Analysis of Automata-theoretic models of Concurrent Recursive Programs

Thesis submitted

in partial fulfilment of the

Degree of Doctor of Philosophy (Ph.D)

by

Prakash Saivasan

Chennai Mathematical Institute

Declaration

The work in this thesis is based on research carried out by me under the supervision and guidance of Prof. K. Narayan Kumar. No part of this thesis has been submitted elsewhere for any other degree of qualification.

Prakash Saivasan
Chennai Mathematical Insitute
Plot H1, SIPCOT IT Park
Siruseri, Kelambakkam
Chennai, India
PIN-603103

Certification

This is to certify that the thesis entitled "Analysis of Automata-theoretic models of Concurrent Recursive Programs " submitted by Mr. Prakash Saivasan is a bona fide record of the research work carried out by him under my supervision and guidance. The content of the thesis in full or parts have not been submitted to any other institute or university for the award of any degree or diploma.

K. Narayan Kumar

Thesis Supervisor

Chennai Mathematical Institute
Plot H1, SIPCOT IT Park
Siruseri, Kelambakkam
Chennai, India
PIN-603103

This work is dedicated to my late grandfather Mr. V. Gopalaswamy

Acknowledgements

I would firstly like to profusely thank my supervisor K. Narayan Kumar. I was indeed very lucky to have a guide as patient and friendly as him. I would also like to thank Mohammed Faouzi Atig who was like a second supervisor to me. I throughly enjoyed working with both of them and it was from them that I have learnt much of what I know. I thank them both for all the support and encouragement that they showed all along. I would also specifically like to thank Ahmed Bouajjani for providing the opportunity to work with him and for all the stimulating research discussions we had. It was indeed a great honour for me to have worked with him. I would also like to thank Paul Gastin and Parosh Abdulla for the interesting discussions we had when I visited them and also for supporting my various research visits. I hope to continue my association and collaboration with all of them for a long time to come.

I would like to thank Madhavan, Suresh, Samir, Jam and KV for teaching me the fundamentals. I would like to thank Sripathy and the administrative staff at CMI for making life at CMI very comfortable. I would also like to thank TCS for supporting me with a scholarship.

I would like to thank my friends Chary, Bakshi, Anbu, Shraddha, Geetha, Pabitra, Santosh and numerous others with whom I have many pleasant memories to share. I would also like to thank Aiswarya, Prateek, Muthu and Sreejith with whom I not only had good times but also the pleasure of discussing computer science. I would also like to thank Kumar and Rajeshwari for the delicious dinners and many memorable moments.

I would like to thank my ex-collegues Gopi and Prashanth from Hewlett Packard for all the support and encouragement. I would also like to thank few of my teachers Veda Mohan, Pramila and R S Milton for inspiring me in their own way.

Lastly I would like to thank my parents, my sister and my grandparents for supporting, encouraging and standing by me always. I would also like to specially express my adoration for my niece Samhita so that she can read about it when she is all grown up.

Contents

Chapter 1

Introduction

In the digital age of internet, mobile computing and cloud computing, the need for developing and quickly deploying complex communicating programs have become an order of the day. With increasing dependency on automated medical equipments, aeroplanes, automobile and financial transactions, the need to check for correctness of programs (also referred to as systems) has gained significance. There are various interesting properties against which one might want to verify such systems. As an example, consider a train that has automated opening and closing of its doors. One may wish to ensure that the door never opens when the train is moving or that the door only opens on the side where a platform is available.

The most extensively used method to ensure correctness of programs is testing. However, while a cleverly constructed set of tests may expose bugs, one cannot obtain a correctness guarantee through this technique. Furthermore testing often fails to find bugs in the concurrent setting where a bug may manifest itself only in the rarest of rare executions (Heisenbugs [74]). *Formal verification* is an alternative technique which establishes the correctness of a program with respect to specifications in a mathematically rigorous manner.

Model checking is a formal verification technique that algorithmically explores all possible system behaviours to check if a given specification holds [53, 57]. The input to the model-checking problem is usually a mathematical model of the system (there are techniques that work directly on the code as well, for e.g. [29, 35, 117]) along with a property expressed in a suitable logic. Examples of such logic are LTL [123] used for specifying properties that each run of the program should satisfy, and CTL [108] for reasoning about the structure of the entire collection of runs of the program.

The models used in model checking can either be finite or infinite. While an accurate modelling of software system usually requires infinite models, hardware systems typically are finite state. One very simple and interesting property is whether a particular state or a location in the program can be reached. The problem of checking such a property is called the reachability problem. Safety properties which say that something bad does not happen can be reduced to the reachability problem. On the other hand, liveness properties which say that something good will eventually happen, require examining infinite runs. An important problem in this setting is the *repeated reachability* problem, which asks if a particular state or a program point can be visited infinitely often. A solution to this is often the key to solving

the model checking problem over infinite behaviours.

A plethora of successful tools for verification of finite state systems have been built e.g. SPIN [82], SMV model checkers[115]. For finite state systems, checking properties of finite behaviours can be reduced to elementary graph theoretic properties while that of infinite behaviours frequently relies on results from the theory of finite state automata over infinite words (and other structures).

Infinite state systems arising from programs are Turing powerful and hence the verification problem is undecidable. The infiniteness arises for a variety of reasons. For e.g.

1. Variables taking values from unbounded or infinite data domain.
2. Recursion: Programs in which procedures can be called recursively can potentially have an unbounded call stack.
3. Heap: Another source of infiniteness that can occur in a program is due to data structures that use dynamically allocated store. See for eg. [5, 113, 127, 128]
4. Programs may deal with real time and continuously evolving variables. See for eg. [10, 11, 120].

In the recent years several techniques have been proposed to circumvent this undecidability, by either considering restricted subclasses of systems (under-approximation) or by relaxing the possible behaviours of the systems (over-approximations). The results in this thesis are along these lines.

Concurrency and communication add a different dimension to the problem. Consider a system with a number of finite state components communicating through shared memory or rendezvous. Even though the composed system is finite state, the number of states is exponential in the number of components. This is called the state-explosion problem and poses the main challenge in the verification of finite state systems. This problem has been addressed through a variety of techniques such as symbolic model checking [46, 115], Partial order techniques [145] and so on. When the components are infinite state the verification problem becomes even undecidable. For example, two recursive programs communicating with each other through a shared memory or by hand shake can simulate a Turing machine. Another example, is the case of (even finite state) systems communicating via FIFO channels. These are Turing powerful as the FIFO channels can easily simulate a tape.

However there are some positive results too. For instance, in the setting of finite state systems communicating via channels that are lossy the reachability problem becomes decidable [7, 8]. The technique used to prove this is based on the theory of well-quasi orders. This technique has been effectively used in developing analysis techniques for a wide variety of systems [1, 9, 68, 70, 129]. Another general technique used to extend verification to beyond finite state systems is that of regular model checking [44, 90, 144]. This technique consists of representing collections of configurations of the system in a finite manner and manipulating these represention to reason about the system. Examples of such representations includes finite state automata, Presburger formulas, semilinear sets and so on. Such techniques have been used to solve verification problems in pushdown systems [41], processes communicating via FIFO channels [6, 42] and parametrised programs [114]. For a detailed survey on regular model checking, we refer the readers to [2].

Let us take a closer look at the infiniteness caused by recursion. Notice that the state space in this case remains infinite even if all variables are drawn from a finite data domain. When all the variables are from a finite data domain the only source of infiniteness is through recursion. Thus, it has a structure similar to that of a pushdown system and thus can be modelled as such systems. The theory of pushdown systems is well studied. For instance the reachability problem, the repeated reachability problems and model checking LTL and CTL formulas over pushdown systems [41, 83, 143] are all decidable. There are also many tools for efficiently model checking pushdown systems for e.g. Bebop, Moped [30, 66].

All the results in this thesis are concerned with formal models of programs that consist of an a priori bounded number of threads or processes that communicate through a shared memory. These threads/processes may be recursive and hence are themselves of infinite state space. The natural formal model of the entire system is that of an multi-pushdown system. Informally, a multi-pushdown system is like a pushdown system except that it is equipped with multiple stacks. The state of the multi-pushdown system incorporates information about the local states of each of the component threads as well as the contents of the shared memory. There is one stack to model the call stack of each thread. As already mentioned, even simple problems such as whether a state is reachable, are undecidable for such systems. Thus, there is no hope of a complete solution. One idea used to circumvent this problem is under-approximation. The idea is to identify a subset of behaviours and restrict the verification only to this subset. An under-approximation is interesting only if the verification problem when restricted to this subset is decidable and in addition the subset covers interesting behaviours. There are various under-approximations that are well studied in literature.

Bounded-context analysis was introduced by [125]. A *context-switch* occurs when the automaton switches from accessing one stack to another (or equivalenty when scheduler switches from one thread to another). Placing an a-priori bound on the number of context-switches along any run results in decidability of reachability and host of other verification problems. Such an under approximation technique has proven to be very useful in finding bugs in real time [118]. It was shown in [95, 105], that checking some of linear time properties under context-bounded setting can be reduced to checking them on a sequential setting. This lead to using plethora of already available tools for model checking concurrent systems. Bounded-context technique has also been successfully used in areas outside of multi-pushdown systems. In [21], a system where multi threaded recursive programs with ability to dynamically fork new threads were considered. In this paper [21], a variant of bounded-context restriction was considered. They analysed behaviours of such system in which, for every process the number of contexts in which it is involved is bounded. In [94], context bounding was used to obtain a decidable underapproximation of concurrent processes communicating via unbounded FIFO channel.

Bounded-phase analysis, which is a generalisation of context switch was introduced in [97]. A *phase* can be thought of as a sequence of moves of a multi-pushdown system in which all pop operations are restricted to a single stack. Now placing an a priori bound on the number of phases along any run results in decidability for a host of verification problems. In [45], *Ordered multi-pushdown* systems were introduced as generalisation of context free

grammar and further studied as model for concurrent programs in [16, 14]. In an ordered multi-pushdown system (OMPDS), there is an inherent ordering on the stacks and in such systems, only executions involving pop operations from least non-empty stacks are allowed. In [18, 101, 102], *Bounded-scope* restriction was studied. In bounded-scope execution, an a priori bound is placed on the number of times a stack can context-switch before which it has to become empty.

In this thesis, we extend the work on some of these restrictions: we propose an algorithm for model checking infinite runs under the bounded-scope restriction. We also propose a new construction to solve the decidability of parity games under the bounded-phase restriction and establish a matching lower bound. We propose a new restriction called adjacent ordered restriction and study the model checking problems under this restriction and give some applications of this model. We also study the analysis of multi-pushdown systems under accelerations i.e. the effect of executing certain sequences of transitions (for e.g. loops) on the configurations.

The incorporation of both the local states of the components as well as the shared memory in the state of an multi-pushdown system means that the communication between the different components is not explicitly represented. This has some disadvantages, for instance, consider the degenerate case, where there are just two threads that do not communicate at all. One should be able to analyse this system, yet they fail to be in any of the classes mentioned above. We study a more fine grained model where each thread is represented as a separate pushdown system and all the communication via the shared memory is explicitly recorded. Even here the two pushdowns can simulate a Turing machine easily. In this setting, we propose a new restriction called *stage bounding* and study the decidability of model checking under this restriction. This work relies on the ability to compute certain regular abstractions of pushdown systems and leads us to one other contribution of this thesis: efficient algorithms for computing abstractions of subclasses of pushdown systems.

We now summarise the contributions of this thesis.

Linear time model checking under bounded scope

Recall that in a bounded-scope execution, an a priori bound is placed on the number of times a stack can context-switch before which it has to become empty. Reachability problem for multi-pushdown systems operating under bounded-scope restriction was shown to be PSPACE-COMPLETE [103]. We first extend the definition of bounded-scope restriction to infinite executions and prove the following results.

- We show how to obtain an exponential procedure for solving the repeated reachability problem. The basic idea of our reduction is to reason in a compositional way about thread interfaces corresponding to the states that are visible at context-switch points. We show that, when all threads are active infinitely often, the interface of each thread can be defined by a finite-state automaton, and then our problem can be reduced to a repeated state reachability problem in the composition of these interface automata. In general, to capture also the case where after some point all threads except one may be stopped, we show that it suffices to analyse infinite runs of a suitable pushdown system. This gives us an EXPTIME

procedure.
- We use the algorithm for repeated reachability to provide an EXPTIME procedure for model checking an LTL formula on bounded-scope computations of a multi-pushdown system. Lower bound immediately follows from the fact that model checking LTL formulas against pushdown system is EXPTIME-COMPLETE.
- We also provide a simple proof of PSPACE hardness for reachability, by reducing the emptiness on n-finite state automata to reachability on multi-pushdown system with bounded-scope restriction.

This is a joint work with M. F. Atig, A. Bouajjani, K. Narayan Kumar and was published in the proceedings of ATVA 2012 [18].

Adjacent ordered multi-pushdown systems

We introduce a variant of multi-pushdown system called the *adjacent ordered multi-pushdown systems*. The restriction imposed by adjacent ordered multi-pushdown system similar to that in ordered multi-pushdown system, in the sense that it allows pop operations to happen on the least non-empty stack. Further it allows push operations only on least non-empty stack or onto stacks immediately adjacent to it. In this model, infinite behaviours may involve infinitely many contexts involving more than one stack as in the case of bounded-scope or ordered multi-pushdown system. This model can also transfer the contents of one stack to another (adjacent) stack. Such a transfer is possible in OMPDS and bounded-phase restrictions but not in the others.

- We show that reachability an adjacent ordered multi-pushdown system is EXPTIME-COMPLETE. This is significantly better than 2ETIME complexity required for solving reachability on multi-pushdown with bounded-phase restriction and ordered-multi pushdown systems.
- We also show that the repeated reachability problem is EXPTIME-COMPLETE. Using the algorithm for repeated reachability, we show how to obtain a procedure for model checking LTL formulas.
- We also show some applications that illustrate the power of this model.
  - We show that reachability in recursive programs communicating via queues, whose connection topology is a forest, can be reduced to reachability on adjacent ordered multi-pushdown system.
  - We also show how to obtain a procedure for solving bounded-phase reachability of a multi-pushdown system, by showing an exponential time reduction to reachability on adjacent ordered multi-pushdown system.

This is a joint work with M. F. Atig and K. Narayan Kumar and was published in the proceedings of DLT 2013 [25]. An extended version was selected and published in a special issue of IJFCS [26]

Acceleration

In the *global model checking problem* the aim is to compute from (a representation of) the set of initial configurations (I) (a representation of) the set of configurations reachable from I (denoted $post^*(I)$). Note that our description of global model-checking does not require that the representations of the initial set I and the reachable set $post^*(I)$ be the same. However, if both sets use the same description, then we say that the representation is stable. Stability is an useful property as it permits us to compose (and hence iterate finitely) the algorithm.

A well known technique used in the verification of infinite state systems is that of loop accelerations. It is similar in spirit to global model checking but with different applications. The idea is to consider a loop of transitions (a finite sequence of transitions that lead from a control state back to the same control state). The aim is to determine the effect of iterating the loop. That is, to effectively construct a representation of the set of configurations that may be reached by valid iterations of the loop. Loop accelerations turns out to be very useful (e.g., [13, 31, 32, 33, 37, 38, 39, 43, 68, 69, 88, 89, 109, 110]) in the analysis of a variety of infinite state systems.

We propose to use accelerations as an under-approximation technique in the verification of MPDSs. We take this further by proposing a technique that composes the iterations of such loops with context bounded runs to obtain a new decidable under-approximation for MPDSs. Observe that there is no bound on the number of context switches under loop iterations while a context bounded run permits unrestricted recursive behaviours, not permitted by loop iterations, thus complementing each other.

- We showing that both regular sets as well as rational sets of configurations are stable w.r.t. bounded-context acceleration.
- We show that under iterations of a loop (a finite sequence of transitions that lead from a control state back to the same control state), the acceleration of a regular set of transitions is always rational while that of a rational set need not be rational.
- We then propose a new representation for configurations called n-CSRE inspired by the CQDDs [43] and the class of bounded semilinear languages [49]. Then, we show that n-CSREs are indeed stable w.r.t iteration of loops. This result also has the pleasant feature that the construction is in polynomial time. However, n-CSREs are not stable w.r.t bounded-context executions.
- We then introduce a joint generalization of both loop iterations and bounded-context executions called bounded context-switch sets. We show that the class of languages defined by n-dimensional constrained automata (the most general class considered here and a n-dimensional version of Parikh automata) is stable w.r.t accelerations via bounded context-switch sets. Since membership is decidable for this class, we obtain a decidability of reachability under this generous class of behaviours.

This is joint work with M. F. Atig and K. Narayan Kumar and will appear in the proceedings of TACAS 2016.

Systems communicating explicitely via shared memory

We adopt a formal model that consists of a network of processes with a shared store ranging over a finite domain. Each of these processes can be a pushdown system, an one-counter system or simply a finite-state system. Each of these processes may perform reads and writes on the shared store. In [64, 96], the problem of parametrised reachability over such models were considered and was shown to be decidable. The parametrised reachability problem asks whether there is a number k such that k identical pushdown systems can co-operate to reach a given control state. Interestingly, the problem is undecidable for a fixed set of processes. We study the reachability on such system. We show that two 1-counter systems sharing only one bit memory are able to simulate any 2-counter machine.

We then restrict the way information flows through the shared memory. The idea we consider is the following: For each computation, consider a decomposition into what we call *stages*, where in each stage only one process is unrestricted while all the others are only allowed to read. Then, we only consider computations up to some fixed bound on the number of stages. Notice that this notion of bounding called the *stage-bounding*, does not restrict the way stacks and counters are accessed. It is rather imposing that writes by different processes to the shared memory cannot interleave in an unbounded manner (while reads are allowed to interleave unboundedly with any kind of operations from any process). The notion stage-bounding is somehow inspired by the notion of context-bounding. However, it is clear that stage-bounding is strictly more general than context-bounding in term of behavior coverage. This is due to the fact that operations (reads and writes) by different processes can alternate unboundedly within one single stage.

- For networks of finite-state systems, we prove that the stage-bounded analysis is NP-complete (while the unbounded analysis is PSPACE-complete as mentioned earlier). So, stage-bounded analysis in this case has the same complexity as context-bounded analysis, while it offers more coverage.
- We show that for systems with precisely two pushdown systems the complexity of stage-bounded analysis is (at least) non-primitive recursive. The decidability in this case is actually still an open problem.
- We prove that for two pushdown systems and one 1-counter system the state reachability problem under stage-bounding is undecidable.
- We prove that for networks with at most one pushdown system and any number of 1-counter systems, stage-bounded analysis is decidable, and we show that it is in NEXPTIME, while it is PSPACE-hard. We establish this decidability result by a non-trivial reduction to the state reachability problem for pushdown systems with reversal-bounded counters (i.e., counters where the number of ascending and descending phases is bounded) [84]. Such a reduction uses the ability to compute downard/upward closures of languages of pushdown systems and one counter systems.

This is a joint work with M. F. Atig, A. Bouajjani, and K. Narayan Kumar and published in proceedings of FSTTCS 2014 [20].

Regular abstractions of one counter automata

We then look at closures on languages of counter automata. A very well known result, the Higman's Lemma , states that any upward closed language has only finite number of minimal elements under the subword relation. As an easy consequence we have that every upward closed language is regular and consequently every downward closed language is regular as well. The downward and upward closures of context free languages are effectively regular and the minimal size of finite state automatas recognising these closures are exponential in size.

- We show that for the counter systems which are subclass of pushdown systems, the upward and downward closures can be computed in POLYNOMIAL-TIME and the resulting automata are POLYNOMIAL in size. The construction for the downward closure is quite involved.

Another abstraction is that of parikh images. The parikh image of a word is a vector that assigns to each letter a natural number giving the number of times it occurs in the word. A parikh image of a language is the collection of the parikh image of the words in the language. A famous theorem of R. Parikh shows that for every context-free language there is a regular language with the same Parikh Image. Parikh images of context-free/regular languages can also be represented using *existential presburger formulas*. Parikh images have been used as an important regular abstraction in the model checking of infinite state systems [23, 60, 130].

It is known that for context-free languages the size of the smallest NFA that is Parikh equivalent may be exponential in the size of the PDA or CFG describing the given language.

- Every one counter automaton has a Parikh equivalent NFA which is subexponential in size. The proof of this result is involved and proceeds by showing a reduction first to Parikh images of reversal bounded one counter automata and then showing that these can be converted to sub-exponential sized Parikh equivlanet NFA.

This is a joint work with M. F. Atig and K. Narayan Kumar and is part of the paper [67]. The paper [67] also includes results proved by D. Chistikov, P. Hofman and G. Zetzsche.

Parity games on bounded-phase multi-pushdown systems

We then consider the problem of solving parity games over a multi-pushdown systems with bounded-phase restriction. The problem of solving parity games on multi-pushdown systems was first studied by A. Seth in [133]. He showed how to obtain a NON-ELEMENTARY decision procedure to solve the problem.

- We provide a simple inductive construction to solve this problem. Our procedure is also has non-elementary complexity.
- We also provide a non-elementary lower bound to the problem by reducing the satisfiability of formulas in first order logic with order ($FO(<)$) over natural numbers to this problem. This answers a question posed by Anil Seth.

This is a joint work with M. F. Atig, A. Bouajjani and K. Narayan Kumar.

Chapter 2

Preliminaries

In this section, we fix some basic definitions and notations that will be used in the entire thesis. We assume here that the reader is familiar with language and automata theory in general.

Notations Let $\mathbb{N}$ denote the non-negative integers. For every $i, j \in \mathbb{N}$ such that $i \leq j$, we use $[i \dots j]$ to denote the set $\{k \in \mathbb{N} \mid i \leq k \leq j\}$.

Let Σ be a finite alphabet. We denote by Σ^* (resp. Σ^ω) the set of all finite (resp. infinite) words over Σ, and by ϵ the empty word. Let u be a word over Σ. We use u^R to denote the reverse of the word u. The length of u is denoted by $|u|$; we assume that $|\epsilon| = 0$ and $|u| = \omega$ if $u \in \Sigma^\omega$. For every $j \in [1 \dots |u|]$, we use $u[j]$ to denote the j^{th} letter of u and $u[i \dots j]$ to denote the sequence starting at i and ending at j. Given any word $w \in \Sigma^*$, we say $u =_{\text{suffix}} w$ iff u is a suffix of w.

Let S be a set of (possibly infinite) words over the alphabet Σ and let $w \in \Sigma^*$ be a word. We define $w.S = \{w.u \mid u \in S\}$. We define the *shuffle* over two words inductively as $\texttt{Shuffle}(\epsilon, w) = \texttt{Shuffle}(w, \epsilon) = \{w\}$ and $\texttt{Shuffle}(a.u', b.v') = a.(\texttt{Shuffle}(u', b.v') \cup b.(\texttt{Shuffle}(a.u', v')$. Given two sets (possibly infinite) of words S_1 and S_2 (over Σ), we define shuffle over these sets as $\texttt{Shuffle}(S_1, S_2) = \bigcup_{u \in S_1, v \in S_2} \texttt{Shuffle}(u, v)$. The `Shuffle` operator for multiple sets can be extended analogously.

We say that a word w over the alphabet Σ is a subword of a word w' if $w = a_1 \dots a_n$ and there are $x_i \in \Sigma^*$, $1 \leq i \leq n+1$ such that if $w' = x_1 a_1 x_2 a_2 \dots x_n a_n x_{n+1}$. We write $w \preceq w'$ to indicate this. It is easy to check that the subword relation is a partial order.

The downward closure of a language $L \subseteq \Sigma^*$ is the language $L{\downarrow} = \{w \mid w \preceq w', \text{for some } w' \in L\}$. Similarly the upward closure of a language $L \subseteq \Sigma^*$ is the language $L{\uparrow} = \{w \mid \exists w' \in L.\ w' \preceq w\}$. A language L is upward closed if $L = L{\uparrow}$ and it is downward closed if $L = L{\downarrow}$. Clearly a language L is upward closed iff its complement $\overline{L}$ is downward closed.

For any alphabet $\Sigma = \{a_1, \cdots, a_n\}$, given a word $w \in \Sigma^*$, the parikh image of w denoted Parikh(w) is a vector $v \in \mathbb{N}^{|\Sigma|}$, where $v = (|w{\downarrow_{a_1}}|, |w{\downarrow_{a_2}}|, \cdots, |w{\downarrow_{a_n}}|)$ is a vector that counts the number of each occurrences of letters from Σ in w. The parikh image of a language Parikh$(L) = \{\text{Parikh}(w) \mid w \in L\}$

Finite-State Automata A *finite-state automaton* is a tuple $A = (Q, \Sigma, \Delta, q_0, F)$ where: (1) Q is the finite non-empty set of states, (2) Σ is the input alphabet, (3) $\Delta \subseteq (Q \times (\Sigma \cup \{\epsilon\}) \times Q)$ is the transition relation, (4) $q_0 \in Q$ is the set of initial states, and (5) $F \subseteq Q$ is the set of final states. The language of finite words accepted (or recognized) by A is denoted by $L(A)$. We may also interpret the set F as a Büchi acceptance condition, and we denote by $L^\omega(A)$ the language of infinite words accepted by A. The size of A is defined by $|A| = (|Q|+|\Sigma|+|\Delta|)$. For any transition $\tau = (q, a, q') \in \Delta$, we let $\Sigma(\tau) = a$, i.e. the input letter of the transition. Given a sequence of transition $T = \tau_1.\tau_2 \cdots \tau_n$, we let $\Sigma(T) = \Sigma(\tau_1).\Sigma(\tau_2).....\Sigma(\tau_n)$

Pushdown-automata A *pushdown automata (PDA)* is a tuple $A = (Q, \Gamma, \Sigma, \delta, s, F)$ where Q is the set of states, Γ is the stack alphabet with a special symbol $\perp$ to facilitate empty test on the stack, Σ is the tape alphabet, $s \in Q$ is the initial state, $F \subseteq Q$ is set of final states and δ is the transition relation. The transition set δ is a subset of $Q \times \mathbf{Op} \times \Sigma_\epsilon \times Q$ with $\mathbf{Op} = \bigcup_{a \in \Gamma \setminus \{\perp\}} \{\mathbf{Push}(a) \cup \mathbf{Pop}(a)\} \cup \{\mathbf{Zero}, \mathbf{Int}\}$

The configuration of PDA A is a pair (q, γ) with $q \in Q$ and $\gamma \in (\Gamma \setminus \{\perp\})^* \perp$. The *initial configuration* is the pair $c_{init} = (s, \perp)$. Given any configuration $c = (q, \gamma)$, we will use $State(c) = q$ and $Stack(c) = \gamma$ to retrieve the state and stack part of the configuration. The transition relation $\xrightarrow{a}_A$, (or $\xrightarrow{\tau}$ when we are interested in the transition used) , $a \in \Sigma$, on the set of configurations is defined as follows:

1. $(q, \alpha\gamma) \xrightarrow{a}_A (q', \gamma)$ if $\tau = (q, \mathbf{Pop}(\alpha), a, q') \in \delta$. Pop move.
2. $(q, \gamma) \xrightarrow{a}_A (q', \beta\gamma)$ if $\tau = (q, \mathbf{Push}(\beta), a, q') \in \delta$. Push move.
3. $(q, \gamma) \xrightarrow{a}_A (q', \gamma)$ if $\tau = (q, \mathbf{Int}, a, q') \in \delta$. Internal move.
4. $(q, \perp) \xrightarrow{a}_A (q', \perp)$ if $\tau = (q, \mathbf{Zero}, a, q') \in \delta$. Emptiness test.

We will use $\rightarrow^*$ to denote the reflexive and transitive closure of $\rightarrow$. Given a set of configurations C, we use $L(A, C)$ to denote the set of words w such that $(s, \perp) \xrightarrow{w}{}^* c$, for some $c \in C$. Given two configurations c_1, c_2, we use $L(A, c_1, c_2)$ to denote the set of words w such that $c_1 \xrightarrow{w}{}^* c_2$. We will use $L(A)$ to mean $\bigcup_{f \in F} L(A, (f, \perp))$.

An infinite computation is said to satisfy Büchi acceptance condition if it visits a state in F infinitely often. We will use $c \xrightarrow{w}_\omega \ldots$ to denote the existence of an infinite run starting at c, which generates w, i.e. a computation of the form $c \xrightarrow{a_1} c_1 \xrightarrow{a_2} \cdots$, where $w = a_1 a_2 \cdots$. We will use $L^\omega(A)$ to denote set of all infinite words generated by infinite computations satisfying Büchi condition i.e. $L^\omega(A) = \{w \mid \exists \pi = c_0 \xrightarrow{w}_\omega \wedge \exists^\infty i \in \mathbb{N}, State(\pi(i)) \in F\}$

When size of stack alphabet $|\Gamma| = 1$, we will refer to pushdown automata as a counter automata, further we will simply refer to $\mathbf{Push}(a), \mathbf{Pop}(a), a \in \Gamma$ as $\mathbf{Inc}, \mathbf{Dec}$ in the counter automata. In this case, we will also refer to number of elements in stack as value of the counter. When ever we are not interested in the input alphabet, we will refer to the pushdown automata as pushdown system (similarly counter automata as counter system). In such a pushdown system, we will omit the reference to Σ and let $A = (Q, \Gamma, \delta, s, F)$, and let the transition relation be $\delta \subseteq Q \times \mathbf{Op} \times Q$. When we are not interested in the set of final states, we will omit it and simply refer to such pushdown systems as $A = (Q, \Gamma, \Sigma, \delta, s)$.

Multi-counter system An n counter system is a tuple $C = (n, Q, \delta, q_0, F)$ where Q is finite non-empty set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is set of final state and $\delta \subseteq Q \times \text{op} \times Q$ is transition relation, where $\text{op} = \bigcup_{i\in[1..n]}\{\mathbf{Inc}_i, \mathbf{Dec}_i, \mathbf{Zero}_i\}$. The configuration of the counter system C is given by a tuple $(q, v_1, v_2, \cdots, v_n)$, where $q \in Q, v_1, v_2, \cdots, v_n \in \mathbb{N}$. The initial configuration C_{init} is given by $(q_0, 0^n)$ and set of final configurations is given by $C_{final} = \{(f, u_1, \cdots, u_n) \mid f \in F, u_1, u_2, \cdots, u_n \in \mathbb{N}\}$. Given two configuration $c_1 = (q, v_1, v_2, \cdots, v_n)$ and $c_2 = (q', u_1, u_2, \cdots, u_n)$, we say $c_1 \xrightarrow{\tau} c_2$ iff one of the following holds.

- $\tau = (q, \mathbf{Inc}_i, q') \in \delta$, for some $i \in [1..n]$, $u_i = v_i + 1$ and for $j \neq i$, $u_j = v_j$
- $\tau = (q, \mathbf{Dec}_i, q') \in \delta$, for some $i \in [1..n]$, $u_i + 1 = v_i$ and for $j \neq i$, $u_j = v_j$
- $\tau = (q, \mathbf{Zero}_i, q') \in \delta$, for some $i \in [1..n]$, $u_i = v_i = 0$ and for $j \neq i$, $u_j = v_j$

We say a sequence $c_1\tau_1 c_2\tau_2 \cdots c_n$ is a computation of C iff for all $i \in [1..n-1]$, $c_i \xrightarrow{\tau_i} c_{i+1}$. We will sometimes refer to such a computation sequence as $c_1 \xrightarrow{\tau_1} c_2 \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_{n-1}} c_n$. The reachability problem for n-counter system asks whether there is a valid computation from the initial configuration to one of the final configurations. It is well known that the reachability problem for even two counter systems is undecidable. We will let $\delta^i_{\mathbf{Inc}} = \delta \cap Q \times \{\mathbf{Inc}_i\} \times Q$ and $\delta_{\mathbf{Inc}} = \bigcup_{i\in[1..n]} \delta^i_{\mathbf{Inc}}$, we will define $\delta^i_{\mathbf{Dec}}$, $\delta^i_{\mathbf{Zero}}$, $\delta_{\mathbf{Zero}}$ and $\delta_{\mathbf{Dec}}$ analogously. We will also let $\delta^i = \delta^i_{\mathbf{Zero}} \cup \delta^i_{\mathbf{Inc}} \cup \delta^i_{\mathbf{Dec}}$.

n-tape finite state automata A n-tape finite state automaton over $\Sigma_1, \ldots, \Sigma_n$ is defined as $A = (Q, \Sigma_1, \ldots, \Sigma_n, \delta, q_0, F)$ where Q is a finite set of states, q_0 is the initial state, F is the set of final states, and $\delta \subseteq (Q \times (\Sigma_1 \cup \{\epsilon\}) \times \cdots \times (\Sigma_n \cup \{\epsilon\}) \times Q)$, is the transition relation. A run π of A over a n-dim word $\mathbf{w}$ over $\Sigma_1, \ldots, \Sigma_n$ is a sequence of transitions $(q_0, \mathbf{u}_1, q_1), (q_1, \mathbf{u}_2, q_2), \ldots, (q_{n-1}, \mathbf{u}_n, q_n) \in \delta$ such that $\mathbf{w} = \mathbf{u}_1\mathbf{u}_2\cdots\mathbf{u}_n$. The run π is accepting if $q_n \in F$. The language of A, denoted by $L(A)$, is the set of n-dim words $\mathbf{w}$ for which there is an accepting run of A over w. A n-dim language is *rational* if it is the language of some n-tape automaton [34]. Observe that 1-tape automata are the standard finite-state automata.

An interesting subclass of rational languages are what are called *recognizable or regular* languages. A n-dim language L is *regular* if it is a finite union of products of n rational 1-dim languages (i.e. $L = \bigcup_{j=1}^{m} L_{(j,1)} \times \cdots \times L_{(j,n)}$ for some $m \in \mathbb{N}$ where $L_{(j,i)}$ is an 1-dim rational language over Σ_i). Observe that if $n = 1$ rational and regular languages are the same. The language $\{(a^i, b^i) \mid i \geq 0\}$ is an example of a rational language that is not regular.

Chapter 3

Shared memory systems

3.1 Introduction

In this chapter, we will introduce a model called the *shared-memory concurrent pushdown systems*. Informally it is a network of processes with a shared store ranging over a finite domain. Each of these processes can either be a pushdown system, a counter system or simply a finite state system. Each of these processes can perform reads and writes to the shared store. We study the reachability problem in the model. First we will prove that in order to get decidability, restricting only the data domain is not enough. Indeed, we show that two 1-counter system communicating via an one bit store are able to simulate any 2-counter machine.

We then restrict the way information flows through shared memory. The idea we consider is the following: For each computation, consider a decomposition into what we call *stages*, where in each stage only one process is unrestricted (i.e. allowed to read and write) while all the others are only allowed to read. Then, we only consider computations up to some fixed bound on the number of stages. Notice that this notion of bounding, called *stage-bounding*, does not restrict the way stacks and counters are accessed. It is rather imposing that writes by different processes to the shared memory cannot interleave in an unbounded manner (while reads are allowed to interleave unboundedly with any kind of operations from any process). The results we establish in this chapter are as follows.

We consider network of finite state systems and show that reachability under stage bounded restriction in this case is NP-COMPLETE (while in the unbounded case it is PSPACE-COMPLETE). So the stage bounded analysis in this case has the same complexity as context-bounded analysis but offers more coverage. However considering networks with just two pushdown makes stage-bounded analysis much harder. We show that precisely with two pushdown systems, the complexity of stage bounded analysis is (at least) non-primitive recursive. The decidability in this case is still an open problem. We prove that for two pushdown and one counter system, the state reachability problem under stage-bounding restriction is undecidable. On the other hand, we will prove that for networks with at most one pushdown system and any number of counter systems, stage bounded analysis is decidable and is in NEXPTIME while it is PSPACE hard. We establish this decidability result by a non-trivial reduction to the state reachability problem for pushdown systems with reversal bounded counters.

Several bounding concepts have been considered in the literature in the last few years such as context-bounding and phase-bounding [98]. Stage-bounded analysis strictly generalizes context-bounded analysis, while it is incomparable with phase-bounding which is based on restricting accesses to stacks (i.e., push and pop operations by different processes in each phase) rather than restricting accesses to the shared memory. Another work based on restricting the access to stacks is for instance scope bounding [100]. Again, the results there are incomparable with those we present here.

In [24], acyclic networks of communicating pushdown systems are considered. While such an acyclic network can encode computations within one stage (since in a stage information flows unidirectionally from the writer to all other processes), it has been shown that switching once between acyclic communication topologies in a network is enough to get undecidability [27]. In contrast, our main result show a case where information flow can be redirected any finite number of times.

In [77, 64], networks of pushdown systems with non-atomic writes are considered. Atomic read-writes cannot be implemented in that model, which means that only a weak form of synchronization is possible. It is shown that for a fixed number of processes the reachability problem is undecidable, while in the parametrized case the problem becomes decidable [77] and is PSPACE-complete [64]. In contrast, our results hold even for the case where atomic read-writes are allowed and show a decidable case for a fixed number of processes. The parametrized case in the context of our stage-bounded analysis is still open and cannot be reduced to the problem considered in [77, 64].

3.2 Shared memory concurrent pushdown System

Definition 1. *A Shared-memory concurrent pushdown System (SCPS) over a set of memory values $\boldsymbol{M}$ is a tuple $(\mathbf{I},\mathbf{P},\boldsymbol{m_0})$ where $\mathbf{I}$ is a finite set of indices and $\mathbf{P}=\{P_i \mid i\in\mathbf{I}\}$ is an $\mathbf{I}$-indexed collection of pushdown systems $P_i=(Q_i,\Gamma_i,\mathscr{O}_{\boldsymbol{M}},\delta_i,s_i)$. The tape alphabet $\mathscr{O}_{\boldsymbol{M}}=\{\mathbf{!}\boldsymbol{m},\mathbf{?}\boldsymbol{m} \mid \boldsymbol{m}\in \boldsymbol{M}\}$ where $\mathbf{!}\boldsymbol{m}$ denotes writing the value $\boldsymbol{m}$ to the shared memory while $\mathbf{?}\boldsymbol{m}$ refers to reading the value $\boldsymbol{m}$ from the shared memory. The value $\boldsymbol{m_0}\in\boldsymbol{M}$ is the initial memory value.*

A configuration of a SCPS $(\mathbf{I},\mathbf{P},\boldsymbol{m_0})$ over $\boldsymbol{M}$ is a triple $(\boldsymbol{q},\boldsymbol{\gamma},\boldsymbol{m})$ where $\boldsymbol{q}$ assigns an element of Q_i to each $i\in\mathbf{I}$, $\boldsymbol{m}\in\boldsymbol{M}$ is the contents of the shared memory and $\boldsymbol{\gamma}$ assigns an element of $((\Gamma_i\setminus\{\bot\})^*\cdot\{\bot\})$ to each $i\in\mathbf{I}$ such that $(\boldsymbol{q}(i),\boldsymbol{\gamma}(i))$ is a configuration of P_i. The initial configuration of the system is the triple $(\boldsymbol{s},\bot,\boldsymbol{m_0})$ where for each i, $(\boldsymbol{s}(i),\bot(i))$ is the initial configuration of P_i.

The transition relation $\xrightarrow{op}_i$, $op\in\mathscr{O}_{\boldsymbol{M}}, i\in\mathbf{I},\tau\in\delta_i$, relating configurations of the SCPS is defined as follows: $(\boldsymbol{q},\boldsymbol{\gamma},\boldsymbol{m})\xrightarrow{op}_i(\boldsymbol{q'},\boldsymbol{\gamma'},\boldsymbol{m'})$ iff $(\boldsymbol{q}(i),\boldsymbol{\gamma}(i))\xrightarrow{op}(\boldsymbol{q'}(i),\boldsymbol{\gamma'}(i))$, $(\boldsymbol{q}(j),\boldsymbol{\gamma}(j))=(\boldsymbol{q'}(j),\boldsymbol{\gamma'}(j))$ for $j\neq i$ and further one of the following holds

1. *$op=\mathbf{?}\boldsymbol{m}$ and $\boldsymbol{m'}=\boldsymbol{m}$ (a read operation)*
2. *$op=\mathbf{!}\boldsymbol{m'}$ (a write operation)*

We write $\xrightarrow{op}$ (or simply $\xrightarrow{op}$) for $\biguplus_{i\in\mathbf{I}}\xrightarrow{op}_i$. This naturally extends to a relation $\xrightarrow{w}{}^*$ for $w\in\mathscr{O}_{\boldsymbol{M}}^*$, $\sigma\in(\delta=\cup_{i\in\mathbf{I}}\delta_i)^*$. We write $(\boldsymbol{q},\boldsymbol{\gamma},\boldsymbol{m})\rightarrow^*(\boldsymbol{q'},\boldsymbol{\gamma'},\boldsymbol{m'})$ if there is some $w\in\mathscr{O}_{\boldsymbol{M}}^*$ such that $(\boldsymbol{q},\boldsymbol{\gamma},\boldsymbol{m})\xrightarrow{w}{}^*(\boldsymbol{q'},\boldsymbol{\gamma'},\boldsymbol{m'})$.

In this chapter, we will call the pushdown systems in an SCPS as a counter if $|\Gamma \setminus \{\perp\}| = 1$ and refer to it as a finite state system (FSS) if $|\Gamma \setminus \{\perp\}| = 0$

Remark: *Communication via shared memory is* unreliable. *This is because, the reader may skip some of the values (lossiness) while reading some values multiple times (stuttering). It is easy to eliminate stuttering errors, using a unidirectional protocol . The writer, writes a delimiter between every adjacent pair of values. The reader only reads values separated by such a delimiter. i.e. an unused symbol (say* \$*) is added to the set of possible memory values. Now, instead of writing a sequence* $m_1, m_2, \ldots, m_k$ *to the memory, the writer writes the sequence* $m_1, \$, m_2, \$, \ldots m_k, \$$. *The reader also expects a* \$ *between every alternate value and hence avoids stuttering errors. Eliminating lossiness would require acknowledgements from the reader.*

3.2.1 The Reachability Problem for SCPS

Given a SCPS $(\mathbf{I}, \mathbf{P}, \boldsymbol{m_0})$ and a configuration $(\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m})$, reachability problem asks whether $(\boldsymbol{s}, \perp, \boldsymbol{m_0}) \rightarrow^* (\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m})$. Unfortunately, this problem is undecidable. Infact it is undecidable even if we allow the memory to be of size one bit.

Theorem 1. *The reachability problem for SCPS is undecidable even when* $|\boldsymbol{M}| = 2$, $|\mathbf{I}| = 2$ *and both the pushdown systems in* $\mathbf{P}$ *are counter systems.*

Proof. The idea is to reduce reachability on a two counter system to reachability on SCPS. We will first describe the proof idea before formalising the same. Fix a 2-counter machine $C = (2, Q, \delta, q_0, F)$. We construct a SCPS $A = ([1,2], \{P_1, P_2\}, 0)$ over memory values $[0,1]$. We will refer to P_1, P_2 as *master* and *slave* respectively. The master simulates the control state of A as well as the value of the counter 1. The job of the slave is to maintain the value of the counter 2. Quite clearly the master can simulate any move that does not involve counter 2. In order to simulate the moves on counter 2 the master communicates with the slave through the shared memory. We show that it is possible for the master to communicate, unambiguously, a value from the set $\{1,2,3\}$ to the slave, standing for increment, decrement and test for zero respectively and also obtain a confirmation from the slave if it is able to complete the operation successfully. First we show how the master may communicate a single value from $\{1,2,3\}$ and then extend it to sequences of such values.

Assume that the memory contains the value 0. To communicate the value $i \in \{1,2,3\}$ the master carries out the sequence of operations $(\mathbf{!1?0})^i.(\mathbf{?1!0})^i$ on the memory. The slave guesses the value j being sent and executes a sequence of the form $(\mathbf{?1!0})^j.(\mathbf{!1?0})^j$. There are three possibilities and we analyze each of them:

1. $i = j$. In this case there is exactly one successful interleaving of the two sequences and it is of the form $(\mathbf{!1}_m\mathbf{?1}_s\mathbf{!0}_s\mathbf{?0}_m)^i.(\mathbf{!1}_s\mathbf{?1}_m\mathbf{!0}_m\mathbf{?0}_s)^i$ (where, the component involved in the memory operation is marked as a subscript). Further it leaves the memory with the value 0.
2. $i < j$. In this case, the interleaved runs reaches a deadlock after a sequence of the from $(\mathbf{!1}_m\mathbf{?1}_s\mathbf{!0}_s\mathbf{?0}_m)^i$ where both components wait for the other one to write the value 1 to proceed further.

3. $i > j$. In this case, the interleaved runs reaches a deadlock after a sequence of the from $(\mathbf{!1}_m\mathbf{?1}_s\mathbf{!0}_s\mathbf{?0}_m)^i(\mathbf{!1}_m\mathbf{!1}_s + \mathbf{!1}_s\mathbf{!1}_m)$ and both components wait for the other one to write the value 0 to proceed further.

Since all unsuccessful runs deadlock, it follows that the protocol can be repeated for any sequence of values and the system will either deadlock or succeed in communicating the sequence correctly to the slave. Finally, handling the confirmation from the slave to the master is also easy. After guessing the next operation the slave attempts to carry out the operation and only on success does it enter the protocol described above. We will now give the detailed construction of $P_1 = (Q_1, \{a, \perp\}, \Sigma_M, \delta_1, q_0)$ and $P_2 = (Q_2, \{a, \perp\}, \Sigma_M, \delta_2, p_0)$. The process P_1 is described below

- The states of P_1 are given by $Q_1 = Q \cup (Q \times S_1)$ where $S_1 = \{w \mid \exists i \in [1\ldots3], w \text{ is a suffix of } (\mathbf{!1?0})^i.(\mathbf{?1!0})^i\}$
- The input alphabet is given by $\Sigma_M = \{?0, !0, ?1, !1\}$
- The initial state of P_1 is q_0 which is also the initial state of our two counter system
- The transition relation δ_1 is defined as below.

  a.1 For all $q, q' \in Q$, if $(q, \mathbf{Zero}_1, q') \in \delta$, then we have $(q, \mathbf{Zero}, \epsilon, q') \in \delta_1$. This transition simulates the zero test on counter-1.

  a.2 For all $q, q' \in Q$, if $(q, \mathbf{Inc}_1, q') \in \delta$, then we have $(q, \mathbf{Push}(a), \epsilon, q') \in \delta_1$. This transition simulates the increment on counter-1.

  a.3 For all $q, q' \in Q$, if $(q, \mathbf{Dec}_1, q') \in \delta$, then we have $(q, \mathbf{Pop}(a), \epsilon, q') \in \delta_1$. This transition simulates the decrement on counter-1.

  a.4 For all $q, q' \in Q$, if $(q, \mathbf{Zero}_2, q') \in \delta$,then we have $(q, \mathbf{Int}, !1, (q', (\mathbf{?0}).(\mathbf{?1!0}))) \in \delta_1$. This transition is added to start communicating with slave process to perform a zero test on its counter.

  a.5 For all $q, q' \in Q$, if $(q, \mathbf{Inc}_2, q') \in \delta$,then we have $(q, \mathbf{Int}, !1, (q', \mathbf{?0}(\mathbf{!1?0})^1.(\mathbf{?1!0})^2)) \in \delta_1$. This transition is added to start communicating with slave process to perform a increment move on its counter.

  a.6 For all $q, q' \in Q$, if $(q, \mathbf{Dec}_2, q') \in \delta$,then we have $(q, \mathbf{Int}, !1, (q', \mathbf{?0}(\mathbf{!1?0})^2.(\mathbf{?1!0})^3)) \in \delta_1$. This transition is added to start communicating with slave process to perform a decrement move on its counter.

  a.7 For all $q \in Q, a \in \Sigma_M, w \in \Sigma_M^*$, we add $((q, a.w), \mathbf{Int}, a, (q, w)) \in \delta_1$. These transitions are added to enable series of communication with slave process through the shared memory operations.

  a.8 For all $q \in Q$, we add $((q, \epsilon), \mathbf{Int}, \epsilon, q) \in \delta_1$. These transitions are fired only on successful completion of communication with slave process (without deadlocking).

