# Games on Multi-stack Pushdown Systems

Anil Seth

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

**Abstract.** Bounded phase multi-stack pushdown automata have been studied recently. In this paper we show that parity games over bounded phase multi-stack pushdown systems are effectively solvable and winning strategy in these games can be effectively synthesized. We show some applications of our result, including a new proof of a known result that emptiness problem for bounded phase multi-stack automata is decidable.

## 1 Introduction

A multi-stack pushdown system (*mpds*) has a finite set of control states and a fixed number of stacks. The transition function of a *mpds* takes as input its control state and topmost symbols of each stack and may (nondeterministically) do a push or a pop operation on any stack along with a possible change in control state of *mpds*. A *mpds* obviously generalizes a pushdown system *pds* as it can have more than one stack. While pushdown systems can be used to model sequential recursive programs, multi-stack pushdown systems can be used to model a class of programs with both 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*. Model checking of programs with threads is an important problem and there have been several recent works, see [1,3,4,5,6], in the area of model checking *mpds* and their variants. Some restrictions however are needed to be imposed on *mpds* to get effectively checkable properties of the model, as even simple properties such as reachability from one configuration to another are undecidable for unrestricted *mpds*. A restriction considered in [4,5] is bounded context switching. In a $k$ context switching *mpds* we consider only those runs of *mpds* which can be divided into $k$ stages, where each stage is a consecutive sequence of moves from the run in which push and pop operations are performed only in one stack. While this seems a strong restriction, it has been useful in practice and reachability analysis with this restriction has helped uncover some errors. Bounded context switching *mpds* admit effective global reachability analysis also. For a *mpds* $M$ and a regular set $\mathcal{C}$ of configurations of $M$, $pre^*(\mathcal{C})$, the set of configurations of $M$ from which a configuration in $\mathcal{C}$ can be reached by $M$, is shown to be regular in [5]. Similarly $post^*(\mathcal{C})$, the set of configurations to which a configuration in $\mathcal{C}$ can be reached by $M$, is also shown to be regular in [5].

S. Artemov and A. Nerode (Eds.): LFCS 2009, LNCS 5407, pp. 395–408, 2009.

In [1], a more liberal version of *mpds* called bounded phase systems have been considered. In a $k-$phase bounded *mpds* only those runs of *mpds* are considered which can be divided into $k$ stages where each stage is a consecutive sequence of moves from the run in which *pop* operations are performed only in one stack (push operations can be performed on any stack in a single phase). The class of bounded phase *mpds* strictly includes the class of bounded context switching *mpds*. In [1] emptiness problem of bounded phase multi-stack automata (*mpda*) is shown to be decidable. Bounded phase *mpds* have also been used in model checking of concurrent queues [3] and bounded phase multi-stack pushdown transducers have been used to give an infinite automaton characterization of complexity class of problems solvable in double exponential time, 2ETIME [2].

In model checking, it is often needed to consider richer properties than reachability. This can be done meaningfully over restricted mpds also. For example, existence of a bad infinite path in a bounded phase *mpds* implies the existence of the same for the unrestricted *mpds* also. A general way to specify verification problems is through infinite games [9], where evaluating a formula in a model is equivalent to deciding if a player has a winning strategy in a game. For example, evaluating a modal $mu$-calculus formula on a transition system can be reduced to solving a parity game on a graph closely related to the transition system. Games can also be used to model reactive systems naturally. Two player parity games over single stack pushdown systems (pds) have been studied in [9] where it is shown that these games can be effectively solved. That is, there is an algorithm which from the description of a game can determine which player has a winning strategy in the game starting from the initial configuration of *pds*.

Two player games over *mpds* have not been studied so far. In this paper, we study two player reachability and parity games over bounded phase *mpds* and show them to be effectively solvable. Our solution is based on a fundamental technique of Walukiewicz [9] which shows how to reduce a parity game on a pushdown system to a parity game over finite state space. In [9] 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 parts. In the first subgame player 1 verifies that if the symbol is popped then it is in one of the guessed states, in the second game it is verified that if the pushed symbol is popped satisfying the guessed conditions then the game is winning for player 0. This does not let the stack grow in any subgame, only topmost symbol of the stack needs to be kept in a game state along with some bounded auxiliary information, thus resulting in a finite state game.

We extend this technique to the case of bounded phase *mpds*. To begin with, we need to store the topmost symbol of each stack. The main difficulty in case of more than one stack is that when a symbol in some stack $i$ is pushed it is not sufficient to guess the states and minimum priorities visited for popping this symbol. The contents of stack $j$, $j \neq i$, can change in an arbitrary way between a push and the corresponding pop in stack $i$. For example, we may have a sequence like $push_i$, $push_j$ $push_j$ $pop_i$, where subscript indicates the

stack on which the operation is performed. In general, we may have an arbitrary number of push operations to stack $j$, $j \neq i$ between a matching push and pop operation in stack $i$. The information about contents of stack $j$ at the time of pop in stack $i$ is needed to simulate the *mpds* transitions after the $pop_i$. The important issue is to see how much information about stack $j$ is needed, will a bounded amount of information suffice? Let us examine this, we need to know at least the topmost symbol of stack $j$, say $\gamma'$, at the time of $pop_i$. In addition to this, to abstract out the contents of stack $j$ below $\gamma'$, we may need something like popping condition for $\gamma'$ after $pop_i$. But this seems circular as in identifying the popping condition for $push_i$ we need to know the popping condition for $\gamma'$ in stack $j$, $j \neq i$. Further, the popping condition for $\gamma'$ of stack $j$, $j \neq i$, needs to be guessed at the time of $push_i$ only. The apparent circularity may be removed by looking at how the phase changes after various *pop* operations involved. We formulate these conditions recursively using recursion on the number of phases. This is the main technical contribution in our reduction of bounded phase *mpds* game to a finite state game. The size of the FSG, we get is rather huge. It is a tower of exponentials of height $k$, the number of phases of *mpds*. This is also the complexity of our decision procedure to solve *mpds* game.

