# Games on Higher Order Multi-stack Pushdown Systems

Anil Seth

Department of Computer Science and Engg.
I.I.T. Kanpur, Kanpur 208016, India
seth@cse.iitk.ac.in

**Abstract.** In this paper we define higher order multi-stack pushdown systems. We show that parity games over bounded phase higher order multi-stack pushdown systems are effectively solvable and winning strategy in these games can be effectively synthesized.

## 1 Introduction

Higher order pushdown systems (*hpds*) are a generalization of pushdown systems (*pds*) in that *hpds* can have nested stacks, such as stack of stacks. The order of an *hpds* depends on the depth of nested stacks allowed by it. Higher order *push* and *pop* operations are provided to push a copy of the topmost stack of any order and to pop it. These models, in their automata form, were introduced in [17] and were further studied in [16,15]. The *hpds* may be used to model higher order recursion, [16,8,4]. In recent years there has been considerable interest in model checking these systems and their variants [10,8,3,9].

Another generalization of pushdown systems is multi-stack pushdown systems (*mpds*). An *mpds* has a finite set of control states and a fixed number, $l$ ($l > 1$), of independent stacks. The transition function of an *mpds* allows for a (nondeterministically chosen) push or a pop operation on any of its stack along with a change in its control state. Multi-stack pushdown systems can be used to model a class of programs with (order-1) recursion and threads. Each thread has its own stack for its procedures calls and communication among threads is through the common finite states of *mpds*. There has been quite some work in model checking *mpds* and its variants in recent years, see [7,5,6,2], as part of model checking of concurrent recursive programs. For effective model checking of *mpds* some restrictions however need to be imposed on *mpds* as even simple properties such as reachability from one configuration to another are undecidable for unrestricted *mpds*. One such restriction, called bounded context switching, was studied in [7]. This was generalized to bounded phases in [5]. The class of bounded phase *mpds* strictly includes the class of bounded context switching *mpds*.

In this paper we define higher order multi-stack pushdown systems (*hmpds*). An order-$n$ *hmpds* ($n$-*hmpds*) has a fixed number (say, $l$) of order-$n$ stacks. The transition function of an $n$-*hmpds* takes as input its control state and topmost symbols of a stack and may (nondeterministically) do a higher order push or

O. Bournez and I. Potapov (Eds.): RP 2009, LNCS 5797, pp. 203–216, 2009.

a pop operation on the stack along with a change in its control state. These systems can model a class of programs with *higher order* recursion and threads. Such programs may naturally arise while considering functional programs with threads as functional programs typically have some higher order operations. The notion of *bounded phase* defined in [5] for multi-stack pushdown automata can be lifted to *hmpds* as follows. In a $k$ phase bounded *hmpds* only those runs of *hmpds* are considered which can be divided into $k$ parts where each part is a consecutive sequence of moves from the run and is called a phase. In a single phase, *pop* operations (of any order) are performed only in one stack while *push* operations (of any order) can be performed on any stack.

Model checking of *hmpds* against a specification can also be formulated as solving a game over configuration graph of the *hmpds*. We show that parity games over bounded phase *hmpds* can be effectively solved. This implies decidability of properties expressed in a rich specification logic, $\mu$-calculus, over configuration graphs of *hmpds*. Parity games over *hpds* were shown to be effectively solvable in [10] and parity games over *mpds* were shown to be effectively solvable in [2].

We solve parity games over *hmpds* by extending the technique of [2]. In order to explain our extension to the technique of [2], we briefly recall this technique. The solution in [2] is based on a fundamental technique of Walukiewicz [12] which shows how to reduce a parity game on a pushdown system to a parity game over a finite state space. In [12] each time a symbol is pushed in the stack, a set of states (along with priorities) is guessed by player 0, the game now divides into two independent parts. In the first sub game player 1 verifies that if the symbol is popped then it is in one of the guessed states, in the second sub game it is verified that if the pushed symbol is popped satisfying the guessed conditions then the game is winning for player 0. The key step in extending the technique of Walukiewicz to the *mpds* case in [2] is to define finite sets $N_{i,h}$ whose elements code relevant information summarizing the play between a push in stack-$i$ and its matching pop operation in phase $h$. These sets also keep necessary information about changes in stacks $t$, $t \neq i$, between a push and a matching pop operation of stack-$i$. $N_{i,h}$ are defined using a careful induction on $h$.

We generalize the set of conditions $N_{i,h}$ from [2] to $N_{i,j,h}$. Each element of $N_{i,j,h}$ codes relevant information for an order-$j$ pop operation of stack-$i$ in phase $h$. This requires us to study evolution of a higher order stack under a sequence of higher order operations. Unlike order-1 case, order-$i$ push and order-$i$ pop operations need not match in a one to one fashion, for $i > 1$. We define a suitable matching which may associate an unbounded many order-$i$ pop operations with a single order-$i$ push operation. We define sets $N_{i,j,h}$ which may contain pop scenarios for all such matching order-$i$ pop operations associated to a single order-$i$ push operation. To take into account relevant differences in these pop operations, though they match the same push operation, some additional information is kept in each element of $N_{i,j,h}$. This requires induction on $j$, for each fixed $h$ in the definition of $N_{i,j,h}$.

Equipped with sets $N_{i,j,h}$ we define a reduction from a $k$ phase *hmpds* game to a finite state game as in [12,11,2]. In this game with each $push_{i,j}$ move $o$ in phase

$p$ player 0 guesses a $\theta \subseteq \cup_{x=p}^{k} N_{i,j,x}$. The guessed set $\theta$ needs to include conditions for each $pop_{i,j}$ operation that may match the $push_{i,j}$ move $o$. The player 1 now either continues the game after $o$ till a matching $pop_{i,j}$ move occurs or it chooses a scenario in $\theta$ and sets the game to a configuration resulting after a *pop* in this scenario. Solving the finite state game leads to deciding which player in the original *hmpds* game has a winning strategy. As a byproduct of this method as in [12,11,2], we also get that winning strategy in a bounded phase *hmpds* game can be executed by a bounded phase higher order multi-stack pushdown automaton (*hmpda*).

The complexity of our algorithm is a tower of exponential of height $n \cdot k$ for solving a $k$ phase $n$-*hmpds* parity game. We do not know if this is optimal, however when specialized to the case of *hpds* ($l = 1$) it reduces to the optimal bound and when specialized to the case of *mpds* ($n = 1$), it reduces to the best known upper bound of [2]. Despite the high computational complexity, we think that from mathematical view point it is interesting to find classes of infinite graphs over which parity games can be solved effectively. Our technique gives a unified proof of solving *hpds* and *mpds* parity games from basic principles.