The process P_2 is described below

- The states of P_2 are given by $Q_2 = \{p_0\} \cup S_2$, where $S_2 = \{w \mid \exists j \in [1\ldots3], w \text{ is a suffix of } (\mathbf{?1!0})^j.(\mathbf{!1?0})^j\}$
- The initial state of P_2 is p_0
- The transition relation δ_1 is defined as below.
  In the following set of transitions, slave guesses the operation that master process wants it to perform and moves to state that executes the appropriate protocol after performing

that operation.

b.1 $(p_0, \mathbf{Zero}, \mathbf{?1}, ((\mathbf{!0}).(\mathbf{!1?0}))) \in \delta_2$

b.2 $(p_0, \mathbf{Push}(a), \mathbf{?1}, (\mathbf{!0}(\mathbf{?1!0})^1.(\mathbf{!1?0})^2)) \in \delta_2$

b.3 $(p_0, \mathbf{Pop}(a), \mathbf{?1}, (\mathbf{!0}(\mathbf{?1!0})^2.(\mathbf{!1?0})^3)) \in \delta_2$

b.4 For all $a \in \Sigma_M$, $w \in \Sigma_M^*$, we add $((a.w), \mathbf{Int}, a, (w)) \in \delta_2$

b.5 We also add $(\epsilon, \mathbf{Int}, \mathbf{?0}, p_0) \in \delta_2$, this transition is fired on successful completion of communication with master process.

The correctness of such a construction follows from the following lemma which relates the runs of the SCPS A constructed with the runs of the two counter system C.

Lemma 1. $(q_0, 0, 0) \rightarrow^*_C (q, v_1, v_2)$ *iff* $((q_0, p_0), (\bot, \bot), 0) \rightarrow^* ((q, p_0), (a^{v_1}\bot, a^{v_2}\bot), 0)$.

With Lemma 1 in place, it is easy to see that reachability of a two counter system C reduces to reachability on SCPS A. This will also complete the proof of Theorem 1. We will now prove Lemma 1, which is not very difficult to see but some what tedious. For this, we will first prove the following lemma which states that the protocol followed by the master and the slave succeeds without deadlocking iff both of them guess a sequence of identical length.

Lemma 2. $((q, \mathbf{?0}(\mathbf{!1?0})^{j-1}.(\mathbf{?1!0})^j), (\mathbf{!0}(\mathbf{?1!0})^{k-1}.(\mathbf{!1?0})^k), a^{n_1}\bot, a^{n_2}\bot), 1) \rightarrow^* ((q, \epsilon), \epsilon), a^{n_1}\bot, a^{n_2}\bot, 0)$, *for some* $x \in [0,1]$ *iff* $j = k$.

Proof. We first prove that for case where $j = k$, we have a successful run. Let $\mu : \Sigma_M \mapsto \Sigma_M$ be a function such that $\mu(?x) = !x$ and $\mu(!x) = ?x$ for any $x \in [0,1]$. Such a function can easily be extended to a sequence $w \in \Sigma_M^*$. We note that $\mu(\mathbf{?0}(\mathbf{!1?0})^{j-1}.(\mathbf{?1!0})^j)) = (\mathbf{!0}(\mathbf{?1!0})^{j-1}.(\mathbf{!1?0})^j)$. We now prove that for any $((q, w_1), w_2, a^{n_1}\bot, a^{n_2}\bot, y)$, such that $w_1 = \mu(w_2)$ and $w_i \in S_i$, we have $((q, w_1), w_2, a^{n_1}\bot, a^{n_2}\bot, y) \rightarrow^* ((q, \epsilon), \epsilon, a^{n_1}, a^{n_2}, 0)$.

Claim 1. *For any* $w_1 \in S_1, w_2 \in S_2$ *such that* $w_1 = \mu(w_2)$, *we have* $((q, w_1), w_2, a^{n_1}\bot, a^{n_2}\bot, y) \rightarrow^* ((q, \epsilon), \epsilon, a^{n_1}, a^{n_2}, 0)$

Proof. We prove this inducting on length of w_1 and w_2

For base case, we consider $w_1 = !0$ and $w_2 = ?0$ (note that the only words of length 1 in $S_1 = !0$ and $S_2 = ?0$). By construction, for all $q \in Q$, we have $\tau_1 = ((q, !0), \mathbf{Int}, !0, (q, \epsilon)) \in \delta_1$ and $\tau_2 = (?0, \mathbf{Int}, ?0, \epsilon) \in \delta_2$. From this it is easy to see that $((q, !0), ?0, a^{n_1}\bot, a^{n_2}\bot, y) \rightarrow^* ((q, \epsilon), \epsilon, a^{n_1}, a^{n_2}, 0)$, by applying τ_1 followed by τ_2 transitions.

Case where $|w_1| = |w_2| > 1$, w.log we will assume that $w_1 = ?x.w_1'$ and $w_2 = !x.w_2'$ for $x \in [0, 1]$ (the other case is similar). Clearly $((q, ?x.w_1'), (!x.w_2'), a^{n_1}\bot, a^{n_2}\bot, y) \rightarrow ((q, ?x.w_1'), (w_2'), a^{n_1}\bot, a^{n_2}\bot, x)$ since $((!x.w_2'), \mathbf{Int}, !x, (w_2')) \in \delta_2$. We also have $((q, ?x.w_1'), (p_0, w_2'), a^{n_1}\bot, a^{n_2}\bot, x) \rightarrow ((q, w_1'), (p_0, w_2'), a^{n_1}\bot, a^{n_2}\bot, x)$, since we have $((q, ?x w_1'), \mathbf{Int}, ?x, (q, w_1')) \in \delta_1$ and the current memory value is x. Note that $w_i' \in S_i$ and $|w_1'| = |w_2'| < |w_1| = |w_2|$. Hence we can apply induction hypothesis to get the required run.

□

We will prove that for cases where $j \neq i$ all possible runs deadlock. Since there are finitely many (six cases) cases to consider, we will analyse each of these cases individually to show that in each case all possible runs necessarily deadlocks.

- Case when $j = 1,\ k = 3$, we have $c = ((q, \mathbf{?0}(\mathbf{!1?0}).(\mathbf{?1!0})^1), (\mathbf{!0}(\mathbf{?1!0})^2.(\mathbf{!1?0})^3), a^{n_1}\bot, a^{n_2}\bot), 1)$. Note that the only possible move enabled here is for P_2 to write the value 0 onto memory, since P_1 is waiting to read value 0 from memory. Hence we have $c = ((q, \mathbf{?0}(\mathbf{!1?0}).(\mathbf{?1!0})^1),\ (\mathbf{!0}(\mathbf{?1!0})^2.(\mathbf{!1?0})^3),\ a^{n_1}\bot, a^{n_2}\bot), 1) \rightarrow c_1 = ((q, \mathbf{?0}(\mathbf{!1?0}).(\mathbf{?1!0})^1), ((\mathbf{?1!0})^2.(\mathbf{!1?0})^3),\ a^{n_1}\bot, a^{n_2}\bot), 0)$. Now note that in c_1, the only possible move is for P_1 to read the memory value 0 and proceed (since P_2 is waiting on a value 1). Hence we have $c_1 \rightarrow c_2 = ((q, (\mathbf{!1?0}).(\mathbf{?1!0})^1), ((\mathbf{?1!0})^2.(\mathbf{!1?0})^3),\ a^{n_1}\bot, a^{n_2}\bot), 0)$. Now note that the memory value is 0 and P_2 is waiting on value 1. Hence the only possible way to proceed is for P_1 to write a 1, proceeding thus, we get the run $c \rightarrow c_1 \rightarrow c_2 \rightarrow^* ((q, (\mathbf{?1!0})^1), ((\mathbf{?1!0}).(\mathbf{!1?0})^3),\ a^{n_1}\bot,\ a^{n_2}\bot), 0)$. Now note that both P_1 and P_2 are waiting on a value 1 and hence they can never proceed further.
- Case where $j = 1, k = 2$ and $j = 2, k = 3$ are similar to the case above.

For the case where $k < j$, we have the following cases to consider

- Case when $k = 1,\ j = 3$ we have $c = ((q, \mathbf{?0}(\mathbf{!1?0})^2.(\mathbf{?1!0})^3), \mathbf{!0}(\mathbf{!1?0})),\ a^{n_1}\bot, a^{n_2}\bot), 1)$. Notice that P_1 is waiting on reading the value 0 and the current memory value is 1, hence the only way the run can proceed is by P_2 writing a 0 and then P_1 reading it. Hence we have $c \rightarrow^* c_1 = ((q, (\mathbf{!1?0})^2.(\mathbf{?1!0})^3),\ (\mathbf{!1?0})),\ a^{n_1}\bot, a^{n_2}\bot), 1)$. Now there are two possible ways to proceed, either P_1 writes a 1 and goes onto waiting on 0, followed by P_2 writing a 1 and going onto wait on value 0 or the other way around. In both cases, both the processes wait on value 0 and hence deadlocks.
- Case where $k = 2, j = 3$ and $k = 1, j = 2$ are similar to the case above.

This completes the proof of Lemma 2

□

Proof of Lemma 1

Proof. ($\Rightarrow$)

We will prove this by induction on length of the computation. Base case involving zero length computation is trivial since $((q_0, p_0), (\bot, \bot), 0) \rightarrow^* ((q_0, p_0), (\bot, \bot), 0)$.

Suppose we have a run $(q_0, 0, 0) \rightarrow^*_C (q_1, v_1', v_2') \xrightarrow{\tau} (q, v_1, v_2)$. If τ is any operation on counter-1, then the proof is trivial since by construction, we added for every transition in C of the form $(q, \mathrm{op}_1, q') \in \delta$ where $\mathrm{op}_1 \in \{\mathbf{Inc}_1, \mathbf{Zero}_1, \mathbf{Dec}_1\}$, a transition of the form $(q, \mathrm{op}, \epsilon, q') \in \delta_1$ where $\mathrm{op} \in \{\mathbf{Push}(a), \mathbf{Pop}(a), \mathbf{Zero}\}$ that can simulate such a move. We will consider $\tau = (q_1, \mathbf{Zero}_2, q)$, rest of the cases are similar. Note that since τ succeeds, we have that $v_2' = v_2 = 0$ and $v_1 = v_1'$.

By induction we have a run $\pi' = ((q_0, p_0), (\bot, \bot), 0) \rightarrow^* ((q_1, p_0), (a^{v_1}\bot, \bot), 0)$. By construction, since we have $\tau \in \delta$, we have the transition $(q_1, \mathbf{Int}, \mathbf{!1}, (q, (\mathbf{?0}).(\mathbf{?1}.\mathbf{!0}))) \in \delta_1$, (see a.1). We also have the transition $(p_0, \mathbf{Int}, \epsilon, (\mathbf{!0}.(\mathbf{!1}.\mathbf{?0}))) \in \delta_2$, (see b.1). Hence we can extend the run π' as follows.

$$((q_1, p_0), (a^{v_1}\bot, \bot), 0) \rightarrow (((q, \mathbf{?0}.(\mathbf{?1!0})),\ p_0), (a^{v_1}\bot, \bot), 1) \rightarrow$$
$$(((q, \mathbf{?0}(\mathbf{?1!0})),\ (\mathbf{!0}.(\mathbf{!1?0}))), (a^{v_1}\bot, \bot), 0).$$

From Lemma 2, we can extend such a run to

$$((q_1,p_0),(a^{\nu_1}\bot,\bot),0)\rightarrow(((q,\mathbf{?0}.(\mathbf{?1!0})),p_0),(a^{\nu_1}\bot,\bot),1)\rightarrow \\ ((q,\mathbf{?0}(\mathbf{?1!0})),(\mathbf{!0}.(\mathbf{!1?0})),(a^{\nu_1}\bot,\bot),0)\rightarrow^*((q,\epsilon),\epsilon,a^{\nu_1}\bot,\bot,0).$$

Now, using the transitions from a.8 and b.5 we can complete the run.

($\Leftarrow$)

Let $T=\{((q,p_0),a^{\nu_1}\bot,a^{\nu_2}\bot,0)\mid q\in Q,\nu_1,\nu_2\in\mathbb{N}\}$, note that T that does not involve intermediate states of the form (q,w) in P_1 or w in P_2 for some $w\in S_1\cup S_2$ and $q\in Q$. For this direction, we will induct on number of time a configuration from set T is seen in the run. Let us consider the computation

$$\pi=((q_0,p_0),(\bot,\bot),0)\rightarrow^*((q,p_0),(a^{\nu_1}\bot,a^{\nu_2}\bot),0)$$

For base case, if number of times a configuration from T seen in π is 1, clearly $q=q_0$ and $\nu_1=\nu_2=0$ i.e. it is a run of length zero. For induction case, let us consider that the number of times a configuration seen in the run is greater than 0. Suppose number of times a configuration from T seen is >1, then the run can be broken up as

$$\pi=c_0=((q_0,p_0),(\bot,\bot),0)\rightarrow^*c_1=((q_1,p_0),(a^{\nu_1^1}\bot,a^{\nu_2^1}\bot),0)\rightarrow^*\cdots \\ \rightarrow^*c_m=((q_m,p_0),(a^{\nu_1^m}\bot,a^{\nu_2^m}\bot),0)\rightarrow^*c=((q,p_0),(a^{\nu_1}\bot,a^{\nu_2}\bot),0).$$

Where $c_0,c_1,\cdots c_m,c$ are all the configurations from T in π.

By induction, we have the run $\sigma=(q_0,0,0)\rightarrow^*{}_C(q_m,\nu_1^m,\nu_2^m)$. If the subcomputation $((q_m,p_0),(a^{\nu_1^m}\bot,a^{\nu_2^m}\bot),0)\rightarrow^*((q,p_0),(a^{\nu_1}\bot,a^{\nu_2}\bot),0)$, does not involve transitions of P_2, then each such a transition is of type (q,op,ϵ,q') for some $op\in\{\mathbf{Zero},\mathbf{Push}(a),\mathbf{Pop}(a)\}$ and it can be simulated by an equivalent transition in the counter system.

Now assume that the π' involves P_2 transitions. Then clearly, the subcomputation $((q_m,p_0),(a^{\nu_1^m}\bot,a^{\nu_2^m}\bot),0)\rightarrow^*((q,p_0),(a^{\nu_1}\bot,a^{\nu_2}\bot),0)$ can be expanded as

$$\pi'=c_1=((q_m,p_0),(a^{\nu_1^m}\bot,a^{\nu_2^m}\bot),0)\rightarrow \\ (((q_1,\mathbf{?0}(\mathbf{!1?0})^{j-1}.(\mathbf{?1!0})^j),p_0,(a^{\nu_1'}\bot,a^{\nu_2'}\bot),1)\rightarrow \\ c_2=(((q_1,\mathbf{?0}(\mathbf{!1?0})^{j-1}.(\mathbf{?1!0})^j),\ (\mathbf{!0}(\mathbf{?1!0})^{k-1}.(\mathbf{!1?0})^k))),(a^{\nu_1'}\bot,a^{\nu_2'}\bot),1)\rightarrow^* \\ c_3=((q_2,\epsilon),\epsilon,(a^{\nu_1''}\bot,a^{\nu_2''}\bot),0)\rightarrow^*c_4=((q,p_0),(a^{\nu_1}\bot,a^{\nu_2}\bot),0)$$

We make the following observations about the computation π'.

- In $c_1\rightarrow^*c_4$, c_1 and c_4 are the only configurations from T.
- In $c_1\rightarrow^*c_2$, involves only transitions form (a.4 or a.5 or a.6) and (b.1 or b.2 or b.3). Hence, we have $\nu_1^m=\nu_1'$.
- From lemma-2, we know that $k=j$
- $c_2\rightarrow^*c_3$ only involve transitions of the form a.7, b.4, from this we get $q_1=q_2$, $\nu_2''=\nu_2'$ and $\nu_1''=\nu_1'$ since these transitions do not involve stack manipulation.

- In $c_3 \rightarrow^* c_4$, the transitions a.8 and b.5 are used. Since c_1, c_4 are only transitions from T, there is no transitions of the form (b.1, b.2 and b.3), from this we can conclude that $v_2'' = v_2$.

Now we can rewrite π' as follows.

$$\pi' = c_1 = ((q_m, p_0), (a^{v_1^m}\bot, a^{v_2^m}\bot), 0) \rightarrow^*$$
$$c_2 = (((q_1, \mathbf{?0}(\mathbf{!1?0})^{j-1}.(\mathbf{?1!0})^j),\ (\mathbf{!0}(\mathbf{?1!0})^{j-1}.(\mathbf{!1?0})^j))), (a^{v_1'}\bot, a^{v_2}\bot), 1) \rightarrow^*$$
$$c_3 = ((q_1, \epsilon), \epsilon, (a^{v_1'}\bot, a^{v_2}\bot), 0) \rightarrow^* c_4 = ((q, p_0), (a^{v_1}\bot, a^{v_2}\bot), 0)$$

We will only consider the case where $j = 1$ (the other cases are similar) and show how to extend σ. Now π' can be written as

$$\pi' = c_1 = ((q_m, p_0), (a^{v_1^m}\bot, a^{v_2^m}\bot), 0) \rightarrow^*$$
$$c_2 = (((q_1, \mathbf{?0}(\mathbf{?1!0})),\ (\mathbf{!0}(\mathbf{!1?0})))), (a^{v_1'}\bot, a^{v_2}\bot), 1) \rightarrow^*$$
$$c_3 = ((q_1, \epsilon), \epsilon, (a^{v_1'}\bot, a^{v_2}\bot), 0) \rightarrow^* c_4 = ((q, p_0), (a^{v_1}\bot, a^{v_2}\bot), 0)$$

We have two cases to consider, depending on whether b.5 was executed before a.8 or not in $c_3 \rightarrow^* c_4$. We will assume that b.5 was executed first, then a.8 is executed immediately after. Then, we have $q_1 = q$ and the following form for the subcomputation.

$$\pi' = c_1 = ((q_m, p_0), (a^{v_1^m}\bot, a^{v_2^m}\bot), 0) \rightarrow^*$$
$$c_2 = (((q, \mathbf{?0}.(\mathbf{?1!0})),\ (\mathbf{!0}.(\mathbf{!1?0})))), (a^{v_1^m}\bot, a^{v_2}\bot), 1) \rightarrow^*$$
$$c_3 = (((q, \epsilon), \epsilon), (a^{v_1^m}\bot, a^{v_2}\bot), 0) \rightarrow c_4 = (((q, \epsilon), p_0), (a^{v_1^m}\bot, a^{v_2}\bot), 0) \rightarrow$$
$$((q, p_0), (a^{v_1^m}\bot, a^{v_2}\bot), 0)$$

Firstly, $\tau_1 = (q_m, \mathbf{Int}, \mathbf{!1}, (q, (\mathbf{?0}).(\mathbf{?1!0}))) \in \delta_1$ and $\tau_2 = (p_0, \mathbf{Zero}, \mathbf{?1}, (\mathbf{!0}).(\mathbf{!1?0})) \in \delta_2$ transitions were used in $c_1 \rightarrow^* c_2$. Since τ_1 was used in such a sub compuation, we know that there is a transition of the form $(q_m, \mathbf{Zero}_2, q) \in \delta$ and from execution of τ_2, from this we can conclude that $v_2^m = v_2 = 0$. From this we get the required run $(q_0, 0, 0) \rightarrow^* (q_m, v_1^m, 0) \rightarrow (q, v_1, 0)$.

In the other case, once $((q', \epsilon), \mathbf{Int}, \epsilon, q') \in \delta_1$ is executed, process P_1 can start executing transitions not involving any operations on counter-2 before process P_2 returns back to p_0 by executing $(\epsilon, \mathbf{Int}, \mathbf{?0}, p_0) \in \delta_2$. In this case, π' is of the form

$$\pi' = ((q_m, p_0), (a^{v_1^m}\bot, a^{v_2^m}\bot), 0) \rightarrow^*$$
$$(((q', \mathbf{?0}(\mathbf{?1!0})),\ (\mathbf{!0}(\mathbf{!1?0})))), (a^{v_1^m}\bot, a^{v_2}\bot), 1) \rightarrow^* (((q', \epsilon), \epsilon), (a^{v_1^m}\bot, a^{v_2}\bot), 0) \rightarrow$$
$$(q', \epsilon, (a^{v_1^m}\bot, a^{v_2}\bot), 0) \rightarrow^* (q'', \epsilon, (a^{v_1}\bot, a^{v_2}\bot), 0) \rightarrow (q, p_0, (a^{v_1}\bot, a^{v_2}\bot), 0)$$

Firstly note that if the master process were to start any communication with slave process before the slave goes back to p_0 state, the computation will dead lock (since the transition that takes slave process from state ϵ to p_0 requires the memory value to be 0).

From this we have that any sub-run of the form $(q',\epsilon,(a^{v_1^m}\bot,a^{v_2}\bot),0)\rightarrow^*(q'',\epsilon,(a^{v_1'}\bot,a^{v_2}\bot),0)$, can involve only transitions that operate on counter-1. Clearly such transitions can be simulated by equivalent transitions in the counter system, leading to a run of the form $(q',v_1^m,v_2)\rightarrow^*(q'',v_1,v_2)$. As in previous case, from execution of τ_1 and τ_2, we have $v_2^m = v_2 = 0$. From this we have the required run $(q_0,0,0)\rightarrow^*(q',v_1^m,0)\rightarrow^*(q'',v_1,0)\rightarrow^*(q,v_1,0)$. This complete the proof of Lemma 1

□

With this, the proof of Theorem 1 is complete. □

3.3 Stage-bounded Computations

We introduce hereafter the concept of stage-bounding. We divide a run into segments, called stages, where in each stage at most one component is allowed to write on the shared memory while there is no restriction on the number of readers. We emphasize that there is no restriction placed on the number of writes or the number of context switches between the different components nor is there any restriction on the accesses to stacks during a stage. We then place an a-priori bound on the number of stages in the run. Formally

Definition 2. *Let* $\rho = \boldsymbol{c_0} \xrightarrow{op_1}_{p_1} \boldsymbol{c_1} \xrightarrow{op_2}_{p_2} \ldots \boldsymbol{c_{n-1}} \xrightarrow{op_n}_{p_n} \boldsymbol{c_n}$ *be a run of the SCPS* $(\mathbf{I},\mathbf{P},\boldsymbol{m})$. *We say that* ρ *is a* p*-run if for all* $1 \le i \le n$, *whenever* $op_i \in \mathbb{W}_{\boldsymbol{M}}$ *(where* $\mathbb{W}_{\boldsymbol{M}} = \{!\boldsymbol{m} \mid m \in \boldsymbol{M}\}$*), we have* $p_i = p$. *That is, all the write transitions are contributed by the same process* p.

We say that ρ *is a* 1*-stage run if it is a* p*-run for some* $p \in \mathbf{I}$ *and a run* ρ *is a* k*-stage run if we may write* $\rho = \boldsymbol{c_0} \xrightarrow{w_1}{}^* \boldsymbol{c_1} \xrightarrow{w_2}{}^* \ldots \boldsymbol{c_{k-1}} \xrightarrow{w_k}{}^* \boldsymbol{c_k}$ *such that each* $\boldsymbol{c_{i-1}} \xrightarrow{w_i}{}^* \boldsymbol{c_i}$ $(1 \le i \le k)$ *is a* 1*-stage run.*

Stage-bounded Reachability Problem: Given a SCPS $(\mathbf{I},\mathbf{P},\boldsymbol{m_0})$, an integer k and a configuration $(\boldsymbol{q},\boldsymbol{\gamma},\boldsymbol{m})$ determine whether there is a k-stage run $(\boldsymbol{s},\bot,\boldsymbol{m_0})\rightarrow^*(\boldsymbol{q},\boldsymbol{\gamma},\boldsymbol{m})$.

3.4 Stage bounded reachability for Communicating FSS

In this section, we show that stage-bounding is relevant even when all components of the SCPS are finite-state. In this case stage bounded reachability problem is indeed easier than the unrestricted reachability problem.

Theorem 2. *The reachability problem for an SCPS where every component is a FSS (finite state system) is* PSPACE-COMPLETE *while the stage bounded reachability problem for SCPS where every component is a FSS is* NP-COMPLETE.

Proof. When there is no bound on the number of stages, it is easy to see that an SCPS with n FSS components is equivalent to the product (intersection) of n FSS and hence the reachability problem is PSPACE-COMPLETE. The details are given below.

The PSPACE algorithm for reachability can easily be obtained by reducing it to language intersection of n finite state automata. For this purpose, we will construct a finite state automata A_i corresponding to each FSS P_i in our SCPS. The finite state automata A_i that we

construct (corresponding to process P_i), has as its input alphabet the memory values tagged with the process index (i.e. $\boldsymbol{M} \times \mathbf{I}$). The state space of such a FSA is given by $Q_i \cup Q_i \times \boldsymbol{M}$, where Q_i is the state space of P_i. The states of the form $(q, \boldsymbol{m}) \in Q_i \times \boldsymbol{M}$ will be used during the write moves to simulate stuttering. The constructed finite state automaton simulates a write move of P_i, of the form $(q, \mathbf{!}\boldsymbol{m}, q')$ through a transition of the form $(q, (\boldsymbol{m}, i), (q', m))$. From states of the form $(q, \boldsymbol{m})$, we have transitions of the form $((q, \boldsymbol{m}), (\boldsymbol{m}, i), (q, \boldsymbol{m}))$ and $((q, \boldsymbol{m}), \epsilon, q)$. These moves allow stuttering of write moves. Any read move of P_i, of the form $(q, \mathbf{?}\boldsymbol{m}, q')$ is simulated by transitions of the form $\{q\} \times (\boldsymbol{M} \times \mathbf{I} \setminus \{i\}) \times \{q'\}$. Here the read move is simulated by reading memory value tagged with any index other than itself. So far all transitions added, simulate moves of FSS. However, we need also moves that mimic moves of other processes. For this, we upward close the finite state automata w.r.t. memory values tagged with index other than itself ($\boldsymbol{M} \times \mathbf{I} \setminus \{i\}$). For this, we have for every $q \in Q_i$, transitions of the form $\{q\} \times (\boldsymbol{M} \times \mathbf{I} \setminus \{i\}) \times \{q\}$. Note that in transitions of A_i, transitions labeled by letters of the form $\boldsymbol{M} \times \{i\}$, correspond to simulating a write of FSS P_i. Whereas transitions labeled by letters of the form $\boldsymbol{M} \times \mathbf{I} \setminus \{i\}$ can either simulate a read move or can mimic a move of another process. So far our construction religiously simulates every move except for own reads, i.e. memory values read by a process, that was written by itself. But notice that such transitions can easily be eliminated from the SCPS by storing the memory values written in the state space. With this observation, the construction is complete. Note that for reachability on SCPS, we are interested in checking whether a particular configuration for each FSS is reachable. A run in such an FSA is accepting if the configuration we are interested in for the corresponding FSS is reached. With such a construction in place, it is easy to see that $\bigcap_{i \in \mathbf{I}} L(A_i) \neq \emptyset$ iff there is a run in SCPS that reaches required state in each FSS.

For the hardness we reduce the emptiness of the intersection of n finite state automata (FSA) to this problem. We use n FSSs. The first one guesses a word in the intersection and apart from simulating the first FSA on this word, it also transfers this word, letter by letter, reliably to the other FSSs using the shared memory. This can be done easily using acknowledgements since there is no bound on the number of stages. The other $n-1$ FSSs simulate one FSA each and hence the emptiness of the intersection reduces to the reachability problem.

To solve the stage bounded reachability problem, we show that it suffices to consider runs where in each stage every one of the readers participates in at most $|A_i|$ transitions, where A_i is the ith automaton. We then use this to show that in addition we may restrict to runs where in each stage the writer participates in at most $O((\sum_i |A_i|)^2)$ transitions.

Let A_i be the ith FSS. Let ρ be the sequence of transitions in some k stage run from c_0 to c_k and let $\rho = \rho_1 \rho_2 \dots \rho_k$ where each ρ_i constitutes a single stage and $c_{i-1} \xrightarrow{\rho_i}{}^* c_i$. We show that we may find a different k stage run $\rho' = \rho'_1 \rho'_2 \dots \rho'_k$, with each ρ'_j constituting a single stage, such that $c_0 \xrightarrow{\rho'_1}{}^* c_1 \xrightarrow{\rho'_2}{}^* \dots \xrightarrow{\rho'_k}{}^* c_k$ and the length of each ρ'_j is polynomial in $\sum_i |A_i|$ and $|\boldsymbol{M}|$. As a first step towards this we show that in any ρ_j, we can bound the number of transitions of any reader i to $|A_i|$. This is because, if there are two occurrences of the same transition τ for some reader i in ρ_j then we may safely delete all the transitions of i in ρ_j between the first and second occurrences, including the first but not the last and obtain a ρ'_j

such that $c_{j-1} \xrightarrow{\rho'_j}{}^{*} c_j$. Thus, the total number of transitions by all the readers in any ρ_j can be bounded by $\sum_i |A_i|$.

Now, suppose there are two occurrences of a transition τ of the writer w in ρ_j (the transition by itself need not necessarily be a write transition) and suppose there are no (read) transitions involving other FSSs (readers) in between. Then we may remove from ρ_j all transitions of the writer starting with first occurrence τ and upto but not including the second occurrence and obtain a ρ'_j such that $c_{j-1} \xrightarrow{\rho'_j}{}^{*} c_j$. This along with the bound on the total number of read transitions means that the number of transitions of the writer in ρ_j needs to be at most $|A_w|.(\sum_i |A_i|)$, where A_w is the writer, and hence quadratic in the sum of the sizes of the FSSs.

Thus we may restrict the length of k stage runs to be at most $k.(\sum_i .|A_i|)^2$ without sacrificing any reachable states and the stage bounded reachability problem is in NP.

□

3.5 Bounded-Stage Reachability of recursive processes

Bounded stage reachability problem is not decidable even when only 2 pushdown and 1 counter processes are involved.

3.5.1 Undecidability of Bounded-Stage Reachability

Unfortunately, stage bounding does not lead to decidability in the general case. We can indeed prove that SCPS with two pushdown systems and one 1-counter system are able to encode the computation of any Turing machine.

Theorem 3. *The* 3*-stage reachability problem for SCPS consisting of two pushdown systems and one counter system is undecidable.*

Proof. We will reduce the halting problem for Turing machines to the stage-bounded reachability problem in a SCPS with two pushdown systems and one counter. We refer to the two pushdowns as the *generator* and the *replayer*. If somehow a writer and a reader could follow a protocol that ensures that every letter that is written is read exactly once then the undecidability would follow quite easily without the counter. However, doing this using shared memory in a stage bounded manner is tricky and details are as follows. In what follows we assume that stuttering errors are eliminated using a suitable delimiter.

The simulation of a (potential) accepting run of the TM is carried out in 4 steps which use 3 stages in all. We fix a suitable encoding of the configurations as a word over some alphabet Γ and assume that this alphabet does not contain the symbol #. In the first step, the generator writes down a (initial) configuration C_1 of the TM in its stack followed by the # symbol. While doing so, it uses the shared memory to send a value, say $\sharp$, to the counter for each letter in C_1. The counter counts the number of such values. Since stuttering has been eliminated, the value of the counter c_1 is $\leq |C_1|$ at the end of this step.

In step 2, the generator guesses a sequence of configurations $C_2, C_3, \ldots C_n$ ending in an accepting configuration, writes them down, separated by #s, in its stack. It also writes the same

sequence to the memory, as it is generated, which in turn is read by the replayer and copied on to its stack. At the end of step 2, the contents of the generator's stack is $x = C_n^R \# C_{n-1}^R \# \ldots C_1^R$ while that of the replayer is $y = D_m^R \# D_{m-1}^R \# \ldots D_1^R$, $m \leq n-1$ and y is a subword of x. It indicates the end of this stage by writing some suitable value to the memory which signals the end of this stage to the replayer and the counter. In all we have used one stage so far.

In step 3, the counter sends its value c_1 to the generator using the shared memory by writing c_1 copies of some fixed value ending with some special value to indicate the completion of this sequence. The generator removes one non-# symbol from his stack for each such value. At the end of this sequence of operations if the top of stack is not a # the generator will reject this run. Thus, a successful completion of this step will mean that $|C_n| \leq c_1$ and thus, $|C_n| \leq |C_1|$. At the end of this step, the contents of the generator's stack is $C_{n-1}^R \# C_{n-2}^R \# \ldots C_1^R$ and the counter is empty. This constitutes the second stage.

In the last step, the replayer removes the contents of its stack one element at a time and writes the removed value to the shared memory for the generator to read. It writes a special end marker at the end of the sequence and enters an accepting state. The sequence read by the generator would therefore be of the form $z = E_p^R \# E_{p-1}^R \# \ldots E_1^R$ (followed by the end marker) where $p \leq m \leq n-1$. Clearly z is a subword of y. The generator, as it reads E_p^R removes symbols from its stack verifying that C_{n-1} may be reached in one step from the configuration E_p (we write $E_p \rightarrow^* C_{n-1}$ to indicate this), entering a reject state if either this is false or if they are not of the same length. It then repeats this procedure for E_{p-1} and C_{n-2} and so on. It enters an accepting state only if it empties its stack at the end of the entire sequence.

Observe that if the generator reaches its accepting state then p has to be $n-1$, $|E_{n-1}| = |C_{n-1}|$, ..., $|E_1| = |C_1|$ and $E_{n-1} \rightarrow^* C_{n-1}$, ..., $E_1 \rightarrow^* C_1$. Further, since z is a subword of y, y is a subword of $C_n^R \# C_{n-1}^R \# \ldots C_2^R$ and $p = n-1$, we have $E_i \preceq D_i \preceq C_{i+1}$ for all $1 \leq i \leq n-1$. Thus,

$$|C_1| = |E_1| \leq |C_2| = |E_2| \leq \ldots \leq |C_{n-1}| = E_{n-1} \leq |C_n|$$

But $|C_n| \leq |C_1|$ and thus,

$$|C_1| = |E_1| = |C_2| = |E_2| \ldots = |C_{n-1}| = |E_{n-1}| = |C_n|$$

Therefore $E_1 = C_2$, $E_2 = C_3$, ..., $E_{n-1} = C_n$ and the result follows. □

3.5.2 Bounded stage reachability for two pushdown case

We already showed that the stage bounded reachability problem for systems with at least two pushdowns and one counter is undecidable. One can ask what happens if we were to restrict ourselves to just two pushdown case. The problem currently remains open. Even if this problem is decidable its complexity cannot be primitive recursive.

The *regular post embedding* problem is the following: Let Σ and Γ be two alphabets. Given two functions $f : \Sigma \rightarrow \Gamma^+$ and $g : \Sigma \rightarrow \Gamma^+$, extended homomorphically to Σ^+, and a regular language $R \subseteq \Sigma^+$, does there exist a $w \in R$ such that $f(w) \preceq g(w)$? As shown in [52], this problem is decidable but cannot be solved by any algorithm with primitive recursive complexity. We reduce the regular post embedding problem to the stage-bounded reachability problem for SCPS with two pushdowns to obtain the following Theorem.

Theorem 4. *The 2-stage bounded reachability problem for SCPS with two pushdowns cannot be solved by any algorithm whose complexity is primitive recursive.*

Proof. The reduction is indeed quite simple. Once again, we refer to the two pushdowns as the *generator* and *replayer*. The generator guesses a word w from R, stores $f(w)$ in its stack and while doing so writes $g(w)$ letter by letter on the shared memory. As always, we assume stuttering errors are eliminated using delimiters. The replayer reads these values and stores them in its stack. At the end of this first stage the contents of the two stacks are $f(w)$ and w' with $w' \preceq g(w)$.

In the second stage, the replayer transfers the contents of its stack to the shared memory, one letter at a time, and the generator pops it stack verifying that it agrees with the letters read from the shared memory. It enters the accepting state only if it empties its stack exactly that the end of this stage. Notice that the values read by the generator w'' is a subword of w' and thus an accepting run verifies that $f(w) = w'' \preceq w' \preceq g(w)$.

Conversely, if $f(w) \preceq g(w)$ there is an accepting run where exactly the letters that do not belong to the embedding of $f(w)$ in $g(w)$ are *lost* i.e. missed by the reader (the replayer in the first stage and the generator in the second stage) in at least one of the two stages. □

3.5.3 Decidability for single pushdown plus counters

The problem is decidable if we restrict ourselves to certain subclasses. The following Theorem describes this class and is the main result of this chapter.

Theorem 5. *The stage bounded reachability problem for SCPS with at most one pushdown system is in* NEXPTIME.

Basically, we show that each counter system can be simulated by an exponential sized bounded-reversal counter system thus reducing the problem to reachability in a *pushdown automaton*[1] (PDA) with reversal bounded counters (which is known to be in NP).

The proof of this Theorem is quite involved and most of what follows in this chapter is devoted to the same. The proof proceeds in a sequence of steps and in each step we provide an informal description of the ideas before providing the formal details The first step is applicable to any SCPS. In this step, we eliminate the shared memory, decouple the different pushdown systems as a collection of pushdown automata (PDA) and reduce the reachability problem for the SCPS to the emptiness of the intersection of these PDAs. This problem, in general, is undecidable, but we will be able to restrict ourselves to the case where the PDAs are of a restricted variety.

In a shared memory system, the sequence of values written by the writer in a stage is not transmitted with precision to the reader as the reader may miss some values while reading others multiple times and this is what permits the decoupling.

We fix an SCPS $S = (\mathbf{I}, \mathbf{P}, \boldsymbol{m_0})$ over the set of memory values $\boldsymbol{M}$ where $\mathbf{P} = \{P_i \mid i \in \mathbf{I}\}$ is an $\mathbf{I}$ indexed collection of pushdown systems $P_i = (Q_i, \Gamma_i, \mathscr{O}_{\boldsymbol{M}}, \delta_i, s_i)$, for the rest of this section. For the moment, consider one stage runs where $p \in \mathbf{I}$ identifies the writer. Suppose we are

[1] We plan to use "automata" instead of systems when they are used as language generators and to avoid ambiguity with the components of the SCPS.

interested in the existence of one stage runs starting at the configuration $((s_i)_{i\in\mathbf{I}},(\rho_i)_{i\in\mathbf{I}},\boldsymbol{m})$ and ending at some configuration $((q_i)_{i\in\mathbf{I}},(\gamma_i)_{i\in\mathbf{I}},\boldsymbol{m'})$. Now, consider the languages L_i, $i\in\mathbf{I}$ (recognising values of memory reads of a reader process), defined as, if $i\neq p$ then

$$L_i=\{\boldsymbol{m_1 m_2\ldots m_n}\mid \boldsymbol{?m_1?m_2\ldots?m_n}\in L(P_i,(s_i,\rho_i),(q_i,\gamma_i))\}$$

and L_p (recognising values of memory writes of a writer process) given by

$$\{\boldsymbol{m.m_1.m_2\ldots m_n.m'}\mid \boldsymbol{?m^*!m_1?m_1{}^*!m_2\ldots!m_n?m_n{}^*!m'?m'^*}\in L(P_p,(s_p,\rho_p),(q_p,\gamma_p))\}$$

Then, the existence of an one stage run from $((s_i)_{i\in\mathbf{I}},(\rho_i)_{i\in\mathbf{I}},\boldsymbol{m})$ to $((q_i)_{i\in\mathbf{I}},(\gamma_i)_{i\in\mathbf{I}},\boldsymbol{m'})$ (with w as the writer) is equivalent to the non-emptiness of

$$St(L_p)\downarrow\cap\bigcap_{i\neq p}L_i\uparrow$$

Moreover, the languages $St(L_p)\downarrow$ and $L_i\uparrow$ can easily be realized as the languages of PDAs A_p and A_i constructed from the PDSs P_p and P_i respectively. These automata maintain the stack and control state of the PDS they simulate as well.

We will first show the construction for a single stage setting and then extend it to multiple stages. We will fix our SCPS as $S=(\mathbf{I},\mathbf{P},\boldsymbol{m_0})$ over the memory domain $\boldsymbol{M}$. In the single stage setting that we consider, we will index the writer process using p and the reader processes by $r_1,\cdots,r_n$. We will show the construction of pushdowns systems A_p and $A_{r_1}\cdots A_{r_n}$, such that checking existence one stage run can be reduced to intersection on these systems.

Construction of pushdown automata A_p corresponding to writer process

$A_p=(P_p,\Gamma,\boldsymbol{M},\delta'_p,s^0_p)$ where

- $P_p=Q_p\times\boldsymbol{M}$, where Q_p is states of the process P_p in S
- $s^0_p=(q^0_p,\boldsymbol{m_0})$, where q^0_p is the initial state of process P_p in S.
- The transition relation δ'_p is defined as below. Along with the transition description, we will also provide a mapping $\boldsymbol{g}:\delta'_p\mapsto\delta_p\cup\{\epsilon\}$, where δ_p is the transitions of process P_p in S.

  a.1 For any memory read transition of the form $\tau=(q,\mathrm{op},\boldsymbol{?m},q')\in\delta_p$ where $\mathrm{op}\in\cup_{a\in\Gamma}\{\mathbf{Push}(a),\mathbf{Pop}(a)\}\cup\{\mathbf{Zero},\mathbf{Int}\}$, we add the transition $\tau'=((q,\boldsymbol{m}),\mathrm{op},\epsilon,(q',\boldsymbol{m}))\in\delta'_p$, we will let $\boldsymbol{g}(\tau')=\tau$.

  a.2 For any memory write transition of the form $\tau=(q,\mathrm{op},\boldsymbol{!m},q')\in\delta_p$ where $\mathrm{op}\in\cup_{a\in\Gamma}\{\mathbf{Push}(a),\mathbf{Pop}(a)\}\cup\{\mathbf{Zero},\mathbf{Int}\}$, we add for all $\boldsymbol{m'}\in\boldsymbol{M}$, the transitions $\tau'=((q,\boldsymbol{m'}),\mathrm{op},\epsilon,(q',\boldsymbol{m}))\in\delta'_p$, and $\tau'=((q,\boldsymbol{m'}),\mathrm{op},\boldsymbol{m},(q',\boldsymbol{m}))\in\delta'_p$ (such a pair allows memory writes to be nondeterministically made visible or not, hence ensuring downward closure), we let $\boldsymbol{g}(\tau')=\tau$.

  a.3 We add also add for all $q\in Q_p$, $\boldsymbol{m}\in\boldsymbol{M}$, the transition $\tau=((q,\boldsymbol{m}),\mathbf{Int},\boldsymbol{m},(q,\boldsymbol{m}))\in\delta'_p$, this allows stuttering of the memory value, we let $\boldsymbol{g}(\tau)=\epsilon$

  Note that by construction, if $w\in L(A_p,c)$ for some configuration c, then any w' that is a subword of w is also in this language (i.e. language of this pushdown automata is downward closed).

Construction of pushdown automata A_{r_i} corresponding to reader process P_{r_i}

$A_{r_i} = (P_{r_i}, \Gamma, \boldsymbol{M}, \delta'_{r_i}, s^0_{r_i})$ where

- $P_{r_i} = Q_{r_i} \times \boldsymbol{M}$, where Q_{r_i} is states of the process P_{r_i} in S
- $s^0_{r_i} = (q^0_{r_i}, \boldsymbol{m_0})$, where $q^0_{r_i}$ is the initial state of process P_{r_i} in S.
- The transition relation δ'_{r_i} is defined as below. Along side the transition description, we will also provide a mapping $\boldsymbol{g} : \delta'_{r_i} \mapsto \delta_{r_i} \cup \{\epsilon\}$, where δ_{r_i} is the transitions of process P_{r_i} in S.

  b.1 For any memory read transition of the form $\tau = (q, \text{op}, \boldsymbol{?m}, q') \in \delta_{r_i}$ where op $\in \cup_{a\in\Gamma}\{\textbf{Push}(a), \textbf{Pop}(a)\} \cup \{\textbf{Zero}, \textbf{Int}\}$, we add the transition $\tau' = ((q, \boldsymbol{m}), \text{op}, \boldsymbol{m}, (q', \boldsymbol{m})) \in \delta'_{r_i}$, we will let $\boldsymbol{g}(\tau') = \tau$.

  b.2 We add also add for all $q \in Q_p$, $\boldsymbol{m}, \boldsymbol{m'} \in \boldsymbol{M}$, the transition $\tau = ((q, \boldsymbol{m}), \textbf{Int}, \boldsymbol{m'}, (q, \boldsymbol{m'})) \in \delta'_p$, this allows upward closure of the memory value, we let $\boldsymbol{g}(\tau) = \epsilon$

Note that for any word w and any configuration c if $w \in L(A_{r_i}, c)$ then any word w' such that w is a subword of w' is also in the language. Now we will like to prove that the existence of a single stage run can be reduced to language intersection problem in the pushdown automata $A_p, A_{r_1}, \cdots A_{r_n}$.

Lemma 3. *There is a* 1*-phase run of the form* $(\boldsymbol{s_0}, \boldsymbol{\bot}, \boldsymbol{m_0}) \rightarrow^* (\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m})$ *in* S *iff there is a word* $w \in \boldsymbol{M}^*$ *such that* $w \in L((s^0_p, \bot), ((\boldsymbol{q}(p), \boldsymbol{m}), \boldsymbol{\gamma}(p))) \cap \bigcap_{i\in[1\ldots n]} L((s^0_{r_i}, \bot), ((\boldsymbol{q}(r_i), \boldsymbol{m}), \boldsymbol{\gamma}(r_i)))$

Proof. ($\Rightarrow$)

We will prove by induction on length of the run that for any 1-phase run of the form $(\boldsymbol{s_0}, \boldsymbol{\bot}, \boldsymbol{m_0}) \rightarrow^* (\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m})$ in S, we can find runs of the form $(s^0_p, \bot) \xrightarrow{w}{}^* ((\boldsymbol{q}(p), \boldsymbol{m}), \boldsymbol{\gamma}(p)))$ and $(s^0_{r_i}, \bot) \xrightarrow{w}{}^* ((\boldsymbol{q}(r_i), \boldsymbol{m}), \boldsymbol{\gamma}(r_i))$ for each $i \in [1 \ldots n]$.

