Global Reachability in Bounded Phase 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 systems (*mpds*) have been studied recently. Given a set $\mathcal{C}$ of configurations of a *mpds* $\mathcal{M}$, let $pre^*_{\mathcal{M}}(\mathcal{C}, k)$ be the set of configurations of $\mathcal{M}$ from which $\mathcal{M}$ can reach an element of $\mathcal{C}$ in at most k phases. In this paper, we show that for any *mpds* $\mathcal{M}$, any regular set $\mathcal{C}$ of configurations of $\mathcal{M}$ and any number k, the set $pre^*_{\mathcal{M}}(\mathcal{C}, k)$, is regular. We use saturation like method to construct a non-deterministic finite multi-automaton recognizing $pre^*_{\mathcal{M}}(\mathcal{C}, k)$. Size of the automaton constructed is double exponential in k which is optimal as the worst case complexity measure.

1 Introduction

Model checking of programs with threads is an important problem that has been considered in several recent works, see [11,7,8,13]. Boolean valued programs with recursion and a fixed number of threads can be modeled as a multi-stack pushdown system (*mpds*). A *mpds* has a finite set of control states and a fixed number of stacks. The transition function of a *mpds* may (nondeterministically) do a push or a pop operation on any stack along with a possible change in control state of *mpds*. Each thread has its own stack for its procedures calls and communication among threads is through the common finite states of *mpds*.

It is well known that even a two stack *pds* can simulate universal models of computation. Therefore to get effectively checkable properties of the model, a restriction called bounded context switching was introduced on *mpds* in [11]. 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. In [11], reachability between configurations of k context switching *mpds* is shown to be decidable. In fact [11], extends the techniques of [4] to show that bounded context switching *mpds* admit effective global reachability analysis also. More precisely, [11] shows that for any *mpds* M and any regular set $\mathcal{C}$ of configurations of M, $pre^*(\mathcal{C}, k)$, the set of configurations of M from which a configuration in $\mathcal{C}$ can be reached by M in at most k context switches, is regular. Similarly $post^*(\mathcal{C}, k)$, the set of configurations to which M can reach from some configuration in $\mathcal{C}$, in at most k context switches, is also shown to be regular in [11]. Bounded context

T. Touili, B. Cook, and P. Jackson (Eds.): CAV 2010, LNCS 6174, pp. 615–628, 2010.

switching model checking has been fruitful in uncovering some errors in software and has received some attention in theory also, see [3,10].

In [7], a generalization of bounded context switching *mpds*, called bounded phase *mpds* have been considered. In a $k-$phase bounded *mpds* also a run is divided into k stages but now in one stage while *pop* operations are performed only in one stack, push operations can be performed on any stack. A stage is called a phase in this case. Bounded phase *mpds* capture a strictly larger class of systems than bounded context switching *mpds*. In [7] emptiness problem of bounded phase multi-stack automata (*mpda*) is shown to be decidable. Apart from being interesting in their own right, bounded phase *mpds* can simulate some other interesting systems also. For example, in [8] networks of finite state processes, where processes can send messages to each other via FIFO queues, are studied. There network architectures are presented where (unbounded) reachability between configurations can be reduced to bounded phase reachability. As a more theoretical result, bounded phase multi-stack pushdown transducers have been used to give an infinite automaton characterization of the complexity class of problems solvable in double exponential time, 2ETIME [9]. In [1], a little more general automata model than *mpda* has been presented and its emptiness problem is shown to be decidable.

In this paper we are interested in global model checking of reachability over bounded phase *mpds*. Let $\mathcal{C}$ be a regular set of configurations of an *mpds*. As mentioned above $pre^*(\mathcal{C}, k)$, and $post^*(\mathcal{C}, k)$ are regular if k is the number of context switches. However, if k is the number of phases of *mpds* then $post^*(\mathcal{C}, k)$, is *not* always regular even for $k = 1$. We present an example, modified from an example mentioned at the end of [7], showing this in section 2.3. This leaves the question if $pre^*(\mathcal{C}, k)$ is regular for a regular set $\mathcal{C}$. By results of [1], reachability in a k phase *mpds* can be simulated by reachability in an order-$2k$ higher order pushdown system. This suggests, by the results of [5], that $pre^*(\mathcal{C}, k)$ for a k phase *mpds* can be constructed in time $exp_{2k}(|Q|)$, where exp_n denotes tower of exponents of height n and $|Q|$ is the number of states in the *mpds*. This seems to be the only known method for constructing $pre^*(\mathcal{C}, k)$. In contrast in this paper, we show that $pre^*(\mathcal{C}, k)$ for a k phase *mpds* can be constructed in time $|Q|^{(c.l)^k}$, where c is a constant and l is the number of stacks of the *mpds*. This complexity is only double exponential in k. It is also the optimal worst case complexity bound as the emptiness problem of k-phase *mpda*, which has a lower bound of double exponential in k [9], can be easily reduced to the problem of constructing an automaton recognizing pre^* for a k phase bounded system. The basic idea of our proof is to construct an automaton recognizing pre^* for a single phase and then iterate this construction k-times to get an automaton recognizing pre^* for a k phase bounded system. The construction for single phase uses a modification of saturation technique of [4] and some ideas from [12].

Let $\mathcal{A}_{pre,k}$ be the finite automaton constructed to accept $pre^*(\mathcal{C}, k)$ for a given *mpds* M and regular set C. We also give an algorithm which from any accepting run of $\mathcal{A}_{pre,k}$ on configuration d constructs a witnessing execution sequence s of transitions of M starting at d and ending at some $e \in \mathcal{C}$.

2 Preliminaries

2.1 Multi-stack Pushdown System

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,* Γ *is the stack alphabet and* q_0 *is the initial state. The transition function* δ *is given as* $\delta = \delta_{ins} \cup \delta_{rem} \cup \delta_{exch}$*, where*

- $\delta_{exch} \subseteq Q \times \Gamma \times Q \times [1 \dots l] \times \Gamma$,
- $\delta_{ins} \subseteq Q \times \Gamma \times Q \times [1 \dots l] \times \Gamma$,
- $\delta_{rem} \subseteq Q \times \Gamma \times Q \times [1 \dots l]$

($\delta_{exch}, \delta_{ins}, \delta_{rem}$ *represent exchange, push and pop operations respectively). An* mpds *operation depends on its control state and topmost symbol of the stack on which the operation is done.*

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

- $(q, s_1, \dots, s_l) \xrightarrow{t} (q', s'_1, \dots, s'_l)$ *if* $t = (q, \gamma, q', i, \gamma') \in \delta_{exch}$, $s_i = \gamma.z_i$, $s'_i = \gamma'.z_i$ *for some* $z_i \in \Gamma^*$ *and* $s'_j = s_j$ *for* $j \neq i$, $1 \leq j \leq l$.
- $(q, s_1, \dots, s_l) \xrightarrow{t} (q', s'_1, \dots, s'_l)$ *if* $t = (q, \gamma, q', i, \gamma') \in \delta_{ins}$, $s_i = \gamma.z_i$, $s'_i = \gamma'.\gamma.z_i$ *for some* $z_i \in \Gamma^*$ *and* $s'_j = s_j$ *for* $j \neq i$, $1 \leq j \leq l$.
- $(q, s_1, \dots, s_l) \xrightarrow{t} (q', s'_1, \dots, s'_l)$ *if* $t = (q, \gamma, q', i) \in \delta_{rem}$, $s_i = \gamma.s'_i$ *and* $s'_j = s_j$ *for* $j \neq i$, $1 \leq j \leq l$.