In [9] it is also shown that the winning strategy in pds game can be executed by a pushdown automaton. This extends naturally to our setting, we show that the winning strategy in bounded phase *mpds* game can be executed by a bounded phase *mpda.* As a special case of our general game, we consider one player reachability game, in which all configurations belong to player 0 only and the winning condition is reaching some specified control state. Existence of winning strategy for player 0 in this game is equivalent to the specified control state being reachable from the initial configuration of *mpds*. In this case the size of our finite state game reduces to double exponential in the number of phases allowed. This special case gives us an alternative proof of one of the main results in [1] that the emptiness problem of bounded phase *mpda* is decidable. Our proof uses quite different technique than the ones used in [1]. In [1] a translation of bounded phase words into finite trees is defined such that the image of this translation is definable in monadic second order logic (MSO) over finite trees. Decidability of MSO on the class of finite trees is then used in [1] to derive the result. The complexity of our decision procedure matches the optimal complexity bound obtained in [3]. Further, our strategy synthesis result when specialized to one player reachability game, can be used to generate counter-examples in automatic verification of safety properties of bounded phase *mpds*.

## 2 Preliminaries

**Definition 1.** *A multi-stack pushdown system (mpds) is given as a tuple* $(Q, \Gamma, l, \delta, q_0)$*, where* $Q$ *is a finite set of states,* $l$ *is the number of stacks,* $\Gamma$ *is the stack alphabet and* $q_0$ *is the initial state. The transition function* $\delta$ *is given as* $\delta = \delta_i \cup \delta_r \cup \delta_e$*, where*

- $\delta_e \subseteq Q \times \Gamma^l \times Q \times [1 \dots l] \times \Gamma$,
- $\delta_i \subseteq Q \times \Gamma^l \times Q \times [1 \dots l] \times \Gamma$,
- $\delta_r \subseteq Q \times \Gamma^l \times [1 \dots l] \times Q$.

*($\delta_e, \delta_i, \delta_r$ represent exchange, push and pop operations respectively). An* mpds *operation depends on its control state and topmost symbols of all its stacks. $\Gamma$ has a special symbol $\perp$ for marking bottom of a stack. $\perp$ can't be pushed, popped or exchanged by any other symbol. For instance, if $(q, \bar{\gamma}, q', j, \gamma) \in \delta_e$, where $\bar{\gamma} = \gamma_1 \dots \gamma_l$ and $\gamma_j = \perp$ then $\gamma = \perp$. Also, if $(q, \bar{\gamma}, i, q') \in \delta_r$ then $\gamma_i \neq \perp$.*

**Definition 2.** *A configuration of multi-stack pushdown system $(Q, \Gamma, l, \delta, q_0)$ is a tuple $(q, s_1, \dots, s_l)$, where $q \in Q$ and $s_i \in \{\perp\} \times (\Gamma - \{\perp\})^*$, for $1 \leq i \leq l$. One step transition $\xrightarrow{t}$ on configurations of mpds is defined as below, where $\bar{\gamma} = \gamma_1 \dots \gamma_l$.*

- $(q, s_1.\gamma_1, \dots, s_l.\gamma_l) \xrightarrow{t} (q', s'_1, \dots, s'_l)$ *if* $t = (q, \overline{\gamma}, q', i, \gamma) \in \delta_e$, $s'_i = s_i.\gamma$ *and* $s'_j = s_j.\gamma_j$ *for* $j \neq i, 1 \leq j \leq l$.
- $(q, s_1.\gamma_1, \dots, s_l.\gamma_l) \xrightarrow{t} (q', s'_1, \dots, s'_l)$ *if* $t = (q, \overline{\gamma}, q', i, \gamma) \in \delta_i$, $s'_i = s_i.\gamma_i.\gamma$ *and* $s'_j = s_j.\gamma_j$ *for* $j \neq i, 1 \leq j \leq l$.
- $(q, s_1.\gamma_1, \dots, s_l.\gamma_l) \xrightarrow{t} (q', s'_1, \dots, s'_l)$ *if* $t = (q, \overline{\gamma}, i, q') \in \delta_r$, $s'_i = s_i$ *and* $s'_j = s_j.\gamma_j$ *for* $j \neq i,\ 1 \leq j \leq l$.

*The initial configuration of* mpds *is defined as* $(q_0, \perp, \dots, \perp)$.