- For base case, consider a zero length run. This trivially follows from a run on ϵ.
- Case where $(\boldsymbol{s_0}, \boldsymbol{\bot}, \boldsymbol{m_0}) \rightarrow^* (\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m})$ is of length greater than 1, then we can clearly split such a run into $(\boldsymbol{s_0}, \boldsymbol{\bot}, \boldsymbol{m_0}) \rightarrow^* (\boldsymbol{q'}, \boldsymbol{\gamma'}, \boldsymbol{m'}) \xrightarrow{\boldsymbol{\tau}} (\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m})$. Now by induction, we have runs of the form

$$(s^0_p, \bot) \xrightarrow{w'}{}^* ((\boldsymbol{q'}(p), \boldsymbol{m'}), \boldsymbol{\gamma'}(p)) \text{ and } \forall\, i \in [1..n] (s^0_{r_i}, \bot) \xrightarrow{w'}{}^* ((\boldsymbol{q'}(r_i), \boldsymbol{m'}), \boldsymbol{\gamma'}(r_i))$$

  We will consider various possibilities for τ and show that in each case, we can extend the run as required.

  – Case where τ is a memory read transition. We have two cases to consider, the transition is that of the writer ($\tau \in \delta_p$) or it is that of a reader ($\tau \in \delta_{r_i}$).

    * Let $\tau = (\boldsymbol{q'}(p), \textbf{Int}, \boldsymbol{?m}, \boldsymbol{q}(p)) \in \delta_p$, then notice that $\boldsymbol{m} = \boldsymbol{m'}$ and that there is a memory read transition of the form $((\boldsymbol{q'}(p), \boldsymbol{m}), \textbf{Int}, \epsilon, (\boldsymbol{q}(p), \boldsymbol{m})) \in \delta'_p$ by a.1. From this we get

$$(s^0_p, \bot) \xrightarrow{w'}{}^* ((\boldsymbol{q}(p), \boldsymbol{m}), \boldsymbol{\gamma}(p))$$

      Also notice that for any reader process r_i, $\boldsymbol{q}(r_i) = \boldsymbol{q'}(r_i)$ and $\boldsymbol{\gamma}(r_i) = \boldsymbol{\gamma'}(r_i)$. Hence by induction we have

$$(s^0_{r_i}, \bot) \xrightarrow{w'}{}^* ((\boldsymbol{q}(r_i), \boldsymbol{m}), \boldsymbol{\gamma}(r_i))$$

      For cases where τ is an operation other than an internal move, the argument is similar.

* Let $\tau = (\boldsymbol{q'}(r_i), \mathbf{Int}, \mathbf{?}\boldsymbol{m}, \boldsymbol{q}(r_i)) \in \delta_{r_i}$, for some $r_i \in \mathbf{I}$. Notice that in this case, $\boldsymbol{m'} = \boldsymbol{m}$ and we have $((\boldsymbol{q'}(r_i), \boldsymbol{m'}), \mathbf{Int}, \boldsymbol{m'}, (\boldsymbol{q}(r_i), \boldsymbol{m'})) \in \delta'_{r_i}$ by b.1. From this, we get

$$(s^0_{r_i}, \perp) \xrightarrow{w'\boldsymbol{m'}}{}^{*} ((\boldsymbol{q}(r_i), \boldsymbol{m'}), \boldsymbol{\gamma}(r_i))$$

For process p, we have the stuttering transition from a.3, and $\boldsymbol{q}(p) = \boldsymbol{q'}(p)$, $\boldsymbol{\gamma}(p) = \boldsymbol{\gamma'}(p)$. From this we get

$$(s^0_p, \perp) \xrightarrow{w'\boldsymbol{m'}}{}^{*} ((\boldsymbol{q}(p), \boldsymbol{m'}), \boldsymbol{\gamma}(p))$$

For all other $r_j \neq r_i$, we have the upward closure transition from b.2. From this and the fact that $\boldsymbol{q}(r_j) = \boldsymbol{q'}(r_j)$, $\boldsymbol{\gamma}(r_j) = \boldsymbol{\gamma'}(r_j)$, the result follows.

– Case where τ is a memory write transition, let $\tau = (\boldsymbol{q'}(p), \mathbf{Int}, \mathbf{!}\boldsymbol{m}, \boldsymbol{q}(p)) \in \delta_p$. Notice that $((\boldsymbol{q'}(p), \boldsymbol{m'}), \mathbf{Int}, \boldsymbol{m}, (\boldsymbol{q}(p), \boldsymbol{m})) \in \delta'_p$ from a.2.

$$(s^0_p, \perp) \xrightarrow{w'\boldsymbol{m}}{}^{*} ((\boldsymbol{q}(p), \boldsymbol{m}), \boldsymbol{\gamma}(p))$$

Further we have for all $r_i \in \mathbf{I}$, the upward closure move of the form $((\boldsymbol{q'}(r_i), \boldsymbol{m'}), \mathbf{Int}, \boldsymbol{m}, (\boldsymbol{q}(r_i), \boldsymbol{m})) \in \delta'_{r_i}$ from b.2. From this, we can easily get the required runs.

($\Leftarrow$)

Before going to prove this direction, we will introduce some notations and claims that we will use later. For the writer we will say that $(q_p, \gamma_p, m) \overset{w}{\rightsquigarrow} (q'_p, \gamma'_p, m')$ for some $w = m_1 m_2 \cdots m_n m'$ iff there is a collection of runs involving only the transitions of p, of the form

$$(\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m}) \xrightarrow{(\mathbf{?}\boldsymbol{m}^*)\mathbf{!}\boldsymbol{m}_1(\mathbf{?}\boldsymbol{m}_1{}^*)}{}^{*} (\boldsymbol{q_1}, \boldsymbol{\gamma_1}, \boldsymbol{m_1}), (\boldsymbol{q'_1}, \boldsymbol{\gamma'_1}, \boldsymbol{m_1}) \xrightarrow{\mathbf{!}\boldsymbol{m}_2(\mathbf{?}\boldsymbol{m}_2{}^*)}{}^{*} (\boldsymbol{q_2}, \boldsymbol{\gamma_2}, \boldsymbol{m_2})$$
$$\xrightarrow{\mathbf{!}\boldsymbol{m}_3(\mathbf{?}\boldsymbol{m}_3{}^*)}{}^{*} \cdots \xrightarrow{\mathbf{!}\boldsymbol{m}_n(\mathbf{?}\boldsymbol{m}_n{}^*)}{}^{*} (\boldsymbol{q_n}, \boldsymbol{\gamma_n}, \boldsymbol{m_n}) \xrightarrow{\mathbf{!}\boldsymbol{m'}(\mathbf{?}\boldsymbol{m'}{}^*)}{}^{*} (\boldsymbol{q'_p}, \boldsymbol{\gamma'_p}, \boldsymbol{m'})$$

such that $\boldsymbol{q}(p) = q_p, \boldsymbol{\gamma}(p) = \gamma_p, \boldsymbol{q'}(p) = q'_p, \boldsymbol{\gamma'}(p) = \gamma'_p$ and for each $i \in [1 \ldots n]$, $\boldsymbol{q'_i}(p) = \boldsymbol{q_i}(p)$ and $\boldsymbol{\gamma'_i}(p) = \boldsymbol{\gamma_i}(p)$. We will call such runs a fractured run.

Similarly for any reader $r \in \mathbf{I}$, we say that $\pi'_r = (q_r, \gamma_r, m) \overset{w}{\rightsquigarrow} (q'_r, \gamma'_r, m')$ for some $w = \boldsymbol{m_0}.\boldsymbol{m_1}.\boldsymbol{m_2} \cdots \boldsymbol{m_n}$, where $\boldsymbol{m_0} = \boldsymbol{m}$ and $\boldsymbol{m_n} = \boldsymbol{m'}$, iff there is a run involving only the transitions of r, of the form

$$(\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m}) \xrightarrow{(\mathbf{?}\boldsymbol{m}^*)}{}^{*} (\boldsymbol{q_1}, \boldsymbol{\gamma_1}, \boldsymbol{m}), (\boldsymbol{q'_1}, \boldsymbol{\gamma'_1}, \boldsymbol{m_1}) \xrightarrow{(\mathbf{?}\boldsymbol{m}_1{}^*)}{}^{*} (\boldsymbol{q_2}, \boldsymbol{\gamma_2}, \boldsymbol{m_1}), \cdots,$$
$$(\boldsymbol{q_n}, \boldsymbol{\gamma_n}, \boldsymbol{m'}) \xrightarrow{(\mathbf{?}\boldsymbol{m'}{}^*)}{}^{*} (\boldsymbol{q'}, \boldsymbol{\gamma'}, \boldsymbol{m'})$$

such that $\boldsymbol{q}(r) = q_r, \boldsymbol{\gamma}(r) = \gamma_r, \boldsymbol{q'}(r) = q'_r, \boldsymbol{\gamma'}(r) = \gamma'_r$ and for each $i \in [1 \ldots n]$, $\boldsymbol{q'_i}(r) = \boldsymbol{q_i}(r)$ and $\boldsymbol{\gamma'_i}(r) = \boldsymbol{\gamma_i}(r)$.

We will also use the following Claim in our proof.

Claim 2. *Suppose there is a fractured run of the writer of the form $\pi_p = (q_p, \gamma_p, \boldsymbol{m}) \overset{w}{\rightsquigarrow} (q'_p, \gamma'_p, \boldsymbol{m'})$, set of fractured runs one per readers (for each $i \in [1 \ldots n]$) of the form $\pi_{r_i} = (q_{r_i}, \gamma_{r_i}, \boldsymbol{m}) \overset{w_{r_i}}{\rightsquigarrow} (q'_{r_i}, \gamma'_{r_i}, \boldsymbol{m'})$. Further, for each r_i, if there is a monotonic map of the form $\boldsymbol{h}_{r_i} : [1 \ldots |w_{r_i}|] \mapsto$*

$[0\ldots|w|]$, *from positions of* w_{r_i} *to positions of* w *such that if* $\boldsymbol{h}_{r_i}(j) = i$ *then we have* $w_{r_i}[j] = w[i]$ *(except for* $h_{r_i}(i) = 0$*, in which case* $w[i] = \boldsymbol{m}$ *), then there is an combined* 1*-stage run in* S *of the form* $(\boldsymbol{q},\boldsymbol{\gamma},\boldsymbol{m}) \rightarrow^* (\boldsymbol{q'},\boldsymbol{\gamma'},\boldsymbol{m'})$*, where* $\boldsymbol{q}(p) = q_p, \boldsymbol{q}(r_i) = q_{r_i}, \boldsymbol{q'}(p) = q'_p, \boldsymbol{q'}(r_i) = q'_{r_i}$, $\boldsymbol{\gamma}(p) = \gamma_p, \boldsymbol{\gamma}(r_i) = \gamma_{r_i}, \boldsymbol{\gamma'}(p) = \gamma'_p, \boldsymbol{\gamma'}(r_i) = \gamma'_{r_i}$.

Proof. (idea)

Let us assume that $w = a_1 a_2 \cdots a_n$. Since there is a monotonic map $\boldsymbol{h}_{r_i}$ from each w_{r_i} to w, it is easy to see that each of these w_{r_i} is of the form $\boldsymbol{m}^* a_{j_1}^* a_{j_2}^* \cdots a_{j_{n_i}}^*$ for some $a_{j_1} a_{j_2} \cdots a_{j_i} \preceq w$. Further $j_1, j_2, \cdots, j_{n_i}$ are the positions into which w_{r_i} maps, via $\boldsymbol{h}_{r_i}$. Hence w_{r_i} can be split as $v_0 v_1 \cdots v_{n_i}$ according to memory values to which they map.

Let σ_{r_i} be the sequence of transitions used in π_{r_i}. Now σ_{r_i} can be split as $\sigma_{r_i} = \sigma_0^{r_i} \sigma_1^{r_i} \cdots \sigma_{n_i}^{r_i}$ such that $\Sigma(\sigma_j^{r_i}) = v_j$.

The global run can now be obtained by first executing sequence of the form $\sigma_0^{r_i}$ that are mapped to position 0 followed by sequence of transitions that generate a_1, followed by sequence of transitions that are mapped to this position (if any), followed by transition that generates a_2 and so on.

□

Coming back to proof of Lemma 3, since we are given that $w \in L(((s_p^0, \boldsymbol{m_0}), \perp), ((q_p, \boldsymbol{m}), \gamma_p))$, then clearly there is a run of the form $((s_p^0, \boldsymbol{m_0}), \perp) \xrightarrow{w}{}^* ((q_p, \boldsymbol{m}), \gamma_p)$. Let $\tau_1^p \tau_2^p \cdots \tau_m^p$ be the sequence of transitions executed in such a run.

Similarly since for all $r \in \mathbf{I}$, we have $w \in L(((s_r^0, \boldsymbol{m_0}), \perp), ((q_r, \boldsymbol{m}), \gamma_r))$, we have a run of the form $((s_r^0, \boldsymbol{m_0}), \perp) \xrightarrow{w}{}^* ((q_r, \boldsymbol{m}), \gamma_r)$. Let $\tau_1^r \tau_2^r \cdots \tau_{m_r}^r$ be the sequence of transitions in such a run.

We will let $\sigma_p = g(\tau_1^p) g(\tau_2^p) \cdots g(\tau_m^p)$ and $v_p = \Sigma(\sigma_p)$. Clearly such a σ_p gives us a run of the form $((s_p^0, \boldsymbol{m_0}), \perp) \overset{v_p}{\rightsquigarrow} ((q_p, \boldsymbol{m}), \gamma_p)$. Let $w = \boldsymbol{m_0}^{n_0} \boldsymbol{m_1}^{n_1} \boldsymbol{m_2}^{n_2} \cdots \boldsymbol{m_k}^{n_k} \boldsymbol{m}^{n_m}$, for some $n_0 \geq 0$ and $n_1, \cdots, n_k, n_m \geq 1$. Note that w is in stuttering downward closure of v_p. This follows from the fact that A_p is similar to P_p, except that it can stutter the memory writes or lose them. From this, we get that v_p can be broken up as $v_p = v_0 \boldsymbol{m_1} v_1 \boldsymbol{m_2} \cdots v_k \boldsymbol{m} v_{k+1}$. Let $v' = \boldsymbol{m_0} v_p$, it is easy to see that there is a natural monotonic map $\boldsymbol{f}$ from positions of w to positions in v' such that if $f(i) = j$ then $w[i] = v'[j]$.

Now for any reader r, let $\sigma_r = g(\tau_1^r) g(\tau_2^r) \cdots g(\tau_{m_{r_i}}^r)$. Let $v_r = \Sigma(\sigma_r)$, it is easy to see that v_r is a subword of w. This follows from the fact that A_r is similar to P_r except that it might read arbitrary values (using the upward closure transitions) without changing configurations. Since σ_r contains transitions of the SCPS S sans the upward closure moves that were added to A_r, we have a local move in SCPS of the form $((s_r^0, \boldsymbol{m_0}), \perp) \overset{v_r}{\rightsquigarrow} ((q_r, \boldsymbol{m}), \gamma_r)$. Further there is a natural monotonic map $\boldsymbol{h}$ from positions of v_r to positions of w such that if $h(i) = j$ then $v_r[i] = w[j]$. Composing these two maps provides us a monotonic map from positions of v_r to positions in v_p. Now using Claim 2, we can get the required run in the SCPS. This completes the proof of Lemma 3.

□

We are however interested in k stage runs where the identity of the writer (and hence the closures to be applied) changes with the stage. We will now show how to extend the above

construction to k stage setting. For this, we will fix the sequence of writers in each stage as $\tau \in \mathbf{I}^k$.

Let $(\boldsymbol{s}, \perp, \boldsymbol{m_0})$ and $(\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m})$ be the initial and target configurations of the SCPS and we wish to determine if there is a k stage run consistent with τ that goes from the initial to the target configuration. We will show how to construct a pushdown automaton A_i^τ that simulates P_i, where its runs break up into k parts, where in the jth part it applies either a stuttering downward closure or upward closure to the behaviour of P_i depending on whether $j = \tau(j)$ or not. We will show that the reachability in the SCPS can be reduced to checking if intersection of the language of these pushdown systems is empty or not. For $i \in [1 \ldots k]$, let $\boldsymbol{M}_i = \boldsymbol{M}^* \cdot (\boldsymbol{M} \times \{i\})$.

1. For any $j \leq k$ if $\tau(j) = i$ then we add the following set of write transitions.

   a.1 For any memory read transition of the form $\tau = (q, \text{op}, \boldsymbol{?m}, q') \in \delta_i$ where op $\in \cup_{a\in\Gamma}\{\mathbf{Push}(a), \mathbf{Pop}(a)\} \cup \{\mathbf{Zero}, \mathbf{Int}\}$, we add the transition $((q, j, \boldsymbol{m}), \text{op}, \epsilon, (q', j, \boldsymbol{m})) \in \delta'_p$.

   a.2 For any memory write transition of the form $\tau = (q, \text{op}, \boldsymbol{!m}, q') \in \delta_p$ where op $\in \cup_{a\in\Gamma}\{\mathbf{Push}(a), \mathbf{Pop}(a)\} \cup \{\mathbf{Zero}, \mathbf{Int}\}$, we add for all $\boldsymbol{m'} \in \boldsymbol{M}$, the transitions $((q, j, \boldsymbol{m'}), \text{op}, \epsilon, (q', j, \boldsymbol{m})) \in \delta'_p$, and $((q, j, \boldsymbol{m'}), \text{op}, \boldsymbol{m}, (q', j, \boldsymbol{m})) \in \delta'_p$ (such a pair allows memory writes to be nondeterministically made visible or not, hence ensuring downward closure).

   a.3 We also add for all $q \in Q_p$, $\boldsymbol{m} \in \boldsymbol{M}$, the transition $((q, j, \boldsymbol{m}), \mathbf{Int}, \boldsymbol{m}, (q, j, \boldsymbol{m})) \in \delta'_p$, this allows stuttering of the memory value.

2. For any $j \leq k$ if $\tau(j) \neq i$ then we add the following set of read transitions.

   b.1 For any memory read transition of the form $\tau = (q, \text{op}, \boldsymbol{?m}, q') \in \delta_{r_i}$ where op $\in \cup_{a\in\Gamma}\{\mathbf{Push}(a), \mathbf{Pop}(a)\} \cup \{\mathbf{Zero}, \mathbf{Int}\}$, we add the transition $((q, j, \boldsymbol{m}), \text{op}, \boldsymbol{m}, (q', j, \boldsymbol{m})) \in \delta'_{r_i}$.

   b.2 We add also add for all $q \in Q_p$, $\boldsymbol{m}, \boldsymbol{m'} \in \boldsymbol{M}$, the transition $((q, j, \boldsymbol{m}), \mathbf{Int}, \boldsymbol{m'}, (q, j, \boldsymbol{m'})) \in \delta'_p$, this allows upward closure of the memory value.

3. In addition, we have for all $q \in Q_i$, $m \in \boldsymbol{M}$ and $j \in [1..k-1]$, the transitions $((q, m, j), \mathbf{Int}, (m.i), (q, m, j+1)) \in \delta_i^\tau$. These set of transitions are used to synchronise the initial memory value at the beginning of each stage.

Using arguments similar to Lemma 3, we can easily prove the following lemma which relates a 1 stage run in the SCPS with a common subword of the pushdown systems constructed.

Lemma 4. *For any $j \in [1..k]$, there is a word $w \in \boldsymbol{M}^*$ such that $w \in \bigcap_{i\in\mathbf{I}} L_{A_i^\tau}(((\boldsymbol{q}(i), \boldsymbol{m}, j), \boldsymbol{\gamma}(i)), ((\boldsymbol{q'}(i), \boldsymbol{m'}, j), \boldsymbol{\gamma'}(i)))$ iff there is a 1-phase run of the form $(\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m}) \rightarrow^* (\boldsymbol{q'}, \boldsymbol{\gamma'}, \boldsymbol{m'})$ in S*

We now prove the following lemma. This lemma relates the emptiness checking of language intersection in the constructed pushdowns with a run in SCPS.

Lemma 5. *For every $p \in \mathbf{I}$, we can construct, in polynomial time in $|S|$, a PDA A_p^τ over the stack alphabet Γ_p, such that, for $c_p = (((\boldsymbol{q}(p), m), k), \boldsymbol{\gamma}(p)), p \in \mathbf{I}$, we have*

1. If $w \in L(A_p^\tau, c_p)$ then $w \in \boldsymbol{M}_1 \cdot \boldsymbol{M}_2 \cdots \boldsymbol{M}_{k-1}.\boldsymbol{M}^*$. (unambiguous breakup)

2. *If $w \in L(A^\tau_p, c_p)$ with $w = w_1 w_2 \dots w_k$, $w_1 \in \boldsymbol{M}^*_1, \dots, w_{k-1} \in \boldsymbol{M}^*_{k-1}$ and $w_k \in \boldsymbol{M}^*$, then for all $w'_1 \in \boldsymbol{M}^*_1, \dots, w'_{k-1} \in \boldsymbol{M}^*_{k-1}$ and $w'_k \in \boldsymbol{M}^*$ such that, either $p = \tau(i)$ and $w'_i \in St(w_i)\downarrow$ or $p \neq \tau(i)$, and $w'_i \in w_i\uparrow$, we have $w'_1.w'_2.\dots.w'_k \in L(A^\tau_p, c_p)$. (closure)*
3. *There is a k stage run from $(\boldsymbol{s}, \perp, \boldsymbol{m_0})$ to $(\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m})$ with $\tau(i)$ as the writer in the i^{th} stage iff $\bigcap_{p\in\mathbf{I}} L(A^\tau_p, c_p) \neq \emptyset$. (decoupling)*

Proof.

1. For each $p \in \mathbf{I}$, if $w \in L(A^\tau_p, c_p)$ then we have a run of the form

$$((s(p), \boldsymbol{m_0}, 1), \perp) \xrightarrow{w}{}^* ((q(p), \boldsymbol{m}, k), \gamma(p))$$

It is easy to see that such a run can be expanded as,

$$\begin{aligned}((s(p), \boldsymbol{m_0}, 1), \perp) &\xrightarrow{w_1}{}^* ((q_1, \boldsymbol{m_1}, 1), \gamma_1) \overset{(\boldsymbol{m_1},1)}{\rightarrow} ((q_1, \boldsymbol{m_1}, 2), \gamma_1) \\ &\xrightarrow{w_2}{}^* ((q_2, \boldsymbol{m_2}, 2), \gamma_2) \overset{(\boldsymbol{m_2},2)}{\rightarrow} ((q_2, \boldsymbol{m_2}, 3), \gamma_2) \xrightarrow{w_3}{}^* \cdots \\ &\overset{(\boldsymbol{m_{k-1}},k-1)}{\rightarrow} ((q_{k-1}, m_{k-1}, k), \gamma_{k-1}) \xrightarrow{w_k}{}^* ((q(p), \boldsymbol{m}, k), \gamma(p))\end{aligned}$$

From this, we know that $w = w_1.(\boldsymbol{m_1}, 1).w_2.(\boldsymbol{m_2}, 2)\dots(\boldsymbol{m_{k-1}}, k).w_k \in \boldsymbol{M}_1 \cdot \boldsymbol{M}_2 \cdots \boldsymbol{M}_{k-1}.\boldsymbol{M}^*$
2. For this, we will first prove the following Claim.

Claim 3. *For any $j \in [1..k]$, and for any $p \in \mathbf{I}$, if $((q, \boldsymbol{m}, j), \gamma) \xrightarrow{w}{}^* ((q', \boldsymbol{m'}, j), \gamma')$, then the following holds.*

a) If $\tau(j) = p$ then for all w' such that $w' \in w\downarrow$, we have a run of the form $((q, \boldsymbol{m}, j), \gamma) \xrightarrow{w'}{}^* ((q', \boldsymbol{m'}, j)\gamma')$
b) If $\tau(j) \neq p$ then for all w' such that $w' \in w\uparrow$, we have a run of the form $((q, \boldsymbol{m}, j), \gamma) \xrightarrow{w'}{}^* ((q', \boldsymbol{m'}, j), \gamma')$

Proof.

a) We will prove this by induction on length of w. Base case is when $w = \epsilon$. In this case, we have nothing to do. For the induction case, we will consider $|w| > 0$. In this case, we will let $w = v.\boldsymbol{m'}$ for some $\boldsymbol{m'} \in \boldsymbol{M}$. Now for any $w' \in w\downarrow$, it is the case that $w' = v'\boldsymbol{m'}$ such that $v' \in v\downarrow$ or $w' \in v\downarrow$. Let the run on w be of the form

$$((q, \boldsymbol{m}, j), \gamma) \xrightarrow{v}{}^* ((q'', \boldsymbol{m''}, j), \gamma'') \overset{\boldsymbol{m'}}{\rightarrow} ((q', \boldsymbol{m'}, j), \gamma')$$

Let the transition used to generate $\boldsymbol{m'}$ be of the form $\tau = ((q'', \boldsymbol{m''}, j), \mathbf{Int}, \boldsymbol{m'}, (q', \boldsymbol{m'}, j))$ (we will only consider this case, the other cases are similar). Now for any $w' \in w\downarrow$, if $w' \in v\downarrow$, by induction we have a run of the form

$$((q, \boldsymbol{m}, j), \gamma) \xrightarrow{v'}{}^* ((q'', \boldsymbol{m''}, j), \gamma'')$$

Combining this with transition of the form, $((q'', \boldsymbol{m''}, j), \mathbf{Int}, \epsilon, (q', \boldsymbol{m'}, j))$ available in a.2, we get the required run

$$((q,\boldsymbol{m},j),\gamma)\xrightarrow{v'}{}^{*}((q'',\boldsymbol{m''},j),\gamma'')\xrightarrow{\epsilon}((q',\boldsymbol{m'},j),\gamma')$$

For case where $w'=v'a$, with $v'\in v\downarrow$, we use the transition τ to extend the run got from induction.

b) This case is tedious but very similar to proof above , hence we will skip the same.

□

Now coming back to proof of Lemma 5, since we have $w\in L(A_p^{\tau},c_p)$, there is a run of the form

$$\begin{aligned}((s(p),\boldsymbol{m_0},1),\perp)\xrightarrow{w_1}{}^{*}((q_1,\boldsymbol{m_1},1),\gamma_1)&\overset{(\boldsymbol{m_1},1)}{\rightarrow}((q_1,\boldsymbol{m_1},2),\gamma_1)\\ &\xrightarrow{w_2}{}^{*}((q_2,\boldsymbol{m_2},2),\gamma_2)\overset{(\boldsymbol{m_2},2)}{\rightarrow}((q_2,\boldsymbol{m_2},3),\gamma_2)\xrightarrow{w_3}{}^{*}\cdots\\ &\overset{(\boldsymbol{m_{k-1}},k-1)}{\rightarrow}((q_{k-1},\boldsymbol{m_{k-1}},k),\gamma_{k-1})\xrightarrow{w_k}{}^{*}((q(p),\boldsymbol{m},k),\gamma(p))\end{aligned}$$

Now using Claim 3, we can replace any w_i in the above run by w_i', where $w_i'\in w_i\downarrow$ if $\tau(i)=p$ and $w_i'\in w_i\uparrow$ otherwise. From this, we get the required result.

3. ($\Rightarrow$) We are given that there is a k-stage run of the form

$$(\boldsymbol{s},\perp,\boldsymbol{m_0})\rightarrow^{*}(\boldsymbol{q},\boldsymbol{\gamma},\boldsymbol{m})$$

Such a run can be split as follows

$$\begin{aligned}c_0=(\boldsymbol{s},\perp,\boldsymbol{m_0})\rightarrow^{*}c_1=(\boldsymbol{q_1},\boldsymbol{\gamma_1},\boldsymbol{m_1})\rightarrow^{*}c_2=(\boldsymbol{q_2},\boldsymbol{\gamma_2},\boldsymbol{m_2})\cdots\\ c_{k-1}=(\boldsymbol{q_{k-1}},\boldsymbol{\gamma_{k-1}},\boldsymbol{m_{k-1}})\rightarrow^{*}c_k=(\boldsymbol{q},\boldsymbol{\gamma},\boldsymbol{m})\end{aligned}$$

Where each $c_i\rightarrow^{*}c_{i+1}$ is a single stage. Now using Lemma 4, we have runs of the form

$$\begin{aligned}((\boldsymbol{s}(p),1,\boldsymbol{m_0}),\perp)\rightarrow^{*}((\boldsymbol{q_1}(p),1,\boldsymbol{m_1}),\boldsymbol{\gamma_1}(p)),\\ (\boldsymbol{q_i}(p),i,\boldsymbol{m_i}),\boldsymbol{\gamma_i}(p))\rightarrow^{*}(\boldsymbol{q_{i+1}}(p),i,\boldsymbol{m_{i+1}}),\boldsymbol{\gamma_{i+1}}(p)),\\ (\boldsymbol{q_k}(p),k,\boldsymbol{m_k}),\boldsymbol{\gamma_k}(p))\rightarrow^{*}(\boldsymbol{q}(p),k,\boldsymbol{m}),\boldsymbol{\gamma}(p))\end{aligned}$$

Combining these runs with transitions from 3, we get the required run.

$\Leftarrow$ Since $\bigcap_{p\in\mathbf{I}}L(A_p^{\tau},c_p)\neq\emptyset$, there is a w such that $w\in\bigcap_{p\in\mathbf{I}}L(A_p^{\tau},c_p)$. Notice that such a w can be split as $w=w_1.\boldsymbol{(m_1,1)}w_2\boldsymbol{(m_2,2)}\cdots w_k$. From this, for each $p\in\mathbf{I}$, we have runs of the form

$$\begin{aligned}((s(p),\boldsymbol{m_0},1),\perp)\xrightarrow{w_1}{}^{*}((q_1(p),\boldsymbol{m_1},1),\gamma_1(p))&\overset{(\boldsymbol{m_1},1)}{\rightarrow}((q_1(p),\boldsymbol{m_1},2),\gamma_1(p))\\ &\xrightarrow{w_2}{}^{*}((q_2(p),\boldsymbol{m_2},2),\gamma_2(p))\cdots\overset{(\boldsymbol{m_{k-1}},k-1)}{\rightarrow}((q_{k-1}(p),\boldsymbol{m_{k-1}},k),\gamma_{k-1}(p))\\ &\xrightarrow{w_k}{}^{*}((q(p),\boldsymbol{m},k),\gamma(p))\end{aligned}$$

From this we get that

- $w_1 \in \bigcap_{p\in\mathbf{I}} L(((s(p), \boldsymbol{m_0}, 1), \bot), ((q_1(p), \boldsymbol{m_1}, 1), \gamma_1(p)))$
- $w_i \in \bigcap_{p\in\mathbf{I}} L(((q_i(p), \boldsymbol{m_i}, i+1), \gamma_i(p)), ((q_{i+1}(p), \boldsymbol{m_{i+1}}, i+1), \gamma_{i+1}(p)))$, for all $i \in [1..n-1]$
- $w_k \in \bigcap_{p\in\mathbf{I}} L(((q_{k-1}(p), \boldsymbol{m_{k-1}}, k), \gamma_{k-1}), ((q(p), \boldsymbol{m}, k), \gamma(p)))$.
  Now applying Lemma 4, we get the following sub-computations of SCPS.
  - $(\boldsymbol{s}, \bot, \boldsymbol{m_0}) \rightarrow^* (\boldsymbol{q_1}, \boldsymbol{\gamma_1}, \boldsymbol{m_1})$
  - $(\boldsymbol{q_i}, \boldsymbol{\gamma_i}, \boldsymbol{m_i}) \rightarrow^* (\boldsymbol{q_{i+1}}, \boldsymbol{\gamma_{i+1}}, \boldsymbol{m_{i+1}})$, for all $i \in [1..n-1]$
  - $(\boldsymbol{q_{k-1}}, \boldsymbol{\gamma_{k-1}}, \boldsymbol{m_{k-1}}) \rightarrow^* (\boldsymbol{q}, \boldsymbol{\gamma}, \boldsymbol{m})$.

  Now combining these runs, gives us the required run. This completes the proof of Lemma 5.

□

Remark: *Note that in our construction we fix a-priori a stage sequence τ and then constructed the pushdown automata A_i^τ (for all $i \in \mathbf{I}$). It is however possible to eliminate the need to fix the stage sequence a-priori. This can be done by letting each process guess a writer at the beginning of each stage and synchronising this guess using an input letter. Thus, we can construct a pushdown automata A_i without needing to fix a stage sequence apriori.*

In the second step we exploit the fact that the language of each A_p^τ is a finite unambiguous concatenation of languages that are upward or downward closed. Towards this we first state two propositions which explain the importance of closures.

Proposition 6 (Downward closure of CFLs [54, 15])**.** *Given a pushdown automaton P and two configurations c_i, c_f, we can construct, in time and space at most exponential in size of P, c_i and c_f, a FSA A with two configurations c_i' and c_f' such that $L(A, c_i', c_f') = L(P, c_i, c_f)\downarrow$.*

Proposition 7 (Upward closure of CFLs [15])**.** *Given a pushdown automaton P and two configurations c_i, c_f, we can construct, in time and space at most exponential in size of P, c_i and c_f, a FSA A with two configurations c_i' and c_f' such that $L(A, c_i', c_f') = L(P, c_i, c_f)\uparrow$.*

This means that, if we are dealing with a single stage then we may replace the PDA A_i, $i \in \mathbf{I}$, described earlier, by exponential sized finite automata B_i, $i \in \mathbf{I}$ (for all i, including the writer p). Thus we have reduced the problem to the emptiness of the intersection for FAs. However the k stage case is somewhat more complex. This is because, as A_i^τ switches from one stage to the next, it has to preserve the configuration of P_i (i.e. the contents of the stack) as well as the contents of the memory. While this is trivial when A_i^τ is a pushdown, it is not possible to do this using finite number of states. However, all is not lost as we may convert A_i^τ into a $2k$-*reversal bounded PDA* B_i^τ. Recall that a run of pushdown automaton is said to be 1-*reversal* if the stack height of the sequence of configurations is either uniformly non-increasing (does not involve a push move) or non-decreasing (does not involve a pop move). A k-*reversal* run is concatenation of k sequences of 1-*reversal* runs. A k-*reversal* bounded PDA is one which only allows at most k-*reversal* runs.

Lemma 6. *For every $p \in \mathbf{I}$, it is possible to construct, in exponential time in the size of A_p^τ, a $2k$ reversal bounded PDA B_p^τ and a configuration c_p', such that $L(B_p^\tau, c_p') = L(A_p^\tau, c_p)$.*

Proof. We first fix a pushdown automaton $A = (Q, \Gamma, \Sigma, \delta, s)$ and show that for any two given configurations $c = (q, \alpha), c' = (q', \beta)$ the closure language $Cl(L(c, c'))$ can be accepted by a 2-reversal bounded automata B (whenever type of closure is not important, we will use $Cl()$ to refer to either the downward or the upward closure). For any run ρ, we say it is a γ-run, $\gamma \in \Gamma^*.\perp$ if γ is the longest common suffix of the stack of every configuration along the run of ρ. We write $\gamma(c, c')$ to refer to the set of words accepted on γ-runs from c to c'. We will represent a γ-run from c to c' over a word w as $c \xrightarrow{w}{}^*{}_\gamma c'$. Let $c = (q, \rho)$ to $c' = (q', \rho')$. Then, $L(c, c')$, the set of words accepted on runs from c to c' is

$$\{x.y \mid x \in \gamma(c, (q'', \gamma)), y \in \gamma((q'', \gamma), c'), \text{ where } \gamma \text{ is a suffix of } \rho \text{ and } q'' \in Q\}$$

For each $\alpha \in \Gamma$ and $q_1, q_2 \in Q$, we let

$$\begin{aligned} L^-_\alpha(q_1, q_2) = &\quad \{w \mid (q_1, \alpha\perp) \xrightarrow{w}{}^* (q_2, \perp) \text{ without using emptiness tests }\} \\ L^+_\alpha(q_1, q_2) = &\quad \{w \mid (q_1, \perp) \xrightarrow{w}{}^* (q_2, \alpha\perp) \text{ without using emptiness tests }\} \\ L^\perp(q_1, q_2) = &\quad \{w \mid (q_1, \perp) \xrightarrow{w}{}^* (q_2, \perp)\} \end{aligned}$$

We can see that the language $\gamma(c, (q'', \gamma))$ (resp. $\gamma((q'', \gamma), c')$) can be rewritten as $\bigcup_{q_1, q_2 \cdots, q_{\ell-1} \in Q} L^-_{\alpha_1}(q, q_1) \cdot L^-_{\alpha_2}(q_1, q_2) \cdots L^-_{\alpha_\ell}(q_{\ell-1}, q'') \cdot L$ and with $\rho = \alpha_1\alpha_2\cdots\alpha_\ell\gamma$ (resp. $\bigcup_{q_1, q_2 \cdots, q_{\ell'-1} \in Q} L \cdot L^+_{\alpha'_1}(q'', q_1) \cdot L^+_{\alpha'_2}(q_1, q_2) \cdots L^+_{\alpha'_{\ell'}}(q_{\ell'-1}, q')$ with $\rho' = \alpha'_{\ell'}\alpha'_{\ell'-1}\cdots\alpha'_1\gamma$) and $L = \{\epsilon\}$ if $\gamma \neq \perp$ and $L = L^\perp(q'', q'')$ otherwise.

Hence, any word $w \in Cl(L(c, c'))$ can be rewritten as the concatenation of three words (i.e., $w = w_1w_2w_3$). The first word w_1 is in $Cl(L^-_{\alpha_1}(q, q_1)) \cdot Cl(L^-_{\alpha_2}(q_1, q_2)) \cdots Cl(L^-_{\alpha_\ell}(q_{\ell-1}, q''_1))$ for some states $q_1, \cdots q_{\ell-1} \in Q$, letters $\alpha_1\alpha_2\cdots\alpha_\ell$ and stack content γ such that $\rho = \alpha_1\alpha_2\cdots\alpha_\ell\gamma$. The second word w_2 is in $Cl(L^\perp(q''_1, q''_2))$ if $\gamma = \perp$, and in $\{\epsilon\}$ otherwise. The last word w_3 is in $Cl(L^+_{\alpha'_1}(q''_2, q'_1)) \cdot Cl(L^+_{\alpha'_2}(q'_1, q'_2)) \cdots Cl(L^+_{\alpha'_{\ell'}}(q'_{\ell'-1}, q'))$ for some some states $q'_1, \cdots q'_{\ell'} \in Q$, letters $\alpha'_1\alpha'_2\cdots\alpha'_{\ell'}$ such that $\rho' = \alpha'_{\ell'}\alpha'_{\ell'-1}\cdots\alpha'_1\gamma$. Note that such a run has at most 3 reversals.

Lemma 7. *Given a pushdown automaton $P = (Q, \Gamma, \Sigma, \delta, s)$, we can construct a 2-reversal bounded PDA $B = (Q', \Gamma, \Sigma, \delta', s')$ such that for any two configurations $c = (q, \rho), c' = (q', \rho') \in \mathscr{C}(P)$, $L(B, ((q, -), \rho), ((q', +), \rho')) = Cl(L(A, c, c'))$, where $(q, +), (q, -) \in Q'$.*

Proof. The languages $L^-_\alpha(q_1, q_2)$, $L^+_\alpha(q_1, q_2)$ and $L^\perp(q_1, q_2)$ are context-free and their upward and downward closures are effectively regular (see Propositions 6,7) and so let $B^-_\alpha(q_1, q_2)$, $B^+_\alpha(q_1, q_2)$ and $B^\perp(q_1, q_2)$ be finite state automata recognising $Cl(L^-_\alpha(q_1, q_2))$, $Cl(L^+_\alpha(q_1, q_2))$ and $Cl(L^\perp(q_1, q_2))$ respectively. We let S subscripted with automata to indicate the states of automata eg. $S_{B^-_\alpha(q_1, q_2)}$ to be the states of $B^-_\alpha(q_1, q_2)$. Similarly, we let $Initial, Final, \delta$ subscripted with the automata to indicate initial state, final state and transitions respectively.

As observed earlier, any word $w \in Cl(L(A, c, c'))$ can be rewritten as the concatenation of three words (i.e., $w = w_1w_2w_3$). The first word w_1 is in $B^-_{\alpha_1}(q, q_1) \cdot B^-_{\alpha_2}(q_1, q_2) \cdots B^-_{\alpha_\ell}(q_{\ell-1}, q''_1)$ for some letters $\alpha_1\alpha_2\cdots\alpha_\ell$ and stack content γ such that $\rho = \alpha_1\alpha_2\cdots\alpha_\ell\gamma$. The second word w_2 is in $B^\perp(q''_1, q''_2)$ if $\gamma = \perp$, and in $\{\epsilon\}$ (with $q''_1 = q''_2$) otherwise. The last word w_3

is in $B^+_{\alpha'_1}(q''_2,q'_1)\cdot B^+_{\alpha'_2}(q'_1,q'_2)\cdots B^+_{\alpha'_{\ell'}}(q'_{\ell'-1},q')$ for some letters $\alpha'_1\alpha'_2\cdots\alpha'_{\ell'}\in\Gamma$ such that $\rho'=\alpha'_{\ell'}\alpha'_{\ell'-1}\cdots\alpha'_1\gamma$. Note that such a run has 3 phases namely decreasing phase, zero phase and increasing phase. Hence our state space consists of states of the regular automata recognising the closure languages along with states of A tagged with phase information. The state space of B is given by $Q\times\{+,-,\perp\}\cup\bigcup_{p,p'\in Q,\alpha\in\Gamma}S_{B^-_\alpha(p,p')}\cup S_{B^+_\alpha(p,p')}\cup S_{B^\perp(p,p')}$. The initial state is given by $(s,-)$ since we always start in a decreasing phase.

In the automata we construct, we intend to simulate the increasing, decreasing and zero phases. Hence we need to add the transitions of these automata to PDS that we construct. Note that the syntax of finite state automata differs from that of PDA. Hence for any $(q,a,q')\in\bigcup_{p,p'\in Q,\alpha\in\Gamma}\delta_{B^-_\alpha(p,p')}\cup\delta_{B^+_\alpha(p,p')}\cup\delta_{B^\perp(p,p')}$ we add the transition $(q,\mathbf{Int},a,q')$ to δ'.

The phase in our automata can transition from decreasing to zero and zero to increasing phase, hence we add for all $p\in Q$, the set of transitions $((p,-),\mathbf{Int},\epsilon,(p,\perp))$, $((p,\perp),\mathbf{Int},\epsilon,(p,+))$. During the decrease phase, the automaton B from the current state (say $(p,-)$) has to guess a return state (say $(p',-)$), pops the top of stack (say α) , simulate the automaton $B^-_\alpha(p,p')$ and finally returns to state $(p',-)$. Hence we include in the transition $((p,-),\mathbf{Pop}(\alpha),\epsilon,Initial_{B^-_\alpha(p,p')})$ and $(Final_{B^-_\alpha(p,p')},\mathbf{Int},\epsilon,(p',-))$. Similarly, we add the transitions $((p,\perp),\mathbf{Zero},\epsilon,Initial_{B^\perp(p,p')})$ and $(Final_{B^\perp(p,p')},\mathbf{Int},\epsilon,(p',\perp))$ and $((p,+),\mathbf{Push}(\alpha),\epsilon,Initial_{B^+_\alpha(p,p')})$ and $(Final_{B^+_\alpha(p,p')},\mathbf{Int},\epsilon,(p',+))$ corresponding to the simulation of zero phase and increase phase. Clearly such a construction allows at most 2-reversals, one during decrease phase and another during the increase phase.
We prove the correctness of the construction below.

- $Cl(L(A,c,c'))\subseteq L(B,((q,-),\rho),((q',+),\rho'))$
  For every $u\in CL(L(A,c,c'))$, we will now show that $u\in L(B,((q,-),\rho),((q',+),\rho'))$. Let $u\in CL(L(A,c,c'))$, then there is a $w\in L(A,c,c')$ such that $c\xrightarrow{w}{}^*c'$ (with $u\in CL(w)$). Clearly such a w can be split into words recognised in decreasing phase w_1, the word recognised in zero phase w_2 and the word recognised in increasing phase w_3. Clearly there is a γ run $(q,\rho)\xrightarrow{w_1}{}^*{}_\gamma(q''_1,\gamma)\xrightarrow{w_2}{}^*{}_\gamma(q''_2,\gamma)\xrightarrow{w_3}{}^*{}_\gamma(q',\rho')$. Note that now u can also be split as $u_1.u_2.u_3$ where for $i\in[1..3]$, $u_i\in Cl(w_i)$. Let $\rho=\alpha_1.\alpha_2.\cdots.\alpha_n.\gamma$, then $(q,\rho)\xrightarrow{w_1}{}^*{}_\gamma(q''_1,\gamma)$ can be represented as

  $$(q,\alpha_1.\alpha_2.\cdots.\alpha_n.\gamma)\xrightarrow{w_1^1}{}^*{}_\gamma(q_1,\alpha_2.\cdots.\alpha_n.\gamma)\xrightarrow{w_1^2}{}^*{}_\gamma\cdots\xrightarrow{w_1^n}{}^*{}_\gamma(q''_1,\gamma)$$

  i.e. the execution can be split into series of runs with one symbol from the stack popped each time. Clearly $u_1=u_1^1.u_1^2\cdots u_1^n$ with $u_1^i\in Cl(w_1^i)$. By definition of $B^-_\alpha(p,p')$, it recognises the closure of words of run from p with α on top of stack and ending at p' with α popped (note that the zero test is not allowed). Clearly $u_1\in B^-_{\alpha_1}(q,q_1)$ and $u_i\in B^-_{\alpha_i}(q_{i-1},q_i)$ and $u_n\in B^-_{\alpha_n}(q_{n-1},q''_1)$. Since we have a transition in our construction of B from state $(q,-)$ on popping α_1 to $Initial_{B^-_{\alpha_1}(q,q_1)}$ and from $Final_{B^-_{\alpha_1}(q,q_1)}$ to $(q_1,-)$, we have a run of the form

  $$((q,-),\alpha_1.\alpha_2.\cdots.\alpha_n.\gamma)\xrightarrow{u_1^1}{}^*(q_1,-),\alpha_2.\cdots.\alpha_n.\gamma)$$

  Reasoning similarly, we can find a corresponding run

  $$((q,-),\alpha_1.\alpha_2.\cdots.\alpha_n.\gamma)\xrightarrow{u_1^1}{}^*((q_1,-),\alpha_2.\cdots.\alpha_n.\gamma)\xrightarrow{u_1^2}{}^*\cdots\xrightarrow{u_1^n}{}^*((q''_1,-),\gamma)$$

Similarly, we can find corresponding runs $((q_1'',\perp),\gamma)\xrightarrow{u_2}{}^*((q_2'',\perp),\gamma)$ and $((q_2'',+),\gamma)\xrightarrow{u_3}{}^*((q',+),\rho')$. This along with the fact that from a decreasing phase we can transition to zero phase and from zero phase to increasing phase, it is easy to see that there is a run in B such that $((q,-),\rho)\xrightarrow{u}{}^*((q',+),\rho'))$.