Our stacks grow from right to left. In the above definition of *mpds,* we do not provide for bottom markers of the stacks explicitly. In fact, we allow a stack to be empty. If stack-i is empty, we represent it by ϵ (the empty string). There is no top of the stack symbol in empty stack-i, we use the convention that $\gamma_i = \epsilon$ in this case. It should be clear from the transition function definition the no *push* or *pop* operation is permitted on an empty stack. However, *push* and *pop* operations on other (non-empty) stacks may still take place.

If a bottom marker is needed, it can be taken as any symbol in Γ with some restriction on transitions involving it.

Definition 3. *A multi-step transition between configurations of mpds, on say sequence* $t_1 t_2 \dots t_n$ *of* mpds *moves,* $c \xrightarrow{t_1 t_2 \dots t_n} d$ *is defined as follows.* $c \xrightarrow{t_1 t_2 \dots t_n} d$ *iff either* $n = 0$ *and* $c = d$ *or there is a* c' *s.t.* $c \xrightarrow{t_1} c'$ *and* $c' \xrightarrow{t_2 \dots t_n} d$*. 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. *A configuration* d *of multi-stack pushdown system* (Q, Γ, l, δ) *is reachable from configuration* c *in* m*-phases if there are* $\alpha_1, \dots, \alpha_m$ *where each* α_i *is a sequence of* mpds *moves with pop moves from at most one stack and* $c \overset{\alpha_1}{\twoheadrightarrow} c_1 \overset{\alpha_2}{\twoheadrightarrow} c_2 \dots \overset{\alpha_m}{\twoheadrightarrow} c_m = d$.

Definition 5. *Let $\mathcal{M}$ be an mpds (Q, Γ, l, δ). Let C be a set of stack configurations of $\mathcal{M}$. We define the following.*

- $pre^*_{\mathcal{M}}(C) = \{c \mid \exists c' \in C[c \twoheadrightarrow c']\}$.
- $pre^*_{\mathcal{M}}(C, k) = \{c \mid \exists c' \in C[c' \textit{is reachable from } c \textit{ in at most } k \textit{ phases}]\}$

The set $pre^*_{\mathcal{M}}(C)$ of configurations may be uncomputable even for a fixed $\mathcal{M}$ and a fixed finite set C. Therefore we defined above $pre^*_{\mathcal{M}}(C, k)$. It is further useful to identify a subset of $pre^*_{\mathcal{M}}(C, 1)$, which consists of configurations of $\mathcal{M}$ from where an element of C may be reached by *pop* operations on stack-i and *push* operations on any stack of $\mathcal{M}$.

Definition 6. *Let $\mathcal{M}$ be an mpds (Q, Γ, l, δ). We define $c \longrightarrow_i c'$ iff there is a sequence $\overline{t}$ of transition of $\mathcal{M}$ in which pop operations occur only on stack-i and $c \xrightarrow{\overline{t}} c'$. Let C be a set of configurations of $\mathcal{M}$.*
We define $pre^*_i(C) = \{c \mid \exists c' \in C[c \longrightarrow_i c']\}$.

2.2 Regular Sets of MPDS Configurations

Definition 7. *Let $\mathcal{M}$ be an mpds (Q, Γ, l, δ) and let $c = (q, s_1, \ldots, s_l)$ be a configuration of $\mathcal{M}$. We define $\#(c) = \#_1 s_1 \#_2 s_2 \#_3 \ldots \#_l s_l \#_{l+1}$.*

Expression $\#(c)$ denotes a representation of contents of stacks in c as a string over alphabet $\Gamma^{\#} = \Gamma \cup \{\#_1, \ldots, \#_{l+1}\}$.

We define regular sets of *mpds* configurations using finite Multi-automata which were introduced in [4] for defining regular sets of configurations of (single stack) pushdown systems. Let $\mathcal{M}$ be an mpds (Q, Γ, l, δ). A finite (multi)automaton for recognizing configurations of $\mathcal{M}$ is given as $\mathcal{A} = (Q_{\mathcal{A}}, \Gamma^{\#}, Q, \delta, F)$, where $Q_{\mathcal{A}}$ is the set of states of $\mathcal{A}$, δ is the transition relation of $\mathcal{A}$, $Q \subseteq Q_{\mathcal{A}}$ is the set of its initial states and F is the set of final states of $\mathcal{A}$.

For any automaton $\mathcal{B}$ and a state q of it, we let $\mathcal{L}(\mathcal{B}, q)$ denote the set of strings accepted by $\mathcal{B}$ when started in state q. The set of configurations $C_{\mathcal{A}}$, accepted by multi-automaton $\mathcal{A}$ above is given as
$C_{\mathcal{A}} = \mathcal{L}(\mathcal{A}) = \{(q, s_1, s_2, \ldots, s_l) \mid \#_1 s_1 \#_2 s_2 \#_3 \ldots \#_l s_l \#_{l+1} \in \mathcal{L}(\mathcal{A}, q), q \in Q\}$.

Definition 8. *A set $\mathcal{C}$ of configurations of $\mathcal{M}$ is regular if there is a finite multi-automaton which accepts $\mathcal{C}$.*

In [11], a regular set C of *mpds* configurations is defined as a finite union of pairs of the form $(q, \prod_{1 \le i \le l} R_i)$, where $q \in Q$ and R_i's are regular sets over Γ. A configuration $c = (q, s_1, \ldots, s_l)$ belongs to $(q', \prod_{1 \le i \le l} R_i)$ iff $q = q'$ and for $1 \le i \le l$, $s_i \in R_i$.

It is not difficult to see that the notion of regularity in [11] and the one introduced in the definition 8 above using multi-automata are equivalent, i.e. they define the same class of sets of *mpds* configurations. This is stated in lemma below.

