except for the $\Phi_{\mathcal{P}}$-vertices all vertices are colored according to the states of $\mathcal{C}$. As by assumption $\rho^{\mathcal{C}}$ is accepting and the $\Phi_{\mathcal{P}}$-vertices have a color that is larger than every color in $\Omega^{\mathcal{C}}$, player 2 wins.

$\Leftarrow$: Assume player 2 wins the game $\mathcal{G}_P$ from the initial configuration. In this case player 2 has a winning strategy φ, where whatever player 1 does, the play fulfills the winning condition.

We have to differ between two cases:

1. Player 1 loses because he made a mistake in the computation of P and chooses an edge which refers to a wrong transition in the run of $\mathcal{P}$. In this case either the play aborts in a vertex of player 1 or the play stays forever in the $\Phi_{\mathcal{P}}$ component in the vertices of the game and so the only color which is seen infinitely often is even.
   This case we want to exclude because here player 2 does not necessarily have a winning strategy and we can immediately determine if this case has appeared because it is the only way the play can abort or the color $2 \cdot n$ appears as smallest color which is seen infinitely often.
2. Player 1 loses because the smallest color which appears infinitely often in the play is even and not $2 \cdot n$, i.e. player 1 chose the correct way to compute P.

We will concentrate on the second case.

As player 2 wins the game from the initial configuration for every possible choice of player 1 we can assume some particular play starting from the initial configuration $((q_0^{\mathcal{P}}, q_0^{\mathcal{C}}, \Phi_{\mathcal{P}}, (0,0,0)), [\ ]_k)$. Due to the definition of the game $\mathcal{G}_P$ and the assumption that player 1 chooses the correct edges in the $\mathcal{P}$ component, the play has to have the following form:

$$\begin{array}{lll}
i = 0: & ((q_0^{\mathcal{P}}, q_0^{\mathcal{C}}, \Phi_{\mathcal{P}}, (0,0,0)), [\ ]_k) & \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} \xrightarrow{p_0} \\
& ((q_{\ell_1}^{\mathcal{P}}, q_0^{\mathcal{C}}, \Phi_1, (p_0,0,0)), s_{\ell_1}) & \xrightarrow{x_0} \\
& ((q_{\ell_1}^{\mathcal{P}}, q_0^{\mathcal{C}}, \Phi_2, (p_0,x_0,0)), s_{\ell_1}) & \xrightarrow{y_0} \\
& ((q_{\ell_1}^{\mathcal{P}}, q_0^{\mathcal{C}}, \Phi_{\mathcal{C}}, (p_0,x_0,y_0)), s_{\ell_1}) & \xrightarrow{\varepsilon} \\
& ((q_{\ell_1}^{\mathcal{P}}, q_1^{\mathcal{C}}, \Phi_{\mathcal{P}}, (p_0,x_0,y_0)), s_{\ell_1}) &
\end{array}$$

$$\begin{array}{lll}
i > 0: & ((q_{\ell_i}^{\mathcal{P}}, q_i^{\mathcal{C}}, \Phi_{\mathcal{P}}, (p_{i-1},x_{i-1},y_{i-1})), s_{\ell_i}) & \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} \xrightarrow{p_i} \\
& ((q_{\ell_{i+1}}^{\mathcal{P}}, q_i^{\mathcal{C}}, \Phi_1, (p_i,x_{i-1},y_{i-1})), s_{\ell_{i+1}}) & \xrightarrow{x_i} \\
& ((q_{\ell_{i+1}}^{\mathcal{P}}, q_i^{\mathcal{C}}, \Phi_2, (p_i,x_i,y_{i-1})), s_{\ell_{i+1}}) & \xrightarrow{y_i} \\
& ((q_{\ell_{i+1}}^{\mathcal{P}}, q_i^{\mathcal{C}}, \Phi_{\mathcal{C}}, (p_i,x_i,y_i)), s_{\ell_{i+1}}) & \xrightarrow{\varepsilon} \\
& ((q_{\ell_{i+1}}^{\mathcal{P}}, q_{i+1}^{\mathcal{C}}, \Phi_{\mathcal{P}}, (p_i,x_i,y_i)), s_{\ell_{i+1}}). &
\end{array}$$

That the correct $P = p_0p_1p_2\dots$ is computed follows by the construction of the game and the assumption on player 1. That the winning condition for the regular P-game is fulfilled for (P, X, Y) where $X = x_0x_1x_2\dots$ and $Y = y_0y_1y_2\dots$ follows from the $\mathcal{C}$ component of the game.