**Definition 3.** *A multi-step transition between configurations of mpds, on say sequence $t_1 t_2 \dots t_n$ of* mpds *moves, $c \overset{t_1 t_2 \dots t_n}{\twoheadrightarrow} d$ is defined as follows. $c \overset{t_1 t_2 \dots t_n}{\twoheadrightarrow} d$ iff either $n = 0$ and $c = d$ or there is a $c'$ s.t. $c \xrightarrow{t_1} c'$ and $c' \overset{t_2 \dots t_n}{\twoheadrightarrow} d$. We write $c \twoheadrightarrow d$ for a multi-step transition from $c$ to $d$ when the sequence of* mpds *moves is not relevant.*

**Definition 4.** *Let $c_0$ be the initial configuration of a* mpds *and let $c$ be a configuration reached from $c_0$ by a sequence $t_1 t_2 \dots t_n$ of* mpds *moves. If the topmost symbol of stack $i$ is $\gamma \neq \perp$, then the push operation corresponding to the topmost symbol of stack $i$ is defined as the last unmatched push operation in stack $i$ in the sequence $t_1 t_2 \dots t_n$.*

The previous definition is usually understood without explicitly mentioning it. It may be noted however that because of exchange operations the push operation corresponding to $\gamma$ may actually have pushed a symbol $\gamma'$, $\gamma' \neq \gamma$ on the stack. For example, after sequence $push_1^{\gamma_1}$, $exch_1^{\gamma_1,\gamma}$, (where $push_i^{\gamma}$ means pushing symbol $\gamma$ to stack $i$, $exch_i^{\gamma,\gamma'}$ means replacing the topmost symbol $\gamma$ of the stack $i$ by $\gamma'$) the topmost symbol on stack 1 is $\gamma$, but it corresponds to $push_1^{\gamma_1}$ operation. Similarly, in sequence $push_1^{\gamma_1}$, $exch_1^{\gamma_1,\gamma}$, $pop_1$, the last pop matches the first push operation $push_1^{\gamma_1}$.

**Definition 5.** *A configuration $d$ of multi-stack pushdown system $(Q, \Gamma, l, \delta)$ is reachable from configuration $c$ in $m$-phases if there are $\alpha_1, \ldots, \alpha_m$ where each $\alpha_i$ is a sequence of* mpds *moves with pop moves from at most one stack and $c \overset{\alpha_1}{\twoheadrightarrow} c_1 \overset{\alpha_2}{\twoheadrightarrow} c_2 \ldots \overset{\alpha_m}{\twoheadrightarrow} c_m = d$ .*

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 [12].

A 2-player $k$-phase *mpds* 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 *mpds*, $Q = Q_0 \oplus Q_1$ 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 *mpds*. 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 $c \to c'$. A play is a sequence of legal moves starting from the initial configuration. A phase in a play is a consecutive sequence of moves such that in this sequence elements from at most one stack are popped (though in a single phase elements may be pushed to any stack). 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.

**Remark**1: Some authors take a play to be a sequence of moves which can not be extended further, we use the descriptor maximal play for this and take a play to be any sequence of moves as defined above.

**Remark**2: We could make the game graph more standard by taking its vertices as triples $(c, p, r)$ where $c$ is an *mpds* configuration $p \leq k$ is the phase in play so far and $r \leq l$ is the number of stack popped last. There is an edge from $(c, p, r)$ to $(c', p', r')$ iff $c \to c'$ and as a result of this transition the phase changes from $p$ to $p'$ and the number of the last popped stack changes from $r$ to $r'$ in the *mpds* configuration.

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 configurations $c$, $c'$ of $\mathcal{G}$, $c \overset{\tau}{\to} c'$ and $c \overset{\tau}{\twoheadrightarrow} c'$ mean that $c'$ is reachable from $c$ in a $\tau$-play in one move or in an arbitrary number of moves respectively.

## 3 Reducing *MPDS* Game to Finite State Game

### 3.1 Intuitive Idea

Let $\mathcal{H} = (Q, \Gamma, l, \delta, q_0)$ be a *mpds* and let $\mathcal{G} = (\mathcal{H}, Q_0, Q_1, max, \Omega : Q \to M)$ 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. We use below notation $\overline{T}$ for a sequence $T_1, \ldots, T_l$ and $\overline{T}[C/i]$ for sequence $T_1, \ldots, T_{i-1}, C, T_{i+1}, \ldots, T_l$, which is the same as $\overline{T}$ except at $i^{th}$ position where it is $C$. We follow [8] 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 $\gamma_i \in \Gamma$, $\overline{B} = B_1, \ldots, B_l$ with each $B_i \in \tau_i$ and $\overline{m} = m_1 \ldots m_l$ with $m_i \in \{0 \ldots max\}$. Intuitively the vertex $Check(q, p, s, \overline{\gamma}, \overline{B}, \overline{m})$ asserts that

- $q$ is the state of the configuration.
- $p$ is the current phase.
- $r$ is the number of stack from which last pop operation was done (initially it is set to 0).
- $\gamma_i$ is the topmost symbol of stack $i$.
- $m_i$ is the minimum priority seen since the push operation corresponding to the topmost symbol of stack $i$ (if topmost symbol is $\perp$, which is never pushed, then $m_i = 0$).
- $B_i \subseteq \cup_{j=1}^{k} N_{i,j}$ gives constraints to be met on popping the topmost element of stack $i$ (if stack $i$ is empty then $B_i = \emptyset$).