- $L(B,((q,-),\rho),((q',+),\rho'))\subseteq Cl(L(A,c,c'))$
  We will now show that for every $u\in L(B,((q,-),\rho),((q',+),\rho'))$, we can find a $w\in L(A,c,c')$ such that $u\in Cl(w)$. Since $u\in L(B,((q,-),\rho),((q',+),\rho'))$, u can be split up as the word recognised in decreasing, zero and increasing phases i.e. $u=u_1.u_2.u_3$ such that

$$(q,-),\rho)\xrightarrow{u_1}{}^*((q_1'',-),\gamma)\xrightarrow{\epsilon}((q_1'',\perp),\gamma)\xrightarrow{u_2}{}^*((q_2'',\perp),\gamma)\xrightarrow{\epsilon}((q_2'',+),\gamma)$$
$$\xrightarrow{u_3}{}^*((q',+),\rho')\ \text{[with } u_2=\epsilon \text{ and } q_1''=q_2'' \text{ if } \gamma\neq\perp\text{]}$$

  Let $\rho=\alpha_1.\alpha_2.\cdots.\alpha_n.\gamma$, then it is easy to see that the decreasing phase can be split up as

$$((q,-),\alpha_1.\alpha_2.\cdots.\alpha_n.\gamma)\xrightarrow{u_1^1}{}^*((q_1,-),\alpha_2.\cdots.\gamma)\xrightarrow{u_1^2}{}^*((q_2,-),\alpha_3.\cdots.\gamma)\cdots$$
$$((q_{n-1},-),\alpha_n.\cdots.\gamma)\xrightarrow{u_1^n}{}^*((q_1'',-),\gamma)$$

  where $u_1^1\in L(B^-_{\alpha_1}(q,q_1))$, $u_1^i\in L(B^-_{\alpha_i}(q_{i-1},q_i))$ and $u_1^n\in L(B^-_{\alpha_n}(q_{n-1},q''))$. By construction, for any $v\in B^-_\alpha(p,p')$ there is a run from $(p,\alpha.\gamma)\xrightarrow{w}{}^*(p',\gamma)$ in A with $v\in Cl(w)$ (since there is no zero test, there is a γ run for any γ). Hence there is a run $(q,\rho)\xrightarrow{w_1}{}^*(q_1'',\gamma)$ in A. Using similar argument we can find runs $(q_1'',\gamma)\xrightarrow{w_2}{}^*(q_2'',\gamma)$ ($w_2=\epsilon$ if $\gamma\neq\perp$) and $(q_2'',\gamma)\xrightarrow{w_3}{}^*(q',\rho')$ in A, with $u_2\in Cl(w_2)$ and $u_3\in Cl(w_3)$. From this we have that $w=w_1.w_2.w_3\in L(A,c,c')$ and $u\in Cl(w)$.

□

With the above lemma in place, we are ready to construct B_p^τ from a given $A_p^\tau=(P,\Gamma,\Sigma,\delta,s)$ such that for any give configuration $c=(q,\gamma)$ of A_p^τ, $L(A_p^\tau,c)=L(B_p^\tau,((q,+),\gamma))$. Note that the states of A_p^τ are of the form $(q,\boldsymbol{m},i)$ i.e. are tagged with memory and the stage information. We let $S_i=\{(q,\boldsymbol{m},i)\mid(q,\boldsymbol{m},i)\in P\}$ i.e. the set of all states that are tagged as stage i. The idea is to first construct a 2 reversal bounded automata for each stage. For this purpose, we construct $A_p^{\tau,i}$ by restricting the states and transition to S_i. i.e. $A_p^{\tau,i}=(S_i,\Gamma,\Sigma,\delta_i,s)$, $s\in S_i$ is any state, the initial state is not important as we will see later, let $\delta_i=\delta\cap(S_i\times\mathbf{Op}\times\Sigma_\epsilon\times S_i)$ i.e. the transitions restricted to stage-i states. Now for such a PDA, by Lemma 7 we can construct a 2-reversal bounded PDA $B_p^{\tau,i}$ such that $Cl(L(A_p^{\tau,i},(q,\rho),(q',\rho')))=L(B^{\tau,i},((q,-),\rho),((q',+),\rho'))$. In fact due to the nature of construction of A_p^τ, if $\tau(i)=p$ then we have that $St(L(A_p^{\tau,i},(q,\rho),(q',\rho')))\downarrow=L(A_p^{\tau,i},(q,\rho),(q',\rho'))$ and if $\tau(i)\neq p$ then $L(A_p^{\tau,i},(q,\rho),(q',\rho'))\uparrow=L(A_p^{\tau,i},(q,\rho),(q',\rho'))$. The idea now is to concatenate appropriate closures of $B_p^{\tau,i}$ for each i. We choose $B_p^{\tau,i}$ to be downward closed when $\tau(i)=p$ and upward closed otherwise. The B_p^τ automaton is now simply defined as union of $B_p^{\tau,i}$ automata for $i\in[1..k]$ along with additional transitions. The states of B_p^τ are union of states of $B_p^{\tau,i}$, the transitions of B_p^τ are the union of transitions of $B_p^{\tau,i}$, in addition to stage change transitions of the form

$(((q,\boldsymbol{m},i),+),\mathbf{Int},(m,i),((q,\boldsymbol{m},i+1),-))$. i.e. for every state-$i$ in increasing phase, we allow it to transition into the next stage $i+1$ in decreasing phase. Note that in such a construction the stack content remains the same across the stage boundaries. With this and Lemma 7, the correctness of the construction is not difficult to see. This completes the proof of Lemma 6

□

Unfortunately, the emptiness of the intersection of even two 2-*reversal bounded* PDAs is undecidable, as can be seen from an easy reduction from the Post's correspondence problem (PCP). The situation is quite different when the PDAs are counters. In fact, we can show:

Lemma 8. *Let k be a natural number. Let A_1 be a $2k$ turn PDA and $A_2,\ldots,A_n$ a sequence of $2k$-reversal bounded counter automata. Let c_i be a configurations of A_i for all $i : 1 \le i \le n$. Then, the problem of checking whether $L(A_1,c_1)\cap\cdots\cap L(A_n,c_n)$ is not empty can be decided in nondeterministic time that is polynomial in the size of A_i, c_i and k, and exponential in n.*

For proving Lemma 8, we will first define PDS with *reversal restricted* counter. We will then show that given a $2k$-reversal bounded PDA and $(n-1)$ number of $2k$-reversal bounded counter automata, deciding whether intersection of languages recognised by these system is empty or not is decidable by reducing it to the reachability problem of a PDS equipped with $3k$ reversal restricted counters.

Definition 3 (Pushdown with reversal-restricted counters)**.** *Let n,k be two natural numbers, $\beta(i) = \{inc(i), dec(i), zero(i) \mid i \in [1..n]\}$, and $\tilde{\beta}(n) = \bigcup_{i=1}^{n}\beta(i)$. Let $A = (Q,\Gamma,\Sigma\cup\tilde{\beta}(n),\delta,s)$ be a PDS equipped with counters. For every $i \in [1..n]$, we say that a run $\rho = (q,\gamma)\xrightarrow{w}{}^{*}{}_{A}(q',\gamma')$ is* (k, i)-reverse run *iff the following conditions hold:*

- $w\downarrow_{\beta(i)} \in St(w_i)$ *for some* $w_i \in (\beta(i))^*$ *such that* $|w_i| \le k$,
- $|w[1..j]\downarrow_{\{dec(i)\}}| \le |w[1..j]\downarrow_{\{inc(i)\}}|$ *for all* $j \in [1..|w|]$, *and*
- $|w[1..j]\downarrow_{\{dec(i)\}}| = |w[1..j]\downarrow_{\{inc(i)\}}|$ *for all* $j \in [1..|w|]$ *such that* $w(j) = zero(i)$.

For every subset $J \subseteq [1..n]$, the run ρ is (k, J)-reverse run *iff it is* (k, i)-reverse *for all $i \in J$. We use $L_{(k,J)}(A,c)$ to denote the set of words w such that there is (k, J)-reverse run of the form* (s, $\perp$) $\xrightarrow{w}{}_{A}\, c$. *For any ($k$, J)-reverse run of the form* $(s,\perp)\xrightarrow{w}{}_{A}\, c$, *we will use $\rho_i(w)$ to indicate the current value of the counter-i. i.e. $\rho_i(w) = |w\downarrow_{\{inc(i)\}}| - |w\downarrow_{\{dec(i)\}}|$*

Reversal restricted reachability Problem: Given two natural numbers n,k and a PDS A and a configuration c, the (k,n)-reversal-restricted-reachability problem is to determine if there is a $(k,[1..n])$-*reverse run* of the form $(s,\perp)\xrightarrow{w}{}^{*}{}_{A}c$. We will also refer to this simply as reachability on pushdown with k reversal restricted counters.

Lemma 9. *Let k be a natural number. Let A_1 be a $2k$ reversal bounded PDA and $A_2,\ldots,A_n$ a sequence of $2k$-reversal bounded counter automata. For $i \in [1..n]$, let $c_i = (q_i^f,\perp)$ be any configurations of A_i. Then, the problem of checking whether $L(A_1,c_1)\cap\cdots\cap L(A_n,c_n)$ is empty or not can be reduced to $3k$ reversal restricted reachability on a PDS with n counters. Furthermore, the size of P is polynomial in the size of A_i and k, and exponential in n.*

Proof. We will fix the $2k$-reversal bounded pushdown automaton to be $A = (Q_1, \Gamma_1, \Sigma, \delta_1, s_1)$ and the $2k$-reversal bounded counter automaton to be $A_i = (Q_i, \{a, \perp\}, \Sigma, \delta_i, s_i)$ for all $i : 2 \leq i \leq n$. We will now show how to construct $S = (Q, \Gamma, \Sigma \cup \tilde{\beta}(n), \Delta, \hat{s})$. The states of S will have the product of states of PDA and the counter automata along with states that can count up to n. i.e. $Q = (Q_1 \times \cdots \times Q_n) \cup (Q_1 \times \cdots \times Q_n \times [1..n] \times \Sigma)$. We need the states that count up to n in order to ensure that any move on input alphabet is made simultaneously by PDA and all the counter automata. The initial state of S, $\hat{s} = (s_1, \cdots, s_n)$ is the initial states of the pushdown automaton and the counter automata. The transition includes set of all epsilon moves of each of the PDA and the counter system along with synchronised moves on any input alphabets. For all $(q_1, \mathbf{Op}, \epsilon, q_1') \in \delta_1$, for all $q_i \in Q_i, i \in [2..n]$, we have the transitions

$$((q_1, q_2, \cdots, q_n), \mathbf{Op}, \epsilon, (q_1', q_2, \cdots, q_n)) \in \Delta$$

For any $(q_i, \mathbf{Op}, \epsilon, q_i') \in \delta_i$ with $i \neq 1$, we have for all $q_j \in Q_j$ such that $j \neq i$, the transitions

$$((q_1, \cdots, q_{i-1}, q_i, q_{i+1}, \cdots, q_n), \mathbf{Int}, x, (q_1, \cdots, q_{i-1}, q_i', q_{i+1}, \cdots, q_n)) \in \Delta$$

where $x = inc(i)$ if $\mathbf{Op} = \mathbf{Push}(a)$, $x = dec(i)$ if $\mathbf{Op} = \mathbf{Pop}(a)$, $x = zero(i)$ if $\mathbf{Op} = \mathbf{Zero}$ and $x = \epsilon$ otherwise.

The transition relations ensures that any move on input symbol is made synchronously by the pushdown automata and all the counter automatas. This is ensured using the counting states which moves from one counter to another and returns to normal state only after each of the counter automaton has executed a transition involving the symbol i.e. for every $b \in \Sigma$, every $(q_1, \mathbf{Op}, b, q_1') \in \delta_1$ and for all $q_i \in Q_i$ $(2 \leq i \leq n)$, we have

$$((q_1, q_2, \cdots, q_n), \mathbf{Op}, b, (q_1', q_2, \cdots, q_n, 2, b)) \in \Delta$$

For every $j \in \{2, ..., n-1\}$, $(q_j, \mathbf{Op}, b, q_j') \in \delta_j$ and for all $q_i \in Q_i$ with $i \neq j$, we have

$$(q_1, \cdots, q_j, \cdots, q_n, j, b), \mathbf{Int}, x_j, (q_1, \cdots, q_j', \cdots, q_n, j+1, b) \in \Delta$$

where $x_j = inc(j)$ if $\mathbf{Op} = \mathbf{Push}(a)$, $x_j = dec(j)$ if $\mathbf{Op} = \mathbf{Pop}(a)$, $x_j = zero(j)$ if $\mathbf{Op} = \mathbf{Zero}$ and $x_j = \epsilon$ otherwise

We also have the transition that takes us back to normal state once all the counter automata have executed a transition involving b. i.e. for all $q_i \in Q_i, i \in [1..n-1]$, $(q_n, \mathbf{Op}, b, q_n') \in \delta_n$ and for x_n defined as above, we have the transitions

$$((q_1, q_2, \cdots, q_n, n, b), \mathbf{Int}, x_n, (q_1, q_2 \cdots, q_n')) \in \Delta$$

Since each counter automaton allows only runs that are at most $2k$-reversals, it is easy to see that for any run of our system, if we project only the operations of a counter (say i), it can be written as concatenation of at most k sequences of the form $inc(i)^*.dec(i)^*.zero(i)^*$. Clearly such a sequence is at most 3 reversal restricted and hence the newly constructed system allows only runs that are at most $3k$ reversal restricted. Furthermore the correctness of our construction follows from the following straight forward Lemma.

Lemma 10. $((s_1,\cdots,s_n),\perp)\xrightarrow{u}{}^*((q_1^f,\cdots,q_n^f),\perp)$ *is a* $(k,[1..n])$-reverse run *in* S *with* $|u\downarrow_{\{inc(i)\}}| = |u\downarrow_{\{dec(i)\}}|$ *for all* $2\le i\le n$ *iff for all* $i\in\{1,...,n\}$, *we have* $(s_i,\perp)\xrightarrow{w}{}^*{}_{A_i}(q_i^f,\perp)$ *with* $w = u\downarrow_\Sigma$.

The decidability of checking whether there exists a $(k,[1..n])$-*reverse run* of the form $((s_1,\cdots,s_n),\perp)\xrightarrow{u}{}^*((q_1^f,\cdots,q_n^f),\perp)$ in S with $|u\downarrow_{\{inc(i)\}}| = |u\downarrow_{\{dec(i)\}}|$ for all $2\le i\le n$ follows from the following lemma. The lemma states that the problem of deciding whether there is a reversal restricted run to a specific configuration is NP-COMPLETE .

Proposition 8 (Reversal restricted reachability Problem [85])**.** *The reachability on pushdown with reversal restricted counters is* NP-COMPLETE.

□

Finally, Lemma 8 is an immediate consequence of Proposition 8 and Lemma 9. This finishes the proof of Theorem 5.

The complexity of such a construction is as follows: The complexity of the $|B|$ automaton constructed in Lemma 7 is exponential on the size of $|P|$, from this, the size of B_p^τ that we construct in Lemma 6 is exponential on the size of A_p^τ. Now in Lemma 9, the size of the reversal bounded system P that we construct is polynomial in size of B_p^τ and exponential on n. Such a P that we construct is specific to the τ we fixed earlier. There are n^k possible choices for τ. Hence the over all complexity for solving the bounded stage reachability problem is equivalent to solving the $3k$ reversal restricted reachability on a pushdown with n counters of size $O(n^k.|S|^{O(|S|.n)})$, where n is the number of processes, $|S| = \Sigma_{p\in\mathbf{I}}|P_p|$ and k is the number of stages. This gives us the NEXPTIME upper bound.

Towards showing lower bounds for the problem, we will reduce the problem of checking emptiness for the intersection of a collection of n finite automata (which is known to be PSPACE complete) to the n-stage bounded reachability problem for SCPS with n counters to obtain the following Theorem.

Theorem 9. *The stage-bounded reachability problem for SCPS consisting only of counter systems is* PSPACE-HARD.

Proof. Let $(A_i)_{1\le i\le n}$ be the given collection of FA. We use n counters C_1 to C_n and n stages. In the first stage the counter C_1 guesses a word w and writes it letter by letter on the memory (taking care to eliminate stuttering) while incrementing its counter by $n-1$ for each such letter. While doing this it also simulates the automaton A_1 on this word verifying that w is accepted by A_1. In this stage, each counter C_i, $2\le i\le n$, reads the values written on the memory by C_1 and verifies that the word w_i it reads is accepted by A_i. It also records the the length of w_i in its counter. Of course, $w_i\preceq w$ and so at the end of this stage, writing c_i for the value of counter i we have $c_1 = (n-1).|w|$ and $|w_i| = c_i\le c_1$ for each $2\le i\le n$.

In stage i, $2\le i\le n$, the counter C_i writes as many values as c_i to the shared memory and C_1 reads and reduces the value of its counter by the number of values it reads. The run is accepting only if C_1 is empty at the end of stage n. Notice that this may happen if and only if $c_i = c_1$ (and the communication in all the stages were loss-free) and thus $w_i = w$ for all $2\le i\le n$. Thus, the emptiness of the intersection reduces to the n-stage reachability in this SCPS. □

3.6 Conclusion

In this chapter, we introduced shared memory concurrent pushdown system and showed that for such models, even one bit shared memory is sufficient to simulate two counter system. We then went on to introduce a restriction called *stage bounding*. We showed that the stage bounding by itself is not enough to get decidability. We showed that two pushdowns and a counter system are enough to get undecidability for reachability under stage bounded restriction. We then showed that if we restrict ourselves to one pushdown and multiple counters, it is possible to decide the reachability problem in NEXPTIME. We first showed that it is possible to reduce the k stage bounded reachability problem on SCPS to language intersection of $2k$ reversal bounded counters automata with a pushdown automata. The size of such $2k$ reversal bounded counters automata that we constructed is exponential in the size of the SCPS. Later we show that deciding intersection of $2k$ reversal bounded counter system with a pushdown system can be reduced to reachability on pushdown with $3k$ reversal restricted counters, which is known to be NP-COMPLETE. The complexity of our construction is exponential in the size of the SCPS and on the number of processes. The exponential dependency on the size of the SCPS arises mainly because we use the exponential time algorithms available for computing the downward and upward closures of pushdown automata. This leads to an exponential sized pushdown with reversal bounded counter system that we construct. However, what we really need is an algorithm that works on counter automata. The question arises as to whether we can do away with this exponent by providing a more efficient downward and upward closures for counter automata. We will show in subsequent chapter that this is indeed possible.

Chapter 4

Regular abstractions of one counter automata

4.1 Introduction

A very well known result called the Higmans's Lemma [81], states that any upward closed language has only finite number of minimal elements under the subword relation. As an easy consequence we have that every upward closed language is regular and consequently every downward closed language is regular as well. Given a language L, a natural problem is then to construct a finite automaton for $L\uparrow$ (upward closure of L) and $L\downarrow$ (downward closure of L) from a finite representation of L. However, this may not always be possible. Emptiness of L is equivalent to the emptiness of $L\uparrow$ or $L\downarrow$ and thus such an effective construction cannot exist for any class for which emptiness is undecidable. Even for classes that have a decidable emptiness problem, the effective construction of such finite automata is an interesting problem.

Another abstraction that may be applied to a language is the Parikh image abstraction. Parikh image of a word $w \in \Sigma^*$ denoted Parikh(w) is a vector $v \in \mathbb{N}^{|w|}$ that counts the number of occurrences letters of Σ in w. The Parikh image of a language L, written Parikh(L) is the set of vectors containing the Parikh images of the words of L.

It has long been known that all three abstractions can be effectively computed for context-free languages (CFL), by the results of van Leeuwen [139] and by what is now referred to as the Parikh theorem [121]. For the Parikh image of CFLs, a number of constructions have been proposed as well [139, 75, 55, 28]; we refer the reader to the paper by Esparza, Ganty, Kiefer, and Luttenberger [62] for a survey and state-of-the-art results: exponential upper and lower bounds on the size of NFA for (L). Algorithms performing these tasks, as well as finite automata recognizing them, are now widely used as building blocks in the language-theoretic approach to verification. Recall that the downward and upward closures were used in chapter 3 for solving the bounded stage reachability problem over shared memory concurrent pushdown systems. There are also other places where computing upward and downward closures occurs as an ingredient in the analysis of systems communicating via shared memory, see, e.g.,[23, 20, 111]. The recent paper [99] shows that for parametrized networks of systems

communicating via shared memory, the decidability hinges on the ability to compute downward closures. Parikh-images as an abstraction in the verification of infinite state systems has been extensively used (see e.g.,[3, 92, 79, 61, 131, 22, 65, 72, 4]).

Effective constructions for the downward closure have been developed for Petri nets [76] and stacked counter automata [147]. The paper [148] gives a sufficient condition for a class of languages to have effective downward closures; this condition has since been applied to higher-order pushdown automata [78]. The effective regularity of the Parikh image is known for phase-bounded and scope-bounded multi-stack visibly pushdown languages [137, 102], and availability languages [4].

The family of languages recognised by one counter automata is more than the class of regular languages but less than the class of context-free languages. From verification perspective, the class of counter automata has proved to be an useful infinite-state model [12, 104]. In this chapter, we consider the complexity of these abstractions on languages accepted by one counter automata.

We first show how to obtain a polynomial sized NFA for $L\uparrow, L\downarrow$, when L is a language of a counter automata. While the construction of $L\uparrow$ is fairly straight forward, the construction of $L\downarrow$ is involved.

As an application, we consider the shared-memory concurrent pushdown systems that we saw in chapter 3. There we showed that the reachability of such systems in the bounded stage setting was NEXPTIME. Specifically, given an SCPS $S = (\mathbf{I}, \mathbf{P}, \boldsymbol{m_0})$, we reduce the k bounded stage reachability problem to a reversal bounded pushdown system, whose size is $O(n^k.|S|^{|S|.n})$, where $n = |\mathbf{I}|$. We show in this chapter on how to eliminate the exponential dependancy on the size of the system. Hence reducing the exponential dependency only on the number of processes.

We then consider the Parikh image abstractions for the languages of the class of counter automata. We provide an quasi-polynomial solution for this problem. Given a counter automata $\mathscr{A}$, we show how to construct a suitable NFA of size $O(|\Sigma|.|\mathscr{A}|^{O(log(|A|))})$. This construction proceeds in two steps. In the first step, we show that it is enough to restrict our attention to only runs with at most polynomially many reversals. The next step works for a reversal bounded pushdown automata as well. We show in this step, that a given reversal bounded pushdown system can be transformed in to another pushdown system (with the same Parikh image) with logarithmic bound on its stack size.

4.2 Counter automata

We first recall that the counter automata is defined as a tuple $C = (Q, \Sigma, \delta, s, F)$, where the transitions can be of the form $\delta \subseteq Q \times \{\mathbf{Int}, \mathbf{Dec}, \mathbf{Zero}, \mathbf{Inc}\} \times \Sigma_\epsilon \times Q$. We first show that in order to compute the upward, downward closure and the Parikh image abstraction, it is sufficient to compute it for only the positive runs. The intuitive explanation for this is, any run of a counter automata can be broken up as a positive part, followed by a zero test part, followed by a positive part, and so on. The part which only performs the zero test part does not require a counter. Hence if each of these positive parts and zero test parts can be abstracted as a finite state automata, then they can be stitched together to get the abstraction of the entire run. We

formalise this idea in the next subsection.

4.2.1 Simplified counter automata

We are interested in computing an efficient finite representation of the downward closure, the upward closure and the Parikh image abstraction for a counter automata. We first show that it suffices to consider only a subclass of counter automata called the *simplified counter automata*, which has the following properties

- There are no zero tests.
- There is a unique final state i.e. $F = \{f\}$.
- Only runs of the from $(s,0) \rightarrow^* (f,0)$ are considered accepting.

We will first prove that it suffices to consider closures on such simplified counter automata. Once we obtain an algorithm for this subclass, it can easily be extended with at-most polynomial blow up to compute closures on the general counter automata. Given a counter automata $C = (Q, \Sigma, \delta, s, F)$, we will let $L^+_{q,q'}(C)$ to be the set of all words accepted by a run from configuration $(q,0)$ to configuration $(q',0)$, not involving any zero test. We will in sequel show that for every pair of states $q, q' \in Q$, if there is an automata $B_{q,q'}$ such that $L(B_{q,q'}) = L^+_{q,q'}(C)\downarrow$, then there is an automata B such that $L(B) = L(C)\downarrow$. Further size of the automata is at most linear in size of $\Sigma_{q,q' \in Q}|B_{q,q'}|$ and size of C. Though we show this only for downward closure, such a Lemma can easily be extended to other abstractions (i.e. upwards closure and Parikh image).

Lemma 11. *Given a counter automata $C = (Q, \Sigma, \delta, s, F)$ and for every $q, q' \in Q$ an NFA $B_{q,q'}$, such that $L(B_{q,q'}) = L^+_{q,q'}(C)\downarrow$, we can construct an automata $B = (Q^B, \Sigma, \Delta^B, s, F)$ such that $L(B) = L(C)\downarrow$. Further $|B| = \Sigma_{q,q' \in Q}|B_{q,q'}| + |C|$*

Proof. Before we prove the Lemma, we will first introduce $\rightarrow^*{}_Z$ to mean a subcomputation of the form $c_1 \rightarrow^* c_2 \cdots \rightarrow^* c_n$ such that value of the counter in each of the configurations is 0. We will let $L^z_{q,q'} = \{w \mid (q,0) \xrightarrow{w}{}^*{}_Z (q',0)\}$ (words of runs not involving the stack). Now, let $w \in L(C)$, then it is easy to see that w can be split as $w = w_0.w_1.w_2 \cdots w_{2n}$ such that for $i \in [0..n]$, $w_{2i} \in L^z_{q_i,q_{i+1}}(C)$ and $w_{2i+1} \in L^+_{q_{2i+1},q_{2i+2}}(C)$ i.e. it can be split as alternating sequence of subwords, one involving no stack operation and the other involving no zero tests. It is also easy to see that $u \in w\downarrow$ iff $u \in w_0\downarrow .w_1\downarrow .w_2\downarrow \cdots w_{2n}\downarrow$. We have assumed that we have a NFA $B_{q,q'}$ such that $L^+_{q,q'}(C)\downarrow = L(B_{q,q'})$. An NFA for $L^z_{q,q'}(C)$ can be obtained by deleting from C all the moves that modify the counter.

Using these facts, we will formally describe the construction of B automata. For this purpose, for any $q, q' \in Q$ we will use $State(B_{q,q'}), Initial(B_{q,q'}), Final(B_{q,q'})$ and $\Delta(B_{q,q'})$ to refer to the states, initial state, final state and transitions of $B_{q,q'}$ respectively. Further we will assume that state space of each $B_{q,q'}$ is distinct.

- The states of B automata are given by $Q^B = Q \cup \bigcup_{q,q' \in Q} State(B_{q,q'})$
- The transition relations are defined as below.

1. For all $q, q' \in Q$, we have $\Delta(B_{q,q'}) \subseteq \Delta^B$.

2. For each transition of the form $(q, \mathbf{Int}, a, q') \in \Delta$ and $(q, \mathbf{Zero}, a, q') \in \Delta$, we have the transition $(q, a, q') \in \Delta^B$ and a transition for downward closure $(q, \epsilon, q') \in \Delta^B$.
3. We further have for all $q, q' \in Q$, the transitions $(q, \epsilon, \mathit{Initial}(B_{q,q'})) \in \Delta^B$ and $(\mathit{Final}(B_{q,q'}), \epsilon, q') \in \Delta^B$

The correctness of such a construction is easy to see, suppose $w \in L(B)$, then there is a run of the form $q_0 \xrightarrow{u_1}{}^* q_1 \xrightarrow{v_1}{}^* q_2 \xrightarrow{u_2}{}^* q_3 \cdots$ such that the transitions used in generating u_i's are from 2 and transitions used in generating v_i's are from 1 and 3. It is easy to see that each transition in the sequence $q_i \xrightarrow{u_i}{}^* q_{i+1}$ can easily be simulated by it corresponding transition in Δ, hence we have the run of the form $(q_i, 0) \xrightarrow{u_i'}{}^* (q_{i+1}, 0)$ where $u_i \preceq u_i'$. The runs of the form $q_i \xrightarrow{v_i}{}^* q_{i+1}$ actually look like $q_i \to \mathit{Initial}(B_{q_i,q_{i+1}}) \xrightarrow{v_i}{}^* \mathit{Final}(B_{q_i,q_{i+1}}) \to q_{i+1}$. Now by definition, corresponding to the run $\mathit{Initial}(B_{q_i,q_{i+1}}) \xrightarrow{v_i}{}^* \mathit{Final}(B_{q_i,q_{i+1}})$, there is a run $(q_i, 0) \xrightarrow{v_i'}{}^* (q_{i+1}, 0)$ such that $v_i \preceq v_i'$. Now combining these runs, we get the required run in counter automata.

For the other direction, suppose $w' \in L(C)\downarrow$, then there is a corresponding run in C of the form $\pi = (q_0, 0) \xrightarrow{w}{}^* (q_n, 0)$, such that $w' \preceq w$. Notice that such a run can be split as $(q_0, 0) \xrightarrow{u_1}{}^*{}_Z (q_1, 0) \xrightarrow{v_1}{}^* (q_2, 0) \xrightarrow{u_2}{}^*{}_Z (q_3, 0) \cdots (q_n, 0)$, where the runs $(q_i, 0) \xrightarrow{v_i}{}^* (q_{i+1}, 0)$ do not involve a zero test. Let $u_1', u_2', \cdots, u_n', v_1', v_2', \cdots, v_n'$ be such that $v_i' \preceq v_i$, $u_i' \preceq u_i$ and $u_1' v_1' u_2' v_2' \cdots u_n' v_n' = w$.

Clearly corresponding to runs of the form $(q_i, 0) \xrightarrow{v_i}{}^* (q_{i+1}, 0)$, for every $v_i'' \preceq v_i$ there is a run of the form $\mathit{Initial}(B_{q_i,q_{i+1}}) \xrightarrow{v_i''}{}^* \mathit{Final}(B_{q_i,q_{i+1}})$. This we get because $L(B_{q_i,q_{i+1}}) = L^+_{q_i,q_{i+1}}(C)\downarrow$. Hence there is a run of the form $\mathit{Initial}(B_{q_i,q_{i+1}}) \xrightarrow{v_i'}{}^* \mathit{Final}(B_{q_i,q_{i+1}})$ in B. Similarly, corresponding to runs of the form $(q_i, 0) \xrightarrow{u_i}{}^*{}_Z (q_{i+1}, 0)$, it is easy to see that we have a run of the form $q_0 \xrightarrow{u_1'}{}^* q_1$ in B. Now combining these runs with transitions in 3, we get the required joint run in B.

□

The above construction can be extended to other closures as well and hence, for the rest of the sections in this chapter, we will limit our attention to the subclass of simplified counter automata. In rest of the chapter, when we say counter automata, we mean a simplified counter automata (unless specified otherwise).

4.3 Computing upward closures

In this subsection, we will show that for any simplified counter automata $\mathscr{A}$, we can construct in polynomial time, a NFA that accepts $L(\mathscr{A})\uparrow$. This easy construction follows the argument traditionally used to bound the length of the shortest accepting run in the pushdown automata. We use the following notation in what follows: for a run ρ and an integer D we write $\rho[D]$ to refer to the run ρ' obtained from ρ by replacing the counter value v by $v + D$ in every configurations along the run.

Lemma 12. *Let $\mathscr{A} = (Q, \Sigma, \delta, s, F)$ be a counter automata and let w be a word accepted by $\mathscr{A}$. Then there is a word $y \preceq w$ in $L(\mathscr{A})$ such that y is accepted by a run where the value of the counter never exceeds $|Q|^2 + 1$.*

Proof. We show that for any accepting run ρ reading a word w, there is an accepting run ρ', reading a word $y \preceq w$, in which the maximum value of the counter does not exceed $|Q|^2 + 1$. We prove this by double induction on the maximum value of the counter and the number of times this value is attained during the run ρ.

If the maximum value is below $|Q|^2 + 1$ there is nothing to prove. Otherwise let the maximum value $m > |Q|^2 + 1$ be attained c times along ρ. We break the run up into segments $\rho = \rho_0\rho_1\rho_2\rho_3 \ldots \rho_m\rho'_{m-1} \ldots \rho'_2.\rho'_1.\rho'_0$ where

1. $\rho_0\rho_1\rho_2 \ldots \rho_m$ is the shortest prefix of ρ after which the counter attains the value m.
2. $\rho_0\rho_1\rho_2 \ldots \rho_i$ is the longest prefix of $\rho_0\rho_1\rho_2 \ldots \rho_m$ after which the counter value is i, $1 \leq i \leq m-1$.
3. $\rho_0\rho_1\rho_2 \ldots \rho_m.\rho'_{m-1} \ldots \rho'_i$, is the shortest prefix of ρ with $\rho_0\rho_1\rho_2 \ldots \rho_m$ as a prefix and after which the counter value is i, $0 \leq i \leq m-1$.

Let the configuration reached after the prefix $\rho_0 \ldots \rho_i$ be (p_i, i), for $1 \leq i \leq m$. Similarly let the configuration reached after the prefix $\rho_0\rho_1\rho_2 \ldots \rho_m.\rho'_{m-1} \ldots \rho'_i$ be (q_i, i), for $0 \leq i \leq m-1$.

Now we make two observations: firstly, the value of the counter never falls below i during the segment of the run $\rho_{i+1} \ldots \rho'_i$ — this is by the definition of the ρ_is and ρ'_is. Secondly, there are $i < j$ such that $p_i = p_j$ and $q_i = q_j$ — this is because $m \geq |Q|^2 + 1$. Together this means that we may shorten the run by deleting the sequence of transitions corresponding to the segment $\rho_{i+1} \ldots \rho_j$ leading from (p_i, i) to (p_j, j) and the sequence corresponding to the segment $\rho'_{j-1} \ldots \rho'_i$ from (q_j, j) to (q_i, i) and still obtain a valid run of the system. That is, $\rho_0\rho_1 \ldots \rho_i\rho_{j+1}[-m]\rho_{j+2}[-m] \ldots \rho'_j[-m]\rho'_{i-1} \ldots \rho'_0$ is a valid run, where $m = j - i$. Clearly the word accepted by such a run, say y' is a subword of w, and further this run has at least one fewer occurrance of the maximal counter value m. Thus the Lemma follows by applying the induction hypothesis to this run and using the fact that the subword relation is transitive. □

The set of words in $L(\mathscr{A})$ accepted along runs where the value of the counter does not exceed $|Q|^2 + 1$ is accepted by an NFA with $|Q|.(|Q|^2 + 1)$ states (it keeps the counter values as part of the state). Combining this with the standard construction for upward closure for NFAs we get

Theorem 10. *There is a polynomial-time algorithm that takes as input a counter automata $\mathscr{A} = (Q, \Sigma, \delta, s, F)$ and computes an NFA with $O(|\mathscr{A}|^3)$ states accepting $L(\mathscr{A})\uparrow$.*

4.4 Computing downward closures

Next we show a polynomial time procedure that constructs an NFA accepting downward closure of the language of a simplified counter automata.

First we extend the definition of configurations to values where the counters come from $\mathbb{Z}$ rather than $\mathbb{N}$. Formally, the set of transitions defines the one step move relation $\overset{\tau}{\Rightarrow}$ (with $\tau \in \delta$) on configurations as follows.

1. $\tau = (q, a, \mathbf{Int}, q')$. Then, $(q,n) \stackrel{\tau}{\Rightarrow} (q', n)$ for all $n \in \mathbb{Z}$. Internal move.
2. $\tau = (q, a, \mathbf{Dec}, q')$. Then, $(q,n) \stackrel{\tau}{\Rightarrow} (q', n-1)$ for all $n \in \mathbb{Z}$. Decrement move.
3. $\tau = (q, a, \mathbf{Inc}, q')$. Then, $(q,n) \stackrel{\tau}{\Rightarrow} (q', n+1)$ for all $n \in \mathbb{Z}$. Increment move.

This extends naturally to sequences of transitions: $(q,n) \stackrel{\epsilon}{\Rightarrow} (q,n)$ and $(q,n) \stackrel{\sigma.\tau}{\Longrightarrow} (q',n')$ if there is (q'',n'') such that $(q,n) \stackrel{\sigma}{\Rightarrow} (q'',n'')$ and $(q'',n'') \stackrel{\tau}{\Rightarrow} (q',n')$. We call this a *free run* on the sequence of transitions σ.

Remark: *We observe that if* $\rho = (q_0,n_0) \stackrel{a_1}{\Rightarrow} (q_1,n_1) \ldots, \stackrel{a_i}{\Rightarrow} (q_i,n_i) \ldots \stackrel{a_k}{\Rightarrow} (q_k,n_k)$ *and* m *is an integer, positive or negative, then* $\rho[m] = (q_0,n_0+m) \stackrel{a_1}{\Rightarrow} (q_1,n_1+m) \ldots, \stackrel{a_i}{\Rightarrow} (q_i,n_i+m) \ldots \stackrel{a_k}{\Rightarrow} (q_k,n_k+m)$ *is also a free run.*

Finally, note that any free run in which the counter values are always ≥ 0 is a run. We first prove a couple of useful lemmas that will lead us to our polynomial time construction.

Let $\mathscr{A} = (Q, \Sigma, \delta, s, F = \{f\})$ be any counter automata and let $K = |Q|$. Consider any run ρ of $\mathscr{A}$ from a configuration (p,i) to a configuration (q,j). If the value of the counter increases (resp. decreases) by at least K in ρ then, it contains a segment that can be *pumped* (or iterated) to increase (resp. decrease) the value of the counter. If the increase in the value of the counter in this iterable segment is k then by choosing an appropriate number of iterations we may increase the value of the counter at the end of the run by any multiple of this k. Quite clearly, the word read along this iterated run will be a superword of word read along ρ. The following Lemmas,whose proof is a simplified version of that of Lemma 12, formalize this.

Lemma 13. *Let* $(p,i) \xrightarrow{x}{}^* (q,j)$ *with* $j - i > |Q|$. *Then, there is an integer* $k > 0$ *such that for each* $N \geq 1$ *there is a run* $(p,i) \xrightarrow{w}{}^* (p', j + N.k)$, *where* $w = y_1.(y_2)^{N+1}.y_3$, *with* $x = y_1y_2y_3$.

Proof. Consider the run $(p,i) \xrightarrow{x}{}^* (q,j)$ and break it up as

$$(p,i) = (p_i,i) \xrightarrow{x_1}{}^* (p_{i+1}, i+1) \xrightarrow{x_2}{}^* (p_{i+2}, i+2) \ldots \xrightarrow{x_j}{}^* (p_j, j) \xrightarrow{x'}{}^* (q,j)$$

where the run $(p_i,i) \xrightarrow{x_1}{}^* \ldots \xrightarrow{x_r}{}^* (p_r,r)$ is the shortest prefix after which the value of the counter attains the value r. Since $j - i > K$ it follows that there are r, r' with $i \leq r < r' \leq j$ such that $p_r = p_{r'}$. Clearly one may iterate the segment of the run from (p_r,r) to (p_r, r') any number of times, say $N \geq 1$, to get a run $(p,i) \xrightarrow{w}{}^* (q, j + (r'-r)N)$. where $w = x_1 \ldots x_r (x_{r+1} \ldots x_{r'})^{N+1} x_{r+1} \ldots x_k$. Setting $k = r' - r$ yields the Lemma. □

An analogous argument shows that if the value of the counter decreases by at least K in ρ then we may iterate a suitable segment to reduce the value of the counter by any multiple of k' (where the k' is the net decrease in the value of the counter along this segment) while reading a superword. This is formalized as

Lemma 14. *Let* $(q', j') \xrightarrow{z}{}^* (p', i')$ *with* $j' - i' > K$. *Then, there is an integer* $k' > 0$ *such that for every* $N \geq 1$ *there is a run* $(q', j' + N.k') \xrightarrow{w = y_1(y_2)^{N+1}y_3}{}^* (p', i')$, *with* $z = y_1y_2y_3$.

Proof. We break the run into segments as:

$$(q', j') = (q_{j'}, j') \xrightarrow{z_{j'-1}}{}^* (q_{j'-1}, j'-1) \xrightarrow{z_{j'-2}}{}^* (q_{j'-2}, j'-2) \ldots \xrightarrow{z_{i'}}{}^* (q_{i'}, i') \xrightarrow{z'}{}^* (p', i')$$

where $(q', j') = (q_{j'}, j') \xrightarrow{z_{j'-1}}{}^{*} (q_{j'-1}, j'-1) \xrightarrow{z_{j'-2}}{}^{*} (q_{j'-2}, j'-2) \ldots \xrightarrow{z_t}{}^{*} (q_t, t)$ is the shortest prefix after which the value of counter is t. Since $j' - i' > K$ it follows that there are t, t' such that $j' \geq t > t' \geq i'$ such that $q_t = q_{t'}$. Then, starting at any configuration (q_t, R) with $R = t + (t - t')N$, $N \in \mathbb{N}$, we may iterate the transitions in the run $(q_t, t) \rightarrow^{*} (q_{t'}, t')$, an additional N times. In particular this yields a run $(q_t, t + (t-t')N) \xrightarrow{z''}{}^{*} (q_{t'}, t')$ where $z'' = (z_{t-1} \ldots z_{t'})^{N+1}$. Observe that $z_{t-1} \ldots z'_t$ is a subword of z''. Finally, notice that this also means that $(q', j' + N.(t-t')) \xrightarrow{z_1 \ldots z_t}{}^{*} (q_t, t + N.(t-t')) \xrightarrow{z''}{}^{*} (q_{t'}, t') \xrightarrow{z_{t'+1} \ldots z_i}{}^{*} (p', i')$. Taking $k' = (t - t')$ completes the proof. □

A consequence of these somewhat innocous Lemmas is the following interesting fact: we can turn a triple consisting of two runs, where the first one increases the counter by at least K and the second one decreases the counter by at least K, and a free-run that connects them, into a real run provided we are content to read a superword along the way.

Lemma 15. *Let $(p, i) \xrightarrow{x}{}^{*} (q, j) \overset{y}{\Rightarrow} (q', j') \xrightarrow{z}{}^{*} (p', i')$, with $j - i > K$ and $j' - i' > K$. Then, there is a run $(p, i) \xrightarrow{w}{}^{*} (p', i')$ such that $xyz \preceq w$.*

Proof. Let the lowest value of counter in the entire run be m. If $m \geq 0$ then the given free run is by itself a run and hence there is is nothing to prove. Let us assume that m is negative.

First we use Lemma 13, to get a k and an x' for any $N > 1$ and a run $(p, i) \xrightarrow{x'}{}^{*} (q, j + k.N)$ with $x \preceq x'$. We can then extend this to a run $(p, i) \xrightarrow{x'}{}^{*} (q, j + k.N) \overset{y}{\Rightarrow} (q', j' + k.N)$, by choose any N such that $k.N > m$. Then, we have that the value of the counter is ≥ 0 in every configuration of this run. Thus $(p, i) \xrightarrow{x'}{}^{*} (q, j + k.N) \xrightarrow{y}{}^{*} (q', j' + k.N)$ for any such N. Now, we apply Lemma 14 to the run $(q', j') \xrightarrow{z}{}^{*} (p', i')$ to obtain the k'. We now set our N to be a value divisible by k', say $k'.I$. Thus, $(p, i) \xrightarrow{x'}{}^{*} (q, j + k.k'.I) \xrightarrow{y}{}^{*} (q', j' + k.k'.I)$ and now we may again use Lemma 14 to conclude that $(q', j' + k.k'.I) \xrightarrow{z''}{}^{*} (p', i')$ with $x \preceq x'$ and $z \preceq z''$. This completes the proof.

□

Interesting as this may be, this Lemma still relies on the counter value being recorded exactly in all the three segments in its antecedent and this is not sufficient. In the next step, we weaken this requirement (while imposing the condition that $q = q'$ and $j = j'$) by releasing the (free) middle segment from this obligation.

Lemma 16. *Let $(p, i) \xrightarrow{x}{}^{*} (q, j), (q, j) \xrightarrow{z}{}^{*} (p', i')$, with $j - i > K$ and $j' - i' > K$. Let there be a free run from q to q that reads y. Then, there is a run $(p, i) \xrightarrow{w}{}^{*} (p', i')$ such that $xyz \preceq w$.*

Proof. Let the given free-run result in $(q, j) \overset{y}{\Rightarrow} (q, j + d)$ (where d is the net effect of the free run on the counter, which may be positive or negative). Iterating this free-run m times yields a free-run $(q, j) \overset{y^m}{\Longrightarrow} (q, j + m.d)$, for any $m \geq 0$. Next, we use Lemma 13 to find a $k > 0$ such that for each $N > 0$ we have a run $(p, i) \xrightarrow{x_N}{}^{*} (q, j + N.k)$ with $x \preceq x_N$. Similarly, we use Lemma 14 to find a $k' > 0$ such that for each $N' > 0$ we have a run $(q, j + N'.k') \xrightarrow{y_{N'}}{}^{*} (p', i')$ with $y \preceq y_{N'}$.

Now, we pick m and N to be multiples of k' in such a way that $N.k + m.d > 0$. This can always be done since k is positive. Thus, $N.k + m.d = N'.k'$ with $N' > 0$. Now we try and

combine the (free) runs $(p,i) \xrightarrow{x_N}{}^* (q, j+N.k)$, $(q, j+N.k) \overset{y^m}{\Longrightarrow} (q, j+N.k+m.d)$ and $(q, j+N'.k') \xrightarrow{y_{N'}}{}^* (p', i')$ to form a run. We are almost there, as $j+N.k+m.d = j+N'.k'$. However, it is not guaranteed that this combined free-run is actually a run as the value of the counter may turn negative in the segment $(q, j+N.k) \overset{y^m}{\Longrightarrow} (q, j+N.k+m.d)$. Let $-N''$ be the smallest value attained by the counter in this segment. Then by replacing N by $N+N''.k'$ and N' by $N'+N''.k$ we can manufacture a triple which actually yields a run (since the counter values are ≥ 0), completing the proof. □

With Lemma 16 in place we can now explain how to relax the usage of counters. Let us focus on runs that are interesting, that is, those in which the counter value exceeds K at some point. Any such run may be broken into 3 stages: the first stage where counter value starts at 0 and remains strictly below $K+1$, a second stage where it starts and ends at $K+1$ and a last stage where the value begins at K and remains below K and ends at 0 (the 3 stages are connected by two transitions, an increment and a decrement). Suppose, we write the given accepting run as $(p,0) \xrightarrow{w_1}{}^* (q,c) \xrightarrow{w_2}{}^* (r,0)$ where (q,c) is a configuration in the second stage. If $a \in \Sigma$ is a letter that may be read in some transition on some free run from q to q. Then, $w_1 a w_2$ is in $L(A)\downarrow$. This is a direct consequence of Lemma 16. It means that in the configurations in the middle stage we may freely read certain letters without bothering to update the counters. This turns out to be a crucial step in our construction. To turn this relaxation idea into a construction, the following seems a natural.

We make an equivalent, but expanded version of $\mathscr{A}$. This version has 3 copies of the state space: The first copy is used as long as the value of the counter stays below $K+1$ and on attaining this value the second copy is entered. The second copy simulates $\mathscr{A}$ exactly but nondeterministically chooses to enter third copy whenever the counter value is moves from $K+1$ to K. The third copy simulates $\mathscr{A}$ but does not permit the counter value to exceed K. For every letter a and state q with a free run from q to q along which a is read on some transition, we add a self-loop transition to the state corresponding to q in the second copy that does not affect the counter and reads the letter a. This idea has two deficiencies: first, it is not clear how to define the transition from the second copy to the third copy, as that requires knowing that value of the counter is $K+1$, and second, this is still a counter automata (since the second copy simply faithfully simulates $\mathscr{A}$) and not an NFA.