Finally to relate this work to reachability problems, the title of this conference, we note that solving reachability problem over *hmpds* is a special case of solving parity games. In particular our results imply that there is an algorithm $\mathcal{A}$ which takes a *hmpds* $\mathcal{H}$ a number $k$ and two configurations $u, v$ of $\mathcal{H}$ and answers *yes* if in the configuration graph of $\mathcal{H}$, $v$ can be reached from $u$ in at most $k$ phases, otherwise $\mathcal{A}$ answers *no*. Further, if the answer is *yes* then a $k$ phase path from $u, v$ can also be produced by the algorithm. Similar result holds for two player reachability between configurations of bounded phase *hmpds*.

## 2 Preliminaries

**Definition 1.** *Let $\Gamma$ be a finite stack alphabet and let $\perp$ be a symbol s.t. $\perp \notin \Gamma$. The Set of order-$i$ stacks over $\Gamma$, $S_i$ for $i \geq 0$ is defined inductively as follows.*

- $S_0 = \Gamma \cup \{\perp\}$ *(we consider an element of $\Gamma$ as an order-0 stack).*
- $S_1 = \{[\perp, s_1, \ldots, s_v] \mid \ s_1 \ldots s_v \in \Gamma,\ v \geq 0\}$.
- $S_{i+1} = \{[s_1, \ldots, s_v] \mid \ s_1 \ldots s_v \in S_i,\ v \geq 1\}$, *for $i \geq 1$.*

*We also define $\perp_0 = \perp$ and for $i \geq 0$, $\perp_{i+1} = [\perp_i]$. Note that $\perp_i \in S_i$.*
*Stack $\perp_i$, for $i > 0$, is called the empty stack of order-$i$.*
*We use $order(s)$ to denote the order of a stack $s$.*

The symbol $\perp$ is used to mark bottom of an order-1 stack.

**Definition 2.** *Let $s$ be an order-$j$ stack for $j \geq 0$. The topmost order-$i$ element ($i \leq j$) of $s$ is defined as*

$$top_i(s) = \begin{cases} s & if \quad order(s) = i \\ top_i(s_u) & if \quad order(s) > i \text{ and } s = [s_1, \ldots, s_u] \end{cases}$$

**Definition 3.** *Operations $push_i, pop_i$ on stacks of order $\geq i$ are defined as follows. Let $s = [s_1, \ldots, s_u]$ and $b \in \Gamma$.*
*$push_1^b(s) = [s_1, \ldots, s_u, b]$, if $order(s) = 1$.*
*$push_1^b(s) = [s_1, \ldots, s_{u-1}, push_1^b(s_u)]$, if $order(s) > 1$.*
*For $i > 1$,*
*$push_i(s) = [s_1, \ldots, s_u, s_u]$, if $order(s) = i$*
*$push_i(s) = [s_1, \ldots, s_{u-1}, push_i(s_u)]$, if $order(s) > i$*
*For $i \geq 1$,*
*$pop_i(s) = [s_1, \ldots, s_{u-1}]$, if $order(s) = i$ and $u > 1$.*
*$pop_i(s) = [s_1, \ldots, s_{u-1}, pop_i(s_u)]$, if $order(s) > i$, $pop_i(s_u)$ is defined.*
*In keeping with conventional notation, a $push_{i+1}$ operation pushes an element of order-$i$ and $pop_{i+1}$ operation pops an element of order-$i$. Note that $pop_i(s)$ is defined iff $top_i(s)$ has more than one element.*

### 2.1 Order-$n$ Multi-stack Pushdown Systems

A order-$n$ multi-stack pushdown system is the same as a multi-stack pushdown system except that each stack is a nested stack of order-$n$. On each stack the *push* and *pop* operations of order-0 to order-$(n-1)$ can be performed. A formal definition is given below.

**Definition 4.** *An order-n multi-stack pushdown system (n-hmpds) is given as a tuple $(Q, \Gamma, \bot, l, \delta, q_0)$, where $Q$ is a finite set of states, $l$ is the number of stacks, $\Gamma$ is the stack alphabet with $\bot$ as in definition 1 and $q_0$ is the initial state. The transition function $\delta$ is given as $\delta = (\bigcup_{j=1}^{n} \delta_{ins,j}) \cup (\bigcup_{j=1}^{n} \delta_{rem,j})$, where*

- $\delta_{ins,j} \subseteq Q \times (\Gamma \cup \bot) \times Q \times [1 \ldots l] \times \Gamma$, $1 \leq j \leq n$.
- $\delta_{rem,j} \subseteq Q \times (\Gamma \cup \bot) \times Q \times [1 \ldots l]$, $1 \leq j \leq n$.

$(q, \gamma, q', i, \gamma') \in \delta_{ins,m}$ denotes *push* transition of order-$(m-1)$ in stack numbered $i$ and $(q, \gamma, q', i) \in \delta_{rem,m}$ denotes *pop* transition of order-$(m-1)$ in stack numbered $i$. These moves are also referred to as $push_{i,m}$ and $pop_{i,m}$ respectively. Values $q, \gamma$ are the state of *hmpds* and the topmost symbol of stack-$i$ before the transition, $q'$ is the state of *hmpds* after the transition. Symbol $\gamma'$ is the symbol pushed when *push* transition is of order-0 ($m = 1$). For $m > 1$, in transition $(q, \gamma, q', i, \gamma') \in \delta_{ins,m}$ symbol $\gamma'$ plays no role. We keep this extra symbol to avoid treating the case $m = 1$ separately. For notational convenience, we stipulate in the following that for $m > 1$, if $(q, \gamma, q', i, \gamma') \in \delta_{ins,m}$ then $\gamma' = \gamma$.

Formal definition of *hmpds* configurations and transitions on them is as follows.

**Definition 5.** *A configuration of $n-$*hmpds *$\mathcal{H} = (Q, \Gamma, l, \delta, q_0)$ is a tuple $(q, s_1, \ldots, s_l)$, where $q \in Q$ and $s_i \in S_n$ for $1 \leq i \leq l$. One step transition on configurations of $\mathcal{H}$ is defined as below.*

- $(q, s_1, \ldots, s_l) \xrightarrow{t} (q', s_1', \ldots, s_l')$ *if $top_0(s_i) = \gamma$, $t = (q, \gamma, q', i, \gamma') \in \delta_{ins,m}$, $s_j' = s_j$ for $j \neq i, 1 \leq j \leq l$ and if $m = 1$ then $s_i' = push_1^{\gamma'}(s_i)$ else $s_i' = push_m(s_i)$.*

– $(q, s_1, \ldots, s_l) \xrightarrow{t} (q', s'_1, \ldots, s'_l)$ *if* $top_0(s_i) = \gamma$, $t = (q, \gamma, q', i) \in \delta_{rem,m}$, $s'_i = pop_m(s_i)$ *and* $s'_j = s_j$ *for* $j \neq i$, $1 \leq j \leq l$.