When an element of stack $i$ is popped, the resulting configuration, say $d$, is in a phase belonging to $[1, k]$. $N_{i,j}$ is the set of conditions related to pop operations of stack $i$ which result in configurations of phase $j$ on popping. In a FSG play for each element pushed in stack $i$ a subset of all popping scenarios $\cup_{j=1}^{k} N_{i,j}$ is guessed. $N_{i,j}$ is defined below simultaneously for $1 \leq i \leq l$ using recursion on $j$, starting from $j = k$ going down to $j = 1$.

**Definition 6.** *In this definition we assume that* $q, \overline{m}, \overline{\gamma}$ *range over* $Q, M^l, \Gamma^l$ *respectively and* $p$ *ranges over* $[1, l]$.

$N_{i,k} = \{(a_1 ... a_{i-1}, (q, k, \overline{\gamma}, \overline{m}), a_{i+1} ... a_l) \mid a_p = \emptyset \ p \neq i\}$
*For* $j$, $k > j \geq 1$,
$N_{i,j} = \{(a_1 ... a_{i-1}, (q, j, \overline{\gamma}, \overline{m}), \ a_{i+1} ... a_l) \mid a_p \subseteq \cup_{r=j+1}^{k} N_{p,r} \ p \neq i\}$.

Each element $u \in N_{ij}$ describes a scenario for popping an element of stack $i$. Assume that the element is popped in *mpds* configuration $c$ and the resulting *mpds* configuration (after pop) is $d$. The tuple $(q, j, \overline{\gamma}, \overline{m})$ in a scenario $u$ stipulates that $\overline{\gamma} = \gamma_1 \ldots \gamma_l$ and $\overline{m} = m_1 \ldots m_l$, where for $1 \leq r \leq l$, $\gamma_r$ is the topmost symbol of stack $r$ in $c$ and $m_r$ is the value in $c$ whose interpretation is as described above. $q$ is the *mpds* state in $d$ and $j$ is the phase of $d$. $a_t$ for $1 \leq t \leq l$, $t \neq i$, is a set of popping conditions for the topmost symbol $\gamma_t$ of stack $t$ in configuration $d$. That is $a_t$ stands for conditions in which $\gamma_t$ can be

popped. This uses recursively the conditions defined for popping a symbol. Note that after popping stack $i$ in $d$ a pop in stack $t$ will result in configuration of phase $> j$. This ensures well defined nature of recursion. Also note that we use $a_t$ as a set of scenarios, instead of a single scenario for popping of $\gamma_t$ in some configuration after $d$. This is necessary in a two player game, because in a two player game player 0 can only guarantee that the resulting *mpds* configuration be one from a set (rather than a uniquely specified) of *mpds* configurations.

In the definition of $N_{i,k}$, $a_t$ for $t \neq i$, is taken to be $\emptyset$ because after popping the symbol in stack $i$ phase $k$ is reached and no other stack can be popped, so popping conditions for topmost symbol of stack $t$ is not of any use now.

### 3.2 The Finite State Game (FSG)

Each *mpds* transition gives rise to some FSG transitions. We group transitions of FSG according to *mpds* transitions (shown in bold) which give rise to them.

1. $(\mathbf{q}, \overline{\gamma}, \mathbf{q'}, \mathbf{i}, \gamma') \in \delta_{\mathbf{e}}$, $1 \leq i \leq l$.
   (a) $Check(q, p, r, \overline{\gamma}, \overline{B}, \overline{m}) \rightarrow Check(q', p, r, \overline{\gamma}[\gamma'/i], \overline{B}, \overline{m'})$, where $m'_j = min(m_j, \Omega(q'))$, for $1 \leq j \leq l$.
2. $(\mathbf{q}, \overline{\gamma}, \mathbf{q'}, \mathbf{i}, \gamma') \in \delta_{\mathbf{i}}$, $1 \leq i < l$.
   (a) $Check(q, p, r, \overline{\gamma}, \overline{B}, \overline{m}) \rightarrow Push_i(p, r, \overline{\gamma}, \overline{B}, \overline{m}, q', \gamma')$
   (b) $Push_i(p, r, \overline{\gamma}, \overline{B}, \overline{m}, q', \gamma') \rightarrow Claim_i(p, r, \overline{\gamma}, \overline{B}, \overline{m}, q', \gamma', C)$, for all $C \subseteq \cup_{j=1}^{k} N_{i,j}$.
   (c) $Claim_i(p, r, \overline{\gamma}, \overline{B}, \overline{m}, q', \gamma', C) \rightarrow Check(q', p, r, \overline{\gamma}[\gamma'/i], \overline{B}[C/i], \overline{m'})$,

$$where \quad m'_j = \begin{cases} min(m_j, \Omega(q')) & if\ j \neq i \\ \Omega(q') & if\ j = i \end{cases}$$

   (d) To check the game after corresponding $pop_i$ operation.
   $Claim_i(p, r, \overline{\gamma}, \overline{B}, \overline{m}, q', \gamma', C) \rightarrow Jump_i(q'', j, \overline{\gamma}, \overline{\gamma''}, \overline{m'}, \overline{B'}, \overline{m})$
   for all $(a_1 ... a_{i-1}, (q'', j, \overline{\gamma''}, \overline{m'}),\ a_{i+1} ... a_l) \in C$
   where $\overline{B'} = (a_1 ... a_{i-1}, B_i, a_{i+1} ... a_l)$.
   (e) $Jump_i(q'', j, \overline{\gamma}, \overline{\gamma''}, \overline{m'}, \overline{B'}, \overline{m}) \rightarrow Check(q'', j, i, \overline{\gamma''}[\gamma_i/i], \overline{B'}, \overline{m''})$,

$$where \quad m''_j = \begin{cases} min(m'_j, \Omega(q'')) & if\ j \neq i \\ min(m_i, m'_i, \Omega(q'')) & if\ j = i \end{cases}$$