Lemma 1. *Let $\mathcal{M}$ be an mpds (Q, Γ, l, δ).*

1. *Let $C = \bigcup(q, \prod_{1 \le i \le l} R_i)$ be a set of configurations of $\mathcal{M}$, where $q \in Q$, R_i's are regular sets over Γ and the union is over finitely many pairs. We can design a multi- automaton $\mathcal{A}$ s.t. $\mathcal{L}(\mathcal{A}) = C$.*
2. *Given a multi- automaton $\mathcal{A}$ accepting a set of configurations of $\mathcal{M}$, we can write $\mathcal{L}(\mathcal{A})$ as a finite union of sets of the form $(q, \prod_{1 \le i \le l} R_i)$, where $q \in Q$ and each R_i is a regular set over Γ.*

Proof. An easy proof is given in the full version. □

It is possible to consider other notions of regularity for *mpds* configurations, for example by reading contents of all stacks simultaneously rather than sequentially. This will be similar to l-ary synchronized rational relations of [14], where l is the number of stacks in the *mpds*. However, in this paper we stick to the notion of regularity presented in definition 8 and work with the multi-automaton formulation only.

For a finite automaton A, we use notations $q \in \delta_A^*(p, w)$, $(p, w, q) \in \delta_A^*$ or $p \xrightarrow[A]{w} q$ to mean that on string w there is a run of A starting in state p and ending in state q.

2.3 An Example for $Post^*(C, 1)$

Let C be a set of configurations of a *mpds*. Analogous to $pre^*(C)$ and $pre^*(C, k)$, the sets $post^*(C)$ and $post^*(C, k)$ are defined based on the set of configurations that can be reached *from* elements of C.

- $post^*(C) = \{d \mid \exists c \in C[c \twoheadrightarrow d]\}$.
- $post^*(C, k) = \{d \mid \exists c \in C[d \text{ is reachable from } c \text{ in at most } k \text{ phases}]\}$

We now present an example, modified from [7], in which $post^*(C, 1)$ is not regular for a regular set C.

Let $M = (\{q_1, q_2, q_3\}, \{a, b, c\}, 3, \delta = \delta_{ins} \cup \delta_{rem})$ be an *mpds*, where $\delta_{ins} = \{(q_2, b, q_3, 2, b), (q_3, c, q_1, 3, c)\}$ and $\delta_{rem} = \{(q_1, a, q_2, 1)\}$.

Mpds M has three stacks, it pops an a from stack-1 and pushes a b on stack-2 and a c on stack-3. This is repeated till stack-1 becomes empty.

Let $C = \{(q_1, \#_1 a^m \#_2 b \#_3 c \#_4) \mid m \ge 0\}$. Clearly, C is a regular set. The set of configurations of $post^*(C)$ in control state q_1 is $D = \{(q_1, \#_1 a^* \#_2 b^m \#_3 c^m \#_4) \mid m \ge 1\}$. The set $post^*(C)$ can not be represented as any multi-automation $\mathcal{B}$ because $\mathcal{B}$ when started in state q_1 is expected to accept set $\{\#_1 a^* \#_2 b^m \#_3 c^m \#_4 \mid m \ge 0\}$, which is *not* a regular set.

Since M has only one phase (it pops from stack 1 only), $post^*(C) = post^*(C, 1)$ for M. Therefore $post^*(C, 1)$ is not regular for M.

3 Intuitive Idea for Regularity of $Pre_i^*(C)$

The main step in our proof of showing $pre^*(C, k)$ regular is to show that $pre_i^*(C)$ is regular. In this section we explain why we may expect that $pre_i^*(C)$ is regular.

In the next few sections we develop these ideas formally to construct a multi-automaton recognizing $pre_i^*(C)$.

Let $\mathcal{M} = (Q, \Gamma, l, \delta)$ be an *mpds* and let $\mathcal{A} = (Q_\mathcal{A}, \Gamma^\#, Q, \delta_\mathcal{A}, \mathcal{F})$ be a nondeterministic finite multi-automaton accepting a set $\mathcal{C}_\mathcal{A}$ of configurations of $\mathcal{M}$.

Consider a run of $\mathcal{M}$ in a phase where *pop* operation is allowed in stack-i only. In this phase contents of the other stacks change in a certain simple way. For example, if stack-j, $j \neq i$, is $\gamma_j.x_j$ at some point in this phase then later during this phase any symbol below γ_j (the topmost symbol) remains unchanged. The symbol γ_j also can't be popped but may change into another symbol because of the exchange move. A *push* instruction may push γ' on stack-j so that its contents become $\gamma'.\gamma_j.x_j$, now $\gamma_j.x_j$ can't change during the rest of this phase. So if at the beginning of the phase, stack-j, $j \neq i$ is $\gamma_j.x_j$ then at any time during this phase it will be $\alpha_j u_j x_j$, for some $\alpha_j \in \Gamma$ and $u_j \in \Gamma^*$.

We need u_j to decide if after the current phase the resulting configuration is in $\mathcal{C}_\mathcal{A}$. The string u_j is not needed in any other way as we are working for a single phase. For this purpose, the relevant information about u_j, is how automaton $\mathcal{A}$ acts on string u_j. This is captured in the pairs (q_j, q'_j) s.t. $q'_j \in \delta^*_\mathcal{A}(q_j, u_j)$. This information is *finitary*. This suggests an automaton, T_i whose states store, apart from a state of $\mathcal{M}$, also for each j, $j \neq i$, α_j and a pair (q_j, q'_j) corresponding to u_j. Automaton T_i will have transitions added corresponding to $\mathcal{M}$'s operations on stack-i in the same ways as saturation procedure does. In addition T_i will also have transitions corresponding to operations of $\mathcal{M}$ on stack-j, $j \neq i$, these transitions will update the triple (α_j, q_j, q'_j) and the current state of $\mathcal{M}$ stored in a state of T_i.

Using nondeterminism it suffices for T_i to keep in it's state for each j, $j \neq i$, only one pair (q_j, q'_j) of states for stack-j, rather than all such pairs for stack-j. Different runs of T_i may keep different pairs (q_j, q'_j) for stack-j, (even for same u_j), and on some run of T_i desired combination of pairs for all stacks will get guessed.

We describe the automaton T_i formally in the next section. In section 5, using T_i we complete the construction of automaton recognizing $pre_i^*(C_\mathcal{A})$.