*The initial configuration of* $n-$hmpds *is defined as* $(q_0, \perp_n, \ldots, \perp_n)$.

Following is the standard definition of the reflexive and transitive closure of relation $\xrightarrow{t}$.

**Definition 6.** *A multi-step transition between configurations of mpds, on say sequence* $t_1 t_2 \ldots t_n$ *of hmpds moves,* $c \xrightarrow{t_1 t_2 \ldots t_n} d$ *is defined as follows.* $c \xrightarrow{t_1 t_2 \ldots t_n} d$ *iff either* $n = 0$ *and* $c = d$ *or there is a* $c'$ *s.t.* $c \xrightarrow{t_1} c'$ *and* $c' \xrightarrow{t_2 \ldots t_n} d$. *We write* $c \twoheadrightarrow d$ *for a multi-step transition from* $c$ *to* $d$ *when the sequence of* hmpds *moves is not relevant.*

Following is a straightforward adaptation of the notion of phases for *mpds* [5] to *hmpds*.

**Definition 7.** *A phase is a sequence of* hmpds *transitions where pop moves (of any order) are performed only on a single stack (though in a single phase push moves may be performed on any stack). A* $k$*-phase bounded run of a* $n$*-hmpds is one which can be partitioned into* $k$ *contiguous segments such that each segment is a single phase.*

It is clear that phase change occurs when a *pop* operation is performed on a stack other than the stack on which it was performed last. We extend the notion of a configuration to $(q, h, r, s_1, \ldots, s_l)$ where $(q, s_1, \ldots, s_l)$ is a configuration as before, $h$ is a natural number recording the phase and $r \in [1, l]$, is the stack number on which the last *pop* operation was performed. In the initial configuration $h = 1$ and $r = 0$.

**Definition 8.** *One step transition on extended configurations of* $\mathcal{H}$ *is defined as* $(q, h, r, s_1, \ldots, s_l) \xrightarrow{t} (q', h', r', s'_1, \ldots, s'_l)$ *where* $(q, s_1, \ldots, s_l) \xrightarrow{t} (q', s'_1, \ldots, s'_l)$ *and*

– *if* $t \in \delta_{ins,m}$ *then* $h' = h$ *and* $r' = r$

– *if* $t = (q, -, q', i) \in \delta_{rem,m}$ *then* $r' = i$ *and* $h' = np(h, r, i) = \begin{cases} h & if\ r = 0\ or\ r = i \\ h + 1 & if\ r \neq i \end{cases}$.

*The initial extended configuration of* $n-$hmpds *is defined as* $(q_0, 0, 1, \perp_n, \ldots, \perp_n)$. *Function* $np(h, r, i)$ *gives the new phase after the transition.*

To consider only $k$-phase bounded runs of hmpds, we restrict ourselves to extended configuration graph where each vertex has phase less than or equal to $k$. In the sequel we use the word configuration for the extended configuration defined above. $top_{i,j}$ refers to $top_j$ of stack-$i$ and $push_{i,j}$, $pop_{i,j}$ refer to transitions of the form $(-, -, -, i, -) \in \delta_{ins,j}$ , $(-, -, -, i) \in \delta_{rem,j}$ respectively.

### 2.2 Parity Games

We assume the reader to be familiar with standard notions of two player parity games, such as game graph, plays, a winning strategy and parity winning condition, see [14].

A 2-player $k$-phase *hmpds* parity game is given as $(\mathcal{H}, Q_0, Q_1, M, \Omega, k)$, where $\mathcal{H} = (Q, \Gamma, l, \delta, q_0)$ is an *hmpds*, $Q = Q_0 \oplus Q_1$ is a partition of states in player 0 and player 1, $M$ is a finite set of priorities and $\Omega : Q \to M$ is a priority assignment to each state in $Q$.

Vertices of our game graph are configurations of the form $(q, h, r, s_1, \ldots, s_l)$, where $h \leq k$, of *hmpds*. Edge relation of this game graph is given by transition relation ' $\rightarrow$' in definition 8. A vertex $(q, -, \ldots, -)$ belongs to player $i$ iff $q \in Q_i$. Priority of a vertex $(q, -, \ldots, -)$ is defined as $\Omega(q)$. A player can move from $c$ to $c'$ only if $c \rightarrow c'$. A play is a sequence of legal moves starting from the initial configuration. By our choice of the vertex set all plays in this game are $k-phase$ bounded. That is a player can not make a move that takes the play into $(k+1)^{th}$ phase.

Winning condition for a maximal play (play which can not be extended further) $\rho$ is defined as follows. If $\rho$ is finite then the player whose turn it is to move at the last vertex of $\rho$ loses. If $\rho$ is infinite then a priority $i \in M$ is said to be visited infinitely often iff there are infinitely many vertices with priority $i$ in $\rho$. $\rho$ is winning for player 0 iff the minimum, among the set of priorities visited infinitely often in $\rho$, is even.

Informally, having a winning strategy for player $i$, means that regardless of player $(1 - i)$'s moves, player $i$ can always play a move such that he wins the resulting play. We will always consider games which start in a predefined initial configuration. A game is called winning for player $i$ if player $i$ has a winning strategy in it starting from the initial configuration.

Given a winning strategy $\tau$ for player 0, in game $\mathcal{G}$, by a $\tau$-play we mean a play of $\mathcal{G}$ in which all moves of player 0 are according to $\tau$. For vertices $c$, $c'$ of $\mathcal{G}$, $c \xrightarrow{\tau} c'$ and $c \xrightarrow{\tau} \!\!\!\!\rightarrow\ c'$ mean that $c'$ is reachable from $c$ in a $\tau$-play in one move or in an arbitrary number of moves respectively.