In the next part of the proof we want to construct a strategy automaton for the player who wins the regular P-game. This automaton is again a HOPDA of level k with tests in OReg_k. The idea for the construction of the strategy automaton for the player winning the game is similar as above and uses the strategy which is delivered by the algorithm of Theorem 3.
We will now show in detail:

> From a positional strategy for the player $i \in \{1,2\}$ winning the game $\mathcal{G}_P$ starting from the initial configuration we can construct a winning strategy for player i in the regular P-game.

We assume here that player 2 wins the game $\mathcal{G}_P$ and so also the regular P-game. If player 1 wins the construction is slightly more complicated as we let player 1 in $\mathcal{G}_P$ also choose the edges for the computation of P and $\mathcal{C}$ but it works similarly.

We will now show how to compute a winning strategy for player 2 in form of a strategy automaton which uses higher-order pushdown stacks with tests. Let φ be the positional winning strategy for player 2 in the game $\mathcal{G}_P$ starting from the initial configuration. In particular, the strategy for player 2 computed by the algorithm of Theorem 3 is defined in this kind of game by two regular sets of stacks respectively configurations[3] $S_0 \subseteq V_2$ and $S_1 \subseteq V_2$ where for all vertices in S_0 player 2 takes the 0 edge and for all vertices in S_1 player 2 takes the 1 edge. Player 2 just has to to follow the game $\mathcal{G}_P$ and mimic the choices of player 2 in $\mathcal{G}_P$.

The idea is to use a HOPDA similar to $\mathcal{P}_\mathcal{G}$ to compute the strategy for player 2. We just add for the P component again the tests of the HOPDSG to guarantee the correct choices and additional tests for the winning strategy S_0, S_1 to compute the output of the correct strategy for player 2. As in S_0 and S_1 also the state of the configuration is stored we have to split the sets such that we have for each state $q \in Q$ sets S_0^q and S_1^q with:

$$s \in S_i^q \text{ iff } push_q(s) \in S_i \text{ for } i \in \{0,1\}, q \in Q.$$

The input for the strategy automaton corresponds to the choices of player 1 in the game $\mathcal{G}_P$. We define the strategy automaton for player 2 by the HOPDA[4] $\mathcal{A}_S = (Q, \Gamma_\mathcal{P}, \Gamma_\mathcal{P}, q_0, \sigma, \mu)$ with test in the set $T_\mathcal{P} \cup \{S_i^q \mid i \in \{0,1\}, q \in Q$ where $T_\mathcal{P}$ is the set of restricted test of the HOPDSG $\mathcal{P}$. The set of states is again defined by $Q = Q_\mathcal{P} \times Q_\mathcal{C} \times \{\Phi_\mathcal{P}, \Phi_1, \Phi_2, \Phi_\mathcal{C}\} \times \{0,1\}^3$ and the start state is $q_0 = (q_0^\mathcal{P}, q_0^\mathcal{C}, \Phi_\mathcal{P}, (0,0,0))$.
The transition function σ is defined as follows:
Computation of the bit of parameter P:

$$\begin{aligned}
\sigma((q_\mathcal{P}, q_\mathcal{C}, \Phi_\mathcal{P}, (b_0,b_1,b_2)), \varepsilon, \gamma, T) &= (q'_\mathcal{P}, q_\mathcal{C}, \Phi_\mathcal{P}, (b_0,b_1,b_2)) \\
&\quad \text{if } (q_\mathcal{P}, \varepsilon, \gamma, T, q'_\mathcal{P}) \in \Delta_\mathcal{P} \\
\sigma((q_\mathcal{P}, q_\mathcal{C}, \Phi_\mathcal{P}, (b_0,b_1,b_2)), 0, \gamma, T) &= (q'_\mathcal{P}, q_\mathcal{C}, \Phi_1, (0,b_1,b_2)) \\
&\quad \text{if } (q_\mathcal{P}, 0, \gamma, T, q'_\mathcal{P}) \in \Delta_\mathcal{P} \\
\sigma((q_\mathcal{P}, q_\mathcal{C}, \Phi_\mathcal{P}, (b_0,b_1,b_2)), 1, \gamma, T) &= (q'_\mathcal{P}, q_\mathcal{C}, \Phi_1, (1,b_1,b_2)) \\
&\quad \text{if } (q_\mathcal{P}, 1, \gamma, T, q'_\mathcal{P}) \in \Delta_\mathcal{P}.
\end{aligned}$$