Suppose we bound the value of the counter by some value U in the second stage. Then we can overcome both of these defects and construct a finite automaton as follows: The state space of the resulting NFA has stages of the form (q,i,j) where $j \in \{1,2,3\}$ denotes the stage to which this copy of q belongs. The value i is the value of the counter as maintained within the state of the NFA. The transitions interconnecting the stages go from a state of the form $(q,K,1)$ to one of the form $(q',K+1,2)$ (while simulating a transition involving an increment) and from a stage of the form $(q,K+1,2)$ to one of the form $(q',K,3)$ (while simulating a decrement). The value of i is bounded by K if $j \in \{1,3\}$ while it is bounded by U if $j=2$. (States of the form $(q,i,2)$ also have self-loop transitions described above.) By using a slight generalization of Lemma 16, which allows for the simultaneous insertion of a number of free runs (or by applying the Lemma iteratively), we can show that any word accepted by such a finite automaton lies in $L(\mathscr{A})\downarrow$. However, there is no guarantee that such an automaton will accept

every word in $L(\mathscr{A})\downarrow$. The second crucial point is that we are able to show that if $U \geq K^2+K+1$ then every word in $L(\mathscr{A})$ is accepted by this 3 stage NFA. We show that for each accepting run ρ in $\mathscr{A}$ there is an accepting run in the NFA reading the same word. The proof is by a double induction, first on the maximum value attained by the counter and then on the number of times this value is attained along the run. Clearly, segments of the run where the value of the counter does not exceed K^2+K+1 can be simulated as is. We then show that whenever the counter value exceeds this number, we can find suitable segments whose net effect on the counter is 0 and which can be simulated using the self-loop transitions added to stage 2 (which do not modify the counters), reducing the maximum value of the counter along the run. We now present the formal details.
We begin by describing the NFA $\mathscr{A}_U$ where $U \geq K+1$.

$$\mathscr{A}_U = (Q_1 \cup Q_2 \cup Q_3, \Sigma, \Delta, i_U, F_U\})$$

where $Q_1 = Q \times \{0 \ldots K\} \times \{1\}$, $Q_2 = Q \times \{0 \ldots U\} \times \{2\}$ and $Q_3 = Q \times \{0 \ldots K\} \times \{3\}$. We let $i_U = (s,$ $0,1)$ and $F_U = \{(f,0,1),(f,0,3) \mid f \in F\}$. The transition relation is the union of the relations Δ_1, Δ_2 and Δ_3 defined as follows:

Transitions in Δ_1:

1. $(q,n,1) \xrightarrow{a} (q',n,1)$ for all $n \in \{0 \ldots K\}$ whenever $(q,\mathbf{Int},a,q') \in \delta$.
   Simulate internal move.
2. $(q,n,1) \xrightarrow{a} (q',n-1,1)$ for all $n \in \{1 \ldots K\}$ whenever $(q,\mathbf{Dec},a,q') \in \delta$.
   Simulate decrement.
3. $(q,n,1) \xrightarrow{a} (q',n+1,1)$ for all $n \in \{0 \ldots K-1\}$ whenever $(q,\mathbf{Inc},a,q') \in \delta$.
   Simulate an increment.
4. $(q,K,1) \xrightarrow{a} (q',K+1,2)$ whenever $(q,\mathbf{Inc},a,q') \in \delta$.
   Simulate an increment and shift to second phase.

Transitions in Δ_2:

1. $(q,n,2) \xrightarrow{a} (q',n,2)$ for all $n \in \{0 \ldots K^2+K+1\}$ whenever $(q,\mathbf{Int},a,q') \in \delta$.
   Simulate internal move.
2. $(q,n,2) \xrightarrow{a} (q',n-1,2)$ for all $n \in \{1 \ldots K^2+K+1\}$ whenever $(q,\mathbf{Dec},a,q') \in \delta$.
   Simulate decrement.
3. $(q,K+1,2) \xrightarrow{a} (q',K,3)$ whenever $(q,\mathbf{Dec},a,q') \in \delta$.
   Simulate decrement and shift to third phase.
4. $(q,n,2) \xrightarrow{a} (q',n+1,2)$ for all $n \in \{0 \ldots K^2+K\}$ whenever $(q,\mathbf{Inc},a,q') \in \delta$.
   Simulate an increment move.
5. $(q,n,2) \xrightarrow{a} (q,n,2)$ whenever $(q,a) \in S$ where $S = \{(q,a) \mid q \overset{w}{\Rightarrow} q, a \preceq w\}$. Freely simulate loops.

Transitions in Δ_3:

1. $(q,n,3) \xrightarrow{a} (q',n,3)$ for all $n \in \{0 \ldots K\}$ whenever $(q,a,\mathbf{Int},q') \in \delta$.
   Simulate internal move.

2. $(q,n,3) \xrightarrow{a} (q',n-1,3)$ for all $n \in \{1 \dots K\}$ whenever $(q,a,\mathbf{Dec},q') \in \delta$.
   Simulate decrement.
3. $(q,n,3) \xrightarrow{a} (q',n+1,3)$ for all $n \in \{0 \dots K-1\}$ whenever $(q,a,\mathbf{Inc},q') \in \delta$.
   Simulate an increment move.

The following Lemma, which is easy to prove, states that the first and third phases simulate faithfully any run where the value of the counter is bounded by K.

Lemma 17.

1. *If* $(q,i,l) \xrightarrow{w}{}^* (q',j,l)$ *in* $\mathscr{A}_U$ *then* $(q,i) \xrightarrow{w}{}^* (q',j)$ *in* $\mathscr{A}$, *for* $l \in \{1,3\}$.
2. *If* $(q,i) \xrightarrow{w}{}^* (q',j)$ *in* $\mathscr{A}$ *through a run where the value of the counter is* $\leq K$ *in all the configurations along the run then* $(q,i,l) \xrightarrow{w}{}^* (q',i,l)$ *for* $l \in \{1,3\}$.

Proof. Follows directly from the construction, we have an equivalent transition manipulating the counter stored in the state, for every transition in the original counter system. □

The next Lemma extends this to runs involving the second phase as well. All moves other than those simulating unconstrained free runs can be simulated by $\mathscr{A}$. The second phase of $\mathscr{A}_U$ can also simulate any run where the counter is bounded by U.

Lemma 18. *1. If* $(q,i,l) \xrightarrow{w}{}^* (q',j,l')$ *is a run of* $\mathscr{A}_U$ *in which no transition from* Δ_2 *of type 5 is used then* $(q,i) \xrightarrow{w}{}^* (q',j)$ *is a run of* $\mathscr{A}$.

2. *If* $\rho = (q_0,i_0) \xrightarrow{a_1}{}^* (q_1,i_1) \xrightarrow{a_2}{}^* \dots \xrightarrow{a_m}{}^* (q_m,i_m)$ *is a run in* $\mathscr{A}$ *in which the value of the counter never exceeds* K^2+K+1 *then* $\rho' = (q_0,i_0,2) \xrightarrow{a_1}{}^* (q_1,i_1,2) \xrightarrow{a_2}{}^* \dots \xrightarrow{a_m}{}^* (q_m,i_m,2)$ *is a run in* $\mathscr{A}_U$.

Proof. The proof of this again directly follows from the fact that for every move in counter system M, we have an equivalent transition in $\mathscr{A}_U$.

□

Now, we are in a take the first step towards generalizing Lemma 16 to prove that $L(\mathscr{A}_U) \subseteq L(\mathscr{A})\downarrow$.

Lemma 19. *Let* $(q,i,2) \xrightarrow{w}{}^* (q',j,2)$ *be a run in* $\mathscr{A}_U$. *Then, there is an* $N \in \mathbb{N}$, *words* $x_0, y_0, x_1, y_1, \dots, x_N$, *and integers* $n_0, n_1, \dots, n_{N-1}$ *such that:*

1. $w \preceq x_0 y_0 x_1 y_1 \dots x_N$.
2. $(q,i) \xRightarrow{x_0 y_0 \dots x_N} (q',j')$ *where* $j' = j + n_0 + n_1 \dots + n_{N-1}$.
3. $(q,i) \xRightarrow{x_0 y_0^{m_0} x_1 y_1^{m_1} \dots x_N} (q',j'')$ *where* $j'' = j + m_0.n_0 + m_1.n_1 \dots + m_{N-1}.n_{N-1}$, *for any* $m_0, m_1, \dots m_{N-1} \geq 1$.

Note that 2 is just a special case of 3 when $m_0, m_1, \cdots m_{N-1} = 1$.

Proof. The run $(q,i,2) \xrightarrow{w}{}^* (q',j,2)$ in $\mathscr{A}_U$ uses only transitions of the types $1,2,4$ and 5. Let N be the number of transitions of type 5 used in the run. We then break up the run as follows:

$$(q,i,2) \xrightarrow{x_0}{}^* (p_0,i_0,2) \xrightarrow{a_0}{}^* (p_0,i_0,2) \xrightarrow{x_1}{}^* (p_1,i_1,2) \dots (p_{N-1},i_{N-1},2) \xrightarrow{a_{N-1}}{}^* (p_{N-1},i_{N-1},2) \xrightarrow{x_N}{}^* (q',j,2)$$

where the transitions on a_i's are the N moves using transitions of type 5 in the run. Let $(p_r, i_r) \overset{y_r}{\Longrightarrow} (p_r, i'_r)$ be a free run with $a_r \preceq y_r$ and let $n_r = i'_r - i_r$. Clearly $w \preceq x_0 y_0 x_1 y_1 \ldots x_N$.

It is quite easy to show by induction on r, $0 \leq r < N$, by replacing moves of types 1,2 and 4 by the corresponding moves in $\mathscr{A}$ and moves of type 5 by the iterations of the free runs identified above that:

$$(q,i) \xrightarrow{x_0}{}^* (p_0, i_0) \overset{y_0^{m_0}}{\Longrightarrow} (p_0, i_0 + m_0.n_0) \overset{x_1}{\Rightarrow} (p_1, i_1 + m_0.n_0) \overset{y_1^{m_1}}{\Longrightarrow} (p_1, i_1 + m_0.n_0 + m_1.n_1)$$
$$\ldots (p_r, i_r + m_0.n_0 \ldots + m_{r-1}.n_{r-1}) \overset{y_r^{m_r}}{\Longrightarrow} (p_r, i_r + m_0.n_0 \ldots m_r.n_r) \xrightarrow{x_{r+1}}{}^*$$
$$(p_{r+1}, i_{r+1} + m_0 n_0 \ldots m_r.n_r)$$

and with $r = N-1$ we have the desired result. □

Now, we use an argument that generalizes Lemma 16 in order to show that:

Lemma 20. *Let w be any word accepted by the automaton $\mathscr{A}_U$. Then, there is a word $w' \in L(\mathscr{A})$ such that $w \preceq w'$.*

Proof. If states in Q_2 are not visited in the accepting run of $\mathscr{A}_U$ on w then we can use Lemma 17 to conclude that $w \in \mathscr{A}$. Otherwise, we break up the run of $\mathscr{A}_U$ on w into three parts as follows:

$$(s,0,1) \xrightarrow{w_1}{}^* (p,K,1) \xrightarrow{a_1} (q,K+1,2) \xrightarrow{w_2}{}^* (r,K+1,2) \xrightarrow{a_2} (t,K,3) \xrightarrow{w_3}{}^* (f,0,3)$$

Using Lemma 17 it follows that $(s,0) \xrightarrow{w_1}{}^* (p,K)$ and $(t,K) \xrightarrow{w_3}{}^* (f,0)$. We then apply Lemmas 13 and 14 to these two segments respectively to identify k and k'. Next we use Lemma 19 to identify the positive integer N, integers $n_0, n_1, \ldots n_{N-1}$ and the free run

$$(q,K+1) \overset{x_0 y_0 \ldots x_N}{\Longrightarrow} (r, K+1+n_0+n_1 \ldots + n_{N-1})$$

with $w_2 \preceq x_0 y_0 x_1 y_1 \ldots x_{N-1} y_{N-1} x_N$. We identify numbers $m, m_0, m_1, \ldots, m_{N-1}$, all ≥ 1, such that $(m-1).k + m_0.n_0 + \ldots m_{N-1}.n_{N-1} = k'.m'$ for some $m' \geq 0$. By taking $m-1$ and each m_i to be some multiple of k' we get the sum $(m-1).k + m_0.n_0 + \ldots m_{N-1} n_{N-1}$ to be a multiple of k', however this multiple may not be positive. Since $k > 0$, by choosing $m-1$ to be a sufficiently large multiple of k' we can ensure that $m' \geq 0$. Using these numbers we construct the free run

$$(q, K+1+(m-1).k) \overset{x_0 y_0^{m_0} x_1 y_1^{m_1} \ldots x_N}{\Longrightarrow} (r, K+1+(m-1).k + m_0 n_0 + \ldots m_{N-1} n_N) =$$
$$(r, K+1+k'.m')$$

Let l be the lowest value attained in this free run. If $l \geq 0$ then

$$(q, K+1+(m-1).k) \xrightarrow{x_0 y_0^{m_0} x_1 y_1^{m_1} \ldots x_N}{}^* (r, K+1+k'.m')$$

and using Lemma 13 and 14 we get

$$(s,0)\xrightarrow{w}{}^{*}(p,K+(m-1).k)\xrightarrow{a_1}(q,K+1+(m-1).k)\xrightarrow{x_0y_0^{m_0}x_1y_1^{m_1}\ldots x_N}{}^{*}(r,K+1+k'.m')$$
$$\xrightarrow{a_2}(t,K+k'.m')\xrightarrow{z}{}^{*}(f,0,3)$$

with $w_1 \preceq w$, $w_2 \preceq x_0y_0^{m_0}x_1y_1^{m_1}\ldots x_N$ and $w_3 \preceq z$ as required.

Suppose $l<0$. Then, we let I be a positive integer such that $I.k+l>0$ and $I=k'.m''$ (i.e. I is divisible by k') which must exist since $k>0$. Then

$$(q,K+1+(m-1).k+I.k)\xRightarrow{x_0y_0^{m_0}x_1y_1^{m_1}\ldots x_N}(r,K+1+I.k+k'.m')$$

is a free run in which the counter values are always ≥ 0 and is thus a run. Once again, we may use Lemmas 13 and 14 (since $I.k$ is a multiple of k') to get

$$(s,0)\xrightarrow{w}{}^{*}(p,K+(m-1).k+I.k)\xrightarrow{a_1}(q,K+1+(m-1).k+I.k)\xrightarrow{x_0y_0^{m_0}x_1y_1^{m_1}\ldots x_N}{}^{*}$$
$$(r,K+1+k'.m'+I.k)\xrightarrow{a_2}(t,K+1+k'.m'+I.k)\xrightarrow{z}{}^{*}(f,0,3)$$

with $w_1 \preceq w$, $w_2 \preceq x_0y_0^{m_0}x_1y_1^{m_1}\ldots x_N$ and $w_3 \preceq z$. This completes the proof of the Lemma. □

Next, we show that if $U \geq K^2+K+1$ then $L(\mathscr{A}) \subseteq L(\mathscr{A}_U)$.

Lemma 21. *Let $U \geq K^2+K+1$. Let w be any word in $L(\mathscr{A})$. Then, w is also accepted by $\mathscr{A}_U$.*

Proof. The proof is accomplished by examining runs of the from $(s,0)\xrightarrow{w}{}^{*}(f,0)$ and showing that such a run may be *simulated* by $\mathscr{A}_U$ transition by transition in a manner to be described below. Any run $\rho=(s,0)\xrightarrow{w}{}^{*}(f,0)$ can be broken up into parts as follow:

$$(s,0)\xrightarrow{x}{}^{*}(g,j)\xrightarrow{y'}{}^{*}(h,j')\xrightarrow{z}{}^{*}(f,0)$$

where, $\rho_1=(s,0)\xrightarrow{x}{}^{*}(g,j)$ is the longest prefix where the counter value does not exceed K, $\rho_3=(h,j')\xrightarrow{z}{}^{*}(f,0)$, is the longest suffix, of what is left after removing ρ_1, in which the value of the counter does not exceed K, and $\rho_2=(g,j)\xrightarrow{y'}{}^{*}(h,j')$ is what lies in between. We note that using Lemma 17 we can conclude that there are runs $(s,0,1)\xrightarrow{x}{}^{*}(g,j,1)$ and $(h,j',3)\xrightarrow{z}{}^{*}(f,0,3)$. Further, observe that if value of the counter never exceeds K then ρ_2 and ρ_3 are empty, $x=w$, $g=f$ and $j=0$. In this case, using Lemma 17, there is a (accepting) run $(s,0,1)\xrightarrow{w}{}^{*}(f,0,1)$.

If the value of the counter exceeds K then $j=j'=K$ and by Lemma 17, $(s,0,1)\xrightarrow{x}{}^{*}(g,K,1)$, $(h,K,3)\xrightarrow{z}{}^{*}(f,0,3)$ and ρ_2 is non-empty. Further suppose that, ρ_2, when written out as a sequence of transitions is of the form

$$\rho_2=(g,K)\xrightarrow{a}(p,K+1)=(p_0,i_0)\xrightarrow{a_1}(p_1,i_1)\xrightarrow{a_2}(p_2,i_2)\ldots\xrightarrow{a_n}$$
$$(p_n,i_n)=(q,K+1)\xrightarrow{b}(h,K)$$

We will show by double induction on the maximum value of the counter value attained in the run ρ_2 and the number of times the maximum is attained that there is a run

$$\rho_2' = (g,K,1) \xrightarrow{a} (p,K+1,2) = (p_0',i_0',2) \xrightarrow{a_1} (p_1',i_1',2) \xrightarrow{a_2} (p_2',i_2',2) \xrightarrow{a_3} \ldots$$
$$\ldots \xrightarrow{a_n} (p_n',i_n',2) = (q,K+1,2) \xrightarrow{b} (h,K,3)$$

such that for all i, $0 \leq i < n$,

1. either $p_i = p_i'$ and $p_{i+1} = p_{i+1}'$ and the ith transition (on a_{i+1}) is of type 1,2 or 4,
2. or $p_i' = p_{i+1}'$, $i_i' = i_{i+1}'$, $p_i' \Rightarrow p_i$ and $p_{i+1} \Rightarrow p_i'$ so that the ith transition (on a_{i+1}) is a transition of type 5.

For the basis, notice that if the maximum value attained is $\leq K^2+K+1$ then, by Lemma 18, there is a run of $\mathscr{A}_U$ that simulates ρ_2 such that item 1 above is satisfied for all i.

Now, suppose the maximum value attained along the run is $m > K^2+K+1$. We proceed along the lines of the proof of Lemma 12. We first break up the run ρ_2 as

$$(g,K) \xrightarrow{a} (p_0,K+1) = (q_{K+1},K+1) \xrightarrow{y_{K+2}}{}^* (q_{K+2},K+2) \xrightarrow{y_{K+3}}{}^* (q_{K+3},K+3)$$
$$\ldots \xrightarrow{y_m}{}^* (q_m,m) \xrightarrow{y'_{m-1}}{}^* (q'_{m-1},m-1) \xrightarrow{y'_{m-1}}{}^* \ldots \xrightarrow{y'_{K+1}}{}^* (q'_{K+1},K+1) \xrightarrow{z}{}^*$$
$$(q,K+1) \xrightarrow{b} (h,K)$$

where

- The prefix upto (q_m,m), henceforth referred to as σ_m, is the shortest prefix after which the counter value is m.
- The prefix upto (q_i,i), $K+1 \leq i < m$ is the longest prefix of σ_m after which the value of the counter is i.
- The prefix upto (q_i',i), $K+1 \leq i < m$ is the shortest prefix of ρ_2 with σ_m as a prefix after which the counter value is i.

By construction, the value of the counter in the segment of the run from $(q_i,i) \rightarrow^* \ldots \rightarrow^* (q_i',i)$ never falls below i. Further, by simple counting, there are i,j with $K+1 \leq i < j \leq m$ such that $q_i = q_j$ and $q_i' = q_j'$. Thus, by deleting the segment of the runs from (q_i,i) to (q_i,j) and (q_j',j) to (q_j',i) we get a shorter run ρ_d which looks like

$$(g,K) \xrightarrow{a}{}^* (p_0,K+1) = (q_{K+1},K+1) \ldots \xrightarrow{y_i}{}^* (q_i,i) \xrightarrow{y_{j+1}}{}^* (q_{j+1},i+1)$$
$$\ldots \xrightarrow{y_m}{}^* (q_m,m-j+i) \xrightarrow{y'_{m-1}}{}^* \ldots\ldots \xrightarrow{y'_j}{}^* (q_j',i) \xrightarrow{y'_{i-1}}{}^* (q'_{i-1},i-1) \ldots$$
$$(q'_{K+1},K+1) \xrightarrow{z}{}^* (q,K+1) \xrightarrow{b}{}^* (h,K)$$

This run reaches the value m at least one time fewer than ρ_2 and thus we may apply the induction hypothesis to conclude the existence of a run ρ_d' of A_{gh} that simulates this run move for move satisfying the properties indicated in the induction hypothesis. Let this run

be:

$$(g,K,1) \xrightarrow{a} (r_0,K+1,2)\ldots \xrightarrow{y_i}{}^{*} (r_i,c_i,2)) \xrightarrow{y_{j+1}}{}^{*} (r_{j+1},c_{j+1},2)\ldots \xrightarrow{y_m}{}^{*}$$
$$(r_m,c_m,2) \xrightarrow{y'_{m-1}}{}^{*} \ldots\ldots \xrightarrow{y'_j}{}^{*} (r'_j,c'_j,2) \xrightarrow{y'_{i-1}}{}^{*} (r'_{i-1},c'_{i-1},2)\ldots$$
$$(r'_{K+1},c'_{K+1},2) \xrightarrow{z}{}^{*} (r',K+1,2) \xrightarrow{b} (h,K)$$

Now, if $(p_l,i_l) \xrightarrow{a_{l+1}} (p_{l+1},i_{l+1})$ was a transition in ρ_2 in the part of the run from (q_i,i) to (q_i,j) then, $q_i \Rightarrow p_l$, $p_l \overset{a_{l+1}}{\Longrightarrow} q_i$ and $p_{l+1} \Rightarrow q_i$. Now, either $r_i = q_i$ or $r_i \Rightarrow q_i$, and $q_{j+1} \Rightarrow r_i$ and (q_i,a_{j+1},op,q_{j+1}) is a transition for some op. In the both cases clearly $r_i \overset{a_{l+1}}{\Longrightarrow} r_i$. Thus every such deleted transition can be simulated by a transition of the form $(r_i,c_i,2) \xrightarrow{a_{l+1}} (r_i,c_i,2)$.

A similar argument shows that every transition of the form $(p_l,i_l) \xrightarrow{a_{l+1}} (p_{l+1},i_{l+1})$ deleted in the segment (q'_j,j) to (q'_j,i) can be simulated by $(r'_j,c'_j,2) \xrightarrow{a_{l+1}} (r'_j,c'_j,2)$. Thus we can extend the run ρ'_d to a run ρ'_2 that simulates ρ_2 fulfilling the requirements of the induction hypothesis. This completes the proof of this Lemma. □

Notice that the size of the state space of $\mathscr{A}_U$ is $K.(K^2+K+1)$ when $U = K^2+K+1$. Since downward closures of NFAs can be constructed by just adding additional (ϵ) transitions, Lemmas 20 and 21 imply that:

Theorem 11. *There is a polynomial-time algorithm that takes as input a simple counter automaton $\mathscr{A} = (Q,\Sigma,\delta,s,F)$ and computes an NFA with $O(|\mathscr{A}|^3)$ states accepting $L(\mathscr{A})\downarrow$.*

4.5 Revisiting shared memory systems

Recall that we defined shared-memory concurrent pushdown systems and a restriction bounded stage on its runs in chapter 3. We also described a procedure for solving the bounded stage reachability on shared memory system when at most one pushdown process was involved. For this, we showed how to reduce the k bounded stage reachability problem on an SCPS S, to reachability on pushdown with reversal bounded counters . The size of the resulting reversal bounded system that we constructed was $O(n^k.|S|^{O(|S|.n)})$, where n is the number of processes in S.

Observe that the exponential dependancy on the number of stages was because we proceeded by fixing the sequence τ of writers for each stage in the construction. This was done for sake of simplifying the proof. This can easily be eliminated, by modifying Lemma-5 such that each process guesses the writer in each stage and all the processes synchronise on the identity of the writer at the beginning of each stage (as they do w.r.t. the value of the memory at the beginning of the stage). This modification reduces the complexity to $O(k.|S|^{O(|S|.n)})$. However note that the complexity is still exponential on size of SCPS and the number of processes.

Next, notice from the calculation in Lemma 6, that the size of the automaton appears in the exponent only because we assumed that the size of the closures on counter systems is

exponential. Thus, the results proved above, reduces the complexity further to $O(k.|S|^{O(n)})$. With this we have the following theorem

Theorem 12. *The stage bounded reachability problem for SCPS with at most one pushdown system is in* NEXPTIME *in the number of processes, while polynomial in the size of system and number of stages. In particular, the problem is in* NP *if the number is processes are fixed.*

4.6 Parikh Images of Reversal Bounded PDAs

In this section we describe an algorithm to construct an NFA Parikh-equivalent to an counter automata $\mathscr{A}$. The NFA has at most $O(|\Sigma|K^{O(\log K)})$ states where $K = |\mathscr{A}|$, a significant improvement over $O(2^{\text{poly}(K,|\Sigma|)})$ for PDA.

We establish this result in two steps. In the first step, we show that we can focus our attention on computing Parikh-images of words recognised along *reversal bounded* runs. A reversal in a run occurs when the counter system switches to incrementing the counter after a non-empty sequence of decrements (and internal moves) or when it switches to decrementing the counter after a non-empty sequence of increments (and internal moves). For a number R, a run is R reversal bounded, if the number of reversals along the run is $\leq R$. Let us use $L_R(\mathscr{A})$ to denote the set of words accepted by $\mathscr{A}$ along runs with at most R reversals.

We construct a new polynomial size counter automata from $\mathscr{A}$ and show that we can restrict our attention to runs with at most R reversals of this counter automata, where R is a polynomial in K. In the second step, from any simple counter automata $\mathscr{A}$ with K states and any integer R we construct an NFA of size $O(K^{O(log(R))})$ whose Parikh image is $L_R(\mathscr{A})$. Combination of the two steps gives a $O(K^{O(logK)})$ construction.

4.6.1 Reversal bounding

We establish that, up to Parikh-image, it suffices to consider runs with $2K^2 + K$ reversals. We use two constructions: one that eliminates *large* reversals (think of a waveform) and another that eliminates *small* reversals (think of the noise on a noisy waveform). For the large reversals, the idea used is the following: we can reorder the transitions used along a run, hence preserving Parikh-image, to turn it into one with few large reversals (a noisy waveform with few reversals). The key idea used is to move each simple cycle at state q with a positive (resp. negative) effect on the counter to the first (resp. last) occurrence of the state along the run. To eliminate the smaller reversals (noise), the idea is to maintain the changes to the counter in the state and transfer it only when necessary to the counter to avoid unnecessary reversals.

Consider a run of $\mathscr{A}$ starting at a configuration (p, c) and ending at some configuration (q, d) such that the value of the counter e in any intermediate configuration satisfies $c - D \leq e \leq c + D$ (where D is some positive integer). We refer to such a run as an *D-bound run.* Reversals along such a run are not important and we get rid of them by maintaining the (bounded) changes to the counter within the state.

We construct a counter automata $\mathscr{A}[D]$ as follows: its states are $Q \cup Q_1 \cup Q_2$ where $Q_1 = Q \times [-D, D]$ and $Q_2 = [-D, D] \times Q$. All transitions of $\mathscr{A}$ are transitions of $\mathscr{A}[D]$ as well and

thus using Q it can simulate any run of $\mathscr{A}$ faithfully. From any state $q \in Q$ the automaton may move nondeterministically to $(q,0)$ in Q_1. The states in Q_1 are used to simulate D-bound runs of $\mathscr{A}$ without altering the counter and by keeping track of the net change to the counter in the second component of the state. For instance, consider the D-bound run of $\mathscr{A}$ described above: $\mathscr{A}[D]$ can move from (p,c) to $((p,0),c)$ then simulate the run of $\mathscr{A}$ to (q,d) to reach $((q,d-c),c)$. At this point it needs to transfer the net effect back to the counter (by altering it appropriately). The states Q_2 are used to perform this role. From a state (q,j) in Q_1, $\mathscr{A}[D]$ is allowed to nondeterministically move to (j,q) indicating that it will now transfer the (positive or negative) value j to the counter. After completing the transfer it reaches a state $(0,q)$ from where it can enter the state q via an internal move to continue the simulation of $\mathscr{A}$.

The nice feature of this simulated run via Q_1 and Q_2 is that there are no reversals in the simulation and it involves only increments (if $d > c$) or only decrements (if $d < c$).

We now formalize the automaton $\mathscr{A}[D]$ and its properties. The counter automata $\mathscr{A}[D] = (Q_D, \Sigma, \delta_D, s, F)$ is defined as follows:

$$Q_D = Q \cup (Q \times \{-D, \ldots, D\}) \cup (\{-D, \ldots, D\} \times Q)$$

and δ_D is defined as follows:

1. $\delta \subseteq \delta_D$. Simulate runs of $\mathscr{A}$.
2. $(q, \mathbf{Int}, \epsilon, (q,0)) \in \delta_D$. Begin a summary phase.
3. $((q,j), \mathbf{Int}, a, (q',j)) \in \delta_D$, if $(q, \mathbf{Int}, a, q') \in \delta$. Simulate an internal move.
4. $((q,j), \mathbf{Int}, a, (q',j+1)) \in \delta_D$, if $(q, \mathbf{Inc}, a, q') \in \delta$. Simulate an increment.
5. $((q,j), \mathbf{Int}, a, (q',j-1)) \in \delta_D$, if $(q, \mathbf{Dec}, a, q') \in \delta$. Simulate a decrement.
6. $((q,j), \mathbf{Int}, \epsilon, (j,q)) \in \delta_D$. Finish summary run.
7. $((j,q), \mathbf{Int}, \epsilon, (j-1,q)) \in \delta_D$, if $j > 0$. Transfer a positive effect.
8. $((j,q), \mathbf{Dec}, \epsilon, (j+1,q)) \in \delta_D$, if $j < 0$. Transfer a negative effect.
9. $((0,q), \mathbf{Int}, \epsilon, q) \in \delta_D$. Transfer control back to copy of $\mathscr{A}$.

The following Lemma is the first of a sequence that relate $\mathscr{A}$ and $\mathscr{A}[D]$.

Lemma 22. *1. For any $p,q \in Q$ and any $c,d \in \mathbb{N}$, if $(p,c) \xrightarrow{w}{}^* (q,d)$ in $\mathscr{A}$ then $(p,c) \xrightarrow{w}{}^* (q,d)$ in $\mathscr{A}[D]$.*

2. For any $p,q \in Q$ and any $c,d \in \mathbb{N}$ if $(p,c) \xrightarrow{w}{}^* (q,d)$ in $\mathscr{A}[D]$ then $(p,c+D) \xrightarrow{w}{}^* (q,d+D)$ in $\mathscr{A}$. In particular, if $(p,0) \xrightarrow{w}{}^* (q,0)$ in $\mathscr{A}[D]$ then $(p,D) \xrightarrow{w}{}^* (q,D)$ in $\mathscr{A}$.

Proof. The first statement simply follows from the fact that $\delta \subseteq \delta_D$.

Let $\rho = (p,c) \xrightarrow{w}{}^* (q,d)$ be a run in $\mathscr{A}[D]$. The second statement is proved by induction on the number of transitions of type 2 taken along ρ (i.e. the number of summary simulations used in ρ). If this number is 0 then all the transitions used are of type 1 thus ρ is a run in $\mathscr{A}$ and thus $\rho[D]$ satisfies the requirements of the Lemma.

Otherwise, let ρ must be of the form

$$\rho = (p,c) \xrightarrow{w_1}{}^* (p_1,c_1) \xrightarrow{\epsilon} ((p_1,0),c_1) \xrightarrow{w_2}{}^* ((0,q_1),d_1) \xrightarrow{\epsilon} (q_1,d_1) \xrightarrow{w_3}{}^* (q,d)$$

where we have identified the first occurrence of the transition of type 2 and as well as the first occurrence of a transition of type 9. Now, by the induction hypothesis, we have runs $(p, c+D) \xrightarrow{w_1}{}^* (p_1, c_1+D)$ and $(q_1, d_1+D) \xrightarrow{w_3}{}^* (q, d+D)$ in $\mathscr{A}$.

From the definition of δ_D, run $((p_1,0),c_1) \xrightarrow{w_2}{}^* ((0,q_1),d_1)$ must be of the form

$$((p_1,0),c_1) \xrightarrow{w_2}{}^* ((p_2,c_2),c_1) \xrightarrow{\epsilon} ((c_2,p_2),c_1) \xrightarrow{\epsilon}{}^* ((0,p_2),c_1+c_2)$$

with $p_2 = q_1$ and $d_1 = c_1 + c_2$ and where the run $((p_1,0),c_1) \xrightarrow{w_2}{}^* ((p_2,c_2),c_1)$ involves only transitions of the form 3, 4 or 5.

Claim: Let $((g,0),e) \xrightarrow{x}{}^* ((h,i),e)$ be a run in $\mathscr{A}[D]$ using only transitions of type 3, 4 or 5. Then $(g,e) \xrightarrow{x}{}^* (h,e+i)$ in $\mathscr{A}$ for any $e \geq D$.

Proof. By induction on the length of the run. The base case is trivial. For the inductive case, suppose $((g,0),e) \xrightarrow{x'}{}^* ((h',i'),e) \xrightarrow{a} ((h,i),e)$ and by the induction hypothesis $(g,e) \xrightarrow{x'}{}^* (h', e+i')$ for any $e \geq D$. Now, if the last transition is an internal transition then, $((h',i'),\mathbf{Int},a,(h,i)) \in \delta$ and $i = i'$. Thus $(h',e+i) \xrightarrow{a} (h,e+i)$ in $\mathscr{A}$. If the last transition is an increment then $((h',i'),\mathbf{Int},a,(h,i)) \in \delta$ and $i = i'+1$. Thus, once again we have $(h',e+i') \xrightarrow{a} (h,e+i)$ in $\mathscr{A}$. Finally, if the last transition is a decrement transition then, $((h',i'),a,\mathbf{Dec},(h,i)) \in \delta$. Then, $i = i'-1$ and $i \geq -D$. Thus, $e+i \geq 0$ and thus $(h',e+i') \xrightarrow{a} (h,e+i)$ in $\mathscr{A}$, completing the proof of the claim. □

Since, $c_1 + D \geq D$, we may apply the claim to conclude that $(p_1, c_1+D) \xrightarrow{w_2}{}^* (p_2 = q_1, c_1 + D + c_2 = d_1 + D)$ in $\mathscr{A}$. This completes the proof of the Lemma. □

Next we show that $\mathscr{A}[D]$ can simulate any D-bound run without reversals.

Lemma 23. *Let $(p,c) \xrightarrow{w}{}^* (q,d)$ be an D-bound run in $\mathscr{A}$. Then, there is a run $(p,c) \xrightarrow{w}{}^* (q,d)$ in $\mathscr{A}[D]$ in which the counter value is never decremented if $c \leq d$ and never incremented if $c \geq d$.*

Proof. The idea is to simply simulate the run as a summary run in $\mathscr{A}[D]$. Let the given run be

$$(p,c) = (p_0,c_0) \xrightarrow{a_1} (p_1,c_1) \xrightarrow{a_2} (p_2,c_2) \ldots \xrightarrow{a_n} (p_n,c_n) = (q,d)$$

Then, it is easy to check that the following is a run in $\mathscr{A}[D]$

$$(p_0,c_0) \xrightarrow{\epsilon} ((p_0,0),c_0) \xrightarrow{a_1} ((p_1,c_1-c_0),c_0) \xrightarrow{a_2} \ldots \xrightarrow{a_n} ((p_n,c_n-c_0),c_0) \xrightarrow{\epsilon} ((c_n-c_0,p_n),c_0)$$

It is also easy to verify that for any configuration with $((j,p),e)$ with $e+j \geq 0$, $((j,p),e) \xrightarrow{\epsilon}{}^* (p, e+j)$ is a run in $A[D]$ consisting only of increments if $j > 0$ and consisting only of decrements if $j < 0$. Since $c_n \geq 0$, $(c_n - c_0) + c_0 \geq 0$ and the result follows. □

Actually this automaton $A[D]$ does even better. Concatenation of D-bound runs is often not an D-bound run but the idea of reversal free simulation extends to certain concatenations. We say that a run $(p_0,c_0) \xrightarrow{w}{}^* (p_n,c_n)$ is an increasing (resp. decreasing) *iterated D-bound run* if it can be decomposed as

$$(p_0,c_0) \xrightarrow{w_1}{}^* (p_1,c_1) \xrightarrow{w_2}{}^* \ldots (p_{n-1},c_{n-1}) \xrightarrow{w_n}{}^* (p_n,c_n)$$

where each $(p_i,c_i) \xrightarrow{w_{i+1}}{}^* (p_{i+1},c_{i+1})$ is an D-bound run and $c_i \leq c_{i+1}$ (resp. $c_i \geq c_{i+1}$). We say it is an iterated D-bound run if it is an increasing or decreasing iterated D-bound run.

Lemma 24. *Let $(p,c) \xrightarrow{w}{}^{*} (q,d)$ be an increasing (resp. decreasing) D-bound run in $\mathscr{A}$. Then, there is a run $(p,c) \xrightarrow{w}{}^{*} (q,d)$ in $\mathscr{A}[D]$ along which the counter value is never decremented (resp. incremented).*

Proof. Simulate each ρ_i by a run that only increments (resp. decrements) the counter using Lemma 23. □

While , as a consequence of item 1 of Lemma 22, we have $L(\mathscr{A}) \subseteq L(\mathscr{A}[D])$, the converse is not in general true as along a run of $\mathscr{A}[D]$ the real value of the counter, i.e. the current value of the counter plus the offset available in the state, may be negative, leading to runs that are not simulations of runs of $\mathscr{A}$. The trick, as elaborated in item 2 of Lemma 22, that helps us get around this is to relate runs of $\mathscr{A}[D]$ to $\mathscr{A}$ with a shift in counter values. We need a bit more terminology to proceed.

We say that a run of is an $D_{\leq}$ run (resp. $D_{\geq}$ run) if the value of the counter is bounded from above (resp. below) by D in every configuration encountered along the run. We say that a run of $\mathscr{A}$ is an $D_{>}$ run if it is of the form $(p,D) \xrightarrow{w}{}^{*} (q,D)$, it has at least 3 configurations and the value of the counter at every configuration other than the first and last is $> D$. Consider any run from a configuration $(p,0)$ to $(q,0)$ in $\mathscr{A}$. Once we identify the maximal $D_{>}$ sub-runs, what is left is a collection of $D_{\leq}$ sub-runs.

Let $\rho = (p,c) \xrightarrow{w}{}^{*} (q,d)$ be a run of $\mathscr{A}$ with $c,d \leq D$. If ρ is a $D_{\leq}$ run then its *D-decomposition* is ρ. Otherwise, its D-decomposition is given by a sequence of runs ρ_0, ρ'_0, $\rho_1, \rho'_1 \ldots \rho'_{n-1}, \rho_n$ with $\rho = \rho_0.\rho'_0.\rho_1.\rho'_1 \ldots .\rho'_{n-1}.\rho_n$, where each ρ_i is a $D_{\leq}$ run and each ρ'_i is a $D_{>}$ run for $0 \leq i \leq n$. Notice, that some of the ρ_i's may be trivial. Since the $D_{>}$ subruns are uniquely identified this definition is unambiguous. We refer to the ρ'_i's (resp. ρ_is) as the $D_{>}$ (resp. $D_{\leq}$) components of ρ.

Observe that the $D_{\leq}$ runs of $\mathscr{A}$ can be easily simulated by an NFA. Thus we may focus on transforming the $D_{>}$ runs, preserving just the Parikh-image, into a suitable form. For D, $\mathscr{R} \in \mathbb{N}$, we say that a $D_{>}$ run ρ is a *$(D,\mathscr{R})$-good run* (think noisy waveform with few reversals) if there are runs $\sigma_1, \sigma_2 \ldots, \sigma_n, \sigma_{n+1}$ and iterated D-bound runs $\rho_1, \rho_2, \ldots, \rho_n$ such that $\rho = \sigma_1\rho_1\sigma_2\rho_2 \ldots \sigma_n\rho_n\sigma_{n+1}$ and $|\sigma_1| + \ldots + |\sigma_{n+1}| + 2.n \leq \mathscr{R}$. Using Lemma 24 and that it is a $D_{>}$ run we show

Lemma 25. *Let $(p,D) \xrightarrow{w}{}^{*} (q,D)$ be an $(D,\mathscr{R})$-good run of $\mathscr{A}$. Then, there is a run $(p,0) \xrightarrow{w}{}^{*} (q,0)$ in $\mathscr{A}[D]$ with atmost $\mathscr{R}$ reversals.*

Proof. Let the given run be ρ. We first shift down ρ to $\rho[-D]$ to obtain a run from $(p,0)$ to $(q,0)$, which is possible since ρ is $D_{>}$ run. We then transform each of the iterated D-bound runs using Lemma 24 so that there are no reversals in the transformed runs. Thus all reversals occur inside the $\sigma_i[-D]$'s or at the boundary and this gives us the bound required by the Lemma. □

So far we have not used the fact that we can ignore the ordering of the letters read along a run (since we are only interested in the Parikh-image of $L(\mathscr{A})$). We show that for any run ρ of $\mathscr{A}$ we may find another run ρ' of $\mathscr{A}$, that is equivalent up to Parikh-image, such that every $D_{>}$ component in the D-decomposition of ρ' is $(D,\mathscr{R})$-good, where $\mathscr{R}$ and D are polynomially related to K.

We fix $D = K$ in what follows. We take $\mathscr{R} = 2K^2 + K$ for reasons that will become clear soon. We focus our attention on some $D_>$ component ξ of ρ which is not $(D,\mathscr{R})$-good. Let $X \subseteq Q$ be the set of states of Q that occur in at least two different configurations along ξ. For each of the states in X we identify the configuration along ξ where it occurs for the very first time and the configuration where it occurs for the last time. There are at most $2|X|(\leq 2K)$ such configurations and these decompose the run ξ into a concatenation of $2|X|+1(\leq 2K+1)$ runs $\xi = \xi_1.\xi_2\ldots\xi_m$ where $\xi_i, 1 < i < m$ is a segment connecting two such configurations. Now, suppose one of these ξ_i's has length K or more. Then it must contain a sub-run $(p,c)\rightarrow^*(p,d)$ with at most K moves, for some $p \in X$ (so, this is necessarily a K-bound run). If $d - c \geq 0$ (resp. $d - c < 0$), then we *transfer* this subrun from its current position to the first occurrence (resp. last occurrence) of p in the run. This still leaves a valid run ξ' since ξ begins with a K as counter value and $|\xi_i| \leq K$. Moreover ξ and ξ' are equivalent upto Parikh-image.

If this ξ' continues to be a $K_>$ run then we again examine if it is $(K,\mathscr{R})$-good and otherwise, repeat the operation described above. As we proceed, we continue to accumulate a increasing iterated K-bound run at the first occurrence of each state and decreasing iterated K-bound run at the last occurrence of each state. We also ensure that in each iteration we only pick a segment that does NOT appear in these $2|X|$ iterated K-bounds. Thus, these iterations will stop when either the segments outside the iterated K-bound are all of length $< K$ and we cannot find any suitable segment to transfer, or when the resulting run is no longer a $K_>$ run. In the first case, we must necessarily have a $(K, 2K^2 + K)$-good run. In the latter case, the resulting run decomposes as usual in $K_\leq$ and $K_>$ components, and we have that every $K_>$ component is strictly shorter than ξ. We formalize the ideas sketched above now.

We begin by proving a Lemma which says that any $K_>$ run ρ can be transformed into a Parikh-equivalent run ξ which is either a $K_>$ run which is $(K, 2K^2 + K)$-good or has a K-decomposition each of whose $K_>$ components are strictly shorter than ρ.

Lemma 26. *Let $\rho = (p,K)\xrightarrow{w}{}^*(q,K)$ be a $K_>$ run in $\mathscr{A}$. Then, there is a run $\xi = (p,K)\xrightarrow{w'}{}^*(q,K)$ in $\mathscr{A}$, with $|\xi| = |\rho|$, Parikh(w) = Parikh(w') such that one of the following holds:*

1. *ξ is not a $K_>$ run. Thus, all $K_>$-components in the K-decomposition of ξ are strictly shorter than ξ (and hence ρ).*
2. *ξ is a $K_>$ run and $\xi = \sigma_1\rho_1\ldots\sigma_n\rho_n$ where $n \leq 2K+1$, each ρ_i is an iterated K-bound run and $|\sigma_i| \leq K$ for each i. Thus, ξ is $(K, 2K^2 + K)$-good.*

Proof. Let $\rho = (p_0,c_0) \xrightarrow{a_1} (p_1,c_1)\ldots \xrightarrow{a_m} (p_m,c_m)$. Let $X \subseteq Q$ be the set of controls states that repeat in the run ρ. We identify the first and last occurrences of each state $q \in X$ along the run ρ, and there are $n = 2.|X| \leq 2K$ such positions. We then decompose the run ρ as follows

$$\begin{aligned}(p_0,c_0) =& (q_0,e_0)\sigma_1(q_1,e_1)\sigma_2(q_2,e_2)\ldots\\ &\ldots(q_{n-1},e_{n-1})\sigma_n(q_n,e_n)\sigma_{n+1}(q_{n+1},e_{n+1}) = (q,d)\end{aligned}$$

where configurations $(q_1,e_1),(q_2,e_2)\ldots(q_n,e_n)$ correspond to the first or last occurrence of states from X. We introduce, for reasons that will become clear in the following, an empty

iterated K-bound run ρ_i following each (q_i, e_i) to get

$$(q_0,e_0)\sigma_1(q_1,e_1)\rho_1(q_1,e_1)\sigma_2(q_2,e_2)\rho_2(q_2,e_2)\dots \\ \dots(q_{n-1},e_{n-1})\sigma_n(q_n,e_n)\rho_n(q_n,e_n)\sigma_{n+1}(q_{n+1},e_{n+1})$$

Let ξ_0 be ρ with the decomposition as written above. We shall now construct a sequence of runs ξ_i, $i \geq 0$, from (p,K) to (q,K), maintaining the length and the Parikh image as an invariant, that is, Parikh(ξ_i) = Parikh(ξ_{i+1}) and $|\xi_i| = |\rho|$. In each step, starting with a $K_>$ run ξ_i, we shall reduce the length of one of the σ_i by some $1 \leq l \leq K$ and increase the length of one iterated K-bound runs ρ_j by l to obtain a run ξ_{i+1}, maintaining the invariant. If this resulting run is not a $K_>$ run then it has a K-decomposition in which every $K_>$ component is shorter than ξ_i (and hence ρ), thus satisfying item 1 of the Lemma completing the proof. Otherwise, after sufficient number of iterations of this step, we will be left satisfying item 2 of the Lemma. Let the $K_>$ run ξ_i be given by

$$(q_0,e_0)\sigma_1^i(q_1,e_1^i)\rho_1^i(q_1,f_1^i)\sigma_2^i(q_2,e_2^i)\rho_2^i(q_2,f_2^i)\dots \\ \dots(q_{n-1},e_{n-1}^i)\sigma_n^i(q_n,e_n^i)\rho_n^i(q_n,f_n^i)\sigma_{n+1}^i(q_{n+1},e_{n+1}^i)$$

where each ρ_j^i is an iterated K-bound run. If the length of $|\sigma_j^i| \leq K$ for each $j \leq n+1$ then, we have already fulfilled item 2 of the Lemma, completing the proof. Otherwise, there is some j such that $|\sigma_j^i| \geq K$. Therefore, we may decompose σ_j^i as

$$(q_{j-1},f_{j-1}^i)\chi_1(r,g)\chi_2(r,g')\chi_3(q_j,e_j^i)$$

where $(r,g)\chi_2(r,g')$ is a run of length $\leq K$ and $r \in X$. There are two cases to consider, depending on whether $g'-g \geq 0$ or $g'-g < 0$.