## 3 Main Ideas in Solving a HMPDS Game

In higher order stacks a sub-stack after being pushed can be copied implicitly several times before being popped. As an example consider an order-2 stack $r = [[\perp a]]$. Here $a$ is a stack symbol which we consider as an order-0 stack. An order$-1$ push operation $o_1$ pushing a symbol $b$, leads to $r_1 = [[\perp ab]]$. Now consider an order-2 push $o_2$ which copies the topmost order-1 stack, this leads to $r_2 = [[\perp ab][\perp ab]]$. In $r_2$ the topmost order-1 stack $[\perp ab]$ is created by $o_2$ by a direct copy of the topmost order-1 stack below. The second occurrences of order-0 stacks $a, b$ however are not created by any order-1 *push* directly. These occurrences get created automatically from the occurrences in the sub-stack being copied by $o_2$. We call such an occurrence an *auto copy* of the corresponding occurrence below. A motivation for considering the *auto copy* relation is to associate

with each sub-stack of a higher order stack a *push* operation. This is needed for matching each *pop* operation with a *push* operation. If $s$ is an *auto copy* of $t$ then we associate the same *push* operation with $s$ as with $t$. So in our example we associate with the second occurrence of $b$, operation $o_1$. A pop operation which pops a sub-stack $z$ is said to match the push operation associated with $z$. This results in matching several pop operations with a single push operation, unlike in order-1 case. For example, if we do an order-1 pop $o_3$ on $r_2$, followed by an order-2 pop $o_4$, followed by an order-1 pop $o_5$ then both $o_3$ and $o_5$ match $o_1$.

As mentioned in the introduction, we generalize the set of conditions $N_{i,h}$ from [2] to $N_{i,j,h}$. Each element of $N_{i,j,h}$ codes relevant information for an order-$j$ pop operation of stack-$i$ in phase $h$. A set of such pop scenarios $\theta \subseteq \cup_{x=1}^{k} N_{i,j,x}$, where $k$ is the number of phases allowed, is associated with an order-$j$ push operation $o$. $\theta$ captures conditions for any pop operation $o'$ matching $o$. If $s$ is a stack created by $o$ then a matching $o'$ may pop either $s$ or a stack $s'$ which is obtained by several steps of *auto copying* of $s$. For an $order-(n-1)$ sub-stack no *auto copy* can be made. An order-$j$ sub-stack $s$, $j < n-1$, however may be *auto copied* by pushing an order-$x$, $x > j$, stack $t$ containing $s$. That is, $s$ may be *auto copied* by a $push_{i,m}$, $m > j+1$, note that in our notation a $push_{i,m}$ pushes a stack of order-$(m-1)$ in stack-$i$. Therefore, if $s'$ is an *auto copy* of $s$ then $s'$ may be contained in different sub-stacks of order $> j$ than $s$ was. So popping conditions for an *auto copy* of $s$ may depend on popping conditions of sub-stacks of order $> j$ containing this *auto copy* of $s$. This is handled by induction on $j$ (from $j = n-1$ down to $j = 0$) while defining $N_{i,j,h}$ for a fixed $h$. Therefore sets $N_{i,j,h}$ are defined by a nested induction. The outer induction on $h$, the phase number, takes care of the contents of the other stacks in a popping scenario, as in [2]. The inner induction on $j$, for a fixed $h$, handles popping conditions of higher order sub-stacks containing an order-$j$ stack.

## 4 Copying by Higher Order Push Operations

In this section we formally define some useful notions related to copying of higher order stacks as they evolve under a sequence of *hpds* transitions. We begin by defining labeled stacks and extend *hpds* transitions to them. Labeled stacks are used as a tool to define a copy relation on sub-stacks of a higher order stack and to define matching between $push_i$ and $pop_i$ operations in a sequence of *hpds* transitions. A sub-stack of a higher order stack $s$ is a stack of any order contained in $s$.

**Definition 9.** *A labeled order-n stack is an order-n stack s such that each sub-stack of s (including s itself) is labeled by a non-negative integer. We show this label as a superscript at the end of a stack. For example* $[[\perp^0\ a^1]^0\ [\perp^0\ a^1]^2]^0$ *is a labeled order-*2 *stack.*

**Definition 10.** *We extend* hpds *transitions to labeled stacks as follows.*
$push_1^b([a_1^{l_1}, \ldots, a_u^{l_u}]^m) = [a_1^{l_1}, \ldots, a_u^{l_u},\ b^l]^m$, *where* $l$ *is a fresh label.*
$push_1^b([s_1^{l_1}, \ldots, s_u^{l_u}]^m) = [s_1^{l_1}, \ldots, s_{u-1}^{l_{u-1}},\ push_1^b(s_u^{l_u})]^m$, *if* $order(s_u) > 1$.
*For* $i > 0$,

$push_{i+1}([s_1^{l_1}, \ldots, s_u^{l_u}]^m) = [s_1^{l_1}, \ldots, s_u^{l_u}, s_u^{l}]^m$, *if* $order(s_u) = i$ *and* $l$ *is a fresh label.*

$push_{i+1}([s_1^{l_1}, \ldots, s_u^{l_u}]^m) = [s_1^{l_1}, \ldots, s_{u-1}^{l_{u-1}}, \ push_{i+1}(s_u^{l_u})]^m$, *if* $order(s_u) > i$.

*Here* $l_1, \ldots, l_u, l, m$ *are labels. A fresh label is a label not occurring in input (including its sub-stacks) to the push operation.*

*The pop operations extend to a labeled stack in a straightforward way, where a stack along with its label is popped. There is no change in the labels of the remaining stacks.*

Each $push_{i+1}$ move creates a stack of order-$i$ with a unique label. It also copies labeled stacks of order $< i$ without changing their labels.

*Example 1.* Let $t = [[\bot^0]^0]^0$ be an order-2 stack. In our examples, stacks grow from left to right. After a $push_1^a$ operation on $t$ we get the stack $t_1 = [[\bot^0 a^1]^0]^0$. A $push_2$ on $t_1$ gives $t_2 = [[\bot^0 a^1]^0[\bot^0 a^1]^2]^0$. A further $push_1^b$ operation on $t_2$ gives $t_3 = [[\bot^0 a^1]^0[\bot^0 a^1 b^3]^2]^0$.

We define below a simple ordering on sub-stacks $s_1$, $s_2$ of a given stack $s$ based on if $s_1$ occurs below $s_2$ in $s$. This is used in a later definition.

**Definition 11.** *Given a stack* $s$ *of order-*$j$ *and its two sub-stacks* $t_1$ *and* $t_2$ *of order-*$i$ *(*$i < j$*), we define "*$t_1 \leq t_2$ *in* $s$*" if either of the following holds.*

- $s = [s_1, \ldots, s_u]$, $t_1 = s_i$ *and* $t_2 = s_j$ *for* $1 \leq i \leq j \leq u$.
- $s = [s_1, \ldots, s_u]$, $t_1$ *is a sub-stack of* $s_i$, $t_2$ *is a sub-stack of* $s_j$, *for* $1 \leq i \leq j \leq u$.
- $s = [s_1, \ldots, s_u]$, $t_1, t_2$ *are sub-stacks of* $s_i$ *for* $1 \leq i \leq u$ *and* $t_1 \leq t_2$ *in* $s_i$.