4 The Automaton T_i

For $1 \leq i \leq l$, we define automaton T_i which is the main component in designing automaton to accept $pre_i^*(C)$. Let

- $S = \Gamma \times Q_\mathcal{A} \times Q_\mathcal{A}$
- $Q_{i1} = \{(a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_l) \mid a_j \in S,\ for\ j \neq i,\ b \in Q\}$
- $Q_{i2} = \{(a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_l) \mid a_j \in S,\ for\ j \neq i,\ b \in Q \times Q_\mathcal{A} \times Q_\mathcal{A}\}$
- $Q_i = Q_{i1} \cup Q_{i2}$

State set of T_i is Q_i. As mentioned above each a_j, $j \neq i$, keeps information (α_j, q_j, q'_j), where $q'_j \in \delta^*_\mathcal{A}(q_j, u_j)$, about contents of stack-j. T_i takes as it's input the contents of stack-i, it is designed using saturation with respect to operations of stack-i. States in Q_{i1} allow T_i to keep current state of $\mathcal{M}$ in its

i^{th} component. The states in Q_{i2} with triple (q, z_1, z_2) in their i^{th} component indicate that $\mathcal{M}$ after applying some transitions can reach a configuration c' in state q with contents of stack-i being x and $z_2 \in \delta^*_{\mathcal{A}}(z_1, x)$. The components a_j, $j \neq i$, of such a Q_{i2} state correspond to contents of stack-j in c'.

We define $T_i = (Q_i, \Gamma^{\#}, \delta_{T_i})$. The transition function δ_{T_i} is defined as $\bigcup_{j \geq 0} \delta_j$, where δ_j, $j = 0, 1, 2, \ldots$ are defined below iteratively. The sequence δ_j, $j = 0, 1, 2, \ldots$ is monotone when viewed as a relation on $Q_i \times \Gamma^{\#} \times Q_i$. This part of the construction is called saturation procedure [4]. Each triple in δ_j corresponds to a transition of $\mathcal{M}$ or $\mathcal{A}$, so we group triples in δ_j according to transition of $\mathcal{M}$ or $\mathcal{A}$, shown in boldface.

We use notation like $\overline{c} = (c_j | 1 \leq j \leq l)$, for a sequence and use $\overline{c}[x/i]$ to mean the sequence which is same as $\overline{c}$ except at index i where it is x. To keep the notation compact we write a state $(a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_l)$ as $\overline{a}[b/i]$. In the following we assume that variable r ranges over integers in $[1, l] - \{i\}$, without explicitly repeating its range in each use. We let $a_r = (\gamma_r, q_r, q'_r)$ in the following transitions.

- $\delta_0 = \emptyset$, For $h \geq 0$, $\delta_{h+1} = \delta_h \cup \{\text{triples given by the following rules}\}$.

1 $(\mathbf{q}, \gamma_{\mathbf{j}}, \mathbf{q}', \mathbf{j}, \gamma') \in \delta_{\mathbf{exch}}$, $\mathbf{j} \neq \mathbf{i}$.
$(\overline{a}[q/i], \epsilon, \overline{a}'[q'/i]) \in \delta_{h+1}$
where $a'_j = (\gamma', q_j, q'_j)$ and $a'_m = a_m$ for $m \in [1, l] - \{j\}$.

2 $(\mathbf{q}, \gamma_{\mathbf{j}}, \mathbf{q}', \mathbf{j}, \gamma') \in \delta_{\mathbf{ins}}$, $\mathbf{j} \neq \mathbf{i}$.
$(\overline{a}[q/i], \epsilon, \overline{a}'[q'/i]) \in \delta_{h+1}$
where $a'_j = (\gamma', q''_j, q'_j)$, $a'_m = a_m$ for $m \in [1, l] - \{j\}$ and $(q''_j, \gamma_j, q_j) \in \delta^*_{\mathcal{A}}$.

3 $(\mathbf{q}, \gamma_{\mathbf{i}}, \mathbf{q}', \mathbf{i}) \in \delta_{\mathbf{rem}}$
$(\overline{a}[q/i], \gamma_i, \overline{a}[q'/i]) \in \delta_{h+1}$

4 $(\mathbf{q}, \gamma_{\mathbf{i}}, \mathbf{q}', \mathbf{i}, \gamma') \in \delta_{\mathbf{exch}}$
$(\overline{a}[q/i], \gamma_i, g) \in \delta_{h+1}$ where $(\overline{a}[q'/i], \gamma', g) \in \delta^*_h$

5 $(\mathbf{q}, \gamma_{\mathbf{i}}, \mathbf{q}', \mathbf{i}, \gamma') \in \delta_{\mathbf{ins}}$
$(\overline{a}[q/i], \gamma_i, g) \in \delta_{h+1}$ where $(\overline{a}[q'/i], \gamma'\gamma_i, g) \in \delta^*_h$

6 $(\mathbf{p}, \gamma_{\mathbf{i}}, \mathbf{p}') \in \delta_{\mathcal{A}}$, $\gamma_i \in \Gamma \cup \{\epsilon\}$

(a) $(\overline{a}[q/i], \epsilon, \overline{a}[(q, p_1, p_1)/i]) \in \delta_{h+1}$ for all $p_1 \in Q_{\mathcal{A}}$.

(b) $(\overline{a}[(q, p_1, p)/i], \gamma_i, \overline{a}[(q, p_1, p')/i]) \in \delta_{h+1}$ for all $p_1 \in Q_{\mathcal{A}}$.

- $\delta_{T_i} = \bigcup_{j \geq 0} \delta_j$,

The automaton T_i simulates transitions of $\mathcal{M}$ and $\mathcal{A}$ on its states. Transitions of T_i can be divided in three groups.

The first group consists of transitions (1) and (2). These transitions are for operations on stack-j, $j \neq i$. Since, T_i does not read contents of stack-j, $j \neq i$, these transitions are ϵ transitions. Let us explain transition (2). In this transition

stack-j becomes $\gamma'\gamma_j u_j$ and we have $q''_j \xrightarrow[\mathcal{A}]{\gamma_j} q_j \xrightarrow[\mathcal{A}]{u_j} q'_j$. We use notation $\delta^*_{\mathcal{A}}$ instead of $\delta_{\mathcal{A}}$ to account for ϵ transitions in $\mathcal{A}$.

The second group consists of the transitions (3), (4) and (5). These are usual saturation operations. These relate to $\mathcal{M}$'s operations of stack-i and consume as input, a symbol of stack-i. The automaton T_i after reading a symbol from stack-i keeps track of the states which $\mathcal{M}$ can reach after this symbol has been popped. For exchange and push on stack-i this is kept track of by new transitions added iteratively, by saturation procedure. Note that the transitions added iteratively include the influence of transitions in *all* groups.

The third group consists of transitions (6a) and (6b). The automaton T_i uses an approach similar to that in [12], at any point (that is after any number of transitions of $\mathcal{M}$), T_i may choose to simulate $\mathcal{A}$ on contents of stack-i. Transition (6a) starts simulating $\mathcal{A}$ from state p on contents of stack-i in the current configuration. Transition (6b) continues the simulation of $\mathcal{A}$ in the last component of triple (q, p_1, p). Note that after applying a transition from this group, T_i can not apply transitions of any other group. Transitions of this group also participate in saturation procedure.