3. $(\mathbf{q}, \overline{\gamma}, \mathbf{i}, \mathbf{q'}) \in \delta_{\mathbf{r}}$, $1 \leq i \leq l$.
   (a) $Check(q, p, r, \overline{\gamma}, \overline{B}, \overline{m}) \rightarrow Win_0$ if $\overline{C}[(q', p', \overline{\gamma}, \overline{m})/i] \in B_i$

(b) $Check(q, p, r, \overline{\gamma}, \overline{B}, \overline{m}) \rightarrow Win_1$ if $\overline{C}[(q', p', \overline{\gamma}, \overline{m})/i] \notin B_i$

$$where \quad p' = \begin{cases} 1 & if\ r = 0 \\ p & if\ r = i \\ p+1 & if\ r \neq i \end{cases},$$

$$p' \leq k \text{ and } \overline{C} = \begin{cases} \overline{B} & if\ \ p' < k \\ \overline{\emptyset} & if\ \ p' = k \end{cases}.$$

Priority of vertex $v$ in FSG, denoted by $\lambda(v)$, is defined as follows. $\lambda(Check(q, \ldots)) = \Omega(q)$, $\lambda(Push_i(\ldots)) = \lambda(Claim_i(\ldots)) = max$ and $\lambda(Jump_i(q, j, \overline{\gamma}, \overline{\gamma'}, \overline{n}, \overline{B}, \overline{m})) = n_i$, where $\overline{n} = n_1 \ldots n_l$. Vertex $Check(q, \ldots)$ belongs to player $j$, $j \in \{0, 1\}$, iff $q \in Q_j$. Vertices $Push_i(...)$, belong to player 0 whereas vertices $Claim_i(...)$, belong to player 1, for $0 \leq i < n$. Each vertex $Jump_i(...)$ has a single outgoing edge so it is immaterial which player is assigned to these vertices.

## 4 Relating Winning in *MPDS* Game and the FSG

Our main theorem is the following.

**Theorem 1.** *A* mpds *game is winning for player* 0 (*from initial configuration* $(q_0, \bot_{\mathcal{S}}, \ldots, \bot_{\mathcal{S}})$) *iff FSG is winning for player* 0 (*from initial configuration* $Check(q_0, 1, 0, \overline{\bot}, \overline{\emptyset}, \overline{0})$). *Further, if* mpds *game is winning for player* 0 *then player* 0 *has a winning strategy in* mpds *game that is computable by a multi-stack automaton.*

*Proof.* We present the construction of strategy automaton below, this is used in proving the direction that a winning strategy for player 0 in FSG implies a winning strategy in *mpds* game. The detailed correctness of this construction as well as the other direction of the theorem that a winning strategy for player 0 in mpds game implies a winning strategy in FSG, are given in full version of this paper. Idea of the proofs is similar to that in [8], but we need to deal with more involved cases as we argue for multi-stack pushdown systems. □

### 4.1 Strategy Automaton

Assuming that there is a winning strategy for player 0 in FSG from $Check(q_0, 1, 0, \overline{\bot}, \overline{\emptyset}, \overline{0})$, we design a $l$ stack pushdown automaton $\mathcal{S}$ which executes a winning strategy $\tau$ of player 0 in *mpds* game from *mpds* configuration $(q_0, \bot_{\mathcal{S}}, \ldots, \bot_{\mathcal{S}})$.

Fix a history free winning strategy $\sigma$ for player 0 in FSG from configuration $Check(q_0, 1, 0, \overline{\bot}, \overline{\emptyset}, \overline{0})$ (such a strategy exists in any parity game, see [12]). Using $\sigma$, we design a deterministic $l$ stack pushdown automaton $\mathcal{S}$ as follows. Apart from the stacks, $\mathcal{S}$ has an input and an output tape. $\mathcal{S}$ reads moves of player 1

from the input tape and outputs moves of player 0 on the output tape. Structure of $\mathcal{S}$'s stacks, at any point in play, is the same as that of *mpds* stacks at that point. For a symbol $\gamma \in \Gamma$ in stack $i$ of mpds, $\mathcal{S}$ stores at the corresponding position in its stack $i$ a tuple of the form $(\gamma, B, m)$, where $B \subseteq \cup_{j=1}^{k} N_{i,j}$ and $m \in \{0, 1, \ldots, max\}$. The additional information $(B, m)$ in stacks of $\mathcal{S}$ records relevant information about the FSG play being simulated. Bottom of stack marker for $\mathcal{S}$ is defined to be $\perp_{\mathcal{S}} = (\perp, \emptyset, 0)$. Control state of $\mathcal{S}$ is of the form $(q, p, r)$, where $q$ is the current state in *mpds* game, $p$ is the phase and $r$ is the stack from which *mpds* play can pop in phase $p$.