*This definition is by induction on* $j - i$.

**Definition 12.** *The initial order-*$n$ *stack* $[\ldots[\bot]\ldots]$ *is labeled as* $[\ldots[\bot^0]^0\ldots]^0$, *that is every sub-stack in it is labeled by* 0. *Let* $s$ *be a labeled order-*$n$ *stack obtained by starting the labeled initial stack and applying some hpds moves. Let* $s_1$, $s_2$ *be two order-*$i$ *sub-stacks of* $s$ *with same labels and let* $s_1 \leq s_2$ *in* $s$. *Then we say that* $s_2$ *is an* auto copy *of* $s_1$ *and denote it as* $s_1 \preceq_i s_2$.

*If a pop move pops a stack with label* $l$ *then we say that it* **matches** *the push move associated with* $l$.

*Example 2.* If on order-3 stack $[[[\bot^0]^0]^0]^0$ the sequence of operations $push_1^a$, $push_1^a$, $push_2$, $push_3$, $pop_1$, $push_1^b$ is performed then we get $t = [[[\bot^0 a^1 a^2]^0[\bot^0 a^1 a^2]^3]^0[[\bot^0 a^1 a^2]^0[\bot^0 a^1 b^5]^3]^4]^0$. Here order-2 stack $[\bot^0 a^1 b^5]^3$ is a copy of the stack $[\bot^0 a^1 a^2]^3$. This shows that a stack can be an *auto copy* of another stack even if their contents are not the same. The *auto copy* relation that we have defined establishes a relation between 'containers' not their contents.

*Example 3.* In the example 1, in $t_3 = [[\bot^0 a^1]^0[\bot^0 a^1 b^3]^2]^0$ the '$a$' (order-0 stack) above is a copy of the '$a$' below. If on $t_3$ we perform the sequence $pop_1$, $pop_1$, $pop_2$, $pop_1$ then both second and the last $pop_1$ operations match the $push_1^a$ operation performed on $t$. Both these operations pop '$a$' associated with this $push_1^a$ operation.

The *auto copy* relation that we have defined can be informally understood as follows. A $push_j$ for $j > 1$, on stack $s$ creates a new instance of $top_{j-1}(s)$ and pushes it on $s$. Creating a new instance of $top_{j-1}(s)$ automatically creates new instances of all stacks (of lower order) contained in $top_{j-1}(s)$. We refer to these new instances of order $< j-1$ as auto copies of their corresponding stacks in $top_{j-1}(s)$. However, we do not consider the new order $j-1$ instance as an auto copy of $top_{j-1}(s)$. The full *auto copy* relation is obtained by taking reflexive, transitive closure of the basic *auto copy* step described above.

There can be many $pop_j$ operations (for $j < n$) matching a single $push_j$ operation. This is a departure from order-1 case. This happens because a stack of order-$(j-1)$ may be *auto copied* several times by $push_r$, $r > j$, operations. A $pop_j$ operation popping any of these *auto copies* is associated with the same $push_j$.

## 5 Reducing *n-HMPDS* Game to Finite State Game

### 5.1 Popping Scenarios

Let $\mathcal{H} = (Q, \Gamma, l, \delta, q_0)$ be *n-mpds* and let $\mathcal{G} = (\mathcal{H}, Q_0, Q_1, M, \Omega : Q \to M)$ be a game structure on $\mathcal{H}$, where $Q = Q_0 \oplus Q_1$ and $M = \{0, \ldots, max\}$ is the set of priorities assigned to vertices of the game graph.

We now define sets $N_{i,j,p}$. These sets are defined simultaneously for all $i$ by a nested induction on $p, j$. The outer induction is on $p$ (starting with $p = k$ down to $p = 1$). For each fixed value of $p$, an induction on $j$ (starting with $j = n-1$ down to $j = 0$) is done. Intuitively $N_{i,j,p}$ is the set of scenarios or constraints to be met for doing a $pop_{i,j+1}$ when the configuration resulting after this *pop* is in phase $p$. This scenario also keeps information about possible scenarios of future *pop* operations. This leads to the induction definition.

**Definition 13.** *In this definition we assume that* $q \in Q$, $\overline{\gamma} \in (\Gamma \cup \{\bot\})^n$ *and* $\overline{m} \in M^{l \times n}$. *Also,* $u \in [1, l]$, $r \in (j, n)$ *and* $h, p \in [1, k]$, *where the intervals shown are integer intervals.*

$$N_{i,n-1,p} = \{(a_1, \ldots, a_{i-1}, (p, q, \overline{\gamma}, \overline{m}), a_{i+1}, \ldots, a_l) \mid a_u \in A_{u,p+1} \text{ for } u \neq i\}$$

$$N_{i,j,p} = \{(a_1, \ldots, a_{i-1}, (p, q, \overline{\gamma}, \overline{m}, b_{i,j+1}, \ldots, b_{i,n-1}), a_{i+1}, \ldots, a_l) \mid a_u \in A_{u,p+1} \text{ for } u \neq i,\ b_{i,r} \subseteq \cup_{x=p}^{k} N_{i,r,x}\}$$

*where* $A_{i,h}$ *are auxiliary sets given as*

$$A_{i,k+1} = \{\emptyset\}, \quad A_{i,h} = \left\{(T_0, \ldots, T_{n-1}) \mid T_j \subseteq \cup_{x=h}^{k} N_{i,j,x}\ \right\}$$

The auxiliary sets $A_{i,h}$ in the above definition are used to keep the definition a bit compact, $A_{i,h}$ do not have any other role.

Each $e = (a_1, \ldots, a_{i-1}, (p, q, \overline{\gamma}, \overline{m}, b_{i,j+1}, \ldots, b_{i,n-1}), a_{i+1}, \ldots, a_l) \in N_{i,j,p}$ represents a scenario for a $pop_{i,j+1}$ operation. Entries $p, q, \overline{\gamma}, \overline{m}$ in $e$ refer to a $pop_{i,j+1}$ transition from configuration $c$ to $c' = (q, p, i, s_1, \ldots, s_l)$, where $\gamma_t = top_{t,0}(c)$, for $1 \leq t \leq l$, $t \neq i$ ($\gamma_i$ is not needed in $e$, we keep it for compact notation), $m_{u,v}$ is the minimum priority visited since $push_{u,v+1}$ corresponding to $top_{u,v}(c)$. Data

$a_u$, $u \neq i$ in $e$ refer to scenarios for $pop_{u,x}$, $0 \leq x < n$, and $b_{i,y}$ refer to scenarios for $pop_{i,y}$, $j+1 < y \leq n$ *after* the $pop_{i,j+1}$ in scenario $e$.