Let (q_B,e_B^i) and (q_E,f_E^i) be the first and last occurrences of r in ξ_i. We will remove the segment of the run given by χ_2 and add it to ρ_B^i if $g' \geq g$ and add it to ρ_E^i otherwise. First of all, since the first and last occurrences of r are distinct, the ρ_B^i will remain a increasing iterated K-bound run while ρ_E^i remains a decreasing iterated K-bound run. Clearly, such a transformation preserves the Parikh image of the word read along the run. It is easy to check that, since ξ_i is a $K_>$ run and the length of χ_2 is bounded by K, the resulting sequence ξ_{i+1} (after adjusting the counter values) will be a valid run, since the counter stays ≥ 0. However, it may no longer be a $K_>$ run. (This may happen, if $e_B^i < g$ and there is a prefix of χ_2 whose net effect is to reduce the counter by more than $e_B^i - K$.) However, in this case we may set ξ_{i+1} is a run from (p,K) to (q,K), with the same length as ξ_i and thus every $K_>$ component in its K-decomposition is necessarily shorter than ξ_i. Thus, it satisfies item 1 of the Lemma.

If ξ_{i+1} remains a $K_>$ run then we observe that $|\sigma_1^i \dots \sigma_n^i| > |\sigma_1^{i+1} \dots \sigma_n^{i+1}|$ and this guarantees the termination of this construction with a ξ satisfying one of the requirements of the Lemma. □

Starting with any run, we plan to apply Lemma 26, to the $K_>$ components, preserving Parikh-image, till we reach one in which every $K_>$ component satisfies item 2 of Lemma 26. To establish the correctness of such an argument we need the following Lemma.

Lemma 27. *Let $\rho = (p,0) \xrightarrow{w}{}^{*} (q,0)$ be a run. If $\rho = \rho_1(r,K)\rho_2$ then every $K_>$ component in the decomposition of ρ is a $K_>$ component of ρ_1 or ρ_2 and vice versa. In particular, if $\rho = \rho_1(r, K)\rho_2(r',K)\rho_3$ then, $K_>$ components of the K-decomposition of ρ are exactly the $K_>$ components of the runs ρ_1, ρ_2 or ρ_3.*

Proof. By the definition of $K_>$ run and K decompositions. □

We can now combine Lemmas 27 and 26 to obtain:

Lemma 28. *Let $\rho = (p,0) \xrightarrow{w}{}^{*} (q,0)$ be any run in $\mathscr{A}$. Then, there is a run $\rho' = (p,0) \xrightarrow{w'}{}^{*} (q,0)$ of $\mathscr{A}$ with Parikh(w) = Parikh(w') such that every $K_>$ component ξ in the canonical decomposition of ρ' is $(K, 2K^2 + K)$-good.*

Proof. The proof is by double induction, on the length of the longest $K_>$ component in ρ that is not $(K, 2K^2 + K)$-good and the number of components of this size that violate it. For the basis case, observe that any $K_>$ component whose length is bounded by $2K^2 + K$ is necessarily $(K, 2K^2 + K)$-good.

For the inductive case, we pick a $K_>$ component ξ in ρ of maximum size apply Lemma 26 and replace ξ by ξ' to get ρ'. If ξ' is $(K, 2K^2 + K)$-good we have reduced the number of components of the maximum size that are not $(K, 2K^2 + 2)$-good in ρ'. Otherwise, ξ' satisfies item 2 of Lemma 26 and thus by Lemma 27 the number of $K_>$ components in the decomposition of ρ' of the size of ξ that are not $(K, 2K^2 + K)$-good is one less than that in ρ. This completes the inductive argument. □

Let $\mathscr{A}^K$ be the NFA simulating the counter system $\mathscr{A}$ when the counter values lie in the range $[0, K]$, by maintaining the counter values in its local state. This automaton is of size $O(K^2)$. Now, suppose for each pair of states $p, q \in Q$ we have an NFA $\mathscr{B}^{pq}$ which is Parikh-equivalent to $L_{2K^2+K}(\mathscr{A}[K]^{p,q})$, where $\mathscr{A}[K]^{p,q}$ is the automaton $\mathscr{A}[K]$ with p as the only initial state and q as the only accepting state. We combine these automata (there are K^2 of them) with $\mathscr{A}^K$ by taking their disjoint union and adding the following additional (internal) transitions. We add transitions from the states of the from (p,K) of $\mathscr{A}^K$, for $p \in Q$ to the initial state of state of all the $\mathscr{B}^{pq}$, $q \in Q$. Similarly, from the accepting states of $\mathscr{B}^{pq}$ we add internal transitions to the state (q,K) in $\mathscr{A}^K$. Finally we deem $(s,0)$ to be the only initial state and $(f,0)$ to be the only final state of the combined automaton. We call this NFA $\mathscr{B}$.

Lemma 29. *Parikh$(L(\mathscr{B}))$ = Parikh$(L(\mathscr{A}))$*

Proof. Let ρ be an accepting run of $\mathscr{A}$ on a word w. We first apply Lemma 28 to construct a run ρ' on a w', with Parikh(w) = Parikh(w'), in whose K-decomposition, every $K_>$ component is $(K, 2K^2 + K)$-good. Let $\chi = (p,K) \xrightarrow{x}{}^{*} (q,K)$ be such a component. Then, by Lemma 25, there is a run $\chi' : (p,0) \xrightarrow{x}{}^{*} (q,0)$ in $\mathscr{A}[K]$ with at most $2K^2 + K$ reversals. Thus, there is a $x' \in L(\mathscr{B}^{pq})$ with Parikh(x) = Parikh(x'). If $(s,0) \xrightarrow{x}{}^{*} (p,K)$ is a $K_\leq$ component of ρ' then $(s,0) \xrightarrow{x}{}^{*} (p,K)$ in $\mathscr{A}^K$. If $(p,K) \xrightarrow{x}{}^{*} (q,K)$ is a $K_\leq$ component of ρ' then $(p,K) \xrightarrow{x}{}^{*} (q,K)$ in $\mathscr{A}^K$ and finally if $(p,K) \xrightarrow{x}{}^{*} (f,0)$ is a $K_\leq$ component of ρ' then $(p,K) \xrightarrow{x}{}^{*} (f,0)$ in $\mathscr{A}^K$. Putting these together we get a run from $(s,0)$ to $(f,0)$ in $\mathscr{B}$ on a word Parikh-equivalent to w' and hence w.

For the converse, any word in $L(\mathscr{B})$ is of the form $x.u_1.v_1.u_2.v_2 \ldots u_n v_n.y$ where $(s, 0) \xrightarrow{x}{}^* (p_1, K)$ in $\mathscr{A}^K$, $(q_n, K) \xrightarrow{y}{}^* (f, 0)$ in $\mathscr{A}^K$, $u_i \in L(\mathscr{B}^{p_i q_i})$ and $(q_i, K) \xrightarrow{v_i}{}^* (p_{i+1}, K)$ in $\mathscr{A}^K$, for each $1 \leq i \leq n$. By construction, there is a run $(s, 0) \xrightarrow{x}{}^* (p_1, K)$ in $\mathscr{A}$ and $(q_n, K) \xrightarrow{y}{}^* (f, 0)$ in M. Further for each i, there is a run $(q_i, K) \xrightarrow{v_i}{}^* (p_{i+1}, K)$ in $\mathscr{A}$ as well. Since $u_i \in L(\mathscr{B}^{p_i q_i})$, by construction of $\mathscr{B}^{p_i q_i}$, there is a run $(p_i, 0) \xrightarrow{u_i'}{}^* (q_i, 0)$ in $\mathscr{A}[K]$ with Parikh(u_i) = Parikh(u_i'). But then, by the second part of Lemma 22, there is a run $(p_i, K) \xrightarrow{u_i'}{}^* (q_i, K)$ in $\mathscr{A}$. Thus we can put together these different segments now to obtain an accepting run in $\mathscr{A}$ on the word $x.u_1'.v_1.u_2'.v_2 \ldots u_n' v_n$. Thus, Parikh$(L(\mathscr{B})) \subseteq$ Parikh$(L(\mathscr{A}))$, completing the proof of the Lemma. □

The number of states in the automaton $\mathscr{B}$ is $\sum_{p,q \in Q} |\mathscr{B}^{pq}| + K^2$. What remains to be settled is the size of the automata $\mathscr{B}^{pq}$. That is, computing an upper bound on the size of an NFA which is Parikh-equivalent to the language of words accepted by an counter automata (in this case $M[K]$) along runs with at most R (in this case $K^2 + K$) reversals. This problem is solved in the next subsection and the solution (see Lemma 32) implies that that the size of $\mathscr{B}^{pq}$ is bounded by $O(|\Sigma| K^{O(log K)})$. Thus we have

Theorem 13. *There is an algorithm, which given an counter automata with K states and alphabet Σ, constructs a Parikh-equivalent NFA with $O(|\Sigma|.K^{O(\log K)})$ states.*

4.6.2 Parikh image under reversal bounds

Here we show that, for any counter system $\mathscr{A}$, with K states and whose alphabet is Σ, and any $R \in \mathbb{N}$, an NFA Parikh-equivalent to $L_R(\mathscr{A})$ can be constructed with size $O(|\Sigma|.K^{O(log K)})$. As a matter of fact, this construction works even for pushdown systems and not just for counter systems.

Let $\mathscr{A}$ be a simple counter system. It will be beneficial to think of the counter as a stack with a single letter alphabet, with pushes for increments and pops for decrements. Then, in any run from $(p, 0)$ to $(q, 0)$, we may relate an increment move uniquely with its *corresponding* decrement move, the pop that removes the value inserted by this push.

Now, consider a *one reversal run* ρ of $\mathscr{A}$ from say $(p, 0)$ to $(q, 0)$ involving two phases, a first phase ρ_1 with no decrement moves and a second phase ρ_2 with no increment moves. Such a run can be simulated, up to equivalent Parikh image (i.e. upto reordering of the letters read along the run) by an NFA as follows: simultaneously simulate the first phase (ρ_1) from the source and the second phase, in reverse order (ρ_2^{rev}), from the target. (The simulation of ρ_2^{rev} uses the transitions in the *opposite* direction, moving from the target of the transition to the source of the transition). The simulation matches increment moves of ρ_1 against decrement moves in ρ_2^{rev} (more precisely, matching the ith increment ρ_1 with the ith decrement in ρ_2^{rev}) while carrying out moves that do not alter the counters independently in both directions. The simulation terminates (or potentially terminates) when a common state, signifying the boundary between ρ_1 and ρ_2 is reached from both ends.

The state space of such an NFA will need pairs of states from Q, to maintain the current state reached by the forward and backward simulations. Since, only one letter of the input

can be read in each move, we will also need two moves to simulate a matched increment and decrement and will need states of the form $Q \times Q \times \Sigma$ for the intermediate state that lies between the two moves.

Unfortunately, such a naive simulation would not work if the run had more *reversals*. For then the ith increment in the simulation from the left need not necessarily correspond to the ith decrement in the reverse simulation from the right. In this case, the run ρ can be written as follows:

$$(p,0)\rho_1(p_1,c) \xrightarrow{\tau_1} (p_1',c+1)\rho_3(p_2',c+1) \xrightarrow{\tau_2} (p_2,c)\rho_4(q_1,c)\rho_5(q,0)$$

where, the increment τ_1 corresponds to the decrement τ_2 and all the increments in ρ_1 are exactly matched by decrements in ρ_5. Notice that the increments in the run ρ_3 are exactly matched by the decrements in that run and similarly for ρ_4. Thus, to simulate such a well-matched run from p to q, after simulating ρ_1 and ρ_5^{rev} simultaneously matching corresponding increments and decrements, and reaching the state p_1 on the left and q_1 on the right, we can choose to now simulate matching runs from p_1 to p_2 and from p_2 to q_1 (for some p_2). Our idea is to choose one of these pairs and simulate it first, storing the other in a stack. We call such pairs *obligations*. The simulation of the chosen obligation may produce further such obligations which are also stored in the stack. The simulation of an obligation succeeds when the state reached from the left and right simulations are identical, and at this point we we may choose to close this simulation and pick up the next obligation from the stack or continue simulating the current pair further. The entire simulation terminates when no obligations are left. Thus, to go from a single reversal case to the general case, we have introduced a stack into which states of the NFA used for the single reversal case are stored. This can be formalized to show that the resulting PDA is Parikh-equivalent to $\mathscr{A}$.

Observe that in this construction each obligation inserted into the stack corresponds to a reversal in the run being simulated, as a matter of fact, it will correspond to a reversal from decrements to increments. Thus it is quite easy to see that the stack height of the simulating run can be bounded by the number of reversals in the original run.

But a little more analysis shows that there is a simulating run where the height of the stack is bounded by $log(R)$ where R is the number of reversals in the original run. Thus, to simulate all runs of $\mathscr{A}$ with at most R reversals, we may bound the stack height of the PDA by $log(R)$.

We show that if the stack height is h then we can choose to simulate only runs with at most $2^{log(R)-h}$ reversals for the obligation on hand. Once we show this, notice that when $h = log(R)$ we only need to simulate runs with 1 reversal which can be done without any further obligations being generated. Thus, the overall height of the stack is bounded by $log(R)$. Now, we explain why the claim made above holds. Clearly it holds initially when $h = 0$. Inductively, whenever we split an obligation, we choose the obligation with fewer reversals to simulate first, pushing the other obligation onto the stack. Notice that this obligation with fewer reversals is guaranteed to contain at most half the number of reversals of the current obligation (which is being split). Thus, whenever the stack height increases by 1, the number of reversals to be explored in the current obligation falls at least by half as required. On the other hand, an obligation (p,q) that lies in the stack at position h from the bottom, was placed there while executing (earlier) an obligation (p',q') that only required 2^{k-h+1} reversals. Since

the obligation (p,q) contributes only a part of the obligation (p',q'), its number of reversals is also bounded by 2^{k-h+1}. And when (p,q) is removed from the stack for simulation, the stack height is $h-1$. Thus, the invariant is maintained.

We now describe the formal construction of the automaton and establish its correctness now. We establish the result directly for a pushdown system. Recall that if Γ is a singleton we have exactly a counter system.

Given a PDA $\mathscr{A} = (Q,\Sigma,\Gamma,\delta,\perp,s,F)$ we construct a new PDA $\mathscr{A}_P$ which simulates runs of $\mathscr{A}$, upto Parikh-images, and does so using runs where the stack height is bounded by $log(R)$ where R is the number of reversals in the run of $\mathscr{A}$ being simulated. $\mathscr{A}_P = (Q_P \cup \{s_P, t_P\}, \Sigma, \Gamma_P, \delta_P, s_P, t_P)$ is defined as follows. The set $Q_P = \Gamma_P$ is given by $(Q \times Q) \cup (Q \times Q \times \Sigma)$. States of the form (p,q) are charged with simulating a well matched run from $(p,\perp)$ to $(q,\perp)$. While carrying out a matched push from the left and a pop from the right, as we are only allowed read one letter of Σ in a single move, we are forced to have an intermediary state to allow for the reading of the letters corresponding to both the transitions being simulated. The states of the form (p,q,a), $a \in \Sigma$, are used for this purpose. The transition relation δ_P is described below:

1. $(s_P, \epsilon, \mathbf{Int}, (s,t)) \in \delta_P$. Initialize the start and target states.
2. $((p,q), \mathbf{Int}, a, (p',q)) \in \delta_P$ whenever $(p, \mathbf{Int}, a, p') \in \delta$. Simulate an internal move from the left.
3. $((p,q), \mathbf{Int}, a, (p,q')) \in \delta_P$ whenever $(q', \mathbf{Int}, a, q) \in \delta$. Simulate an internal move from the right.
4. $((p,q), \mathbf{Int}, a, (p',q',b)) \in \delta_P$ whenever $(p, \mathbf{Push}(x), a, p')$, $(q', \mathbf{Pop}(x), a, q) \in \delta$ for some $x \in \Gamma$. Simulate a pair of matched moves, a push from the source and the corresponding pop from the target, first part.
5. $((p,q,b), \mathbf{Int}, b, (p,q)) \in \delta_P$ whenever $b \in \Sigma$. Second part of the move described in previous item.
6. $((p,q), \mathbf{Push}((q',q)), \epsilon, (p,q')) \in \delta_P$ for every state $q' \in Q$. Guess a intermediary state where a pop to push reversal occurs. Simulate first half first and push the second as an obligation on the stack.
7. $((p,q), \mathbf{Push}((p,q')), \epsilon, (q',q)) \in \delta_P$ for every state $q' \in Q$. Guess a intermediary state where a pop to push reversal occurs. Simulate second half first and push the first as an obligation on the stack.
8. $((p,p), \mathbf{Pop}((p',q')), \epsilon, (p',q')) \in \delta_P$. Current obligation completed, load next one from stack.
9. $((p,p), \mathbf{Zero}, \epsilon, t_P) \in \delta_P$. All segments completed successfully, so accept.

The following Lemma shows that every run of $\mathscr{A}_P$ simulates some run of $\mathscr{A}$ upto Parikh-image. In what follows, we recall that a run ρ is a γ-run for some $\gamma \in \Gamma^*\perp$ if γ is a suffix of the stack contents in every configuration in ρ (denoted $\rightarrow_\gamma$).

Lemma 30. *Let $\beta \in \Gamma_P^*\perp$. Let $((p,q),\beta) \xrightarrow{w}{}^* ((r,r),\beta)$ be a β-run in $\mathscr{A}_P$, for some p,q and r in Q. Then, for every $\gamma \in (\Gamma \setminus \{\perp\})^*\perp$ there is a run $(p,\gamma) \xrightarrow{w'}{}^* (q,\gamma)$ in $\mathscr{A}$ such that Parikh$(w') =$ Parikh(w). Thus, if $w \in L(\mathscr{A}_P)$ then there is a w' in $L(\mathscr{A})$ with Parikh$(w) =$ Parikh(w').*

Proof. Proof of the Lemma directly follows from the following Claim.

Claim 4. *If there is a run of the form* $((p,q),\beta)\xrightarrow{v}{}^{*}((p',q'),\beta)$ *in* $\mathscr{A}_P$*, then for every* $\gamma \in (\Gamma\setminus\{\bot\})^*$*, there are runs of the form* $(p,\gamma\bot)\xrightarrow{v_1}{}^{*}(p',\alpha\gamma\bot)$ *and* $(q',\alpha\gamma\bot)\xrightarrow{v_2}{}^{*}(q,\gamma\bot)$*, for some* $\alpha \in (\Gamma\setminus\{\bot\})^*$ *such that* $Parikh(v) = Parikh(v_1.v_2)$.

Proof. We will now prove this by inducting on stack height and on length of the run. Suppose the stack was never used (always remained β), then the proof is easy to see. Then we proceed by length of the run.

The only interesting case is when the run is of the form $((p,q),\beta)\xrightarrow{v_1}{}^{*}((p_1,q_1),\beta)\xrightarrow{a_1a_2}{}^{*}((p',q'),\beta)$, where the transition used to execute the sub-run $((p_1,q_1'),\beta)\xrightarrow{a_1a_2}{}^{*}((p',q'),\beta)$ are $\tau_1 = ((p_1,q_1),\mathbf{Int},a_1,(p',q',a_2)$ (from 4) followed by $\tau_2 = ((p',q',a_2),\mathbf{Int},a_2,(p',q'))\in\delta_P$ (from 5) . In this case, clearly there are transitions of the form $(p_1,\mathbf{Push}(x),a_1,p')$ and $(q',\mathbf{Pop}(x),a_1,q_1)\in\delta$. From this we have $(p_1,\gamma\bot)\rightarrow^{*}(p',x\gamma\bot)$ and $(q',x\gamma\bot)\rightarrow^{*}(q_1,\gamma\bot)$. Combining this with the run got by induction, gives us the required run.

Let us assume that stack was indeed used, then the run $((p,q),\beta)\xrightarrow{v}{}^{*}((p',q'),\beta)$ can be split as

$$((p,q),\beta)\xrightarrow{v_1}{}^{*}((p_1,q_1),\beta)\rightarrow((p_2,q_2),(t_1,t_2)\beta)\xrightarrow{v_2}{}^{*}$$
$$((r_1,r_1),(t_1,t_2)\beta)\rightarrow((t_1,t_2),\beta)\xrightarrow{v_3}{}^{*}((p',q'),\beta)$$

We have two cases to consider, either $q_1 = t_2$ or $p_1 = t_1$. We will consider the case where $q_1 = t_2$, the other case is analogous. In this case, clearly $p_2 = p_1$ and $t_1 = q_2$. Hence the run is of the form

$$((p,q),\beta)\xrightarrow{v_1}{}^{*}((p_1,q_1),\beta)\rightarrow((p_1,q_2),(q_2,q_1)\beta)\xrightarrow{v_2}{}^{*}$$
$$((r_1,r_1),(q_2,q_1)\beta)\rightarrow((q_2,q_1),\beta)\xrightarrow{v_3}{}^{*}((p',q'),\beta)$$

Now consider the sub-run of the form

$$((p,q),\beta)\xrightarrow{v_1}{}^{*}((p_1,q_1),\beta)$$

clearly such a run is shorter and hence by induction we have a corresponding runs of the form $(p,\gamma\bot)\xrightarrow{v_1'}{}^{*}(p_1,\alpha''\gamma\bot)$ and $(q_1,\alpha''\gamma\bot)\xrightarrow{v_1''}{}^{*}(q,\gamma\bot)$, for all $\gamma\in(\Gamma\setminus\{\bot\})^*$ some $\alpha''\in(\Gamma\setminus\{\bot\})^*$ and such that $\text{Parikh}(v_1) = \text{Parikh}(v_1'.v_1'')$.

Consider the sub-run of the form

$$((p_1,q_2),(q_2,q_1)\beta)\xrightarrow{v_2}{}^{*}((r_1,r_1),(q_2,q_1)\beta)$$

clearly stack height of such a run is shorter by 1. Hence by induction, we have a corresponding runs of the form, $(p_1,\gamma\bot)\xrightarrow{v_2'}{}^{*}(r_1,\alpha'\gamma'\bot)$ and $(r_1,\alpha'\gamma'\bot)\xrightarrow{v_2''}{}^{*}(q_2,\gamma'\bot)$ for some $\alpha'\in(\Gamma\setminus\{\bot\})^*$ and all $\gamma'\in(\Gamma\setminus\{\bot\})^*$, such that $\text{Parikh}(v_2) = \text{Parikh}(v_2'.v_2'')$. Hence we also have the run $(r_1,\alpha'\alpha''\gamma\bot)\xrightarrow{v_2''}{}^{*}(q_2,\alpha''\gamma\bot)$ and a run of the form $(r_1,\alpha'\alpha''\gamma\bot)\xrightarrow{v_2''}{}^{*}(q_2,\alpha''\gamma\bot)$.

consider the sub-run of the form

$$((q_2, q_1), \beta) \xrightarrow{v_3}{}^{*} ((p', q'), \beta)$$

clearly such a run is shorter in length, hence by induction, we have corresponding runs $(q_2, \gamma' \perp) \xrightarrow{v_3'}{}^{*} (p', \alpha\gamma' \perp)$ and $(q', \alpha\gamma' \perp) \xrightarrow{v_3''}{}^{*} (q_1, \gamma' \perp)$, for some $\alpha \in (\Gamma \setminus \{\perp\})^*$ and all $\gamma' \in (\Gamma \setminus \{\perp\})^*$ and such that Parikh(v_3) = Parikh$(v_3'.v_3'')$. Hence we also have $(q_2, \alpha''\gamma \perp) \xrightarrow{v_3'}{}^{*} (p', \alpha\alpha''\gamma \perp)$ and $(q', \alpha\alpha''\gamma \perp) \xrightarrow{v_3''}{}^{*} (q_1, \alpha''\gamma \perp)$

Now combining these sub-runs, we get the required run. □

□

In the other direction, we show that every run of $\mathscr{A}$ is simulated upto Parikh-image by $\mathscr{A}_P$ with a stack height that is logrithmic in the number of reversals. The next Lemma shows how $\mathscr{A}_R$ simulates runs of $\mathscr{A}$ and provides bounds on stack size in terms of the number of reversals of the run in $\mathscr{A}$.

Lemma 31. *Let $(p, \alpha) \xrightarrow{w}{}^{*} (q, \alpha)$ be a α-run of $\mathscr{A}$ with R reversals with $\alpha \in \Gamma^*.\perp$. Then, for any $\gamma \in \Gamma_P^* \perp$, there is a γ-run $((p, q), \gamma) \xrightarrow{w'}{}^{*} ((r, r), \gamma)$ with Parikh(w) = Parikh(w'). Further for any configuration along this run the height of the stack is no more than $|\gamma| + log(R + 1)$.*

Proof. The proceeds by a double induction, first on the number of reversals and then on the length of the run.

For the base case, suppose $R = 0$. If the length of the run is 0 then the result follows trivially. Otherwise, we will first recall that we use $\xrightarrow{\tau}$ to refer to transition relation labeled with the transitions (as opposed to letters). Let the α-run ρ, $\alpha \in \Gamma^* \perp$ be of the form:

$$(p, \alpha) = (p_0, \alpha_0) \xrightarrow{\tau_1} (p_1, \alpha_1) \xrightarrow{\tau_2} \ldots \xrightarrow{\tau_n} (p_n, \alpha_n) = (q, \alpha)$$

If τ_1 is an internal move $(p_0, \mathbf{Int}, a_1, p_1)$ then $((p_0, p_n), \mathbf{Int}, a_1, (p_1, p_n))$ is a transition δ_P (of type 2). Thus

$$((p_0, p_n), \gamma) \xrightarrow{a_1} ((p_1, p_n), \gamma)$$

is a valid move in $\mathscr{A}_P$. Let $w = a_1 w_1$. Then, by induction hypothesis, there is a γ-run

$$((p_1, p_n), \gamma) \xrightarrow{w_1'}{}^{*} ((r, r), \gamma))$$

with Parikh(w_1') = Parikh(w_1), whose stack height is bounded by $|\gamma|$. Putting these two together we get a γ-run

$$((p_0, p_n), \gamma) \xrightarrow{a_1.w_1'}{}^{*} ((r, r), \gamma)$$

with Parikh(w) = Parikh$(a_1.w_1')$ whose stack height is bounded by $|\gamma|$ as required.

If τ_n is an internal transition $(p_{n-1}, a_n, \mathbf{Int}, p_n)$ then

$$((p_0, p_n), a_n, \mathbf{Int}, (p_0, p_{n-1})) \in \delta_P$$

is a transition of of type 3. Thus, $((p_0, p_n), \gamma) \xrightarrow{a_n} ((p_0, p_{n-1}), \gamma)$ is a move in $\mathscr{A}_P$. Further, by the induction hypothesis, there is a word w_2 with $w = w_2.a_n$ and a γ-run $((p_0, p_{n-1}), \gamma) \xrightarrow{w_2'}{}^{*} ((r, r), \gamma)$ with Parikh(w_2) = Parikh(w_2'). Then, since Parikh$(a_n.w_2')$ = Parikh$(w_2.a_n)$, we can put these two together to get the requisite run. Once again the stack height is bounded by $|\gamma|$.

Since the given run is a α-run, the only other case left to be considered is when τ_1 is a push move and τ_n is a pop move. Thus, let $\tau_1 = (p_0, \mathbf{Push}(x_1), a_1, p_1)$ and $\tau_n = (p_{n-1}, \mathbf{Pop}(x_n), a_n, p_n)$. We claim that $x_1 = x_n$ and as a matter fact the value x_1 pushed by τ_1 remains in the stack all the way till end of this run and is popped by τ_n. If the x_1 was popped earlier in the run than the last step, then the stack height would have necessarily reached $|\alpha|$ at this pop, and therefore there will necessarily be a subsequent push of x_n. But this contradicts the fact that $R = 0$. Thus, we have the following moves in $\mathscr{A}_P$.

$$((p_0, p_n), \gamma) \xrightarrow{((p_0,p_n),\mathbf{Int},a_1,(p_1,p_{n-1},a_n))} ((p_0, p_{n-1}, a_n), \gamma)$$
$$\xrightarrow{((p_1,p_{n-1},a_n),\mathbf{Int},a_n,(p_1,p_{n-1})}((p_1, p_{n-1}), \gamma)$$

Let $w = a_1 w_3 a_n$. Then applying the induction hypothesis we get a γ-run $((p_1, p_{n-1}), \gamma) \xrightarrow{w_3'}{}^{*} ((r, r), \gamma)$ where the stack height is never more than $|\gamma|$. Combining these two gives us a γ-run $((p_0, p_n), \gamma) \xrightarrow{a_1 a_n w_3'}{}^{*} ((r, r), \gamma)$ where the stack height is never more than $|\gamma|$. Observing that Parikh$(a_1 a_n w_3')$ = Parikh$(a_1 w_3 a_n)$ gives us the desired result.

Now we examine runs with $R \geq 1$. And once again we proceed by induction on the length l of runs with R reversals. For $R \geq 1$ there are no runs of length $l = 0$ and so the basis holds trivially. As usual, let

$$(p, \alpha) = (p_0, \alpha_0) \xrightarrow{\tau_1} (p_1, \alpha_1) \xrightarrow{\tau_2} \ldots \xrightarrow{\tau_n} = (p_n, \alpha_n) = (q, \alpha)$$

be an α-run with R reversals. If either τ_1 or τ_n is an internal move then the proof can proceed by induction on l exactly along the same lines as above and the details are omitted. Otherwise, since this is a α-run, τ_1 is a push move and τ_n is a pop move. Let $\tau_1 = (p_0, a_1, push(x_1), p_1)$ and $\tau_n = (p_{n-1}, a_n, pop(x_n), p_n)$. Now we have two possibilities.

Case 1: The value x_1 pushed in τ_1 is popped only by τ_n. This is again easy, as we can apply the same argument as in the case $R = 0$ to conclude that,

$$((p_0, p_n), \gamma) \xrightarrow{((p_0,p_n),\mathbf{Int},a_1,(p_1,p_{n-1},a_n))} ((p_0, p_{n-1}, a_n), \gamma)$$
$$\xrightarrow{((p_1,p_{n-1},a_n),\mathbf{Int},a_n,(p_1,p_{n-1})}((p_1, p_{n-1}), \gamma)$$

Again, with $w = a_1 w_3 a_2$, and applying the induction hypothesis to the shorter run $(p_1, \alpha_1) \xrightarrow{w_3}{}^{*} (p_{n-1}, \alpha_{n-1})$ with exactly R reversals, we obtain a γ-run

$$((p_1, p_{n-1}), \gamma) \xrightarrow{w_3'}{}^{*} ((r, r), \gamma)$$

in which the height of the stack is bounded by $|\gamma| + log(R+1)$. Combining these gives us the γ-run with stack height bounded by $|\gamma| + log(R+1)$, $((p_0, p_n), \gamma) \xrightarrow{a_1 a_n w_3'}{}^{*} ((r, r), \gamma)$ as required.

Case 2: The value x_1 pushed in τ_1 is popped by some τ_j with $j < n$. Then we break the run into two α-runs, $\rho_1 = (p_0, \alpha_0) \xrightarrow{a_1 \ldots a_j}{}^{*} (p_j, \alpha_j)$ and $\rho_2 = (p_j, \alpha_j) \xrightarrow{a_{j+1} \ldots a_n}{}^{*} (p_n, \alpha_n)$. Note that $\alpha = \alpha_0 = \alpha_j = \alpha_n$. Let $a_1 \ldots a_j = w_1$ and $a_{j+1} \ldots a_n = w_2$. Let the number of reversals of ρ_1 and ρ_2 be R_1 and R_2 respectively. First of all, we observe that $R_1 + R_2 + 1 = R$. Thus $R_1, R_2 < R$ and further either $R_1 \leq R/2$ or $R_2 \leq R/2$.

Suppose $R_1 \leq R/2$. Then, by the induction hypothesis, there is an $((p_j, p_n)\gamma)$-run

$$\rho_1' \;=\; (((p_0, p_j), (p_j, p_n).\gamma) \xrightarrow{w_1'}{}^{*} ((r', r'), (p_j, p_n).\gamma))$$

with $\text{Parikh}(w_1) = \text{Parikh}(w_1')$ and whose stack height is bounded by

$$\begin{aligned} |(p_j, p_n).\gamma| + log(R_1 + 1) &= |\gamma| + 1 + log(R_1 + 1) \\ &\leq |\gamma| + 1 + log(R + 1) - 1 \\ &= |\gamma| + log(R + 1) \end{aligned}$$

Similarly, by the induction hypothesis, there is an γ-run $\rho_2' = ((p_j, p_n), \gamma) \xrightarrow{w_2'}{}^{*} ((r, r), \gamma)$ whose number of reversals is bounded by $|\gamma| + log(R_2 + 1) \;\leq\; |\gamma| + log(R + 1)$ and for which $\text{Parikh}(w_2') = \text{Parikh}(w_2)$.

We have everything in place now. We construct the desired run by first using a transition of type 6, following by ρ_1', followed by a transition of type 8, followed by a simulation of ρ_2' to obtain the following:

$$\begin{aligned} ((p_0, p_n), \gamma) \xrightarrow{((\boldsymbol{p_0}, \boldsymbol{p_n}), \textbf{Push}((\boldsymbol{p_j}, \boldsymbol{p_n})), \boldsymbol{\epsilon}, (\boldsymbol{p_0}, \boldsymbol{p_j}))} ((p_0, p_j), (p_j, p_n).\gamma) \xrightarrow{w_1'}{}^{*} \\ ((r', r'), (p_j, p_n)\gamma) \xrightarrow{((\boldsymbol{r'}, \boldsymbol{r'}), \textbf{Pop}((\boldsymbol{p_j}, \boldsymbol{p_n})), \boldsymbol{\epsilon}, (\boldsymbol{p_j}, \boldsymbol{p_n}))} ((p_j, p_n), \gamma) \xrightarrow{w_2'}{}^{*} ((r, r), \gamma) \end{aligned}$$

This runs satisfies all the desired properties. The case where $R_2 \leq R/2$ is handled similarly using moves of type 7 instead of type 6 and using the fact the $\text{Parikh}(w_2'.w_1') = \text{Parikh}(w_1'.w_2')$. This completes the proof of the Lemma. □

As we did for counter systems we let $L_R(\mathscr{A})$ refer to the language of words accepted by $\mathscr{A}$ along runs with at most R reversals. Now, for a given R, we can simulate runs of $\mathscr{A}_P$ where stack height is bounded by $log(R)$, using an NFA by keeping the stack as part of the state. The size of such an NFA is $O(|Q_P||\Gamma_P|^{O(log(R))}) = O(|\Sigma||Q|^{O(log(R))})$. Let $\mathscr{A}_R$ be such an NFA. Then by Lemma 30, we have $\text{Parikh}(L(\mathscr{A}_R)) \subseteq \text{Parikh}(L(\mathscr{A}))$ and by Lemma 31 we also have $\text{Parikh}(L_R(\mathscr{A})) \subseteq \text{Parikh}(L(\mathscr{A}_R))$. By keeping track of the reversal count in the state, we may construct an $\mathscr{A}'$ with state space size $O(R.|Q|)$ such that that $L(\mathscr{A}') = L_R(\mathscr{A}') = L_R(\mathscr{A})$. Thus, we have

Lemma 32. *There is a procedure that takes a simple OCA $\mathscr{A}$ with K states and whose alphabet is Σ, and a number $R \in \mathbb{N}$ and returns an NFA Parikh-equivalent to $L_R(\mathscr{A})$ of size $O(|\Sigma|.(RK)^{O(log(R))})$.*

4.7 Conclusion

In this chapter, we studied language theoretic features of context-free languages. We first showed that the downward and upward closed (w.r.t. subword relation) language of a counter automata can effectively be represented by a polynomial sized NFA.We then showed that given a reversal bounded pushdown system, we can effectively obtain an Parikh equivalent, sub exponential sized finite state automaton. Using this we showed that an Parikh equivalent finite state representation of language of counter system is at most sub-exponential in the size of the counter system. We conjecture that such a finite state representation recognising the Parikh image abstraction of counter system is tight. We further believe that the lower bound can be obtained from a class of counter systems, described in Figure 4.1.

The counter system C_n, (for any $n \in \mathbb{N}$) operates on the alphabets $\{a_1, \cdots, a_n, b_1, \cdots, b_n\}$. It proceeds in phases and accepts a word of the form $c_1^* c_2^* \cdots c_n^*$, where c_i either a_i or b_i. In each phase it either reads a word of the form a_i^* or of the form b_i^*. If it reads a word of the form a_i^* then, the counter is incremented (denoted in the figure as $+$) for each word read. Similarly if b_i^* is chosen in a phase then the counter is decremented (denoted in the figure as $-$) for each b_i that is read.

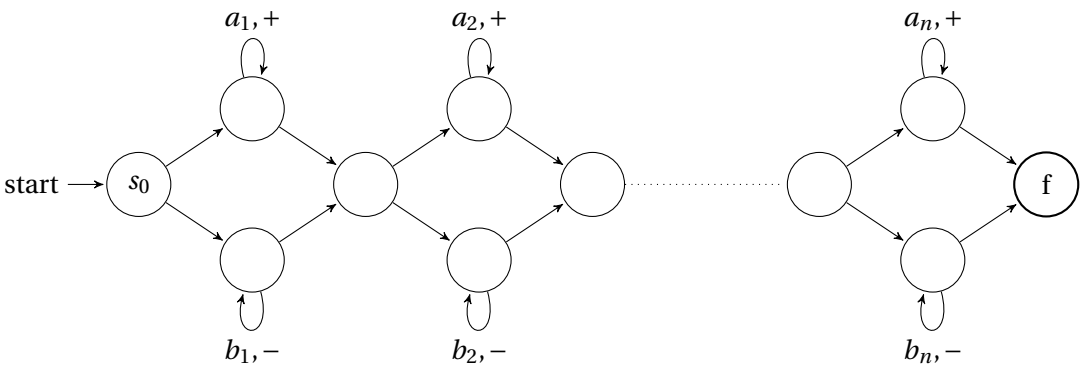

Figure 4.1: C_n

We do not know how to construct polynomial sized NFAs for this family of counter systems and believe that such a family of NFAs does not exist.

Chapter 5

Multi-pushdown systems (MPDS)

5.1 Introduction

In this chapter, we describe the *multi-pushdown* system model and some related results. This will be useful in the following chapters where we describe out results on this model. Informally a multi-pushdown system is a generalisation of pushdown systems equipped with multiple stacks. Here, each recursive thread is modelled using a pushdown stack. Thus it naturally generalises the use of pushdown systems to model sequential recursive program to those with bounded number of recursive threads. In full generality, MPDS are not analysable as even two stacks are sufficient to simulate a Turing machine [126]. The focus therefore has been on identifying restrictions on behaviours of such systems that leads to decidability of verification problems. We first introduce the MPDS model and then discuss some of the well known results.

5.2 Multi-pushdown system

Definition 4 (MPDS)**.** *A* Multi-PushDown System *(MPDS) is a tuple* $\mathrm{M} = (n, Q, \Gamma, \Delta, q_0, \gamma_0)$ *where:*

1. $n \geq 1$ *is the number of stacks*
2. Q *is the non-empty set of states,*
3. Γ *is the finite set of stack symbols, containing a special symbol* $\perp$.
4. $q_0 \in Q$ *is the initial state,* $\gamma_0 \in (\Gamma_\epsilon = \Gamma \cup \{\epsilon\})$ *is the initial stack symbol*
5. $\Delta \subseteq Q \times \bigcup_{i\in[1..n]} \mathbf{Op}_i \times Q$ *is the transition relation, where* $\mathbf{Op}_i = \{\mathbf{Push}_i(a), \mathbf{Pop}_i(a) \mid a \in \Gamma \setminus \{\perp\}\} \cup \{\mathbf{Zero}_i, \mathbf{Int}_i\}$.

We will use $\mathbf{Op}$ *to denote set of all transitions i.e.* $\mathbf{Op} = \bigcup_{i\in[1..n]} \mathbf{Op}_i$ *and* Δ_i *to mean* $\Delta_i = \Delta \cap (Q \times \mathbf{Op}_i \times Q)$.

A configuration of the MPDS M is a (n+1) tuple $(q, w_1, w_2, \cdots, w_n)$ with $q \in Q$, and w_1, $w_2, \ldots, w_n \in \Gamma^* \perp$. The set of configurations of the MPDS M is denoted by $\mathscr{C}(M)$. The *initial configuration* c_M^{init} of the MPDS M is $(q_0, \perp, \ldots, \perp, \gamma_0\perp)$. Given $\tau = (q, op, q') \in \Delta$. Given two

configurations $c = (q, \gamma_1, \cdots, \gamma_n)$ and $c' = (q', \gamma'_1, \cdots, \gamma'_n)$ (where for all $i \in [1..n]$, $\alpha_i, \gamma_i \in \Gamma^* \perp$) we say $c \xrightarrow{\tau} c'$ iff one of the following holds.

- $\tau = (q, \mathbf{Push}_i(a), q')$, $\gamma'_i = a.\gamma_i$ and $\forall\, j \in [1..n] \setminus \{i\}, \gamma'_j = \gamma_j$
- $\tau = (q, \mathbf{Pop}_i(a), q')$, $\gamma_i = a.\gamma'_i$ and $\forall\, j \in [1..n] \setminus \{i\}, \gamma'_j = \gamma_j$
- $\tau = (q, \mathbf{Zero}_i, q')$, $\gamma'_i = \gamma_i = \perp$ and $\forall\, j \in [1..n] \setminus \{i\}, \gamma'_j = \gamma_j$
- $\tau = (q, \mathbf{Int}_i, q')$ for some $i \in [1..n]$ and $\forall\, j \in [1..n], \gamma'_j = \gamma_j$

For any subset $T \subseteq \Delta$, we define $\rightarrow_T$ as $\bigcup_{\tau \in T} \xrightarrow{\tau}_M$. We write $\rightarrow^*{}_T$ to denote the reflexive and transitive closure of the relation $\rightarrow_T$. For every sequence of transitions $\rho = \tau_1 \tau_2 \ldots \tau_m \in T^*$ and two configurations $c, c' \in \mathscr{C}(M)$, we write $c \xrightarrow{\rho}{}^*{}_T c'$ to denote that one of the following two cases holds:

1. $\rho = \epsilon$ and $c = c'$
2. There are configurations $c_0, \cdots, c_m \in \mathscr{C}(M)$ such that $c_0 = c$, $c' = c_m$, and $c_i \xrightarrow{\tau_{i+1}}{}_T c_{i+1}$ for all $i \in [0..m-1]$.

Given a configuration $c = (q, w_1, w_2, \cdots, w_n)$, we use $\mathit{Stack}_i(c)$ to denote the stack-i content i.e. $\mathit{Stack}_i(c) = w_i$ and $\mathit{State}(c)$ to denote the state q. A *computation* π of M starting from a configuration c is a (possibly infinite) sequence of the form $c_0 \xrightarrow{\tau_1} c_1 \xrightarrow{\tau_2} \cdots$ such that $c_0 = c$ and $c_{i-1} \xrightarrow{\tau_i} c_i$ for all $1 \leq i \leq |\tau_1 \tau_2 \cdots|$. We use $\mathit{Conf}(\pi)$, $\mathit{State}(\pi)$, and $\mathit{Trace}(\pi)$ to denote the sequences $c_0 c_1 \cdots$, $\mathit{State}(c_0)\mathit{State}(c_1)\cdots$, and $\tau_1 \tau_2 \cdots$ respectively. Given a finite computation $\pi_1 = c_0 \xrightarrow{t_1} c_1 \xrightarrow{t_2} c_2 \cdots \xrightarrow{t_m} c_m$ and a (possibly infinite) computation $\pi_2 = c_{m+1} \xrightarrow{t_{m+2}} c_{m+2} \xrightarrow{t_{m+3}} \cdots$, π_1 and π_2 are said to be *compatible* if $c_m = c_{m+1}$. Then, we write $\pi_1 \bullet \pi_2$ to denote the computation $\pi \stackrel{\text{def}}{=} c_0 \xrightarrow{t_1} c_1 \xrightarrow{t_2} c_2 \cdots \xrightarrow{t_m} c_m \xrightarrow{t_{m+2}} c_{m+2} \xrightarrow{t_{m+3}} \cdots$.

There are various interesting questions that one can ask about this model. *Reachability problem* asks whether a given configuration c is reachable from the initial configuration. *Repeated reachability problem* on MPDS M asks whether given a set of states $F \subseteq Q$, if there is an infinite computation π of M such that some state of F is visited infinitely often. We also consider model checking LTL formulas on MPDS. We first fix set of atomic propositions AP, an LTL formula φ, an MPDS $M = (n, Q, \Gamma, \Delta, q_0, \gamma_0)$ and a labelling function $\tau : Q \mapsto 2^{AP}$. The labelling function is extended to any configuration $c \in \mathscr{C}(M)$ as, $\tau(c) = \tau(\mathit{State}(c))$. It is easy to see that $(\mathscr{C}(M), \rightarrow, \tau)$ is a labelled transition system. One can then ask if every infinite path through this system satisfies a given LTL formula.

Since MPDS by itself is Turing powerful, our only hope is to study the above questions on restricted behaviours of the MPDS. we will discuss below some of the well know restrictions and related results.

5.2.1 Bounded Context

Qadeer and Rehof introduced *bounded-context* restriction in [124]. They also showed that question of whether a given configuration is reachable by a *bounded-context* computation for some a-priori fixed bound is decidable. Informally a context is a sequence of transitions involving only one stack. In a *bounded-context* computation, there is an a-priori bound on the number of contexts that can appear in it.