Note that only transitions in step (4), (5) depend on transitions present in the previous stage. Other transitions are in δ_h for all $h > 0$.

T_i does not have any initial or final state. It's purpose is not to recognize any language instead it is used as a component in the automaton to recognize $pre^*_i(\mathcal{C}_{\mathcal{A}})$. In the next section we prove some crucial properties of automaton T_i.

4.1 Properties of T_i

States of T_i abstract configurations of c. The relations between the two is described in the definitions below.

Definition 9. *Let $c = (q, \overline{\gamma v})$ be an mpds configuration, where $\overline{\gamma v} = \gamma_1 v_1, \ldots, \gamma_l v_l$. Let $e = \overline{a}[q/i]$ be a state of T_i, where $a_r = (\gamma_r, q_r, q'_r)$ for $r \in [1, l] - i$. We define $c \approx e$ (read as c is compatible with e) if $q'_r \in \delta^*_{\mathcal{A}}(q_r, v_r)$ for $r \in [1, l] - i$.*

Definition 10. *Let $c = (q, \overline{\gamma v}[w/i])$ be an mpds configuration, where $\overline{\gamma v} = \gamma_1 v_1, \ldots, \gamma_l v_l$. Let $e = \overline{a}[(q, z_1, z_2)/i]$ be a state of T_i. We define $c \approx_1 e$ (read as c is compatible with e) if $c \approx e$ and $z_2 \in \delta^*_{\mathcal{A}}(z_1, w)$.*

In relationship $c \approx e$ contents of stacks $[1, l] - \{i\}$ are abstracted by a run of $\mathcal{A}$. As for a given $v \in \Gamma^*$, there can be many pairs (q, q') s.t. $q' \in \delta^*_{\mathcal{A}}(q, v)$, there are many e for a given c s.t. $c \approx e$. Relation $c \approx e$ is independent of contents of stack-i. In $c \approx_1 e$ contents of stack-i are also abstracted by a run of $\mathcal{A}$.

The lemma below shows that transition rules of T_i capture the effect of transitions of $\mathcal{M}$ and transitions of $\mathcal{A}$ on actual configurations, in sound and complete way, on abstracted states of T_i.

Lemma 2. *Let $c = (q, \overline{\gamma v}[w/i])$ be an mpds configuration. Then the following hold.*

1. *If* $c \longrightarrow_i c'$, $c' \approx_1 e'$ *then there is an* e, $c \approx e$ *and* $(e, w, e') \in \delta^*_{T_i}$.
2. *If* $c \approx e$, $(e, w, e') \in \delta^*_{T_i}$ *where* $e' \in Q_{i2}$ *then there is a* c' *s.t.* $c \longrightarrow_i c'$ *and* $c' \approx_1 e'$.

Proof. The proof is a bit tedious but works out as expected. We use induction and case analysis in same way as in saturation proofs of [12]. Details are in full version. □

5 Regularity of $Pre^*_i(C_{\mathcal{A}})$

In this section we construct an automaton P_i, using T_i of the previous section, to accept $pre^*_i(C_{\mathcal{A}})$.

Let $\mathcal{A}$ be a multi-automaton, $\overline{\gamma} = \gamma_1 \dots \gamma_l$ and $e = \overline{a}[(q, z_1, z_2)/i]$, where $a_r = (\alpha_r, p_r, q_r)$. We know that e encodes information related to *some run* of $\mathcal{A}$, on stack contents modified during phase-i (where *pop* on stack-i only is allowed). Definition 11 below, refines the acceptance by $\mathcal{A}$ to acceptance via a run which is consistent with the information in e.

First some notation, given a run ρ of $\mathcal{A}$ on input y, if x has unique occurrence in y then we let $\rho(x)$ be the state reached in run ρ at the end of x.

Definition 11. *Let* $(q, \overline{\alpha u}[w'/i]) \approx_1 e$. *We say that* $(q, \overline{\alpha u v}[w'/i]) \in \mathcal{L}(\mathcal{A})$ ***via*** e *if there is an accepting run* ρ *of* $\mathcal{A}$ *on input* $\#(q, \overline{\alpha u v}[w'/i])$, *starting at state* q *s.t.* $\rho(\#_r \alpha_r) = p_r$, $\rho(\#_r \alpha_r u_r) = q_r$, *for* $r \neq i$, *and* $\rho(\#_i) = z_1$, $\rho(\#_i w') = z_2$.

The definition above can be seen as stating that a configuration is accepted by $\mathcal{A}$ using a run that is in some specified states at some specific points in the input.

Let e be as above, we wish to design an automaton which accepts a configuration c such that if in phase-i, c is transformed according to information in e then the resulting configuration is accepted by $\mathcal{A}$ via e. We call this multi-automaton $\mathcal{A}^{\overline{\gamma}}_e$ which works as follows. Automaton $\mathcal{A}^{\overline{\gamma}}_e$ on input $\#(p, \overline{\gamma v}[w/i])$, simulates $\mathcal{A}$ from start state q with the following modifications.

On reading $\#_j \gamma_j$, $j \neq i$, $\mathcal{A}^{\overline{\gamma}}_e$ simulates $\mathcal{A}$ on $\#_j \alpha_j$ instead of on $\#_j \gamma_j$. If after having been simulated on $\#_j \alpha_j$ state of $\mathcal{A}$ is not p_j then $\mathcal{A}^{\overline{\gamma}}_e$ aborts otherwise $\mathcal{A}^{\overline{\gamma}}_e$ changes the state of $\mathcal{A}$ in simulation to q_j and continues the simulation of $\mathcal{A}$ on the input following $\#_j \gamma_j$. (This ensures that on $\#_j \gamma_j$, $\mathcal{A}^{\overline{\gamma}}_e$ simulates $\mathcal{A}$ on all $\#_j \alpha_j u_j$, s.t. $p_j \xrightarrow[\mathcal{A}]{u_j} q_j$.)

On reading $\#_i \gamma_i$, if state of the simulated $\mathcal{A}$ is not z_1 then $\mathcal{A}^{\overline{\gamma}}_e$ aborts otherwise $\mathcal{A}^{\overline{\gamma}}_e$ changes the state of $\mathcal{A}$ to z_2 and $\mathcal{A}$ ignores the input till it sees $\#_{i+1}$. (This ensures that on $\#_i w \#_{i+1}$, $\mathcal{A}^{\overline{\gamma}}_e$ simulates $\mathcal{A}$ on all $\#_i w' \#_{i+1}$, where $z_1 \xrightarrow[\mathcal{A}]{w'} z_2$.)