Configuration of $\mathcal{S}$ is defined as its state along with the contents of its stacks. It subsumes the current *mpds* configuration in it. We define a function $f$ from configurations of $\mathcal{S}$ to vertices of FSG. If $c$ is a configuration of $\mathcal{S}$ in which state of $\mathcal{S}$ is $(q, p, r)$ and top of the stack $j$ symbol is $(\gamma_j, B_j, m_j)$, for $1 \leq j \leq l$, then $f(c) = Check(q, p, r, \overline{\gamma}, \overline{B}, \overline{m})$. In strategy execution the following invariant is maintained: if $\mathcal{S}$ is in configuration $c$ then there is a $\sigma$ play from $f(c_0)$ to $f(c)$, where $c_0$ is the initial configuration of $\mathcal{S}$.

Suppose $\mathcal{S}$ is in configuration $c$ as above. The next move in the *mpds* play is either read from the input tape (if $q \in Q_1$) or $\mathcal{S}$ mimics the $\sigma$ move (if $q \in Q_0$) from the corresponding FSG configuration $f(c) = Check(q, p, r, \overline{\gamma}, \overline{B}, \overline{m})$. (Moves from an *mpds* configuration and from the corresponding $Check(\ldots)$ vertex in FSG are in $1-1$ correspondence by design of FSG.)

### 4.2 Operation of $\mathcal{S}$

Initially, $\mathcal{S}$ is in state $(q_0, 1, 0)$ and each stack of $\mathcal{S}$ is $\perp_{\mathcal{S}}$. At any arbitrary point in *mpds* play if $\mathcal{S}$ is in configuration $c$ in which state of $\mathcal{S}$ is $(q, p, r)$ and top of the stack $j$ symbol is $(\gamma_j, B_j, m_j)$, for $1 \leq j \leq l$. Then for each *mpds* move played in the *mpds* game, we describe actions performed by $\mathcal{S}$. Stack and state updation of $\mathcal{S}$ below is grouped by the moves of *mpds* regardless of whichever player plays.

**[I]** $(q, \overline{\gamma}, q', i, \gamma') \in \delta_e$, $1 \leq i \leq l$.

$\mathcal{S}$ sets top of the stack $i$ symbol to $(\gamma', B_i, min(m_i, \Omega(q')))$ and top of the stack $j$ (for $j \neq i$) symbol to $(\gamma_j, B_j, min(m_j, \Omega(q')))$ and changes its control state to $(q', p, r)$.

**[II]** $(q, \overline{\gamma}, q', h, \gamma') \in \delta_i$, $1 \leq h \leq l$.

Let $C \subseteq \cup_{t=1}^{k} N_{h,t}$ be as in transition (2.b) in corresponding $\sigma$ play. $\mathcal{S}$ pushes $(\gamma', C, \Omega(q'))$ on stack $h$, topmost symbol of the stack $j$ (for $j \neq h$) is changed to $(\gamma_j, B_j, min(m_j, \Omega(q')))$ and $\mathcal{S}$ changes its control state to $(q', p, r)$.

**[III]** $(q, \overline{\gamma}, i, q') \in \delta_r$, $1 \leq i \leq l$.

$\mathcal{S}$ pops the topmost symbol from stack $i$. Let the resulting topmost symbol in stack $i$ (after the above pop) be $(\gamma', D,\ n)$. $\mathcal{S}$ modifies it to $(\gamma', D, min(m_i, n, \Omega(q')))$. $\mathcal{S}$ modifies the topmost symbol of

stack $j$ (for $j \neq i$) to $(\gamma_j, B_j, min(m_j, \Omega(q')))$ and enters control state $(q', p', i)$, where

$$p' = \begin{cases} 1 & if\ r = 0 \\ p & if\ r = i \\ p+1 & if\ r \neq i \end{cases}.$$

### 4.3 Complexity of Solving the Game

By the reduction in section 3, to solve the *mpds* game it suffices to solve the associated FSG. In this section we estimate the size of FSG and the complexity of solving it. Let us begin by defining 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 *mpds* and $\mathcal{G}$ be an *mpds* game on $\mathcal{H}$ as in section 3.1. For a set $A$, we let $|A|$ denote its cardinality. Then by definition 6,

$|N_{i,k}| = |Q|.k.|M|^l.|\Gamma|^l$, for all $i$.

$|N_{i,j}| \leq t.(\prod_{p=1}^{l} 2^{\Sigma_{r=j+1}^{k}|N_{p,r}|})$, for $1 \leq j < k$ where $t = |Q|.k.|M|^l.|\Gamma|^l$.

$|N_{i,j}| \leq t.(\prod_{p=1}^{l} 2^{k|N_{p,j+1}|})$, using $|N_{p,j+1}| > |N_{p,r}|$ for $r > j+1$

$|N_{i,j}| \leq t.(2^{l.k|N_{i,j+1}|})$, because by symmetry $|N_{p,j+1}| = |N_{i,j+1}|$, for $1 \leq p \leq l$.

$|N_{i,j}| \leq (2^{c.l.k|N_{i,j+1}|})$ for a constant $c$, using $m.2^m \leq 2^{2m}$.

This leads to $|N_{i,1}| = exp_{k-1}(O(z))$ and $|B_1| = exp_k(O(z))$, where $z = l.k^2.|Q|.|M|^l.|\Gamma|^l$. The number of vertices in FSG is therefore $exp_k(O(z))$.

It is known that a game graph with $n$ vertices, $m$ edges and $d$ priorities can be solved in time $O(m.n^d)$, see [10].