Definition 5. Contexts: *A* context *of a stack $i \in [1..n]$ is a computation of the form $\pi = c_0 \xrightarrow{t_1} c_1 \xrightarrow{t_2} \cdots$ such that Trace$(\pi) \in \Delta_i^* \cup \Delta_i^\omega$. We define* $\texttt{Initial}(\pi)$ *to be the configuration at the beginning of π (i.e.,* $\texttt{Initial}(\pi) = c_0$*). Furthermore, for any finite context $\pi = c_0 \xrightarrow{t_1} c_1 \xrightarrow{t_2} \cdots \xrightarrow{t_m} c_m$, we use* $\texttt{Target}(\pi)$ *to denote the configuration at the end of the context π (i.e.,* $\texttt{Target}(\pi) = c_m$*). Similarly, we use Context(π) to denote the active stack of the context (i.e. Context$(\pi) = i$ is the stack on which it operates).*

Context Decomposition: *Every computation can be seen as a concatenation of a sequence of contexts $\pi_1 \bullet \pi_2 \bullet \ldots$. In particular, every computation π can be written as a sequence $\pi_1 \bullet \pi_2 \bullet \ldots$ such that for all i, π_i and π_{i+1} are not contexts of the same stack. We refer to this as the* context decomposition *of π.*

Context-bounded Computations: *Given $k \in \mathbb{N}$, a computation $\pi = c_0 \xrightarrow{t_1} c_1 \xrightarrow{t_2} \cdots$ is said to be k* context-bounded *if it has a context decomposition $\pi = \pi_1 \bullet \pi_2 \bullet \ldots \bullet \pi_l$ consisting of at most k contexts (i.e. $l \leq k$). Thus in a context-bounded computation the number of switches between the stacks is bounded by $(k-1)$.*

Results: S. Qadeer and J. Rehof showed that, given any configuration, deciding whether or not it is reachable through a bounded-context computation is NP-COMPLETE [124]. In it was shown [118] that, in most practical cases context bounding is an effective way to capture bugs. In [105] it was shown that decidability of reachability on multi-threaded systems, operating under the context bounded restriction can be reduced to decidability on a sequential program. This enabled multi-threaded programs to be analysed under sequential setting, using plethora of already available tools. In [21], a system where multi-threaded recursive programs with ability to dynamically fork new threads and a variant of bounded-context restriction wass considered. Such a restriction allows only those behaviours in which, for every process the number of contexts it is in involved is bounded. They showed that reachability problem under this restricted setting is EXPSPACE COMPLETE. In [17], problem of finding a fair ultimately periodic executions under a context bounded restriction for multi-pushdown systems was considered and solved.

5.2.2 Bounded Phase

The restriction bounded-phase was introduced in [97]. Informally a phase is a sequence of operations in which the **Pop** operations are performed on only one stack. In a *bounded-phase* computation, there is an a-priori bound on the number of phases that it can involve.

Definition 6. Phase *A* Phase *of a stack $i \in [1..n]$ is a computation that involves pops (and zero test) only from stack-i i.e. it is a computation of the form $\pi = c_0 \xrightarrow{t_1} c_1 \xrightarrow{t_2} \cdots$ in which Trace$(\pi) \in \Delta^{\downarrow_i}$. Where $\Delta^{\downarrow_i} = \Delta \cap (Q \times (op \setminus \bigcup_{j \neq i} \cup_{a \in \Gamma} \{\mathbf{Pop}_j(a)\} \cup \{\mathbf{Zero}_j\}) \times Q)$.*

Bounded Phase computation *Given $k \in \mathbb{N}$, a computation $\pi = c_0 \xrightarrow{t_1} c_1 \xrightarrow{t_2} \cdots$ is said to be k* Phase-bounded *if it can be seen as concatenation of atmost k-*Phases *i.e. $\pi = \pi_1 \bullet \pi_2 \bullet \ldots \bullet \pi_l$ such that $\pi_1, \cdots, \pi_l$ are* Phases *and $l \leq k$.*

Results: The question of whether a given configuration is reachable through a *bounded-phase* computation for some fixed a-priori bound was shown to be 2ETIME COMPLETE [97]. In the same paper, the authors also proved the Parikh theorem and closure under boolean operations for the class of languages accepted by a bounded-phase MPDS. Given a set of configurations C, we define $Pre^*(C)$ with respect to *k-bounded-phase* restriction as a set of configurations c', from which a configuration $c \in C$ can be reached through a *k-bounded-phase* computation. Global model checking problem asks whether given a regular set of configuration C, $Pre^*(C)$ is also effectively regular. In [134], the global model checking problem and repeated reachability problem for MPDS with bounded-phase restriction were solved. This lead to decidability of model checking LTL logic against *bounded-phase* computations. In [133] parity games over MPDS with bounded-phase restrictions were considered and a NON-ELEMENTARY decision procedure for solving it was established. We revisit parity games over MPDS in a later chapter of the thesis. In [112, 56] it was shown that a bounded-phase executions of an MPDS machine has bounded tree-width/split-width, as an application of this, the decidability of many linear time properties over bounded-phase executions could be obtained.

5.2.3 Bounded Scope

The *bounded-scope* restriction was introduced in [100], and we describe this restriction formally below. We introduce some notation for this purpose. For any $i \in [1..n]$ and for any two contexts π_1 and π_2 of stack i, we write $< \pi_1, \pi_2 >_i$ to denote that $Stack_i(\texttt{Initial}(\pi_2)) = Stack_i(\texttt{Target}(\pi_1))$. This notation is extended in the straightforward manner to any bigger sequence. Given a run of MPDS π and its context decomposition $\pi = \pi_1 \bullet \pi_2 \bullet \cdots$, we let $Comp_i(\pi) =< \pi_{i_1}, \pi_{i_2}, \cdots >_i$ (with $i_1 < i_2 < i_3 < \cdots$) to be the maximal subsequence of i-contexts of the decomposition.

Definition 7. Cluster *A* cluster ρ *of a stack* $i \in [1..n]$ *of size* $j \in \mathbb{N}$ *(also referred to as* j*-cluster) is a sequence of finite contexts* $< \pi_1, \pi_2, \ldots, \pi_j >_i$ *of the stack* i *such that* $Stack_i(\texttt{Initial}(\pi_1)) = Stack_i(\texttt{Target}(\pi_j)) = \perp$ *(i.e., the stack* i *at the beginning of the context* π_1 *and at the end of the context* π_j *is empty).*

Scope-Bounded Computations *Intuitively, in a scope-bounded computation, any value that is pushed in a stack* i *is removed within* k *contexts involving this stack* i*. Equivalently, we require that for any* scope-bounded *computation, if the computation is finite, then it is just a concatenation of clusters of size at most* k *i.e., a computation* π *is said to be a* k *scope-bounded computation if for each* $i \in [1..n]$*,* $Comp_i(\pi)$*, can be seen as a sequence of clusters of size at most* k *(i.e.* $Comp_i(\pi) =< \rho_1, \rho_2, \cdots >_i$*, where each* $\rho_1, \cdots, \rho_{m_i}$ *is a* k *cluster).*

Results: In [102], the scope-bounded restriction was introduced and the reachability under this restriction was shown to be PSPACE complete. Later in [56, 103], it was shown that the tree-width/split-width of computations of such a system is bounded. In [102], language theoretic properties of bounded-scope system were studied. It was shown that the scope-bounded MPDS are determinizable. Further it was shown that the class of languages of a

scope-bounded MPDS are closed under intersection and complementation. Further a sequentialisation construction of MPDS with scope-bounded restriction was shown, leading to Parikh theorem for such languages.

5.2.4 Ordered multi-pushdown run

Ordered multi-pushdown system was originally introduced in [45] and the verification and model checking problems on these structures were studied in [16]. Informally, in an ordered MPDS execution, the pop operations are allowed only on the least non-empty stack. We would like to note here that, we always refer to any stack by the position it occupies in the configuration. From this, there is also an implicit ordering on the stacks. Before we formally present the definition, we introduce a funtion $\mathbf{Act} : \mathscr{C}(M) \mapsto [1..n]$ that takes as an input, a configuration of MPDS and outputs the least non-empty stack (or the active stack) of that configuration. We define $\mathbf{Act}(c) = j$ if $c \in Q \times \bot^{j-1} \times (\Gamma^+\bot) \times (\Gamma^*\bot)^{n-j}$ and $\mathbf{Act}(c) = n$ if $c \in Q \times \{\bot\}^n$.

Definition 8. *Any execution $\pi = c_1 \xrightarrow{\tau_1}_M c_2 \xrightarrow{\tau_2} \cdots$ of given MPDS $M = (n, Q, \Gamma, \Delta, q_0, \gamma_0)$, is said to be an order-restricted execution iff for all $i \in [1..|\pi|]$, if $\tau_i \in (Q \times \cup_{a\in\Gamma}\mathbf{Pop}_j(a) \times Q) \cap \Delta_j$, for some $j \in [1..n]$ then $\mathbf{Act}(c_i) = j$, i.e. the pop operation is allowed only on the least non-empty stack.*

Results: In [16] it was shown that the reachability problem under this restriction is decidable and 2ETIME COMPLETE. It was shown in [14] that the repeated reachability problem for such models can be reduced to a sequence of reachability queries and hence model checking the LTL formulas on ordered restriction computations of MPDS is decidable. In [112, 56], the tree-width and split-width of a ordered restricted executions of an MPDS was shown to be bounded.

Chapter 6

Linear time model checking under bounded scope

6.1 Introduction

There have been many works to address the problem of detecting safety bugs in shared memory multi-threaded programs. However besides safety, it is crucial to also ensure whether concurrent programs satisfy certain liveness properties. Then one interesting question is what would be a suitable concept for restricting behaviours of the multi-threaded programs when reasoning about liveness properties and more generally about any omega-regular property expressible in linear time temporal logic such as LTL or by a Büchi automata.

While context-bounding is quite useful for detecting safety bugs for which it is sufficient to consider finite computations, this concept is not very appropriate for reasoning about liveness properties for which it is necessary to consider infinite behaviours. The reason for this is because context bounding does not give a chance for every thread to be executed infinitely often. Any context-bounded infinite execution eventually degenerates to that of a pushdown system. In this respect, the scope-bounded restriction [103], is more suitable for reasoning about liveness since it allows behaviours with unbounded context-switches between threads.

In this chapter, we show how to obtain an decision procedure for model checking a LTL formula against scope-bounded executions of an MPDS. For this, we first define what a *scope-bounded* infinite run of an MPDS means. A *bounded-scope* repeated reachability problem asks if there is an infinite *bounded-scope* computation that visits a given good state, infinitely often. We then go onto show that the *bounded-scope* repeated reachability problem is decidable. We show that this problem can be reduced to checking emptiness on a Büchi pushdown automata. We later show how to use this result to model check an LTL formula.

We also present an alternate proof of hardness for the bounded-scope reachability problem. The original proof of hardness in [103] was by an involved reduction from the emptiness problem for a space bounded Turing machine to the bounded-scope reachability problem on an MPDS system. In this chapter, we first show a simpler proof to obtain the lower bound. For this, we show an easy reduction from the emptiness of the intersection of n finite state automata, to the scope-bounded reachability problem on an MPDS.

In [14] the model checking problem of ordered multi-pushdown system is shown to be decidable in 2ETIME-COMPLETE. It is not very clear how this decidability result is related to the one we prove in this chapter. Simulating scope-bounded computation using the ordered multi-pushdown system computation does not seem possible. More over the complexity of the model checking problem for ordered multi-pushdown system is clearly higher than model checking under bounded-scope restriction.

A procedure for detecting termination bugs using repeated context-bounded reachability has been proposed in [17]. The idea there is to focus on checking the existence of fair context-bounded *ultimately periodic* non-terminating computations i.e., infinite computations of the form uv^ω where u and v are finite computation segments with a bounded number of context-switches. The model checking procedure described in this chapter is more general than the procedure in [17] since scope-bounded and ω regular behaviours are more general than ultimately periodic computations and context-bounding.

Remark: *In [101], the problem of model checking infinite scope-bounded executions of a multi-pushdown system, against a powerful logic called multi-CaRet (which is an extension of CaRet logic) was considered and was shown to be* EXPTIME-COMPLETE. *Our work is independent of their work and both results were obtained and published concurrently.*

6.2 Hardness for scope-bounded reachability

In this section, we show an alternate proof of hardness for scope-bounded reachability.

Theorem 14. *Given multi-pushdown system M, a constant k and a state q, checking if q is reachable by a k-scope-bounded computation from the initial configuration is* PSPACE HARD.

Proof. (sketch) Let us fix n finite state automata $A_1 \cdots A_n$ where $A_i = (Q_i, \Sigma, \delta_i, q_i^0, f_i^0)$. We know that the problem of checking whether $\bigcap_{i=1}^n L(A_i) \neq \emptyset$ is PSPACE HARD. We show how to reduce this problem to checking whether a state is reachable via a finite *n-bounded-scope* computation of an MPDS M. The idea is to construct an MPDS with n stacks. Initially each of its n stacks are populated with the initial states of the automata (i^{th} stack is initialised with q_i^0). At start of each round, an input letter is guessed and systematically, the state stored in each of the stack is replaced with a new state. The new state is the result of applying the guessed input letter to the current state. For e.g. if stack-i has state q_i in its stack and if the guessed input letter is an a, q_i is replaced with q_i' if $(q_i, a, q_i') \in \delta_i$. Clearly at the end of k rounds, if the word guessed so far is w, then for each $i \in [1..n]$, stack-i holds the state q_i where $q_i = \delta_i^*(q_i^0, w)$. If at the end of some round if all the stacks contain a final state, then we know that $\bigcap_{i=1}^n L(A_i) \neq \emptyset$. This can easily be arranged by non-deterministically checking at the end of a round, if all the stacks contain a final state and moving to a new state say f in the MPDS. Note that in each round, the state contained in each stack is popped and then is replaced by the next state (and hence the stack is emptied). Hence the problem of checking if the intersection of the language of n finite state automata is empty can be reduced to 1 *scope-bounded* reachability (of state f) in an MPDS with n stacks.

□

6.3 Infinite scope-bounded computations

In this section, we introduce the definition of scope-bounded computation for infinite case.

Scope-Bounded Computations: Let π be any infinite computation, it is an infinite k *scope-bounded* computation if it can be context decomposed as $\pi_1 \bullet \pi_2 \bullet \dots$ such that one of the following holds.

1. (Case where there are infinitely many context switches) For each $i \in [1..n]$, $Comp_i(\pi)$, can be seen as a sequence of clusters of size at most k (i.e. $Comp_i(\pi) =< \rho_1, \rho_2, \cdots >_i$, where each $\rho_1, \cdots, \rho_{m_i}$ is a k cluster). Moreover, there are at least two distinct indices $i, j \in [1..n]$ such that σ_i and σ_j are infinite (and all the stacks for which σ_l is finite are empty beyond a point).
2. (Case where beyond some point all stacks except i are empty and there is a final infinite context involving the stack i) For all $j \neq i$, $Comp_j(\pi) =< \rho_1\rho_2\cdots\rho_{m_j} >_j$ (where each ρ_1, $\cdots, \rho_{m_j}$ is a k cluster) and $Comp_i(\pi) =< \rho_1, \rho_2, \cdots \rho_{m_i}, \sigma'_i >_i$,(where each $\rho_1, \cdots, \rho_{m_i}$ is a k cluster), $\sigma'_i =< \pi'_1, \pi'_2 \cdots, \pi'_\ell >_i$ is a sequence of contexts with $\ell \leq k$ and π'_ℓ is an infinite context.

6.4 Model checking LTL on bounded scope executions

In this section, we show how to model check LTL formulas over scope-bounded computations of MPDS. For this, we first show how to solve the repeated reachability problem and then we show how to use this to solve the model checking problem.

6.4.1 Bounded scope repeated reachability

Bounded scope repeated reachability problem asks, given an MPDS $M = (n, Q, \Gamma, \Delta, q_{\mathsf{init}}, \gamma_0)$, a number $k \in \mathbb{N}$ and a set of final states F, whether there is an infinite k bounded-scope run, starting from the initial configuration that visits some final state $f \in F$ infinitely often. The following Theorem states that this problem can be reduced to checking emptiness of a Büchi pushdown system.

Theorem 15. *Let $k \in \mathbb{N}$ be a natural number, $\mathrm{M} = (n, Q, \Gamma, \Delta, q_{\mathsf{init}}, \gamma_0)$ an MPDS, and $F \subseteq Q$ a set of states. Then it is possible to construct a Büchi pushdown automaton P such that M has a k scope-bounded computation that visits some state in F infinitely often if and only if the language $L^\omega(\mathrm{P})$ is not empty. Moreover, the size of P is $O(|F|(k|\mathrm{M}|)^{dkn})$ for some constant d.*

We first informally sketch the proof before formalising the same. Firstly it is easy to see that we may restrict ourself to checking whether a single final state $f \in F$ is visited infinitely often along an infinite run. This is because, by definition we need only a single state to repeat infinitely many times to satisfy the Büchi condition. If we can solve the problem for the single state case, then we can repeat our check for each state in F. So w.l.o.g., we will assume that $F = \{f\}$ for some $f \in Q$.

Before we go on to describe the proof, we introduce some notations below. Given a finite context π, we first define an abstraction as $\mathbf{Abs}(\pi) = (\mathit{State}(\texttt{Initial}(\pi)), \mathit{Context}(\pi), \text{flg}, \mathit{State}(\texttt{Target}(\pi)))$, i.e. it is a tuple that records the initial state (as first component of tuple), the final state (as last component of the tuple), the active stack of the context (as second component of the tuple) and information on whether f was seen during the execution of the context (i.e. flg = 1 if π visited f, 0 otherwise). Given an infinite context of the form $\pi = c_1 \rightarrow c_2 \rightarrow c_3 \rightarrow \cdots$, its abstraction is defined as $\mathbf{Abs}(\pi) = \mathbf{Abs}(c_1 \rightarrow c_2).\mathbf{Abs}(c_2 \rightarrow c_3)\cdots$ i.e. the abstraction of its one step computations. Given a sequence of contexts $\pi = \pi_1 \bullet \pi_2 \bullet \cdots$, the abstraction of such a sequence is defined as a word of the form $\mathbf{Abs}(\pi) = \mathbf{Abs}(\pi_1)\mathbf{Abs}(\pi_2)\cdots$, we call the resulting word the *abstraction sequence* of π. Given a k cluster of the form $\rho =< \pi_1, \pi_2, \cdots \pi_l >_i$ ($l \leq k$), we let $\mathbf{Abs}(\rho) = \mathbf{Abs}(\pi_1)\mathbf{Abs}(\pi_2)\cdots\mathbf{Abs}(\pi_l)$. Given a (possibly infinite) sequence of abstractions w of the form $w = (q_1, i_1, f_1, q_1')(q_2, i_2, f_2, q_2')(q_3, i_3, f_3, q_3')\cdots$, we say it is well-formed if $q_1 = q_{\text{init}}$ and further for all $i \geq 2$, we have $q_i = q'_{i-1}$. Given two configurations $c, c' \in \mathscr{C}(M)$, we say $c \equiv_i c'$ iff $\mathit{State}(c) = \mathit{State}(c')$ and $\mathit{Stack}_i(c) = \mathit{Stack}_i(c')$. Given two clusters $\rho =< \pi_1, \pi_2 \cdots, \pi_l >_i$ and $\rho' =< \pi'_1, \pi'_2, \cdots, \pi'_l >_i$ containing equal number of contexts, we say they are *i-equivalent* iff the following conditions hold.

- $\mathbf{Abs}(\rho) = \mathbf{Abs}(\rho')$
- For all $j \neq i$, we have for all $i' \in [1..l]$, $\mathit{Stack}_j(\texttt{Initial}(\pi_{i'})) = \mathit{Stack}_j(\texttt{Initial}(\pi'_{i'}))$

Now consider any k scope-bounded infinite computation π. Clearly it can be context decomposed into infinite number of finite contexts ($\pi = \pi_1 \bullet \pi_2 \bullet \cdots$, where $\pi_1, \pi_2, \cdots$ are finite contexts) or into finite number of contexts ending in an infinite context ($\pi = \pi_1 \bullet \pi_2 \cdots \bullet \pi_\ell$, where $\pi_1, \pi_2, \cdots \pi_{\ell-1}$ are finite context and π_ℓ is an infinite context).

Let us first consider a k scope-bounded computation π of an MPDS M, involving infinitely many finite contexts. Note that this corresponds to the computation having infinitely many context switches. Let $\sigma_i = \mathit{Comp}_i(\pi)$, clearly by definition, each σ_i is a sequence of (possibly infinite) k clusters i.e. for all $i \in [1..n]$, we have $\sigma_i = < \rho_1^i, \rho_2^i, \cdots >_i$, where each ρ_j^i is a k-cluster. Further we have that at least two indices i, j such that σ_i, σ_j are infinite. Consider $\mathbf{Abs}(\pi)$, clearly such an abstraction sequence is well formed. Given any well formed infinite word $w \in (Q \times [0..n] \times [0,1] \times Q)^\omega$, when can we say that it is an abstraction of some k scope-bounded run of an MPDS? Firstly note that if we were to replace any k-cluster ρ corresponding to stack-i in π by an i-equivalent k-cluster ρ', the run is still a valid k scope-bounded run. Hence it is enough to check whether there are abstraction sequences of k-clusters, one for each stack and whether the given word is in the shuffle of such sequences. The following Lemma formalises this.

Lemma 33. *M has a k scope-bounded computation $\pi = \pi_1 \bullet \pi_2 \bullet \cdots$, visiting infinitely often the state f that can be decomposed into infinitely many finite contexts if and only if there is a well formed word w such that*

- *There are infinitely many indices $j \in \mathbb{N}$ such that $w[j] \in Q \times [1..n] \times \{1\} \times Q$*
- *For every stack $i \in [1..n]$, there is a (possibly infinite) sequence σ_i of k clusters of the stack i, such that $w \in \texttt{Shuffle}(\{\mathbf{Abs}(\sigma_1)\}, \ldots, \{\mathbf{Abs}(\sigma_n)\})$.*

Proof. ($\Rightarrow$)

For each $i \in [1..n]$, let $\sigma_i = \mathit{Comp}_i(\pi)$. By definition of scope-bounded run, we have for each $i \in [1..n]$, σ_i is a (possibly infinite) sequence of k-clusters (i.e. $\sigma_i =< \rho^i_1, \rho^i_2, \cdots >_i$, where $\rho^i_1, \rho^i_2, \cdots$ are k-clusters). It is easy to see that $\mathbf{Abs}(\pi) \in \mathtt{Shuffle}(\{\mathbf{Abs}(\sigma_1)\}, \ldots, \{\mathbf{Abs}(\sigma_n)\})$ and that $\mathbf{Abs}(\pi)$ is indeed well formed. Hence the required word is $w = \mathbf{Abs}(\pi)$.

($\Leftarrow$)

We assume a well formed word $w \in \mathtt{Shuffle}(\{\mathbf{Abs}(\sigma_1)\}, \ldots, \{\mathbf{Abs}(\sigma_n)\})$ where each σ_i is a concatenation of possibly infinite sequence of k clusters and show how to construct a MPDS run π from it. Firstly note that each of the σ_i is a sequence of contexts of the form $< \pi^i_1, \pi^i_2, \cdots >_i$. Let $\sigma = \pi_1, \pi_2, \pi_3, \cdots$ be a sequence of contexts such that $\sigma \in \mathtt{Shuffle}(\sigma_1, \ldots, \sigma_n)$ and $\mathbf{Abs}(\sigma) = w$. It is easy to note that w determines such a σ uniquely. Though we have this sequence of contexts, it need not be compatible. Hence we are not immediately promised an infinite run from this sequence of contexts. However all is not lost. The below Lemma 34 states that if there is a context of stack i starting from a particular configuration, then there is a context starting from any configuration that is i-equivalent to it.

Lemma 34. *Given any context π (with $\mathit{Context}(\pi) = i$), let $u = \mathit{Trans}(\pi)$ be the sequence of transitions of π then from any configuration c such that $c \equiv_i \mathit{Initial}(\pi)$. There is a valid run of the form $c \xrightarrow{u}{}^* c'$, with $c' \equiv_i \mathit{Target}(\pi)$. Further we have that for all $j \neq i$, $c' \equiv_j c$.*

Proof. We prove this by induction on the length of the computation in the context. The base case is a zero length computation, which is trivial. For the induction case, we will assume that the length of the context is greater than 1. In this case, the context π can be split as $\pi = d \rightarrow^* d'' \xrightarrow{\tau} d'$. By induction, we have for any $c \equiv_i d$, a computation of the form $c \rightarrow^* c''$ such that for all $j \neq i$, $c'' \equiv_j c$ and $d'' \equiv_i c''$. The case where τ is an internal move is easy to see. We now consider the case where τ is a zero move and rest of the cases are similar. Suppose $\tau = (q, \mathbf{Zero}_i, q')$, since $c'' \equiv_i d''$, we have $\mathit{State}(c'') = \mathit{State}(d'')$ and $\mathit{Stack}_i(c'') = \mathit{Stack}_i(d'') = \bot$. From this we have $c'' = (q, \gamma_1, \cdots, \gamma_{i-1}, \bot, \cdots, \gamma_n) \xrightarrow{\tau} c' = (q', \gamma_1, \cdots, \gamma_{i-1}, \bot, \cdots, \gamma_n)$. Further, it is easy to see that $c' \equiv_i d'$ and for all $j \neq i$, $c' \equiv_j c$. □

Now using the above Lemma, we will show how to construct a valid scope-bounded computation from σ (recall that $\sigma = \pi_1, \pi_2, \pi_3, \cdots$ is a sequence of contexts such that $\sigma \in \mathtt{Shuffle}(\sigma_1, \ldots, \sigma_n)$ and $\mathbf{Abs}(\sigma) = w$). Let $\alpha = \alpha_1, \alpha_2 \cdots$ be a sequence, such that $\alpha_i = \mathit{Trans}(\pi_i)$. Now the existence of a k scope-bounded run follows easily from the fact that, using Lemma 34, we can inductively construct for any prefix $\alpha_1, \cdots \alpha_n$ of α, a run from the initial configuration c^{Init}_M of the form $c^{Init}_M \xrightarrow{\alpha_1 \cdots \alpha_n}{}^* c_n$ such that $c_n \equiv_i \mathit{Target}(\pi_n)$, where $i = \mathit{Context}(\pi_n)$ and for all $j \neq i$ we have $c_{n-1} \equiv_j c_n$.

□

We will now show that the abstractions of set of all k scope-bounded computations involving infinitely many finite contexts is regular. For this, we will construct a Büchi automaton that will recognise well formed words $w \in (Q \times [1..n] \times [0,1] \times Q)^\omega$ such that there are infinitely many indices $j \in \mathbb{N}$ such that $w[j] \in P \times [0..n] \times \{1\} \times P$ and $w \in \mathtt{Shuffle}(\{\mathbf{Abs}(\sigma_1)\}, \ldots, \{\mathbf{Abs}(\sigma_n)\})$, where σ_i is a (possibly infinite) sequence of clusters of stack i. Then by using

Lemma 33, we get the existence of k bounded-scope run involving infinitely many context-switches.

We will firstly show that for every $k \in \mathbb{N}$ and $i \in [1..n]$, the set $L_i^k(M)$ of all the finite words of the form $\mathbf{Abs}(\rho)$ where ρ is a k cluster of stack i can be seen as the language of a finite state automaton. Note that such a language is finite and hence regular. The problem is to show that it is effectively regular and to compute the complexity of the automata recognising such a language. For this, we build a pushdown system recognising the abstractions of clusters and then restricting ourselves to only those words whose length is less than or equal to k.

Let $\mathtt{Ctx}_i^{q,q'}$ be set of all sequences of contexts such that, for any $j \in \mathbb{N}$ if $(\pi_1, \pi_2, \cdots, \pi_j) \in \mathtt{Ctx}_i^{q,q'}$ then the following holds.

- $Context(\pi_1) = \cdots = Context(\pi_j) = i$.
- For all $l \in [1..j-1]$, we have $Stack_i(\mathtt{Target}(\pi_l)) = Stack_i(\mathtt{Initial}(\pi_{l+1}))$.
- $State(\mathtt{Initial}(\pi_1)) = q$, $State(\mathtt{Target}(\pi_n)) = q'$ and $Stack_i(\mathtt{Initial}(\pi_1)) = Stack_i(\mathtt{Target}(\pi_n)) = \bot$

Let $s_i^{q,q'} = \{\mathbf{Abs}(\pi_1, \pi_2, \cdots, \pi_j) \mid (\pi_1, \pi_2, \cdots, \pi_j) \in \mathtt{Ctx}_i^{q,q'}, j \in \mathbb{N}\}$. Such a set captures all abstractions of clusters of stack-i that starts at q and end in q'.

Lemma 35. *The set $S_i^{q,q'}$ is context free.*

Proof. Any element of $S_i^{q,q'}$ is of the form, $\mathbf{Abs}(\rho)$, where $\rho = (\pi_1, \cdots, \pi_n) \in \mathtt{Ctx}_i^{q,q'}$. From this we know that each $\pi_1, \cdots, \pi_n$ is a context of stack i and hence during its execution, no stack other than i is used. This means that, a single stack is sufficient to simulate the moves. We show how to construct a pushdown system that will simulate each of these contexts, using its local states and its stack. The states of the pushdown system that we construct will be of the form (q, q', i). In this, the first component is used to store the starting state of the context, second component is used to store the state reached while simulating the context and the last component records whether the final state f was seen during the simulation. The pushdown system will simulate a move of the context by performing any stack operation of the context on its stack. Further it will update the second component of the state space to reflect the effect of the move. If a final state is ever seen, it is recorded in the third component. The pushdown system can nondeterministically guess from any configuration of the form $((q, q', \mathrm{flg}), \gamma)$, the completion of a context. In this case, it outputs the abstraction (q, i, flg, q') and starts simulating a new context. The details are formalised below.

The Pushdown system is defined as $P_i(q, q') = ((Q \times Q \times [0,1]) \cup \{e\}, (Q \times \{i\} \times [0,1] \times Q), \Gamma, \delta^{(q,q')}, (q, q, 0))$. The states of the pushdown system records along with the the start state of the context, current state, and whether the state f was seen during the execution of the context. The input alphabet is set of all possible abstractions of context i. The transition relation $\delta^{(q,q')}$ is defined as below.

1. For any transition $(q_1, \mathbf{Push}_i(\alpha), q_1') \in \Delta$, we add for all $p \in Q$, the transitions $((p, q_1, x), \mathbf{Push}(\alpha), \epsilon, (p, q_1', y))$ to $\delta^{(q,q')}$. Further if $q_1 = f$ or $q_1' = f$ then we let $y = 1$ and we let $y = x$ otherwise. We add similar transitions for operations such as $\mathbf{Pop}_i, \mathbf{Zero}_i$ and $\mathbf{Int}_i$. These set of transitions simulate the context specific moves.

2. For all $q_1, q_2, q_3 \in Q$, we add $((q_1, q_2, x), \mathbf{Int}, (q_1, i, x, q_2), (q_3, q_3, 0))$ to $\delta^{(q,q')}$. These set of transitions, makes a non-deterministic jump (indicating end of the current context and beginning of a new one).
3. For all $q_1 \in Q$, we add $((q_1, q', x)), \mathbf{Zero}, (q_1, i, x, q'), e)$. These set of transitions ends the cluster when required state with empty stack is reached.

Correctness of such a construction is easy to see from the following Lemma.

Lemma 36. *Any string* $w = (q_1, i, x_1, q'_1).(q_2, i, x_2, q'_2).....(q_m, i, x_m, q'_m) \in L(P(q, q'), e)$ *iff there is a context sequence of the form* $\sigma =< \pi_1, \pi_2, \cdots, \pi_m >_i$, *such that* $\mathbf{Abs}(\sigma) \in S_i^{q,q'}$ *and* $w = \mathbf{Abs}(\sigma)$.

Proof. For this, we first prove the following Claim that relates the run of P_i to run of M.

Claim 5. *For any* $\gamma_1, \gamma_2 \in (\Gamma \setminus \{\bot\})^*$ *and* $x \in [0,1]$, $((p,p,0), \gamma_1\bot) \xrightarrow{\epsilon}{}^*{}_{P_i(q,q')} ((p,q,x), \gamma_2\bot)$ *(using transitions only from 1) iff there is a context of stack-*i $\pi = c \rightarrow^*{}_M c'$ *such that* $\text{State}(c) = p$, $\text{State}(c') = q$, $\text{Stack}_i(c) = \gamma_1\bot$, $\text{Stack}_i(c') = \gamma_2\bot$ *and for all* $j \neq i$, $\text{Stack}_j(c) = \text{Stack}_j(c') = \bot$. *Further if* $x = 1$ *then there is a* c'' *with* $\text{State}(c'') = f$ *such that* π *can be written as* $\pi = c \rightarrow^* c'' \rightarrow^* c'$

Proof. ($\Rightarrow$) We prove this by induction on the length of the computation. The base case being zero length computation is trivial. For the inductive case, we assume that $((p,p,0), \gamma_1\bot) \rightarrow^*{}_{P_i(q,q')} ((p,q,x), \gamma_2\bot)$ is of size greater than zero. Then the computation can be split as

$$((p,p,0), \gamma_1\bot) \rightarrow^*{}_{P_i(q,q')} ((p,q',y), \gamma'_2\bot) \xrightarrow{\tau}{}_{P_i(q,q')} ((p,q,x), \gamma_2\bot)$$

By induction, we have a run of the form

$$(p, \bot^{i-1}, \gamma_1\bot, \bot^{n-i}) \rightarrow^*{}_M (q', \bot^{i-1}, \gamma'_2\bot, \bot^{n-i})$$

Let $\tau = ((p,q',y), \mathbf{Push}(\alpha), (p,q,x))$ (rest of the cases are similar and easy). Such a transition was added in the first place due to the existence of a transition in M of the form $(q', \mathbf{Push}_i(\alpha), q)$. Moreover, if $y = 0$ and $x = 1$ then we have $q = f$ or $q' = f$. From these, we can extend the run as follows.

$$(p, \bot^{i-1}, \gamma_1\bot, \bot^{n-i}) \rightarrow^*{}_M (q', \bot^{i-1}, \gamma'_2\bot, \bot^{n-i}) \rightarrow (q, \bot^{i-1}, \gamma_2\bot, \bot^{n-i})$$

($\Leftarrow$)

We again prove this direction by inducting on the length of the computation. Base case of length zero computation is trivial. For inductive case, we assume $(p, \bot^{i-1}, \gamma_1\bot, \bot^{n-i}) \rightarrow^* (q, \bot^{i-1}, \gamma_2\bot, \bot^{n-i})$ is a computation of length greater than zero. Hence we can split this computation as

$$(p, \bot^{i-1}, \gamma_1\bot, \bot^{n-i}) \rightarrow^* (q', \bot^{i-1}, \gamma'_2\bot, \bot^{n-i}) \xrightarrow{\tau} (q, \bot^{i-1}, \gamma_2\bot, \bot^{n-i})$$

For some $q' \in Q$ and $\gamma'_2 \in (\Gamma \setminus \{\bot\})^*$. By induction, we have a run of the form

$$((p,p,0), \gamma_1\bot) \rightarrow^*{}_{P_i(q,q')} ((p,q',y), \gamma'_2\bot)$$

Let $\tau = (q', \mathbf{Push}_i(\alpha), q)$ (the other cases are similar). If $y = 0$ and $q = f$ or $q' = f$ then we have the transition $((p, q', 0), \mathbf{Push}(\alpha), (p, q, 1)) \in \delta^{(q,q')}$. If $y = 0$ and $q \neq f$ and $q' \neq f$ then we have the transition of the form $((p, q', 0), \mathbf{Push}(\alpha), (p, q, 0)) \in \delta^{(q,q')}$. If $y = 1$, we have the transition $((p, q', 1), \mathbf{Push}(\alpha), (p, q, 1)) \in \delta^{(q,q')}$. Now using one of these transitions, we can get the required run.

$$((p, p, 0), \gamma_1 \perp) \rightarrow^*_{P_i(q,q')} ((p, q, x), \gamma_2 \perp)$$

□

Now, proof of the Lemma 36 is an immediate consequence of the following Claim 5 and the transitions from 2,3. This completes the proof of Lemma 36.

□

This completes the proof of 35.

□

Given a pushdown automata P and an integer k, one can easily construct an finite state automaton B accepting the set of k length words accepted by P, i.e. $L(B) = \{w \mid |w| \leq k \wedge w \in L(P)\}$. This can easily be obtained by converting the given pushdown automaton to a context free grammar and then simulating all the derivations of length bounded by k. The size of such a finite state automaton is at most exponential in k.

Corollary 1. *$L_i^k(q, q') = \{w \mid w \in S_i^{q,q'} \wedge |w| \leq k\}$ i.e. the set of all k-clusters of stack i is effectively regular. The complexity of such a construction is at most exponential in k.*

We now show how to construct a Büchi automata recognising the abstractions of a k scope-bounded infinite computation that has infinitely many contexts. Towards constructing the Büchi automata, we first fix the finite state automaton obtained from corollary 1. Let $B_i(q, q')$ be the finite state automata recognising the language $L_i^k(q, q')$. Let B_i be an automata such that $L(B_i) = \bigcup_{q,q' \in Q} L(B_i(q, q'))$ i.e. consolidated automata recognising all interface languages between all pairs of states for stack-i, let $B_i = (Q^{B_i}, (Q \times \{i\} \times [0, 1] \times Q), \delta^{B_i}, q_0^{B_i}, F^{B_i})$. Further we will, w.l.o.g. assume that the initial is not present in the set of final states.

Lemma 37. *Given an MPDS $\mathrm{M} = (n, Q, \Gamma, \Delta, s_0, \gamma_0)$ and a state $f \in Q$, the problem of checking whether there is a well-formed word w such that*

1. *$w \in \mathtt{Shuffle}(\{\mathbf{Abs}(\sigma_1)\}, \ldots, \{\mathbf{Abs}(\sigma_n)\})$, where for each $i \in [1..n]$, σ_i is a sequence of k clusters of stack i (with at least two of them being infinite)*
2. *There are infinitely many $j \in \mathbb{N}$ such that $w[j] \in (Q \times [1..n] \times \{1\} \times Q)$*

can be reduced to the emptiness problem for a Büchi automaton B whose size is $O((k|\mathrm{M}|)^{dkn})$ for some constant d.

Proof. For any $X \subseteq [1..n]$, let $L_X = \{w \mid w \in \mathtt{Shuffle}(\{\mathbf{Abs}(\sigma_1)\}, \ldots, \{\mathbf{Abs}(\sigma_n)\}) \wedge \exists^\infty j \in \mathbb{N}, w[j] \in (Q \times [1..n] \times \{1\} \times Q)\}$. It is the set of all well formed words in shuffle of clusters $\sigma_1, \cdots \sigma_n$, such that for each $i \in X$, σ_i is an infinite sequence of k clusters and for each $j \notin X, \sigma_j$ is an finite sequence of k clusters. Clearly, what we require is a Büchi automata B such that $L(B) = \bigcup_{X \subseteq [1..n], |X| > 1} L_X$. We show how to construct such a Büchi automata recognising the

language L_X, for some fixed X. Then the required B automaton can easily be obtained by taking union over all such X.

The idea to construct B_X, is to run the B_i (for all $i \in [1..n]$) automatas in parallel and stitch up the k-clusters obtained to form a well-formed word. We also need to ensure that, in any execution, those components outside of X terminate eventually and those in X execute infinitely often.

Ensuring that components outside of X terminate eventually is done by setting up a special state $\perp$. Now the Büchi acceptance condition can be setup in such a way that for all components not in X, only $\perp$ is seen infinitely often.

For components in X, we need to ensure that every one of them is executed infinitely often. This is done by ensuring that for every $j \in X$, the final state and initial state of B_j is seen infinitely often. For this, we assume an implicit ordering on the elements of X say $X = \{x_1, \cdots x_m\}$. The Büchi system we construct will remember as part of its state space, the last element of X that has visited a final state. Further the Büchi acceptance condition will ensure that each of these elements are repeated in the state infinitely often.

The automata needs to accept sequence of abstractions that has to be well formed. Given any abstraction of the form (p_1, i, x, p_2), by target state we mean the state p_2 and by source state, we mean the state p_1. The accepted sequence of abstractions is ensured to be well formed by storing in the state space the target state of the last seen abstraction and accepting the next abstraction only if its source state matches the stored state. The construction is formalised below.

The automata $B_X = (Q^{B_X}, (Q \times \{i\} \times [0,1] \times Q), \delta^{B_X}, q_0^{B_X}, F^{B_X})$ is defined as

1. The set of states of B_X are $Q^{B_X} = Q^{B_1} \cup \{\perp\} \times \cdots \times Q^{B_n} \cup \{\perp\} \times Q \times X \times [0,2]$. Informally, the states include the product of states of all B_i ($i \in [1..n]$) and a special $\perp$ which will be used by the indices $i \notin X$, to indicate completion of the finite computation. We further record the last seen target state in the abstraction sequence seen so far (as $(n+1)$'st component), this information will be used in ensuring that the sequence is well formed. We also have a component to ensure that each element in X is seen infinitely often. Lastly we have in the tuple, flags [0..2] to record if f was visited. As we will see later, it also serves the purpose of ensuring that each element in X is seen infinitely often.
2. The initial state of B_X is $q_0^{B_X} = (q_0^{B_1}, \cdots, q_0^{B_n}, s_0, x_1, 0)$.
3. The set of final states of B_X are $F^{B_X} = \{(q_1, \cdots, q_n, p, x_m, 2) \mid p \in Q,\ \forall i \in [1..n] \setminus X,\ q_i = \perp \wedge q_{x_m} \in F^{B_{x_m}}\}$. The final state ensures that all the components that were outside of X have terminated. Further as we will see later, such a final state ensures that between any two appearances of it, each component in X has made progress and that we have seen state f at least once.
4. The transition relation δ^{B_X} is defined as below.

a.1 If $(q_i,\ (p, i, b, p'),\ q_i') \in \delta^{B_i}$ then we add for all states $(q_1, \cdots, q_i, \cdots, q_n, p, x, z) \in Q^{B_X}$, the transition $((q_1, \cdots, q_i, \cdots, q_n, p, x, z),\ (p, i, b, p'),\ (q_1, \cdots, q_i', \cdots, q_n, p', x, y)) \in \delta^{B_X}$, where $y = b \vee z$ (is 1 if either of b, z is 1). Such a transition simulates one move of δ^{B_i}. It further records in flag y whether f was seen.

a.2 For every $i \notin X$ and for all $q_i \in F^{B_i}$, we add $((q_1, \cdots, q_n, p, x, z),\ \epsilon,\ (q_1, \cdots, q_{i-1}, \perp, q_{i+1},$

$\cdots, q_n, p, x, z)) \in \delta^{B_X}$ signifying possible end of the finite computation. We also add the transition from final state to initial state, this will start a new cluster for this thread. $((q_1, \cdots, q_i, \cdots, q_n, x, y), \epsilon, (q_1, \cdots, q_0^{B_{x_i}}, \cdots, q_n, x, y)) \in \delta^{B_X}$

a.3 For every $x_i \in X, x_i \neq x_m$ and for all $q_{x_i} \in F^{B_{x_i}}$ we add $((q_1, \cdots, q_{x_i}, \cdots, q_n, x_i, y), \epsilon, (q_1, \cdots, q_0^{B_{x_i}}, \cdots, q_n, x_{i+1}, y)) \in \delta^{B_X}$. This will ensure that we have transitioned from the final to initial state for $x_i \in X$.

a.4 We also add for all $q_{x_m} \in F^{B_{x_m}}$, $((q_1, \cdots, q_n, x_m, 1), \epsilon, (q_1, \cdots, q_n, x_m, 2))$ to δ^{B_X} and $((q_1, \cdots, q_{x_m}, \cdots, q_n, x_m, 2), \epsilon, (q_1, \cdots, q_0^{B_{x_m}}, \cdots, q_n, x_1, 0))$ to δ^{B_X} . Note that by adding the special state with flag 2 and making it the final state, we are forcing the automaton to move from x_m to x_1. This ensures that each of the elements in X are seen infinitely often.

To establish correctness of the construction, we will prove the following Lemma.

Lemma 38. *$w \in L(B)$ iff the following holds*

1. *w is well formed*
2. *For each $i \in [1..n]$, there is a sequence σ_i of clusters of the stack i such that $w \in$ $\texttt{Shuffle}(\{\mathbf{Abs}_{q_f}(\sigma_1)\}, \ldots, \{\mathbf{Abs}_{q_f}(\sigma_n)\})$.*
3. *There are infinitely many indices $j \in \mathbb{N}$ such that $w[j] \in Q \times [1..n] \times \{1\} \times Q$.*

Proof. ($\Rightarrow$)

1. Let $w \in L(B)$, it is easy to see why w is well formed. Firstly notice that all moves on the letters from w are by using the transition in a.1. In each of these transitions, the target state of previously seen input is recorded (i.e. if the previously seen input is $(p, i, 1, p')$ then p' is recorded in the $(n+1)$'st component of the state). Further notice that the transition can be fired only if the previously recorded target state matches the source state of the input currently read (i.e. the transition is enabled on input $(p, i, 1, p')$ only if the $(n+1)$'st component of the current state is p). From this it is easy to see that for any input w that is accepted, the source state of $w[i]$ for every i matches with the target state of $w[i-1]$. Hence any word w that is accepted, is well formed.
2. Follows directly from the fact that we chose $|X| > 1$ and from the following easy to see Claim. Here, we will use $w\downarrow_i$ to mean $w\downarrow_{(Q\times\{i\}\times[0,1]\times Q)}$

   Claim 6. *If $w \in L(B_X)$ then for all $i \in X$, we have $w\downarrow_i \in L(B_i)^\omega$ and for all $i \notin X$, we have $w\downarrow_i \in L(B_i)^*$.*

3. Since $w \in L(B)$, it satisfies Büchi condition specified. Hence there are infinitely many positions where state in F^B is visited. Between any two visits to these final states, the last element of the tuple resets to 0 and changes to 1 before going back to 2. Clearly there was at least one abstraction which was form $Q \times [1..n] \times \{1\} \times Q$ between any two visits to final state.

($\Leftarrow$)

For this direction, we are given $w \in (Q \times [1..n] \times [0,1] \times Q)^\omega$ such that

- w is well formed
- $w \in \texttt{Shuffle}(\{\mathbf{Abs}(\sigma_1)\}, \ldots, \{\mathbf{Abs}(\sigma_n)\})$, where σ_i is a sequence of k cluster of stack i.

- There are infinitely many indices $j \in \mathbb{N}$ such that $w[j] \in Q \times [1 \dots n] \times \{1\} \times Q$

We show that there is a corresponding accepting run in B_X for w, where $X \subseteq [1..n]$ is the set of indices such that $X = \{x \mid \sigma_x \text{ is infinite}\}$. We will now show inductively that for any $i \in \mathbb{N}$, there is a run of the form $\pi_i = q_0^B \xrightarrow{w'}{}^* (q_1, \cdots, q_n, x_j, z, q)$, where $w' = w[1 \dots i]$, for each $x \in X$, we have $q_0^{B_x} \xrightarrow{w'\downarrow_x}{}^* q_x$ and for $y \notin X$, if $w'\downarrow_y \neq w\downarrow_y$ then $q_0^{B_y} \xrightarrow{w'\downarrow_y}{}^* q_y$ else $q_y = \perp$. We also have for every $i < j$, π_i to be a prefix of π_j.

Base case is trivial.

For any j, let $\pi_j = q_0^B \xrightarrow{w'}{}^* (q_1, \cdots, q_n, x_j, z, p)$ (with $w' = w[1 \dots j]$) be the run given by the induction hypothesis. We extend such a run to $j+1$. We will assume that $w[j+1] = (p, i, s, p')$.