Finally, $\mathcal{A}^{\overline{\gamma}}_e$ accepts if it reaches accepting state of $\mathcal{A}$ in the simulation at the end of the given input.

The following Lemma is clear from the construction of $\mathcal{A}^{\overline{\gamma}}_e$.

Lemma 3. *For any* $(q, \overline{\alpha u}[w'/i]) \approx_1 e$, $(p, \overline{\gamma v}[w/i]) \in \mathcal{L}(\mathcal{A}^{\overline{\gamma}}_e)$ *iff* $(q, \overline{\alpha u v}[w'/i]) \in \mathcal{L}(\mathcal{A})$ *via* e.

Following lemma justifies the construction of $\mathcal{A}_e^{\overline{\gamma}}$ as it gives a criteria for a configuration c to be in $pre_i^*(\mathcal{C}_\mathcal{A})$ using $\mathcal{A}_e^{\overline{\gamma}}$.

Lemma 4. *Let $\mathcal{A}$ be a multi-automaton accepting a set $\mathcal{C}_\mathcal{A}$ of configurations of $\mathcal{M}$. Then $c = (p, \overline{\gamma v}[w/i]) \in pre_i^*(\mathcal{C}_\mathcal{A})$ iff there are $e \in Q_{i1}$ and $e' \in Q_{i2}$ s.t. $(p, \overline{\gamma}[w/i]) \approx e$, $(e, w, e') \in \delta_{T_i}^*$ and $c \in L(\mathcal{A}_{e'}^{\overline{\gamma}})$.*

Proof. Proof is given in full version of this paper. □

Using the characterization in Lemma 4, we design a multi-automaton P_i to accept $pre_i^*(C)$ in the following lemma.

Lemma 5. *Let $\mathcal{A}$ be a multi-automaton accepting a set $\mathcal{C}_\mathcal{A}$ of configurations of $\mathcal{M}$. There is a multi-automaton P_i which accepts $pre_i^*(C_\mathcal{A})$.*

Proof. We design P_i to accept $pre_i^*(C)$ using the criteria in Lemma 4. Automaton P_i on input $c = (q, \overline{\gamma v}[w/i])$, non-deterministically guesses $e \in Q_{i1}$ and $e' \in Q_{i2}$ and separately verifies each of the three conditions (i) $(q, \overline{\gamma}[w/i]) \approx e$ (ii) $(e, w, e') \in \delta_{T_i}^*$ (iii) $c \in L(\mathcal{A}_{e'}^{\overline{\gamma}})$ as it scans the input.

In more detail, P_i when started from state q, guesses $e = \overline{a}[q/i]$, where $a_r = (\beta_r, p_r, q_r)$, $p_r \xrightarrow[\mathcal{A}]{\epsilon} q_r$ for $r \neq i$. This is hardwired in P_i as guessing of e and e' is hardwired in P_i. Condition (i) is now verified by an automaton which checks that for each $r \neq i$ the symbol just after $\#_r$ is β_r. Condition (ii) is verified by running automaton T_i from state e on the string enclosed between $\#_i$ and $\#_{i+1}$ and accepting if it reaches e'. Condition (iii) is verified by running automaton $\mathcal{A}_{e'}^{\overline{\gamma}}$, from state q on $\#c$.

The verification phase consists of running these three automata simultaneously and accepting if each of them accepts. □

6 Regularity of $Pre^*(C, k)$

From the previous section, for each i, $1 \leq i \leq l$, we have multi-automaton $P_i(\mathcal{A}) = (Q_i, \Gamma^\#, \delta_i, Q, F_i)$ which accepts $pre_i^*(\mathcal{C}_\mathcal{A})$. Consider a multi-automaton $P(\mathcal{A}) = ((\oplus_{i=1}^l Q_i) \oplus Q, \Gamma^\#, (\oplus_{i=1}^l \delta_i) \cup \{(q, \epsilon, q^i) | q \in Q\}, Q, \cup_{i=1}^l F_i)$, where symbol $\oplus$ stands for a disjoint union.

$P(\mathcal{A})$ is obtained by adding a new copy of states Q as initial states of $P(\mathcal{A})$ and ϵ transitions from each initial state $q \in Q$ to corresponding initial state of each multi-automaton $P_i(\mathcal{A})$. The other transitions in $P(\mathcal{A})$ are the transitions present in each $P_i(\mathcal{A})$.

In state $q \in Q$, on input $\#(c)$, $P(\mathcal{A})$ nondeterministically chooses i and simulates $P_i(\mathcal{A})$ on $\#(c)$. Thus the language accepted by $P(\mathcal{A})$ is $\bigcup_i pre_i^*(\mathcal{C}_\mathcal{A})$. That is $P(\mathcal{A})$ accepts the set of configurations of $\mathcal{M}$ from which $\mathcal{M}$ can reach an element of $\mathcal{C}_\mathcal{A}$ in a single phase.

Finally, we define $P^0 = \mathcal{A}$ and $P^{j+1}(\mathcal{A}) = P(P^j(\mathcal{A}))$, for $j \geq 0$. The automaton $P^k(\mathcal{A})$, which is defined by iterating the operator P, k-times, accepts the set of configurations of $\mathcal{M}$ from which $\mathcal{M}$ can reach an element of $\mathcal{C}_\mathcal{A}$ in at most k phases. That is $P^k(\mathcal{A})$ accepts $pre^*(C_\mathcal{A}, k)$.

6.1 Complexity

Let the number of states in multi-automaton $\mathcal{A}$ be $|Q_{\mathcal{A}}|$ and the number of states in *mpds* $\mathcal{M}$ be $|Q|$. It is clear that $|Q| \leq |Q_{\mathcal{A}}|$, as $Q \subseteq Q_{\mathcal{A}}$. A simple counting shows the following.

Number of states in T_i is $O(|\Gamma|^l.|Q_{\mathcal{A}}|^{2l+1})$.

Number of states in $\mathcal{A}^{\overline{\gamma}}_{e'}$ is $O(|Q_{\mathcal{A}}|)$.

Number of states in $P_i(\mathcal{A})$ is $O((|\Gamma|^l \cdot |Q_{\mathcal{A}}|^{2l+1})^2 \cdot (|\Gamma|^l \cdot |Q_{\mathcal{A}}|^{2l+1} \cdot |Q_{\mathcal{A}}|)) = O(|\Gamma|^{3l} \cdot |Q_{\mathcal{A}}|^{7l})$.