Player 1 moves, i.e. chooses bit $b'_1 \in \{0,1\}$:

$$\sigma((q_\mathcal{P}, q_\mathcal{C}, \Phi_1, (b_0,b_1,b_2)), b'_1, \tau, \emptyset) = (q_\mathcal{P}, q_\mathcal{C}, \Phi_2, (b_0,b'_1,b_2)).$$

Player 2 moves:

$$\begin{aligned}
&\sigma((q_\mathcal{P}, q_\mathcal{C}, \Phi_2, (b_0,b_1,b_2)), 0, \tau, S_0^{(q_\mathcal{P}, q_\mathcal{C}, \Phi_2, (b_0,b_1,b_2))}) \\
&\qquad = (q_\mathcal{P}, q_\mathcal{C}, \Phi_\mathcal{C}, (b_0,b_1,0)) \\
&\sigma((q_\mathcal{P}, q_\mathcal{C}, \Phi_2, (b_0,b_1,b_2)), 1, \tau, S_1^{(q_\mathcal{P}, q_\mathcal{C}, \Phi_2, (b_0,b_1,b_2))}) \\
&\qquad = (q_\mathcal{P}, q_\mathcal{C}, \Phi_\mathcal{C}, (b_0,b_1,1)).
\end{aligned}$$

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

[4] In this HOPDA we have instead of a transition relation Δ the transition function δ which gets as input the choices of player 1 and the output function μ which delivers the choices for player 2. Besides of this their definition is similar and according to the usual definition of strategy automata.

Evaluation of the winning condition:

$$\sigma((q_{\mathcal{P}}, q_{\mathcal{C}}, \Phi_{\mathcal{C}}, (b_0, b_1, b_2)), \varepsilon, \tau, \emptyset) = (q_{\mathcal{P}}, q'_{\mathcal{C}}, \Phi_{\mathcal{P}}, (b_0, b_1, b_2))$$
$$\text{if } \delta_{\mathcal{C}}(q_{\mathcal{C}}, (b_0, b_1, b_2)) = q'_{\mathcal{C}}.$$

The output function μ is defined by:

$$\mu((q_{\mathcal{P}}, q_{\mathcal{C}}, \Phi_2, (b_0, b_1, b_2)), S_0^{(q_{\mathcal{P}}, q_{\mathcal{C}}, \Phi_2, (b_0, b_1, b_2))}) = 0$$
$$\mu((q_{\mathcal{P}}, q_{\mathcal{C}}, \Phi_2, (b_0, b_1, b_2)), S_1^{(q_{\mathcal{P}}, q_{\mathcal{C}}, \Phi_2, (b_0, b_1, b_2))}) = 1.$$

According to the proof for the solution of the game we can conclude that the defined automaton computes a winning strategy for player 2 in the P-regular game, if player 2 wins the higher-order pushdown parity game $\mathcal{G}_P$ with the strategy given by S_0, S_1.

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 [13, 14] on regular P-games, using automata theoretic concepts and ideas that go back to Siefkes [17]. 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 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. This could be the case if winning conditions are considered that does not take into account the hole complexity of the parameter.

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). In this setting for example the collapsible pushdown systems [10] come into account as they are strictly more expressive then the higher-order pushdown systems. They have also higher-order pushdown stacks as storeage but they additionally attach to every symbol in a stack some link to a stack situated somewhere below it. The link is used when the new operation "collapse" is executed to return back to the linked stack.

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. Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 452–461, 2008.
11. P. Hänsch. Infinite games with parameters. Master's thesis, Lehrstuhl für Informatik 7, RWTH Aachen, Aachen, Germany, 2009.
12. R. McNaughton. Testing and generating infinite sequences by a finite automaton. In *Inform. Contr.*, volume 9, pages 521–530, 1966.
13. A. Rabinovich. Church synthesis problem with parameters. In *Proc. CSL*, volume 4207 of *LNCS*, pages 546–561, Heidelberg, 2006. Springer.
14. A. Rabinovich. Church synthesis problem with parameters. *Logical Methods in Computer Science*, 3(4:9):1–24, 2007.
15. 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.
16. H. Rogers. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill, New York, 1967.
17. D. Siefkes. The recursive sets in certain monadic second order fragments of arithmetic. *Archiv math. Logik*, 17:71–80, 1975.
18. W. Thomas. The theory of successor with an extra predicate. *Math. Ann.*, 237:121–132, 1978.
19. S. Wöhrle. *Decision problems over infinite graphs: Higher-order pushdown systems and synchronized products.* PhD thesis, RWTH Aachen, 2005.