Number of edges in FSG is bounded by $[exp_k(O(z))]^2$, which is the same as $exp_k(O(z))$. Number of distinct priorities in FSG is $|M|$. It follows that our FSG can be solved and winning strategy can be constructed in time bounded by $exp_k(O(z.|M|))$, where $z = l.k^2.|Q|.|M|^l.|\Gamma|^l$ as mentioned above.

## 5 One Player Case

Let $\mathcal{H}$ be an *mpds* and $\mathcal{G}$ be an *mpds* game on $\mathcal{H}$ as in section 3.1. In this section, we consider the special case when all configurations belong to player 0, that is $Q = Q_0$. In this case popping conditions in definition 6 can be simplified as follows.

**Definition 7.** *In this definition we assume that* $q, \overline{m}, \overline{\gamma}$ *range over* $Q, M^l, \Gamma^l$ *respectively and* $p$ *ranges over* $[1, l]$.

$N_{i,k} = \{(a_1...a_{i-1}, (q, k, \overline{\gamma}, \overline{m}), a_{i+1}...a_l) \mid a_p = \emptyset\ p \neq i\}$

*For* $j$, $k > j \geq 1$,

$N_{i,j} = \{(a_1...a_{i-1}, (q, j, \overline{\gamma}, \overline{m}),\ a_{i+1}...a_l) \mid a_p \in \cup_{r=j+1}^{k} N_{p,r}\ \ p \neq i\}$.

$B_i \in \cup_{j=1}^{k} N_{i,j}$

Complexity analysis in this case becomes:

$|N_{i,k}| = |Q|.k.|M|^l.|\Gamma|^l$, for all $i$.

$|N_{i,j}| \leq \prod_{p=1}^{l}(\Sigma_{r=j+1}^{k}|N_{p,r}|)$, for $1 \leq j < k$.

$|N_{i,j}| \leq (\prod_{p=1}^{l} k.|N_{p,j+1}|)$ using $|N_{p,j+1}| > |N_{p,r}|$ for $r > j+1$.

$|N_{i,j}| \leq (k.|N_{i,j+1}|)^l$, because by symmetry $|N_{p,j+1}| = |N_{i,j+1}|$, for $1 \leq p \leq l$.

This leads to $|N_{i,1}| = z^{l^{O(k)}}$ and $|B_1| = z^{l^{O(k)}}$, for $l \geq 2$ where $z = |Q|.|M|.|\Gamma|$.

The number of vertices in FSG is therefore $z^{l^{O(k)}}$ and it can be solved in time $z^{|M|.(l^{O(k)})}$, where $z = |Q|.|M|.|\Gamma|$.

### 5.1 Bounded Phase Multi-stack Pushdown $\omega$-Automata

We may also consider bounded phase multi-stack pushdown $\omega$-automata on infinite words.

**Definition 8.** *A bounded phase multi-stack pushdown parity $\omega$-automaton is given as a tuple $(Q, \Sigma, \Gamma, l, k, \delta, q_0, M, \Omega)$, where $Q$ is a finite set of states, $\Sigma$ is an input alphabet and $k$ is a bound on the number of phases. $\Gamma, q_0, l$ are the same as in definition 1. $\delta = \delta_i \cup \delta_r \cup \delta_e$ is also the same as in definition 1,2 except that each transition of the automaton also depends on the current symbol being read from the input tape apart from the state and the topmost symbols of all stacks. Therefore*

- $\delta_e \subseteq Q \times \Sigma \times \Gamma^l \times Q \times [1 \dots l] \times \Gamma$,
- $\delta_i \subseteq Q \times \Sigma \times \Gamma^l \times Q \times [1 \dots l] \times \Gamma$,
- $\delta_r \subseteq Q \times \Sigma \times \Gamma^l \times [1 \dots l] \times Q$.

A configuration of this automaton is the same as a *mpds* configuration along with position of input head on the input tape. With each move the input head moves one position to the right. A run of this automaton is a sequence of configurations starting with the initial configuration and in which for any two consecutive configurations the successor configuration is a result of some $\delta-$transition on the predecessor configuration. A run is accepting if the number of phases in it are $\leq k$ and it satisfies the parity acceptance condition given by $\Omega$. We have the following theorem about such automata.

**Theorem 2.** *Emptiness problem for bounded phase multi-stack pushdown $\omega-$automata with parity acceptance condition is decidable in time $(|Q|.|M|.|\Gamma|)^{|M|.(l^{O(k)})}$, where $Q, \Gamma, l, k, M$ are as in definition 8 and $l \geq 2$.*

*Proof.* Consider a bounded phase multi-stack $\omega$-automaton $\mathcal{M}$ as in definition 8. We erase all input symbols from transitions of $\mathcal{M}$ and let all states belong to player 0. This gives a one player *mpds* game. Winning in this game for player 0 is equivalent to $\mathcal{M}$ having an accepting run on some input, that is $\mathcal{L}(\mathcal{M}) \neq \emptyset$. By section 5, this *mpds* game can be solved in time $(|Q|.|M|.|\Gamma|)^{|M|.(l^{O(k)})}$. $\square$

### 5.2 Reachability in Bounded Phase *mpds*

In this section we study applications of our results to reachability problem among configurations of *mpds*. Reachability easily reduces to parity winning condition.