(The number of different e, e' pairs is $O((|\Gamma|^l.|Q_{\mathcal{A}}|^{2l+1})^2)$. For each guess of e, e' pair, $P_i(\mathcal{A})$ has a copy of T_i and $\mathcal{A}^{\overline{\gamma}}_{e'}$ to simulate these automata on the input.)

Number of states in $P(\mathcal{A})$ is $O(l.|\Gamma|^{3l}.|Q_{\mathcal{A}}|^{7l})$.

This can be taken as $O(|Q_{\mathcal{A}}|^{c.l})$, for some constant c, if $|\Gamma|$ is taken as constant or $|Q_{\mathcal{A}}| \geq |\Gamma|$.

The number of states in $P^k(\mathcal{A})$ is therefore $O(|Q_{\mathcal{A}}|^{(c.l)^k})$.

We sum all this up in the main theorem below.

Theorem 1. *Let $\mathcal{A}$ be a multi-automaton with $|Q_{\mathcal{A}}|$ many states recognizing a set $C_{\mathcal{A}}$ of configurations of a l stack mpds $\mathcal{M}$. There is a multi-automaton with $O(|Q_{\mathcal{A}}|^{(c.l)^k})$ states (where c is a constant independent of $\mathcal{A}$ and $\mathcal{M}$), which recognizes $pre^*_{\mathcal{M}}(C_{\mathcal{A}}, k)$ and can be constructed in $O(|Q_{\mathcal{A}}|^{(c.l)^k})$ time.*

Note that the automaton constructed permits incremental construction as the number of phases is increased. Suppose we have the automaton $P^k(\mathcal{A})$ to recognize $pre^*(C_{\mathcal{A}}, k)$, now to construct the automaton $P^{k+1}(\mathcal{A})$ to recognize $pre^*(C_{\mathcal{A}}, k+1)$, we do not need to the start the construction from scratch instead we may just apply operator P to $P^k(\mathcal{A})$. This feature may be useful in practice.

7 Extracting a Witness Sequence of Transitions

Given an accepting run R of $P^k(\mathcal{A})$ on $c = (q, \overline{\gamma v})$, we show how to effectively construct a sequence $\overline{t}$ of transitions of $\mathcal{M}$, s.t. $c \xrightarrow{\overline{t}} c'$ for a $c' \in \mathcal{A}$.

We begin by observing that Lemma 2 part (2) and Lemma 3 are constructive in the following sense.

Lemma 6. *Let $c = (q, \overline{\gamma v})$ be an mpds configuration and let $e = \overline{a}[q/i] \in Q_{i1}$ where $a_r = (\gamma_r, p_r, q_r)$. Let $c \approx e$. Further a run $e \xrightarrow[T_i(\mathcal{A})]{\gamma_i v_i} e'$ of $T_i(\mathcal{A})$ be given for some $e' = \overline{a'}[(q', z_1, z_2)/i] \in Q_{i2}$, where $a'_r = (\alpha_r, p'_r, q'_r)$.*

Then we can effectively construct $\overline{t}$ and c' s.t. $c \xrightarrow{\overline{t}}_i c' = (q', \overline{\alpha u v})[w/i]$. Further, we can effectively construct runs $p'_r \xrightarrow[\mathcal{A}]{u_r v_r} q'_r$ and $z_1 \xrightarrow[\mathcal{A}]{w} z_2$.

Proof. Easily follows from the proof of Lemma 2, part (2). Details are given in full version. □

Following lemma is immediate from the construction of $\mathcal{A}^{\gamma}_{e'}$ in section 5. Instead of $\mathcal{A}$, we consider an arbitrary multi-automaton $\mathcal{B}$ accepting configurations of a *mpds*.

Lemma 7. *Let* $e' = \overline{a'}[(q, z_1, z_2)/i] \in Q_{i2}$ *where* $a'_r = (\alpha_r, p'_r, q'_r)$. *Also let* $(q, \overline{\alpha u}[w/i]) \approx_1 e'$. *From an accepting run of* $\mathcal{B}^{\gamma}_{e'}$ *on* $c = (p, \overline{\gamma v})$ *and runs* $p'_r \xrightarrow[\mathcal{B}]{u_r} q'_r$ *and* $z_1 \xrightarrow[\mathcal{B}]{w} z_2$ *we can easily construct an accepting run of* $\mathcal{B}$ *on* $(q, \overline{\alpha u v})[w/i])$.

Proof. Let an accepting run of $\mathcal{B}^{\gamma}_{e'}$ on $\#(p, \overline{\gamma v})$ be given. From this we can extract a run R of $\mathcal{B}$ on c with modifications introduced by $\mathcal{B}^{\gamma}_{e'}$. In the transitions between p'_r and q'_r in R, we insert the run $p'_r \xrightarrow[\mathcal{B}]{u_r} q'_r$ and in transition between z_1 and z_2 in R, we insert the run $z_1 \xrightarrow[\mathcal{B}]{w} z_2$. This gives an accepting run of $\mathcal{B}$ on $\#((q, \overline{\alpha u v})[w/i])$. □

Equipped with the two Lemma above, we can extract the witness run as follows. Let an accepting run R of $P^k(\mathcal{A})$ on $c = (p, \overline{\gamma y})$ be given. By definition of $P^k(\mathcal{A})$, we have a run R of $P(B)$ on $c = (p, \overline{\gamma y})$, where $B = P^{k-1}(\mathcal{A})$. By examining the first transition in R, we get an i s.t. there is an accepting run R_i of $P_i(B)$ on c. By examining R_i we get

1. $e = \overline{a}[p/i]$, where $a_r = (\gamma_r, p_r, q_r)$ and $p_r \xrightarrow[B]{\epsilon} q_r$ for $r \neq i$.
2. $e' = \overline{a'}[(q, z_1, z_2)/i]$, where $a'_r = (\alpha_r, p'_r, q'_r)$.
3. Run $e \xrightarrow[T_i(B)]{\gamma_i y_i} e'$ of $T_i(B)$.
4. Accepting run of $B^{\gamma}_{e'}$ on $c = (p, \overline{\gamma y})$.