Aachener Informatik-Berichte

This list contains all technical reports published during the past five years. A complete list of reports dating back to 1987 is available from http://aib.informatik.rwth-aachen.de/. To obtain copies consult the above URL or send your request to: Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Email: `biblio@informatik.rwth-aachen.de`

2004-01 *	Fachgruppe Informatik: Jahresbericht 2003
2004-02	Benedikt Bollig, Martin Leucker: Message-Passing Automata are expressively equivalent to EMSO logic
2004-03	Delia Kesner, Femke van Raamsdonk, Joe Wells (eds.): HOR 2004 – 2nd International Workshop on Higher-Order Rewriting
2004-04	Slim Abdennadher, Christophe Ringeissen (eds.): RULE 04 – Fifth International Workshop on Rule-Based Programming
2004-05	Herbert Kuchen (ed.): WFLP 04 – 13th International Workshop on Functional and (Constraint) Logic Programming
2004-06	Sergio Antoy, Yoshihito Toyama (eds.): WRS 04 – 4th International Workshop on Reduction Strategies in Rewriting and Programming
2004-07	Michael Codish, Aart Middeldorp (eds.): WST 04 – 7th International Workshop on Termination
2004-08	Klaus Indermark, Thomas Noll: Algebraic Correctness Proofs for Compiling Recursive Function Definitions with Strictness Information
2004-09	Joachim Kneis, Daniel Mölle, Stefan Richter, Peter Rossmanith: Parameterized Power Domination Complexity
2004-10	Zinaida Benenson, Felix C. Gärtner, Dogan Kesdogan: Secure Multi-Party Computation with Security Modules
2005-01 *	Fachgruppe Informatik: Jahresbericht 2004
2005-02	Maximillian Dornseif, Felix C. Gärtner, Thorsten Holz, Martin Mink: An Offensive Approach to Teaching Information Security: "Aachen Summer School Applied IT Security"
2005-03	Jürgen Giesl, René Thiemann, Peter Schneider-Kamp: Proving and Disproving Termination of Higher-Order Functions
2005-04	Daniel Mölle, Stefan Richter, Peter Rossmanith: A Faster Algorithm for the Steiner Tree Problem
2005-05	Fabien Pouget, Thorsten Holz: A Pointillist Approach for Comparing Honeypots
2005-06	Simon Fischer, Berthold Vöcking: Adaptive Routing with Stale Information
2005-07	Felix C. Freiling, Thorsten Holz, Georg Wicherski: Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks
2005-08	Joachim Kneis, Peter Rossmanith: A New Satisfiability Algorithm With Applications To Max-Cut
2005-09	Klaus Kursawe, Felix C. Freiling: Byzantine Fault Tolerance on General Hybrid Adversary Structures
2005-10	Benedikt Bollig: Automata and Logics for Message Sequence Charts
2005-11	Simon Fischer, Berthold Vöcking: A Counterexample to the Fully Mixed Nash Equilibrium Conjecture