The motivation for $a_u$, $u \neq i$ in $e$ is similar to that in the definition of $N_{i,p}$ in [2], except that in stack $u$ now not just $pop_{u,0}$ but $pop_{u,0}, \ldots, pop_{u,n-1}$ moves are possible. A tuple in set $A_{u,h}$ gives popping scenarios for $pop_{u,0}, \ldots, pop_{u,n-1}$ when such a pop move results in a configuration of phase $\geq h$. These are denoted by $T_0, \ldots, T_{n-1}$ respectively. Any pop operation in stack-$u$, $u \neq i$, subsequent to $pop_{i,j+1}$ move in phase $p$, can only be made in a phase $> p$. Therefore the entries $a_u$ in $e$ are taken from sets $A_{u,p+1}$.

For each $b_{i,r}$, $j+1 \leq r < n$, note that $N_{i,j,p}$ contains scenarios about popping not just a single stack but also about all *auto copies* of it. This is because constraints for all matching $pop_{i,j+1}$ are to be guessed at the time of a $push_{i,j+1}$. As discussed in section 3, if a sub-stack $t$ is popped by a $pop_{i,j+1}$ then order-$(j+1)$,...,order-$(n-1)$ stacks, $t_{j+1}, \ldots, t_{n-1}$ containing $t$ need not be the same stacks in which push associated to $t$ was done. Stack $t$ may be an *auto copy* of stack $s$, where $s$ is the stack actually pushed by the push associated to $t$. Values $q, \overline{\gamma}, \overline{m}$ allowed in a scenario which pops $t$ also depend on popping scenarios for $t_{j+1}, \ldots, t_{n-1}$ containing $t$. Scenario $e$ contains an allowable combination of $q, \overline{\gamma}, \overline{m}$ and popping scenarios for $t_{j+1}, \ldots, t_{n-1}$ explicitly. These Scenarios are $b_{i,j+1}, \ldots, b_{i,n-1}$ in the definition above. In general these are some subsets of $\cup_{x=p}^{k} N_{i,j+1,x}, \ldots, \cup_{x=p}^{k} N_{i,n-1,x}$ respectively.

For the base case, $e \in N_{i,x,k}$ and $a_u = \emptyset$ for $u \neq i$ as in this case $k$ is the last phase so a pop in any stack other than $i$ will not occur after this pop.

Below, we use notation like $\overline{B} = (B_{i,j} | 1 \leq i \leq l, 0 \leq j < n)$, for double indexed sets. We also use $\overline{B}[C/(i,j)]$ to mean the indexed set which is same as $\overline{B}$ except at index $(i,j)$ where it is $C$. For single indexed set we use sequence like notation with $\overline{T}$ and $\overline{T}[C/i]$ as obvious counterparts of $\overline{B}$, $\overline{B}[C/(i,j)]$respectively. For double indexed sets $\overline{B}$ as above we use $\overline{B_i}$ for $B_{i,0}, \ldots, B_{i,n-1}$.

We follow [11] in presentation of our finite state game. Most important vertices of the finite state game (FSG) are of the form $Check(q, p, r, \overline{\gamma}, \overline{B}, \overline{m})$, where $q \in Q$, $p \in [1,k]$, $r \in [0,l]$, $\overline{\gamma} = \gamma_1 \ldots \gamma_l$ with each $\gamma_i \in \Gamma \cup \{\perp\}$. Finally $\overline{B} = (B_{i,j} | 1 \leq i \leq l, 0 \leq j < n)$ and $\overline{m} = (m_{i,j} | 1 \leq i \leq l, 0 \leq j < n)$. Intuitively vertex $Check(q, p, r, \overline{\gamma}, \overline{B}, \overline{m})$ asserts the following about a *hmpds* configuration $c$.

- $q$ is the state of the configuration.
- $p$ is the current phase.
- $r$ is the number of stack on which last pop operation was done (initially it is set to 0).
- $\gamma_i$ is the topmost symbol of stack $i$.
- $B_{i,j} \subseteq \cup_{x=1}^{k} N_{i,j,x}$ is a set of scenarios for $pop_{i,j}$ move on $c$.
- $m_{i,j}$ records the minimum priority visited in the play (till the current instant) since the $push_{i,j}$ move associated to $top_{i,j}(c)$ was played.

Apart from *Check* vertices there are also some auxiliary vertices.

A bit more notation. We use $E \uparrow p$ for $E \cap (\cup_{x=p}^{k} N_{i,j,x})$. For a sequence $\overline{T} = (T_i \mid 1 \leq i \leq m)$, $\overline{T} \uparrow p$ stands for $(T_i \uparrow p \mid 1 \leq i \leq m)$. For a double index $\overline{T}$, $\overline{T} \uparrow p$ is defined similarly.

### 5.2 The Finite State Game (FSG)

Each *hmpds* transition gives rise to some FSG transitions. We group transitions of FSG according to *hmpds* transitions (shown in bold).

1. $(\mathbf{q}, \gamma_\mathbf{i}, \mathbf{q}', \mathbf{i}, \gamma') \in \delta_{\mathbf{ins,j}}$ where $1 \leq i \leq l$ and $1 \leq j \leq n$.

   This gives rise to transitions:
   (a) $Check(q, p, r, \overline{\gamma}, \overline{B}, \overline{m}) \rightarrow Push_{i,j}(p, r, \overline{\gamma}, \overline{B}, \overline{m}, q', \gamma')$
   (b) $Push_{i,j}(p, r, \overline{\gamma}, \overline{B}, \overline{m}, q', \gamma') \rightarrow Claim_{i,j}(p, r, \overline{\gamma}, \overline{B}, \overline{m}, q', \gamma', C)$, for $C \subseteq \cup_{h=p}^{k} N_{i,j-1,h}$
   (c) $Claim_{i,j}(p, r, \overline{\gamma}, \overline{B}, \overline{m}, q', \gamma', C) \rightarrow Check(q', p, r, \overline{\gamma}[\gamma'/i], \overline{B}[C/(i, j-1)], \overline{m'})$,

$$where \quad m'_{x,y} = \begin{cases} \Omega(q') & \text{if } x = i \text{ and } y = j-1 \\ min(m_{i,j}, \Omega(q')) & \text{otherwise} \end{cases}$$

   (d) To check the game after a matching $pop_{i,j}$ operation.