**Definition 9.** *Let $\mathcal{H}$ be a* mpds *with $Q$ as its finite set of states. A set $\mathcal{C}$ of configurations of $\mathcal{H}$ is regular if there is a finite multi-automaton which accepts string $s_1\#s_2\#\ldots\#s_l$ starting from state $q$ iff $(q, s_1, s_2, \ldots, s_l) \in \mathcal{C}$.*

The finite multi-automaton in the above definition is just a finite automaton which has one initial state for each $q \in Q$. Finite multi-automata were introduced in [11].

A slightly general version of reachability problem for *mpds* can be defined as the following decision problem.

**Definition 10.** *(Regular reachability problem) Given an* mpds $\mathcal{M}$ *and a regular set $R$ of configurations of $\mathcal{M}$, is there a $r \in R$ and such that configuration $r$ is reachable from the initial configuration.*

**Theorem 3.** *Regular reachability problem for bounded phase* mpds *is decidable in time $(|Q|.|\Gamma|)^{l^{O(k+l)}}$, where $|Q|$ is the sum of states in input* mpds *and in input multi-automaton accepting $R$, $\Gamma$ is stack alphabet, $k$ is the bound on phases and $l \geq 2$ is the number of stacks in the input* mpds.

*Proof.* The idea is to add transitions to the input *mpds* to check if its configuration is in $R$. This can be done in $2l$ phases. Easy details of this proof are given in full version. □

**Corollary 1.** *Emptiness problem of bounded phase nondeterministic multi-stack pushdown automata is decidable in $(|Q|.|\Gamma|)^{l^{O(k+l)}}$ time where $|Q|$ is the number of states, $\Gamma$ is stack alphabet, $k$ is bound on phases and $l \geq 2$ is the number of stacks in input* mpda.

*Proof.* Let given multi-stack pushdown automaton accept by reaching the final state $q_f$. We convert this *mpda* to *mpds* by erasing input symbols from transitions. Emptiness problem of the given automaton is the same as regular reachability problem of the resulting *mpds* with $R = \{q_f\} \times (\Gamma^*)^l$. □

If we keep fixed all parameters of *mpda* except the number of phases allowed, by the above corollary we get the time complexity as a function of $k$, to be $2^{2^{O(k)}}$. This is same as the complexity of emptiness checking of an *mpda* in [2].

Consider a multi-stack pushdown system $\mathcal{M}$ with 3 stacks. Define it's transition function so that if stack 1 is empty then it has no transition, otherwise it pops a symbol from stack 1 and pushes a $b$ on stack 2 and a $c$ on stack 3. After this $\mathcal{M}$ again checks stack 1 and repeats the same sequence of actions. If $\mathcal{M}$ starts with initial configuration $(q_0, \bot.a^n, \bot, \bot)$ then it reaches $(q_0, \bot, \bot.b^n, \bot.c^n)$. This shows that $post^*$ of a regular set is not regular. We do not have any example to show that $pre^*$ of a regular set is not regular. In fact, we think that $pre^*$ of a regular set is regular for bounded phase *mpds*.

## 6 Conclusion

In this paper we have shown that parity games over bounded phase multi-stack pushdown systems can be effectively solved. The complexity of our algorithm is a tower of exponentials of the height same as the number of phases allowed. An open question is that if this complexity can be improved or is there a matching lower bound. In [7,8], winning regions in parity games over pds have been shown to be regular. The same question can also be asked for two player parity games over bounded context switching *mpds* and over bounded phase *mpds*. We think that our techniques can be used to show that winning region in these games is also regular. It will also be interesting to know if MSO theory of configuration graphs of bounded phase *mpds* is decidable.

Finally, we have also mentioned application of our results in deciding emptiness of multi-stack bounded phase $\omega-$automata. It seems interesting to study the class of languages recognized by such $\omega-$automata.

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

## References

1. Madhusudan, P., Parlato, G., La Torre, S.: A Robust Class of Context-Sensitive Languages. In: 22nd IEEE Symp. on Logic in Computer Science (LICS) Wroclaw, Poland (2007)
2. Madhusudan, P., Parlato, G., La Torre, S.: An Infinite Automaton Characterization of Double Exponential Time. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 33–48. Springer, Heidelberg (2008)
3. 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)
4. Lal, A., Reps, T.: Reducing Concurrent Analysis Under a Context Bound to Sequential Analysis. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 37–51. Springer, Heidelberg (2008)
5. Qadeer, S., Rehof, J.: Context-bounded model checking of concurrent software. In: Proceedings of the 11th International Symposium on Tools and Algorithms for the Construction and Analysis of Systems (2005)
6. Kahlon, V., Ivancic, F., Gupta, A.: Reasoning About Threads Communicating via Locks. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 505–518. Springer, Heidelberg (2005)
7. Serre, O.: Note on Winning Positions on Pushdown Games with Omega-Regular Conditions. Information Processing Letters 85(6), 285–291 (2003)
8. Cachat, T.: Uniform solution of parity games on prefix-recognizable graphs. In: Proc. INFINITY. ENTCS, vol. 68(6) (2002)
9. Walukiewicz, I.: Pushdown processes: games and model checking. Information and computation 164, 234–263 (2001)

10. 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)
11. Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: Applications to model checking. In: Proc. Concur, pp. 135–150 (1997)
12. 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)