2005-12	Neeraj Mittal, Felix Freiling, S. Venkatesan, Lucia Draque Penso: Efficient Reductions for Wait-Free Termination Detection in Faulty Distributed Systems
2005-13	Carole Delporte-Gallet, Hugues Fauconnier, Felix C. Freiling: Revisiting Failure Detection and Consensus in Omission Failure Environments
2005-14	Felix C. Freiling, Sukumar Ghosh: Code Stabilization
2005-15	Uwe Naumann: The Complexity of Derivative Computation
2005-16	Uwe Naumann: Syntax-Directed Derivative Code (Part I: Tangent-Linear Code)
2005-17	Uwe Naumann: Syntax-directed Derivative Code (Part II: Intraprocedural Adjoint Code)
2005-18	Thomas von der Maßen, Klaus Müller, John MacGregor, Eva Geisberger, Jörg Dörr, Frank Houdek, Harbhajan Singh, Holger Wußmann, Hans-Veit Bacher, Barbara Paech: Einsatz von Features im Software-Entwicklungsprozess - Abschlußbericht des GI-Arbeitskreises “Features”
2005-19	Uwe Naumann, Andre Vehreschild: Tangent-Linear Code by Augmented LL-Parsers
2005-20	Felix C. Freiling, Martin Mink: Bericht über den Workshop zur Ausbildung im Bereich IT-Sicherheit Hochschulausbildung, berufliche Weiterbildung, Zertifizierung von Ausbildungsangeboten am 11. und 12. August 2005 in Köln organisiert von RWTH Aachen in Kooperation mit BITKOM, BSI, DLR und Gesellschaft fuer Informatik (GI) e.V.
2005-21	Thomas Noll, Stefan Rieger: Optimization of Straight-Line Code Revisited
2005-22	Felix Freiling, Maurice Herlihy, Lucia Draque Penso: Optimal Randomized Fair Exchange with Secret Shared Coins
2005-23	Heiner Ackermann, Alantha Newman, Heiko Röglin, Berthold Vöcking: Decision Making Based on Approximate and Smoothed Pareto Curves
2005-24	Alexander Becher, Zinaida Benenson, Maximillian Dornseif: Tampering with Motes: Real-World Physical Attacks on Wireless Sensor Networks
2006-01 *	Fachgruppe Informatik: Jahresbericht 2005
2006-02	Michael Weber: Parallel Algorithms for Verification of Large Systems
2006-03	Michael Maier, Uwe Naumann: Intraprocedural Adjoint Code Generated by the Differentiation-Enabled NAGWare Fortran Compiler
2006-04	Ebadollah Varnik, Uwe Naumann, Andrew Lyons: Toward Low Static Memory Jacobian Accumulation
2006-05	Uwe Naumann, Jean Utke, Patrick Heimbach, Chris Hill, Derya Ozyurt, Carl Wunsch, Mike Fagan, Nathan Tallent, Michelle Strout: Adjoint Code by Source Transformation with OpenAD/F
2006-06	Joachim Kneis, Daniel Mölle, Stefan Richter, Peter Rossmanith: Divide-and-Color
2006-07	Thomas Colcombet, Christof Löding: Transforming structures by set interpretations
2006-08	Uwe Naumann, Yuxiao Hu: Optimal Vertex Elimination in Single-Expression-Use Graphs
2006-09	Tingting Han, Joost-Pieter Katoen: Counterexamples in Probabilistic Model Checking

2006-10	Mesut Günes, Alexander Zimmermann, Martin Wenig, Jan Ritzerfeld, Ulrich Meis: From Simulations to Testbeds - Architecture of the Hybrid MCG-Mesh Testbed
2006-11	Bastian Schlich, Michael Rohrbach, Michael Weber, Stefan Kowalewski: Model Checking Software for Microcontrollers
2006-12	Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker: Replaying Play in and Play out: Synthesis of Design Models from Scenarios by Learning
2006-13	Wong Karianto, Christof Löding: Unranked Tree Automata with Sibling Equalities and Disequalities
2006-14	Danilo Beuche, Andreas Birk, Heinrich Dreier, Andreas Fleischmann, Heidi Galle, Gerald Heller, Dirk Janzen, Isabel John, Ramin Tavakoli Kolagari, Thomas von der Maßen, Andreas Wolfram: Report of the GI Work Group "Requirements Management Tools for Product Line Engineering"
2006-15	Sebastian Ullrich, Jakob T. Valvoda, Torsten Kuhlen: Utilizing optical sensors from mice for new input devices
2006-16	Rafael Ballagas, Jan Borchers: Selexels: a Conceptual Framework for Pointing Devices with Low Expressiveness
2006-17	Eric Lee, Henning Kiel, Jan Borchers: Scrolling Through Time: Improving Interfaces for Searching and Navigating Continuous Audio Timelines
2007-01 *	Fachgruppe Informatik: Jahresbericht 2006
2007-02	Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl: SAT Solving for Termination Analysis with Polynomial Interpretations
2007-03	Jürgen Giesl, René Thiemann, Stephan Swiderski, and Peter Schneider-Kamp: Proving Termination by Bounded Increase
2007-04	Jan Buchholz, Eric Lee, Jonathan Klein, and Jan Borchers: coJIVE: A System to Support Collaborative Jazz Improvisation
2007-05	Uwe Naumann: On Optimal DAG Reversal
2007-06	Joost-Pieter Katoen, Thomas Noll, and Stefan Rieger: Verifying Concurrent List-Manipulating Programs by LTL Model Checking
2007-07	Alexander Nyßen, Horst Lichter: MeDUSA - MethoD for UML2-based Design of Embedded Software Applications
2007-08	Falk Salewski and Stefan Kowalewski: Achieving Highly Reliable Embedded Software: An empirical evaluation of different approaches
2007-09	Tina Kraußer, Heiko Mantel, and Henning Sudbrock: A Probabilistic Justification of the Combining Calculus under the Uniform Scheduler Assumption
2007-10	Martin Neuhäußer, Joost-Pieter Katoen: Bisimulation and Logical Preservation for Continuous-Time Markov Decision Processes
2007-11	Klaus Wehrle (editor): 6. Fachgespräch Sensornetzwerke
2007-12	Uwe Naumann: An L-Attributed Grammar for Adjoint Code
2007-13	Uwe Naumann, Michael Maier, Jan Riehme, and Bruce Christianson: Second-Order Adjoints by Source Code Manipulation of Numerical Programs