   $Claim_{i,j}(p, r, \overline{\gamma}, \overline{B}, \overline{m}, q', \gamma', C) \rightarrow Jump_{i,j}(q'', h, \overline{\gamma}, \overline{\gamma''}, \overline{m'}, \overline{B'}, \overline{m})$
   for any $(a_1, \ldots, a_{i-1}, z, a_{i+1}, \ldots, a_l) \in C$ where
   $a_r = (B'_{r,0}, \ldots, B'_{r,n-1})$ for $1 \leq r \leq l$, $r \neq i$
   $z = (h, q'', \overline{\gamma''}, \overline{m'}, B'_{i,j}, \ldots, B'_{i,n-1})$,
   $B'_{i,t} = B_{i,t}$ for $0 \leq t < j$.

   (e) $Jump_{i,j}(q'', h, \overline{\gamma}, \overline{\gamma''}, \overline{m'}, \overline{B'}, \overline{m}) \rightarrow Check(q'', h, i, \overline{\gamma''}[\gamma_i/i], \overline{B'}, \overline{m''})$,

$$where \quad m''_{x,y} = \begin{cases} min(m'_{x,y}, \Omega(q'')) & if\ x \neq i\ or\ y \geq j \\ min(m_{x,y}, m'_{i,j-1}, \Omega(q'')) & if\ x = i\ and\ y < j \end{cases}$$

2. $(\mathbf{q}, \gamma_\mathbf{i}, \mathbf{q}', \mathbf{i}) \in \delta_{\mathbf{rem,j}}$, $1 \leq i \leq l$.

   This gives rise to transitions:
   (a) $Check(q, p, r, \overline{\gamma}, \overline{B}, \overline{m}) \rightarrow Win_0$ if $D \in B_{i,j-1}$ and $p' \leq k$
   (b) $Check(q, p, r, \overline{\gamma}, \overline{B}, \overline{m}) \rightarrow Win_1$ if $D \notin B_{i,j-1}$ and $p' \leq k$

   where $p' = np(p, r, i)$ and if $p' \leq k$ then
   - $D = (C_1, \ldots, C_{i-1}, z, C_{i+1}, \ldots, C_l)$
   - $z = (p', q', \overline{\gamma}, \overline{m}, B'_{i,j}, \ldots, B'_{i,n-1})$
   - $C_r = (B'_{r,0}, \ldots, B'_{r,n-1})$ for $1 \leq r \leq l$, $r \neq i$

$$and\ B'_{x,y} = \begin{cases} B_{x,y} \uparrow (p'+1) & \text{if } x \neq i \\ B_{x,y} \uparrow p' & \text{if } x = i \text{ and } y \geq j \end{cases}$$

Priority of a vertex $v$ in FSG, denoted by $\lambda(v)$, is defined as follows.

- $\lambda(Check(q, \ldots)) = \Omega(q)$, $\lambda(Push_{i,j}(\ldots)) = \lambda(Claim_{i,j}(\ldots)) = max$.
- $\lambda(Jump_{i,j}(q, h, \overline{\gamma}, \overline{\gamma'}, \overline{m'}, \overline{B}, \overline{m})) = m'_{i,j-1}$,
  where $\overline{m'} = (m'_{x,y} \mid x \in [1, l], y \in [0, n))$.

Vertices of the form $Check(q, \ldots)$ belong to player-$z$, $z \in \{0, 1\}$, iff $q \in Q_z$. Vertices $Push_{i,j}(...)$, belong to player-0 whereas vertices $Claim_{i,j}(...)$, belong to player-1. Vertices $Jump_{i,j}(...)$ belong to player 0. Vertices $Win_0$ and $Win_1$ belong to player 1 and player 0 respectively. As there are no transitions from $Win_0$ and $Win_1$, by our convention $Win_0$ and $Win_1$ are winning for player 0 and player 1 respectively.

We explain the transition rules of FSG in some detail below.

- Rule (1.a) transfers the game to player-0's vertex ($Push_{i,j}$ vertex) regardless of the player to whom *Check* vertex in belongs. This is because *player 0* only can make a claim about popping scenarios.
- In rule (1.b), $C$ is the set of scenarios for all $pop_{i,j}$ matching the $push_{i,j}$. The same set of popping scenarios is maintained for any *auto copy* of this stack.
- In rule (1.c), player 1 sets the game to the configuration after the $Push_{i,j}$ move. Sub-stacks $top_{x,y}$ in the new configuration are *auto copies* of sub-stacks $top_{x,y}$, in configuration before, for $x \neq i$ or $y \neq j$. Therefore $B_{x,y}$, $m_{x,y}$ for $x \neq i$ or $y \neq j$ remain unchanged.
- In rule (1.d-e), the game is verified after a matching $pop_{i,j}$ move with popping scenario in $C$. We show the transitions for an arbitrary scenario $(a_1, \ldots, a_{i-1}, z, a_{i+1}, \ldots, a_l) \in C$. Let $c'$ be a configuration arising after a *pop* in this scenario. The phase of $c'$ is $h$ and $m'_{x,y}$ are priorities corresponding to $top_{x,y}(c')$, for $x \neq i$ or $y \geq j$.
  Note that if a $push_{i,j}$ is done in configuration $c$ then for $0 \leq t < j$, $top_{i,t}(c')$ is an *auto copy* of $top_{i,t}(c)$. Therefore we have $B'_{i,t} = B_{i,t}$ as the set of popping scenarios remain same for the *auto copies*.
  The value $m'_{i,j-1}$ gives the minimum priority visited between $push_{i,j}$ and the matching $pop_{i,j}$ in the play. This explains the expression $m''_{i,t}$, for priorities corresponding to $top_{i,t}(c')$ for $0 \leq t < j$.
  The $Jump$ vertex is to capture the min priority between $push_{i,j}$ and the matching $pop_{i,j}$ (it is $m'_{i,j-1}$ in the present case) in FSG path (1.a-b-d-e).
- In rule (2), condition $p' \leq k$ refers to the fact that a *pop* move is possible only if the resulting phase is $\leq k$. Given $p' \leq k$, the transition (2.a) represents the case where *pop* move satisfies the popping condition, the transition (2.b) represents complement of this case.
  The main step is to define $D \in N_{i,j,p}$ based on various popping conditions in the current configuration of the play. In $B_{x,y}$ there may be some popping scenarios for phase $< p'$. We remove these scenarios by using operator $\uparrow$. More specifically, we keep scenarios of phase $\geq p'$ for popping in stack $i$ and scenarios of phase $> p'$ for popping in stacks other than $i$ as after the present *pop*, other *pop*s can occur only in these phases. The $B'_{x,y}$ defined in this way form the desired components of $D$ so that $D \in N_{i,j,p}$.