By Lemma 6 (taking $v_r = \epsilon$), we can effectively get $\overline{t_k}$, c'_k s.t. $(p, \overline{\gamma}) \xrightarrow{\overline{t_k}}_i c'_k = (q, \overline{\alpha u})[w/i]$ and $(q, \overline{\alpha u}[w/i]) \approx_1 e'$, and also runs $p'_r \xrightarrow[B]{u_r} q'_r$ and $z_1 \xrightarrow[B]{w} z_2$. It follows that $c \xrightarrow{\overline{t_k}}_i c_k = (q, \overline{\alpha u y})[w/i]$. As $c = (p, \overline{\gamma y}) \in L(B^{\overline{\gamma}}_{e'})$, by Lemma 7, we can effectively get an accepting run of B on $c_k = (q, \overline{\alpha u y}[w/i])$. As $c \xrightarrow{\overline{t_k}}_i c_k$ and $B = P^{k-1}(\mathcal{A})$ has an accepting run on c_k, we can repeat the reasoning to get $c_k \xrightarrow{\overline{t_{k-1}}}_{i'} c_{k-1}$, for some i' and an accepting run of $P^{k-2}(\mathcal{A})$ on c_{k-1}. Continuing the process, finally we get t_1 s.t. $c_2 \xrightarrow{\overline{t_1}}_j c_1$, for some j, and $c_1 \in \mathcal{A}$. The desired witness sequence is $\overline{t} = \overline{t_k}, \ldots, \overline{t_1}$. This gives us the theorem below.

Theorem 2. *Let* $\mathcal{A}$, $C_{\mathcal{A}}$ *and* $\mathcal{M}$ *be as in theorem 1. Let* $P^k(\mathcal{A})$ *be the multi-automaton constructed in the proof of theorem 1 to recognize* $pre^*_{\mathcal{M}}(C_{\mathcal{A}}, k)$. *Given an accepting run* R *of* $P^k(\mathcal{A})$ *on configuration* d, *a sequence* $\overline{t}$ *of transitions of* $\mathcal{M}$ *can be effectively constructed, s.t.* $d \xrightarrow[\mathcal{M}]{\overline{t}} d'$ *for some* $d' \in \mathcal{A}$.

Preliminary analysis of complexity of the above procedure for extracting a witness sequence, indicates that it is linear in length of the run R but triple exponential in the number of phases for fixed $\mathcal{M}$ and $\mathcal{A}$.

8 Conclusion

We have shown that the set $pre^*(C, k)$, for a k phase bounded *mpds* is regular and an automaton representation of this set can be constructed efficiently. We have also given a procedure to extract from any accepting run of this automaton on a configuration d, a witness sequence of transitions of *mpds* to show that $d \in pre^*(C, k)$. Note that these results also apply to systems which can be simulated by phase bounded *mpds*. As an example, these results apply of message passing systems in [8]. Representation of configurations of these systems involves representing contents of message queues between processes, each such queue can be represented by two stacks as in the simulation of [8].

We note that the construction of pre^* also gives a new proof for checking emptiness of a bounded phase *mpda*. Let $M = (Q, \Sigma, \Gamma, l, q_0, \delta, q_f)$ be a l stack *mpda*, where q_0 is the initial state, acceptance is by final state q_f and other symbols have the usual meaning. We obtain a *mpds* $M1$ from M by erasing input symbols from transitions in δ. Let $L(M, k)$ be the language of words recognized by M in k-phases. Now, $L(M, k) = \emptyset$ iff $I \in pre^*_{M1}(C_f, k)$, where $C_f = (q_f, \#_1\Gamma^*\#_2 \ldots \#_l\Gamma^*\#_{l+1})$ is the set of final configurations of M and $I = (q_0, \#_1\#_2 \ldots \#_l\#_{l+1})$ is the initial configuration M. Therefore emptiness of $L(M, k)$ can be determined by checking membership of I in finite multi-automaton for $pre^*_{M1}(C_f, k)$. This algorithm is very different from the other known methods [7,13], for checking emptiness of a *mpda*.

It may be interesting to see if results similar to ours can also be shown for a little more general model, the multi-pushdown systems of [1], where stacks are ordered and a pop operation is always on the first non-empty stack. There is no bound on the number of phases in that system.

Another natural question is to extend these results to the setting of two players. The present approach, if extended successfully, will give a effective history free strategy in two player reachability games over bounded phase *mpds* whereas techniques of [13] can give only a strategy computable by a multi-stack pushdown automaton.

Recently, a saturation based proof has been given in [6] for constructing winning regions in parity games over a single stack pushdown system. It may be natural to see if method of [6] combined with ideas of this paper can give a saturation based construction for representation of winning regions in parity games over bounded phase *mpds*.

While the worst case complexity of our construction is optimal, it may be possible to improve this construction so that it works more efficiently on some inputs. In this direction, it will be interesting to see if automata constructions of sections 4,5 and 6 can be combined into a single construction where we start with a small number of states and new states are added only whenever they are needed.

Another way to control size of the automaton constructed may be to consider approximations of bounded phase reachability. The construction for a single phase is single exponential. It may be combined with usual bounded context switching construction. That is, for the first few iterations we do the phase switching construction and then on the automaton so obtained we do the usual

construction for bounded context switching. This will keep the double exponential growth in check. For improving the complexity in a single phase we may restrict push in a single phase to some small number of stacks rather than allowing it on all stacks. Size of the automaton constructed for $pre_i^*(C)$ is exponential in the number of stacks on which push operations are allowed in phase-i.

Acknowledgments

Financial support for this work was provided by Research I Foundation.

References

1. Atig, M.F., Bollig, B., Habermehl, P.: Emptiness of Multi-pushdown Automata Is 2ETIME-Complete. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 121–133. Springer, Heidelberg (2008)
2. Atig, M.F., Bouajjani, A., Qadeer, S.: Context-Bounded Analysis for Concurrent Programs with Dynamic Creation of Threads. In: TACAS 2009. LNCS, vol. 5505, pp. 107–123. Springer, Heidelberg (2009)
3. Lal, A., Reps, T.W.: Reducing Concurrent Analysis under a Context Bound to Sequential Analysis. Formal Methods in System Design 35(1), 73–97 (2009)
4. Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: Applications to model checking. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 135–150. Springer, Heidelberg (1997)
5. Hague, M., Ong, C.-H.L.: Symbolic Backwards-Reachability Analysis for Higher-Order Pushdown Systems. Logical Methods in Computer Science 4 (2008)
6. Hague, M., Ong, C.-H.L.: Winning Regions of Pushdown Parity Games: A Saturation Method. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 384–398. Springer, Heidelberg (2009)
7. 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)
8. 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)
9. 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)
10. La Torre, S., Madhusudan, P., Parlato, G.: Reducing Context-Bounded Concurrent Reachability to Sequential Reachability. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 477–492. Springer, Heidelberg (2009)
11. 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)
12. Seth, A.: An Alternative Construction in Symbolic Reachability Analysis of Second Order Pushdown Systems. Int. J. Found. Comput. Sci. (IJFCS) 19(4), 983–998 (2008)
13. 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)
14. Thomas, W.: A short introduction to infinite automata. In: Kuich, W., Rozenberg, G., Salomaa, A. (eds.) DLT 2001. LNCS, vol. 2295, pp. 130–144. Springer, Heidelberg (2002)