2007-14	Jean Utke, Uwe Naumann, Mike Fagan, Nathan Tallent, Michelle Strout, Patrick Heimbach, Chris Hill, and Carl Wunsch: OpenAD/F: A Modular, Open-Source Tool for Automatic Differentiation of Fortran Codes
2007-15	Volker Stolz: Temporal assertions for sequential and concurrent programs
2007-16	Sadeq Ali Makram, Mesut Güneç, Martin Wenig, Alexander Zimmermann: Adaptive Channel Assignment to Support QoS and Load Balancing for Wireless Mesh Networks
2007-17	René Thiemann: The DP Framework for Proving Termination of Term Rewriting
2007-18	Uwe Naumann: Call Tree Reversal is NP-Complete
2007-19	Jan Riehme, Andrea Walther, Jörg Stiller, Uwe Naumann: Adjoints for Time-Dependent Optimal Control
2007-20	Joost-Pieter Katoen, Daniel Klink, Martin Leucker, and Verena Wolf: Three-Valued Abstraction for Probabilistic Systems
2007-21	Tingting Han, Joost-Pieter Katoen, and Alexandru Mereacre: Compositional Modeling and Minimization of Time-Inhomogeneous Markov Chains
2007-22	Heiner Ackermann, Paul W. Goldberg, Vahab S. Mirrokni, Heiko Röglin, and Berthold Vöcking: Uncoordinated Two-Sided Markets
2008-01 *	Fachgruppe Informatik: Jahresbericht 2007
2008-02	Henrik Bohnenkamp, Marielle Stoelinga: Quantitative Testing
2008-03	Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, Harald Zankl: Maximal Termination
2008-04	Uwe Naumann, Jan Riehme: Sensitivity Analysis in Sisyphe with the AD-Enabled NAGWare Fortran Compiler
2008-05	Frank G. Radmacher: An Automata Theoretic Approach to the Theory of Rational Tree Relations
2008-06	Uwe Naumann, Laurent Hascoet, Chris Hill, Paul Hovland, Jan Riehme, Jean Utke: A Framework for Proving Correctness of Adjoint Message Passing Programs
2008-07	Alexander Nyßen, Horst Lichter: The MeDUSA Reference Manual, Second Edition
2008-08	George B. Mertzios, Stavros D. Nikolopoulos: The λ-cluster Problem on Parameterized Interval Graphs
2008-09	George B. Mertzios, Walter Unger: An optimal algorithm for the k-fixed-endpoint path cover on proper interval graphs
2008-10	George B. Mertzios, Walter Unger: Preemptive Scheduling of Equal-Length Jobs in Polynomial Time
2008-11	George B. Mertzios: Fast Convergence of Routing Games with Splittable Flows
2008-12	Joost-Pieter Katoen, Daniel Klink, Martin Leucker, Verena Wolf: Abstraction for stochastic systems by Erlang's method of stages
2008-13	Beatriz Alarcón, Fabian Emmes, Carsten Fuhs, Jürgen Giesl, Raúl Gutiérrez, Salvador Lucas, Peter Schneider-Kamp, René Thiemann: Improving Context-Sensitive Dependency Pairs
2008-14	Bastian Schlich: Model Checking of Software for Microcontrollers
2008-15	Joachim Kneis, Alexander Langer, Peter Rossmanith: A New Algorithm for Finding Trees with Many Leaves