## 6 Relating Winning in *n-HMPDS* Game and the FSG

Our main theorem is the following.

**Theorem 1.** *A* hmpds *game is winning for player* 0 (*from initial configuration* $(q_0, \perp_n, \ldots, \perp_n)$) *iff FSG is winning for player* 0 (*from initial configuration*

$Check(q_0, 1, 0, \overline{\perp}, \overline{\emptyset}, \overline{0})$). *Further, if* hmpds *game is winning for player* 0 *then player* 0 *has a winning strategy in* hmpds *game that is computable by an order-n multi-stack automaton.*

*Proof.* (FSG to *hmpds* game) Assuming that there is a winning strategy for player-0 in FSG from $Check(q_0, 1, 0, \overline{\perp}, \overline{\emptyset}, \overline{0})$, we design an $l$ stack deterministic $n-$hmpda $\mathcal{S}$ which executes a winning strategy $\tau$ of player 0 in *hmpds* game starting from *hmpds* configuration $(q_0, \perp_n, \ldots, \perp_n)$. The automaton $\mathcal{S}$ is an $l$ stack deterministic hmpda with an input and an output tape. It reads moves of player 1 from the input tape and outputs moves of player 0 on the output tape. Detailed construction of $\mathcal{S}$ and the correctness proof of $\mathcal{S}$ is given in full version of this paper [1].
(*Hmpds* game to FSG) Proof of this direction is given in [1].

Idea of the proofs in both directions is similar to that in [11,2], but we need to deal with operations on higher order stacks. □

### 6.1 Complexity of Solving the Game

By the reduction in section 5, to solve a *hmpds* game it suffices to solve an associated FSG. In this section we estimate size of the FSG and the complexity of solving it. Let us define a class of functions $exp_n(m)$ iteratively as follows. $exp_1(m) = 2^m$ and for $n \geq 1$, $exp_{n+1}(m) = 2^{exp_n(m)}$. Roughly, $exp_n(m)$ is a tower of exponentials of height $n$. Let $\mathcal{H}$ be an *hmpds* and $\mathcal{G}$ be an *hmpds* game on $\mathcal{H}$ as in section 5.1. For a set $A$, we let $|A|$ denote its cardinality.

By a simple complexity analysis, whose details are omitted due to lack of space, we get $|N_{i,0,k-1}| \leq exp_{n-1}(z)$ and the number of vertices in FSG is $exp_{n \cdot k}(O(z))$, where $z = |Q| \cdot |M|^{l \times n} \cdot |\Gamma|$ and $|M| > 1$. It follows by [13] that our FSG can be solved and the winning strategy can be constructed in time bounded by $exp_{n \cdot k}(O(z))$, with $z$ as above.

We can code $e \in N_{i,j,k}$ more economically by noting that we need to keep only $m_{i,j}, \ldots, m_{i,n-1}$ in it, in particular $m_{u,x}$, $u \neq i$ need not be stored as there is no *pop* in stack $u$ after a *pop* corresponding to $e$. This leads to size of FSG and the time to solve it as $exp_{n \cdot k}(O(|Q| \cdot |M| \cdot l \cdot |\Gamma|))$.

## 7 Conclusion

In this paper we have defined higher order multi-stack pushdown systems (hmpds). We have shown that parity games on bounded phase *hmpds* are effectively solvable and a winning strategy executable by higher order multi-stack automata can be synthesized effectively. It remains open if the complexity bound given in the paper to solve these games can be improved. Recently we have also shown that winning regions in parity games on bounded phase hmpds are regular.

**Acknowledgments.** Financial support for this work is provided by Research I Foundation.

## References


1. Seth, A.: Games on Higher Order Multi-Stack Pushdown Systems, full version, http://www.cse.iitk.ac.in/users/seth/RP09/fullversion
2. Seth, A.: Games on Multi-Stack Pushdown Systems. In: Artemov, S., Nerode, A. (eds.) LFCS 2009. LNCS, vol. 5407, pp. 395–408. Springer, Heidelberg (2008)
3. Carayol, A., Hague, M., Meyer, A., Ong, C.-H.L., Serre, O.: Winning Regions of Higher-Order Pushdown Games. In: Proc: LICS 2008, pp. 193–204. IEEE Computer Society, Los Alamitos (2008)
4. Hague, M., Murawski, A.S., Luke Ong, C.-H., Serre, O.: Collapsible Pushdown Automata and Recursion Schemes. In: Proc: LICS 2008, pp. 452–461. IEEE Computer Society, Los Alamitos (2008)
5. Madhusudan, P., Parlato, G., La Torre, S.: A Robust Class of Context-Sensitive Languages. In: Proc: LICS 2007, pp. 161–170. IEEE Computer Society, Los Alamitos (2007)
6. Madhusudan, P., Parlato, G., La Torre, S.: Context-Bounded Analysis of Concurrent Queue Systems. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 299–314. Springer, Heidelberg (2008)
7. Qadeer, S., Rehof, J.: Context-bounded model checking of concurrent software. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 93–107. Springer, Heidelberg (2005)
8. Knapik, T., Niwinski, D., Urzyczyn, P., Walukiewicz, I.: Unsafe Grammars and Panic Automata. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1450–1461. Springer, Heidelberg (2005)
9. Bouajjani, A., Meyer, A.: Symbolic reachability analysis of higher-order context-free processes. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 135–147. Springer, Heidelberg (2004)
10. Cachat, T.: Higher order pushdown automata, the caucal hierarchy of graphs and parity games. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 556–569. Springer, Heidelberg (2003)
11. Cachat, T.: Uniform solution of parity games on prefix-recognizable graphs. In: Proc. Infinity. ENTCS, vol. 68(6) (2002)
12. Walukiewicz, I.: Pushdown processes: games and model checking. Information and computation 164, 234–263 (2001)
13. Jurdziński, M.: Small Progress Measures for Solving Parity Games. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 290–301. Springer, Heidelberg (2000)
14. Thomas, W.: Languages, automata and logic. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. III, pp. 389–455. Springer, New York (1997)
15. Engelfriet, J.: Iterated pushdown automata and complexity classes. In: STOC 1983: Proceedings of the fifteenth annual ACM symposium on Theory of computing, pp. 365–373. ACM Press, New York (1983)
16. Damm, W., Goerdt, A.: An automata-theoretical characterization of the OI-hierarchy. Information and Control 71(1-2), 1–32 (1986)
17. Maslov, A.N.: Multilevel stack automata. Problems of Information Transmission 15, 1170–1174 (1976)