Note that for each $t \in [1..n]$, w' can be split as $u_1^t, \cdots . u_\ell^t$ for some $\ell \in \mathbb{N}$ such that for each $u_j, j \in [1..\ell - 1]$, we have $u_j \downarrow_t \in L(B_t)$ (except for u_ℓ, all other words are from the language of B_t). Further for u_ℓ, we have a run of the form $q_0^{B_t} \xrightarrow{u_\ell\downarrow_t}{}^* q_t$. This follows from the nature of w we have assumed and the fact that, for $t \notin X$, we have $\sigma_t \downarrow_t \in L(B_t)^*$ and for $t \in X$, we have $\sigma_t \downarrow_t \in L(B_t)^\omega$. Now for i, we can split w' as $w' = u_1^i \cdots u_\ell^i$. Let the run on B_i over $u_\ell^i.(p, i, s, p')\downarrow_i$ be of the form $q_0^{B_i} \xrightarrow{u_\ell.(p,i,s,p')\downarrow_i}{}^* q_i'$, where the last transition used is $(q_i, (p, i, s, p'), q_i') \in \delta^{B_i}$. By definition we have the transition $((q_1, \cdots, q_i, \cdots, q_n, x_j, z, p), (p, i, s, p'), (q_1, \cdots, q_i', \cdots, q_n, x_j, y, p')) \in \delta^B$. Hence we can extend the run on $w'.(p, i, s, p')$ as

$$\pi' = q_0^B \xrightarrow{w'}{}^* (q_1, \cdots, q_i, \cdots, q_n, x_j, z, p) \xrightarrow{(p,i,s,p')} (q_1, \cdots, q_i', \cdots, q_n, x_j, y, p')$$

where $y = z \vee b$. Further,

- if $i \notin X$ and $w'.(p, i, s, p')\downarrow_i = \sigma_i$ then, we terminate the execution of component i

$$\pi' = q_0^B \xrightarrow{w'.(q,i,f,q')}{}^* (q_1, \cdots, q_i', \cdots, q_n, x_j, y, p') \rightarrow (q_1, \cdots, q_{i-1}, \perp, q_{i+1} \cdots, q_n, x_j, y, p')$$

- if $q_i' \in F^{B_i}$ and $x_j = i \neq x_m$ then, we extend the run as

$$\pi' = q_0^B \xrightarrow{w'.(p,i,f,p')}{}^* (q_1, \cdots, q_i', \cdots, q_n, x_j, y, p') \rightarrow (q_1, \cdots, q_0^{B_i}, \cdots, q_n, x_{j+1}, y, p')$$

- if $q_i' \in F^{B_i}$ and $i \notin X$ then, we extend the run as

$$\pi' = q_0^B \xrightarrow{w'.(p,i,f,p')}{}^* (q_1, \cdots, q_i', \cdots, q_n, x_j, y, p') \rightarrow (q_1, \cdots, q_0^{B_i}, \cdots, q_n, x_j, y, p')$$

- If $i = x_m$, $q_i' \in F^{B_i}$ and $y = 1$ then, we extend the run as

$$\pi' = q_0^B \xrightarrow{w'.(p,i,s,p')}{}^* (q_1, \cdots, q_i', \cdots, q_n, x_m, 1, p') \rightarrow (q_1, \cdots, q_i', \cdots, q_n, x_1, 2, p') \rightarrow (q_1, \cdots, q_0^{B_i}, \cdots, q_n, x_1, 0, p')$$

Firstly note that in our construction, for each $j \in \mathbb{N}$ we are extending the run that was obtained inductively for $j-1$ and together they define a single infinite run. By definition of

our X, all components outside X do not have an infinite sequence of clusters. Hence for all the components in the state corresponding to any $y \notin X$ will reach $\perp$ eventually. For every component x in X, $w\downarrow_x$ is a concatenation of an infinite sequence of clusters. Each such cluster is accepted by a B_x automata. From these, it is easy to see that the constructed run is accepting in B_X. This completes the proof of Lemma 38. □

Size of each of B_i is $O(k.|M|^{dk})$, where $d \in \mathbb{N}$ is some constant. Hence size of $|B| = O(k|M|^{dkn})$. This completes the proof of Lemma 37.

□

Now what is left to consider is a k scope-bounded computation of an MPDS M, say $\pi = \pi_1 \bullet \pi_2 \bullet \cdots \pi_m$ that can be decomposed into finitely many contexts ending in an infinite context. As in previous case, we would like to know when a well formed word $w \in (Q \times [0..n] \times [0,1] \times Q)^\omega$ characterises the abstraction of an infinite k scope-bounded computation. Firstly, w.l.o.g we assume that the infinite context in such computations occurs only for stack-1 (i.e. $Context(\pi_m) = 1$). Notice that for any k scope-bounded computation π, by definition, for each $j \in [2 \ldots n]$, $Comp_j(\pi) =< \sigma_1\sigma_2\cdots\sigma_{m_j} >_j$ (where each $\sigma_1, \cdots, \sigma_{m_j}$ are k clusters), and $Comp_1(\pi) =< \sigma_1, \sigma_2, \cdots \sigma_{m_i}, \rho >_1$,(where each $\sigma_1, \cdots, \sigma_{m_i}$ are k clusters), $\rho =< \pi'_1, \pi'_2 \cdots, \pi'_\ell >_i$ is a sequence of contexts, where $\ell \leq k$ and π'_ℓ is an infinite context. Using this information, in the following Lemma, we characterise the existence of k scope-bounded run as a well-formed infinite word of the form $w \in (Q \times [0..n] \times [0,1] \times Q)^\omega$.

Lemma 39. *Given an MPDS M, there is an infinite k-scope-bounded computation $\pi = \pi_1 \bullet \pi_2 \bullet \cdots \pi_m$ that can be decomposed into finitely many contexts (ending in a infinite context) visiting infinitely often the state f if and only if there is a well formed word $w \in (Q \times [0..n] \times [0,1] \times Q)^\omega$ such that*

- *There are infinitely many indices $j \in \mathbb{N}$ such that $w[j] \in Q \times [1..n] \times \{1\} \times Q$*
- *For every $i \in [1..n]$, there is a finite sequence σ_i of k clusters of the stack i, further for stack-1, there is is a sequence of at most $\ell \leq k$ contexts $< \pi'_1, \pi'_2 \cdots, \pi'_\ell >_1$, with π'_ℓ being an infinite context such that $w \in \texttt{Shuffle}(\{\textbf{Abs}(< \sigma_1, \pi'_1, \cdots, \pi'_{\ell-1} >_1)\}, \{\textbf{Abs}(\sigma_2)\}, \ldots, \{\textbf{Abs}(\sigma_n)\}).\textbf{Abs}(\pi'_\ell)$.*

Proof. The proof of this Lemma is very similar to proof of 33. Hence we omit the same.

□

We now show that checking existence of such a well formed word can be reduced to checking emptiness on an Büchi pushdown automata.

Lemma 40. *Given an MPDS $\mathrm{M} = (n, Q, \Gamma, \Delta, q_0, \gamma_0)$ and a final state f, the problem of checking whether there is an infinite word $w \in (Q \times [0..n] \times [0,1] \times Q)^\omega$ such that*

- *w is well-formed*
- *There are infinitely many indices $j \in \mathbb{N}$ such that $w[j] \in Q \times [1..n] \times \{1\} \times Q$*
- *For every $i \in [1..n]$, there is a finite sequence σ_i of k clusters of the stack i, further for stack-1, there is ρ such that $\rho =< \pi_1, \pi_2 \cdots, \pi_\ell >_1$ is a sequence of $\ell \leq k$ contexts, with π_ℓ being an infinite context such that $w \in \texttt{Shuffle}(\{\textbf{Abs}(< \sigma_1, \pi_1, \cdots, \pi_{\ell-1} >_1)\}, \{\textbf{Abs}(\sigma_2)\}, \ldots, \{\textbf{Abs}(\sigma_n)\}).\textbf{Abs}(\pi_\ell)$.*

can be reduced to the emptiness problem for a Büchi pushdown automaton. The size of such a Büchi pushdown automaton will be $O((k|\mathrm{M}|)^{dkn})$ for some constant d.

Proof. As in the proof of Lemma 37 , it is possible to construct a finite state automaton B_i accepting exactly all the finite words of the form $\mathbf{Abs}(\sigma)$, where σ is a cluster of size at most k of the stack $i \in [1..n]$. On the other hand, we can construct a Büchi pushdown automaton P accepting the set of infinite words of the form $\mathbf{Abs}(< \pi_1, \pi_2, \cdots, \pi_\ell >_1)$, where $\pi_1, \cdots, \pi_\ell$ is a sequence of contexts of the first stack such that π_ℓ is an infinite context and $\ell \leq k$. Finally, we can use standard automaton constructions, to show that we can construct a Büchi pushdown P that accepts all the well-formed words w satisfying the required properties. The details are formalised below.

In the below construction, we will use B_i^* to denote the automata obtained by adding an epsilon transition from final states to the initial state of B_i automaton i.e. $B_i^* = (Q^{B_i}, (Q \times \{i\} \times [0,1] \times Q), \delta^{B_i^*}, q_0^{B_i}, F^{B_i})$, where $\delta^{B_i^*} = \delta^{B_i} \cup \{(f, \epsilon, q_0^{B_i}) | f \in F^{B_i}\}$, note that $L(B_i^*) = L(B_i)^*$.

The required Büchi pushdown automata is given by $P = (Q^P, (Q \times [1..n] \times [0,1] \times Q), \Gamma, \delta^P, q_0^P, f^P)$. where

- $Q^P = (Q^{B_1^*} \cup Q \times [1..k]) \times Q^{B_2^*} \cup \{\bot\} \times \cdots \times Q^{B_n^*} \cup \{\bot\} \times Q \times [0,1])$ is the set of control states (we will assume that only stack-1 can have infinite context). The first n components in the state are for simulating the B_i automata (and additionally the pushdown system during the last k context execution, in case of stack-1). The penultimate component is used to ensure that the word accepted is well-formed. The last component is used during the execution of the k-context of stack-1.
- Γ is the set of stack alphabet of MPDS M.
- $q_0^P = (q_0^{B_1}, q_0^{B_2}, \cdots, q_0^{B_n}, q_0^M, 0)$ is the initial state.
- $f^P = ((f,k), \bot, \cdots, \bot, 1)$ is the final state.
- The transition relation δ^P is given as follows.

  b.1 For all $i \in [1..n]$, if $(q_i, (q, i, x, q'), q_i') \in \delta^{B_i^*}$ then we add for all $j \neq i, q_j \in Q^{B_j^*}$ the transitions $((q_1, \cdots, q_{i-1}, q_i, \cdots, q_n, q, 0), \mathbf{Int}, (q, i, x, q'), (q_1, \cdots, q_i', \cdots, q_n, q', 0)) \in \delta^P$. We simulate each B_i^* in a well formed manner.

  b.2 For every $i \in [2..n]$, $q_i \in F^{B_i^*}$, we add for all $p \in (Q^{B_1^*} \cup Q \times [1..k-1])$ and $q_i \in Q^{B_i^*}$, we add $((p, q_2 \cdots, q_i, \cdots, q_n, q, 0), \mathbf{Int}, \epsilon, (p, q_2, \cdots, q_{i-1}, \bot, q_{i+1}, \cdots, q_n, q, 0)) \in \delta^P$, to end the finite cluster sequence.

  b.3 Further we add for all $q_1 \in F^{B_1^*}$, $((q_1, q_2 \cdots, q_n, q, 0), \mathbf{Int}, \epsilon, ((q_1, 1), q_2, \cdots, q_n, q_1, 1)) \in \delta^P$ to denote the end of finite cluster sequence and beginning of the last sequence involving k contexts for stack-1.

  b.4 For every transition of the form $(q, 1, \mathbf{Push}_1(\alpha), q') \in \Delta$, we add the following transitions. We also add similar transitions for $\mathbf{Pop}_1, \mathbf{Int}_1, \mathbf{Zero}_1$ operations.

  - for each $i < k$ and for all $j > 1, s_j \in Q^{B_j^*} \cup \{\bot\}$, the transitions $(((q,i), s_2, \cdots, s_n, p, 1), \mathbf{Push}(\alpha), \epsilon, ((q', i), s_2, \cdots, s_n, p, 1)) \in \delta^P$. These set of transitions simulate the last sequence of stack-1 contexts, upto the very last infinite context.
  - The transitions $(((q,k), \bot, \cdots, \bot, q, 1), \mathbf{Push}(\alpha), (q, 1, x, q'), ((q', k), \bot, \cdots, \bot, q', 1)) \in \delta^P$, where $x = 1$ if $q = f$ and $x = 0$ otherwise. These transitions simulate the last infinite context.

b.5 We add for all $i \in [2..n]$, $s_i \in Q^{B_i^*} \cup \{\bot\}$ and $j \leq k$, $(((q_1, j), s_2, \cdots, s_n, q, 0), \mathbf{Int}, \epsilon, ((q_1, j), s_2, \cdots, s_n, q, 1)) \in \delta^P$. These set of transitions mark beginning of context π'_j.

b.6 We add for all $j < k$ and $i \in [2..n]$, $s_i \in Q^{B_i^*} \cup \{\bot\}$, the transition $(((q, j), s_2, \cdots, s_n, q', 1), \mathbf{Int}, (q', 1, x, q), ((q, j+1), s_2, \cdots, s_n, q, 0)) \in \delta^P$ for all $q \in Q$. These set of transitions mark end of context π'_j.

b.7 We further add for all $q \in Q, j \leq k$, the transitions $(((q, j), \bot, \cdots, \bot, q, 1), \mathbf{Int}, \epsilon, (q, k), \bot, \cdots, \bot, q, 0) \in \delta^P$. These set of transitions mark the beginning of the infinite context.

We prove in sequel, the correctness of our construction. The following Lemma relates an infinite run in the constructed pushdown system to an infinite context in the multi-pushdown system.

Lemma 41. *Given any configuration $c = (((q, k), \bot^{n-1}, q, 1), \gamma\bot)$ of P, an infinite word $w \in L^\omega(P, c)$ iff there is an infinite context π starting from $(q, \gamma\bot, \bot^{n-1})$ of stack 1, such that $\mathbf{Abs}(\pi) = w$.*

Proof. This directly follows from the construction of the pushdown system, where states of the form $((q, k), \bot^n, p, 1)$ simulates an infinite context of MPDS, move by move. □

Lemma 42. *$w \in L(P)$ iff $w = u.v$ such that $u \in \texttt{Shuffle}(\{\mathbf{Abs}(\sigma_1, <\pi_1, \cdots, \pi_{\ell-1}>_1)\}, \{\mathbf{Abs}(\sigma_2)\}, \ldots, \{\mathbf{Abs}(\sigma_n)\})$, $v = \mathbf{Abs}(\pi_\ell)$, where $\pi_1, \pi_2 \cdots, \pi_\ell, \ell \leq k$ are context of stack-1 (with π_ℓ being an infinite context) and $\sigma_i, i \in [1..n]$ is finite sequence of k-clusters of stack-i.*

Proof. ($\Rightarrow$)

Suppose $w \in L(P)$, clearly there is a run $\pi = ((q_0^{B_1}, q_0^{B_2}, \cdots, q_0^{B_n}, q_0^M, 0), \bot) \xrightarrow{u}{}^* (((q, \ell-1), \bot^{n-1}, q, x), \gamma\bot) \rightarrow (((q, k), \bot^{n-1}, q, 1), \gamma\bot) \xrightarrow{v}{}^* \ldots$ for some $x \in [0, 1]$ and $\ell \leq k$. Clearly by Lemma 41, we have a computation π_ℓ in M starting from $(q, \gamma\bot, \bot, \cdots, \bot)$ such that $\mathbf{Abs}(\pi_\ell) = v$. We will show that there are $\sigma_1, \cdots \sigma_n, \pi_1, \cdots, \pi_{\ell-1}$ such that $u \in \texttt{Shuffle}(\{\mathbf{Abs}(<\sigma_1, \pi_1, \cdots, \pi_{\ell-1}>_1)\}, \{\mathbf{Abs}(\sigma_2)\}, \ldots, \{\mathbf{Abs}(\sigma_n)\})$. For this, we will present below a set of very easy to see claims. Here, the proofs are omitted since they are either straight forward or very similar to the ones we saw in Lemma 37.

Claim 7. *For any $w \in L(P)$, w is well formed.*

Claim 8. *If $w \in L(P)$, then for any $i \neq 1$ and we have that $w \downarrow_i \in L(B_i^*)$*

Claim 9. *For any $w = u_1.u_2.v$ such that $\pi = ((q_0^{B_1}, q_0^{B_2}, \cdots, q_0^{B_n}, q_0^M, 0), \bot) \xrightarrow{u_1}{}^* ((q_1, \cdots, q_n, p, 0), \bot) \rightarrow (((p, 1), q_2, \cdots, q_n, p, 1), \bot) \xrightarrow{u_2}{}^* (((q, j), \bot^{n-1}, q, x), \bot) \rightarrow (((q, k), \bot^{n-1}, q, 1), \gamma\bot) \xrightarrow{v}{}^* \ldots$ for some $j < k$, $q_i \in Q^{B_i^*}, p, q \in Q$, we have that $u_1 \downarrow_1 \in L(B_1^*)$*

Claim 10. *For all $j > 1, s_j \in Q^{B_i^*} \cup \{\bot\}, q, q' \in Q$, we have $(((q, i), s_2, \cdots, s_n, q, 1), \gamma\bot) \xrightarrow{\epsilon}{}^*{}_P ((q', i), s_2, \cdots, s_n, q, 1), \gamma'\bot) \xrightarrow{(q,1,x,q')}{}^*{}_P (((q', i), s_2, \cdots, s_n, q', 0), \gamma'\bot)$ iff there is a context $\pi_i = (q, \gamma\bot, \gamma_2\bot, \cdots, \gamma_n\bot) \rightarrow^*{}_M (q', \gamma'\bot, \gamma_2\bot, \cdots, \gamma_n\bot)$ for all $\gamma_1, \cdots, \gamma_n \in (\Gamma \setminus \{\bot\})^*$.*

From Claim 8, clearly there is a finite sequence of clusters σ_i for each $i \in [2..n]$ such that $\mathbf{Abs}(\sigma_i) = u \downarrow_i$. From Claim 9 and 10, we have $\sigma_1, \pi_1 \cdots, \pi_{\ell-1}$ and u_1, u_2 such that $u_1 \downarrow_1 =$

$\mathbf{Abs}(\sigma_1)$, $u_2\downarrow_1 = \mathbf{Abs}(< \pi_1, \pi_2 \cdots, \pi_{\ell-1} >_1)$. Hence we have that $u \in \texttt{Shuffle}(\{\mathbf{Abs}(< \sigma_1, \pi_1, \cdots, \pi_{\ell-1} >_1)\}, \{\mathbf{Abs}(\sigma_2)\}, \ldots, \{\mathbf{Abs}(\sigma_n)\})$.

($\Leftarrow$)

Let $w = u.v$ be well formed sequence of interfaces such that $u \in \texttt{Shuffle}(\{\mathbf{Abs}(< \sigma_1, \pi_1, \cdots, \pi_{\ell-1} >_1)\}, \{\mathbf{Abs}(\sigma_2)\}, \ldots, \{\mathbf{Abs}(\sigma_n)\})$, $v = \mathbf{Abs}(\pi_\ell)$, where $\pi_1, \pi_2 \cdots, \pi_\ell$ are contexts of stack-1 and $\sigma_i, i \in [1..n]$ is a finite sequence of clusters of stack-i.

Firstly note that it is enough to show that there is a computation

$$\pi = ((q_0^{B_1^*}, \cdots, q_0^{B_n^*}, 0, q_0^M), \bot) \xrightarrow{u}{}^*{}_P (((q, \ell), \bot^{n-1}, q, x), \gamma\bot)$$

for some $\ell < k$. Since combining this with the transition $(((q, \ell), \bot^{n-1}, q, x), \mathbf{Int}, \epsilon, (((q, k), \bot^{n-1}, q, 1))$ and Lemma 41, will give us the desired run. Towards this, we will inductively show that for every prefix u' of u (i.e. $u' = u[1 \ldots j]$ for some j)), there is a run of the form $((q_0^{B_1^*}, \cdots, q_0^{B_n^*}, 0, q_0^M), \bot) \xrightarrow{u'}{}^*{}_P ((s, q_2, \cdots, q_n, q, 0), \gamma\bot)$ (where $u' = u[1 \ldots j]$)) and the following properties hold.

- For $i \in [2..n]$, if $u'\downarrow_i \neq \sigma_i$ then $q_0^{B_i^*} \xrightarrow{u'\downarrow_i}{}^* q_i$ else if $u'\downarrow_i = \sigma_i$ then $q_i = \bot$
- if $u'\downarrow_1$ is a prefix of $\mathbf{Abs}(\sigma_1)$ then we have that $s = q_1$, $q_0^{B_1^*} \xrightarrow{u'\downarrow_i}{}^* q_1$ and $\gamma = \epsilon$. Else if $u'\downarrow_1$ is equal to $\mathbf{Abs}(< \sigma_1, \pi_1, \cdots, \pi_{j-1}, \pi_j >_1)$, for some $j < k$, then $s = (State(\texttt{Target}(\pi_j)), j+1)$ and $\gamma = Stack(\texttt{Target}(\pi_j))$

We will now show how to construct such a run.

- Base case being run of length 0 is simple.
- Let $u'(p, i, y, p')$ be any prefix of w, with $i \neq 1$. Let

  $$((q_0^{B_1^*}, \cdots, q_0^{B_n^*}, 0, q_0^M), \bot) \xrightarrow{u'}{}^* ((q, q_2, \cdots, q_n, 0, p), \gamma.\bot)$$

  be the run got by induction. Such a run can be extended to $u'a$ (where $a = (p, i, y, p')$) as follows.

  $$\pi' = ((q_0^{B_1^*}, \cdots, q_0^{B_n^*}, 0, q_0^M), \bot) \xrightarrow{u'}{}^* ((q, q_2, \cdots, q_n, 0, p), \gamma.\bot) \xrightarrow{a} ((q, q_2, \cdots, q_i', \cdots, q_n, 0, p'), \gamma\bot) \text{ where } q_i' = \delta^{B_i^*}(q_i, a)$$

  Further, if $u'.a\downarrow_i = \mathbf{Abs}(\sigma_i)$ (note that in this case, $q_i' \in F^{B_i^*}$ since $w'.a\downarrow_i \in L(B_i^*)$) then we will use the transition in b.2 and let the extended run to be

  $$\pi' = ((q_0^{B_1^*}, \cdots, q_0^{B_n^*}, 0, q_0^M), \bot) \xrightarrow{u'}{}^* ((q, q_2, \cdots, q_n, 0, p), \gamma\bot) \xrightarrow{a} ((q, q_2, \cdots, q_i', \cdots, q_n, 0, p'), \gamma\bot) \rightarrow ((q, q_2, \cdots, q_{i-1}, \bot, q_{i+1}, \cdots, q_n, 0, p'), \gamma\bot)$$

- In case of $u'a$, with $a = (p, 1, x, p')$ being the prefix, there there are two distinct possibilities. i.e. $u'.a\downarrow_1$ is a prefix of $\mathbf{Abs}(\sigma_1)$ or $u'.a\downarrow_1 = \mathbf{Abs}(< \sigma_1, \pi_1, \cdots, \pi_j >_1)$ for some $j < k$. We will show how to extend in each of these two cases.
  - The case where $u'.a\downarrow_1$ is a prefix of $\mathbf{Abs}(\sigma_1)$ is similar to one discussed above for $a = (p, j, y, p')$ with $j \in [2..n]$.

– We will now consider the case where $u'.a\downarrow_1 = \mathbf{Abs}(<\sigma_1,\pi_1,\cdots,\pi_j>_1)$ for some $j<k$. We have two cases to consider, namely $j=1$ and $j>1$. We only consider the case where $j=1$ (since case where $j>1$ is similar and straight forward.). For this, let the run got by induction be as follows.

$$((q_0^{B_1^*},\cdots,q_0^{B_n^*},0,q_0^M),\perp)\xrightarrow{u'}{}^{*}((q,q_2,\cdots,q_n,0,p),\gamma.\perp)$$

clearly $\gamma=\epsilon$. Let $\pi_1=(q,\perp,\gamma_2,\cdots,\gamma_n)\rightarrow^*{}_M(q',\gamma'\perp,\gamma_2,\cdots,\gamma_n)$. Then by Claim 10, we have a run of the form $(((q,1),q_2,\cdots,q_n,1,p),\perp)\rightarrow^*(((q',1),q_2,\cdots,q_n,1,p),\gamma'\perp)$. Combining this with the transition in b.3 of the form $((q,q_2,\cdots,q_n,0,p),\mathbf{Int},\epsilon,((q,1),q_2,\cdots,q_n,1,p)$, we get the following run, which completes the proof.

$$\begin{aligned}((q_0^{B_1^*},\cdots,q_0^{B_n^*},0,q_0^M),\perp)\xrightarrow{u'}{}^{*}((q,q_2,\cdots,q_n,0,p),\perp)\rightarrow\\ (((q,1),q_2,\cdots,q_n,1,p),\perp)\rightarrow^*(((q',1),q_2,\cdots,q_n,1,p),\gamma'\perp)\\ \xrightarrow{a}(((q',1),q_2,\cdots,q_n,0,p),\gamma'\perp)\end{aligned}$$

This completes the proof of Lemma 42

□

This completes the proof of Lemma 40 □

Now, the required Büchi pushdown system to complete the proof of Theorem 15 is obtained by taking union of the Büchi automata and the Büchi pushdown automata that we constructed above.

6.4.2 LTL Model checking

We consider in this section the linear-time model checking problem for MPDS's under scope-bounding. We consider that we are given ω-regular properties expressed in linear-time propositional temporal logic (LTL) [122]. Let us fix a set of atomic propositions *Prop*, and let $k\in\mathbb{N}$ be a natural number. The *k scope-bounded model-checking* problem is the following: Given a formula φ (in LTL) with atomic propositions from *Prop*, and a MPDS $M=(n, Q,\Gamma,\Delta,q_{\text{init}},\gamma_0)$ along with a labeling function $\Lambda: Q\rightarrow 2^{Prop}$ associating to each state $q\in Q$ the set of atomic propositions that are true in it, check whether all infinite k-scope-bounded computations of M from the initial configuration c_M^{init} satisfy φ.

To solve this problem, we adopt an automata-based approach similar to the one used in [40] to solve the analogous problem for pushdown systems. We construct a Büchi automaton $\mathscr{B}_{\neg\varphi}$ over the alphabet 2^{Prop} accepting the negation of φ [141, 140]. Then, we compute the product of the MPDS M and the Büchi automaton $\mathscr{B}_{\neg\varphi}$ to obtain a MPDS $M_{\neg\varphi}$ with a Büchi accepting set of states F and leaving us with the task of checking if any of its its k scope-bounded runs is accepting. Hence, we can reduce our model-checking problem to the k scope-bounded repeated reachability problem for MPDSs, which, by Theorem 15, can be solved.

Theorem 16. *The problem of scope-bounded model checking LTL properties of multi-pushdown systems is EXPTIME-complete.*

The lower bound of Theorem 16 follows immediately from the fact that the model-checking problem for LTL for pushdown systems (i.e., MPDS with only one stack) are EXPTIME-complete [40].

For the upper bound, it is well known that, given a MPDS M and an ω-regular formula φ, it is possible to construct a MPDS M' and a set of repeating states F of M' such that the problem of scope-bounded model checking of M w.r.t. the formula φ is reducible to the k-scope-bounded repeated state reachability problem of a MPDS M' w.r.t. F. Moreover, the size of M' is exponential in the size of φ and polynomial in the size of M and k. Applying Theorem 15 to the MPDS M' and F, we obtain our complexity result.

6.5 Conclusion

In this chapter, we established that the repeated reachability problem and the model checking linear-time properties (expressed as formulas of LTL) against scope-bounded executions of multi pushdown system are decidable in EXPTIME. Model checking LTL properties are also EXPTIME-COMPLETE.

Chapter 7

Adjacent ordered MPDS

7.1 Introduction

In this chapter, we introduce a restricted variant of multi-pushdown system called the *adjacent ordered multi-pushdown system* (AOMPDS). Informally, an adjacent ordered multi-pushdown system allows pop operations only on the least non-empty stack (active stack) and restricts every other operation to the least non-empty stack or its adjacent stacks. In this chapter, we will show that for such systems, the control state reachability problem is EXPTIME COMPLETE. This is significantly better than the 2-ETIME complexity required for solving the control state reachability problem under the ordered restriction or the bounded-phase restriction. We also note that such a system allows transfer of content from the least non-empty stack to the next stack (adjacent higher numbered stack). This is not possible under the bounded-context restriction. In fact, to the best of our knowledge, it is the first restriction that allows transferring the contents of a stack and yet has an EXPTIME procedure to solve the control state reachability problem.

We next provide a EXPTIME procedure to solve the repeated reachability problem on such systems. As an application of this, we also get a procedure to model check LTL properties against the runs of an adjacent ordered MPDS. We note that in case of both bounded-phase and bounded-context infinite executions, the run eventually degenerates to effectively using only a single stack. Even though under bounded-phase restriction, an infinite run can involve pushing elements infinitely often onto multiple stacks, eventually the content of only one stack can effectively be read. Ordered restriction that we discussed about earlier, allows infinite runs effectively involving multiple stacks. However the complexity required to model check LTL properties against such a restriction is very high (2ETIME-COMPLETE). Similar to bounded-scope restriction that we saw earlier, AOMPDS allows infinite runs involving multiple stacks, and yet has EXPTIME complexity.

Later in this chapter, we illustrate the power of AOMPDS using some applications. We first show that reachability on recursive programs communicating via queues [138], whose connection topology is a forest can be reduced to reachability on an AOMPDS with at most polynomial blowup. We also show that bounded-phase reachability on an MPDS can be reduced to reachability on AOMPDS, with at most exponential blowup.

7.2 Adjacent ordered multi-pushdown system

In this section, we use a slightly different model of MPDS, where in a move it is possible to examine the top of each stack and modify more than one stack at a time (i.e. the ability to re-write). Such a definition of MPDS has been used in literature earlier [14, 16]. This simplifies our constructions and proofs. As in the case of pushdown systems, such a definition is equivalent to the one which uses only push/pop operations. In section 7.5, we describe how AOMPDS an also be seen as a restriction on the behaviours of the push-pop style MPDS used elsewhere in this thesis.

Definition 9. *An* Adjacently Odered Multi-PushDown System *(AOMPDS) is a tuple* $\mathscr{A} = (n, Q, \Gamma, \Delta, q_0, \gamma_0)$ *where:*

- $n \geq 1$ *is the number of stacks,*
- Q *is the non-empty set of states,*
- Γ *is the stack alphabet containing the special symbol* $\perp$ *to mark the bottom of stack,*
- $q_0 \in Q$ *is the initial state,*
- $\gamma_0 \in \Gamma$ *is the initial stack symbol,*
- $\Delta \subseteq ((Q \times (\Gamma_\epsilon)^n) \times (Q \times (\Gamma^*)^n))$ *is the transition relation such that if* $((q, \gamma_1, \gamma_2, \ldots, \gamma_n), (q', \alpha_1, \alpha_2, \ldots, \alpha_n))$ *is in* Δ *then , there is an index* $i \in [1..n]$ *such that* $\gamma_1 = \cdots = \gamma_{i-1} = \perp$, $\gamma_i \in (\Gamma \setminus \{\perp\})$, *and* $\gamma_{i+1} = \cdots = \gamma_n = \epsilon$ *and further one of the following properties holds:*
  - **Operate on the stack** i**:**
    For all $j < i$, *we have* $\alpha_j = \perp$, *for all* $j > i$, *we have* $\alpha_j = \epsilon$ *and* $\alpha_i \in (\Gamma \setminus \{\perp\})^*$ *with* $|\alpha_i| \leq 2$. *We will refer to such transitions as* $\Delta_{(i,i)}$
  - **Push on the stack** $j = i - 1$ **:**
    $\alpha_j \in (\Gamma \cdot \{\perp\})$, *we have* $\alpha_i = \epsilon$, *for all* k *such that* $k \neq j$ *and* $k < i$, *we have* $\alpha_k = \perp$ *and for* $k > i$, *we have* $\alpha_k = \epsilon$. *We will refer to such transitions as* $\Delta_{(i,i-1)}$
  - **Push on the stack** $j = i + 1$ **:**
    $\alpha_i = \epsilon$, *for all* $k < i$, *we have* $\alpha_k = \perp$, *for all* k *such that* $j \neq k$ *and* $k > i$, *we have* $\alpha_k = \epsilon$ *and* $\alpha_j \in \Gamma$. *We will refer to such transitions as* $\Delta_{(i,i+1)}$

A configuration of an AOMPDS $\mathscr{A}$ is a (n+1) tuple $(q, w_1, w_2, \cdots, w_n)$ with $q \in Q$, and w_1, $w_2, \ldots, w_n \in (\Gamma \setminus \{\perp\})^* \perp$. We will use $\mathscr{C}(\mathscr{A})$ as in case of MPDS to denote set of configurations of the AOMPDS $\mathscr{A}$. The *initial configuration* $c^{\text{init}}_{\mathscr{A}}$ of the AOMPDS $\mathscr{A}$ is $(q_0, \perp, \ldots, \perp, \gamma_0\perp)$. If $t = ((q, \gamma_1, \ldots, \gamma_n), (q', \alpha_1, \ldots, \alpha_n))$ is an element of Δ, then $(q, \gamma_1 w_1, \ldots, \gamma_n w_n) \xrightarrow{t}_{\mathscr{A}} (q', \alpha_1 w_1, \ldots, \alpha_n w_n)$ for all $\gamma_1 w_1, \ldots, \gamma_n w_n \in (\Gamma \setminus \{\perp\})^* \perp$.

7.2.1 Reachability on AOMPDS

Theorem 17. *The reachability problem for Adjacent Ordered Multi-Pushdown System is* EXPTIME-COMPLETE.

Upper Bound: Let $\mathscr{A} = (n, Q, \Gamma, \Delta, q_0, \gamma_0)$ be an AOMPDS with $n > 1$ (the case where $n = 1$ boils down to the reachability of pushdown systems which is well-known to be in PTIME). The proof of EXPTIME-containment is through an inductive construction that reduces the

reachability problem for $\mathscr{A}$ to the reachability problem for a pushdown system with only an exponential blow up in size. The key step is to show that we can reduce the reachability problem for $\mathscr{A}$ to the reachability problem on an $(n-1)$-AOMPDS. The feature of our reduction is that there is no blowup in the state space and the size of the stack alphabet increases quadratically in the number of states. A non-linear blow up in the number of states will result in a complexity higher than EXPTIME.

We plan to use a single stack to simulate both the first and second stacks of $\mathscr{A}$. It is useful to consider runs of $\mathscr{A}$ to understand how this works. Any run ρ of $\mathscr{A}$ starting at the initial configuration naturally breaks up into segments $\sigma_0\rho_1\sigma_1\ldots\rho_k\sigma_k$ where the segments ρ_i contain configurations where stack 1 is non-empty while in any configuration in the σ_i's, the stack 1 is empty. Clearly the content of stack 1 at the beginning of ρ_i contains exactly two symbols, and we assume it to be $a_i\bot$. We further assume that ρ_i begins at control state q_i and the segment σ_i in state q_i'. What is the contribution of the segment ρ_i, which is essentially the run of a pushdown automaton starting and ending at the empty stack configuration, to this run?

Firstly, it transforms the local state from q_i to q_i'. Secondly, a word w_i is pushed on to stack 2 during this segment. It also, consumes the value a_i from stack 1 in this process, but that is not relevant to the rest of the computation. To simulate the effect of ρ_i it would thus suffice to jump from state q_i to q_i' and push the word w_i on stack 2. There are potentially infinitely many possible runs of the form ρ_i that go from q_i to q_i' while removing a_i from stack 1 and thus infinite possibilities for the word that is pushed on stack 2. However, it is easy to see that this set of words $L(q_i,a_i,q_i')$ is a CFL. If the language $L(q_i,a_i,q_i')$ is a regular language, we could simply *summarize* this run by depositing a word from this language on stack 2 and then proceed with the simulation of stack 2. However, since it is only a CFL this is not possible. Instead, we have to interleave the simulation of stack 2 with the simulation of stack 1, using stack 2, and there is no a priori bound on the number of switches between the stacks in such a simulation.

To simulate the effect of ρ_i, we jump directly to q_i' and push a non-terminal symbol (from the appropriate CFG) that generates the language $L(q_i,a_i,q_i')^R$ (reverse, because stacks are last in first out). Now, when we try to simulate σ_i', we might encounter a nonterminal on top of stack 2 instead of a terminal symbol belonging to stack 2. In this case, we rewrite the nonterminal using one of the rules of the CFG applicable to this nonterminal. In effect, we produce a left-most derivation of a word from $L(q_i,a_i,q_i')$ in a lazy manner, interspersed within the execution involving stack 2, generating terminals only when they need to be consumed. This is the main idea in the construction that is formalized below.

We define $\Delta_1 = \Delta_{(1,1)} \cup \Delta_{(1,2)}$, $\Delta_i = \Delta_{(i,i)} \cup \Delta_{(i,i+1)} \cup \Delta_{(i,i-1)}$ for all $2 \leq i < n$, and $\Delta_n = \Delta_{(n,n)} \cup \Delta_{(n,n-1)}$.

We construct a context-free grammar $G_{\mathscr{A}} = (\mathrm{NT}, (\Gamma \setminus \{\bot\}), P)$ from the AOMPDS $\mathscr{A}$. The set of non-terminals is $\mathrm{NT} = (Q \times (\Gamma \setminus \{\bot\}) \times Q)$. The set of productions P is defined as the smallest set of rules satisfying:

1. For every two states $p, p' \in Q$, and every transition $((q,\gamma,\epsilon,\ldots,\epsilon),(q',\gamma_1\gamma_2,\epsilon,\ldots,\epsilon))$ in Δ such that $\gamma,\gamma_1,\gamma_2 \in (\Gamma \setminus \{\bot\})$, we have $(q,\gamma,p) \Rightarrow_{G_{\mathscr{A}}} (q',\gamma_1,p')(p',\gamma_2,p)$
2. For every state $p \in Q$, and every transition $((q,\gamma,\epsilon,\ldots,\epsilon),(q',\gamma',\epsilon,\ldots,\epsilon))$ in Δ such that γ,

$\gamma' \in (\Gamma \setminus \{\perp\})$, we have $(q,\gamma,p) \Rightarrow_{G_{\mathscr{A}}} (q',\gamma',p)$

3. For every transition $((q,\gamma,\epsilon,\ldots,\epsilon),(q',\epsilon,\epsilon,\ldots,\epsilon))$ in Δ such that $\gamma \in (\Gamma \setminus \{\perp\})$, we have $(q,\gamma,q') \Rightarrow_{G_{\mathscr{A}}} \epsilon$
4. For every transition $((q,\gamma,\epsilon,\ldots,\epsilon),(q',\epsilon,\gamma',\epsilon,\ldots,\epsilon))$ in Δ such that $\gamma,\gamma' \in (\Gamma \setminus \{\perp\})$, we have $(q,\gamma,q') \Rightarrow_{G_{\mathscr{A}}} \gamma'$.

Then, it is easy to see that the context-free grammar summarizes the effect of the first stack on the second one. Formally, we have:

Lemma 43. *The context free language $L_{G_{\mathscr{A}}}((q,\gamma,q'))$ is equal to the set of words $\{w^R \in (\Gamma \setminus \{\perp\})^* \mid \exists \rho \in \Delta_1^*.\ (q,\gamma\perp,w_2,\ldots,w_n) \xrightarrow{\rho}_{\mathscr{A}} (q',\perp,ww_2,\ldots,w_n)\}$.*

The proof of the above Lemma is straight forward and very similar to the classical result of converting a pushdown to a context free grammer, hence we omit the same. We are now ready to show that reachability problems on $\mathscr{A}$ can be reduced to reachability problems on an $(n-1)$-AOMPDS $\mathscr{N}$. Further, the number of states of $\mathscr{N}$ is linear in $|Q|$, size of the stack alphabet of $\mathscr{N}$ is $O(|Q|^2.|\Gamma|)$ and the number of transitions is $O(|Q|^3.|\Gamma|.|\Delta|)$. The upper-bound claimed in Theorem 17 then follows by simple induction.

Let $F \subseteq Q$ be the set of states whose reachability we are interested in, we show how to construct $(n-1)$-AOMPA $\mathscr{N}$ such that the reachability question on $\mathscr{A}$ can be reduced to reachability question on $\mathscr{N}$. Formally, $\mathscr{N}$ is defined by the tuple $(n-1,Q,\Gamma \cup \mathrm{NT},\Delta',q_0,\gamma_0)$ where Δ' is defined as the smallest set satisfying the following conditions:

1. For any transition $((q,\perp,\gamma_2,\ldots,\gamma_n),(q',\perp,\alpha_2,\ldots,\alpha_n)) \in \Delta$, we have $((q,\gamma_2,\ldots,\gamma_n),(q',\alpha_2,\ldots,\alpha_n)) \in \Delta'$
2. For any transition $((q,\perp,\gamma_2,\epsilon,\ldots,\epsilon),(q',\gamma\perp,\epsilon,\ldots,\epsilon)) \in \Delta_{(2,1)}$, we have $((q,\gamma_2,\epsilon,\ldots,\epsilon),(q'',(q',\gamma,q''),\epsilon,\ldots,\epsilon)) \in \Delta'$ for all $q'' \in Q$
3. For any production rule $X \Rightarrow_{G_{\mathscr{A}}} w$ and state $q \in Q$, we have $((q,X,\epsilon,\ldots,\epsilon),(q,w^R,\epsilon,\ldots,\epsilon)) \in \Delta'$.

Lemma 44. *A state $f \in F$ is reachable in $\mathscr{A}$ iff f is reachable in $\mathscr{N}$.*

The fact that even a single contiguous segment of moves using stack 1 in $\mathscr{A}$ may now be interleaved arbitrarily with executions involving other stacks in $\mathscr{N}$, makes the proof some what involved. Towards the proof, we define a relation between the configurations of $\mathscr{N}$ and $\mathscr{A}$ systems. For any configuration $c \in \mathscr{C}(\mathscr{A})$ and $d \in \mathscr{C}(\mathscr{N})$, we say cRd iff one of the following is true:

1. d is of the form $(q,\perp,w_3,\cdots,w_n)$ and c is of the form $(q,\perp,\perp,w_3,\cdots,w_n)$
2. d is of the form $(q,\eta_1 v_1 \eta_2 v_2 \cdots \eta_m v_m \perp,w_3,\cdots,w_n)$ and c is of the form $(q,\perp,u_1 v_1 u_2 v_2 \cdots u_m v_m \perp,w_3,\cdots,w_n)$ where $v_1,u_1,v_2,u_2,\ldots,v_m,u_m \in (\Gamma \setminus \{\perp\})^*$, $\eta_1,\eta_2,\ldots,\eta_m \in \mathrm{NT}^*$ and $\eta_k \Rightarrow^*_{G_{\mathscr{A}}} u_k^R$ for all $k \in [1..m]$.

Thus, cRd verifies that it is possible to replace the nonterminals appearing in stack 2 in d by words they derive (and by tagging an additional empty stack for the missing stack 1) to obtain c. We now show that this relation faithfully transports runs (from the initial configuration) in both directions. This is the import of Lemmas 46 and 47, which together guarantee that the state reachability in $\mathscr{A}$ reduces to state reachability in $\mathscr{N}$. We will start by stating the

following simple Lemma. The proof of the Lemma follows directly from the fact that we can simply perform a left most derivation of the CFG.

Lemma 45. *Let $c_1 \in \mathscr{C}(\mathscr{A})$ be a configuration of $\mathscr{A}$ such that $c_{\mathscr{A}}^{init} \rightarrow^*_{\mathscr{A}} c_1$. For every configuration $d_1 \in \mathscr{C}(\mathscr{N})$ such that $c_1 R d_1$, if c_1 is of the form $(q, \bot, a w_2, w_3, \ldots, w_n)$ for some $a \in (\Gamma \setminus \{\bot\})$, then there is a configuration $d_2 \in \mathscr{C}(\mathscr{N})$ such that $c_1 R d_2$, $d_1 \rightarrow^*_{\mathscr{N}} d_2$, and d_2 is of the form $(q, a w_2', w_3, \ldots, w_n)$.*

Proof. If d_1 is already of the form $(q, a w_2', w_3, \ldots, w_n)$ then we have nothing to prove, else since $c_1 R d_1$, we have $d_1 = (q, \eta_1 v_1 \eta_2 v_2 \cdots \eta_n v_n, w_3, \ldots, w_n)$ for some $\eta_i \in \mathrm{NT}^*$, such that $\eta_i \Rightarrow^*_{G_{\mathscr{A}}} u_k^R$ and $u_1 v_1 u_2 v_2 \cdots u_n v_n = a w_2$. Since $\eta_1 \Rightarrow^* u_1^R$, there is a derivation $\eta_1 \Rightarrow^* \eta_1'^R a \Rightarrow^* u_1^R$. Combining this with production rule 3, we get $d_1 \rightarrow^*_{\mathscr{N}} d_2 = (q, a\eta_1' v_1 \eta_2 v_2 \cdots \eta_n v_n, w_3, \ldots, w_n)$, clearly $c_1 R d_2$.

□

Lemma 46. *Let $c_1, c_2 \in (Q \times \{\bot\} \times (Stack(\mathscr{A}))^{n-1})$ be two configurations such that $c_{\mathscr{A}}^{init} \rightarrow^*_{\mathscr{A}} c_1$ and $c_{\mathscr{A}}^{init} \rightarrow^*_{M} c_2$. If $c_1 \xrightarrow{\rho}_{\mathscr{A}} c_2$, with $\rho \in \cup_{i=3}^{n} \Delta_i \cup (\Delta_{(2,1)} \Delta_1^*) \cup \Delta_{(2,2)} \cup \Delta_{(2,3)}$, then for every configuration $d_1 \in \mathscr{C}(\mathscr{N})$ such that $c_1 R d_1$, there is a configuration $d_2 \in \mathscr{C}(\mathscr{N})$ such that $c_2 R d_2$ and $d_1 \rightarrow^*_{\mathscr{N}} d_2$.*

Proof. Let $c_1, c_2 \in \mathscr{C}(\mathscr{A})$ be two configurations such that $c_{\mathscr{A}}^{init} \rightarrow^*_{\mathscr{A}} c_1$ and $c_{\mathscr{A}}^{init} \rightarrow^*_{\mathscr{A}} c_2$. Let $c_1 \xrightarrow{\rho}_{\mathscr{A}} c_2$, with $\rho \in \cup_{i=3}^{n} \Delta_i \cup (\Delta_{(2,1)} \Delta_1^*) \cup \Delta_{(2,2)} \cup \Delta_{(2,3)}$. Let $d_1 \in \mathscr{C}(\mathscr{N})$ such that $c_1 R d_1$.

- Case $\rho \in \cup_{i=3}^{n} \Delta_i$ By the definition of AOMPDS we know that the first and second stack of $\mathscr{A}$ and the first stack of $\mathscr{N}$ are empty in the configuration c_1 and d_1 respectively. This implies that c_1 is d_1 extended with an empty stack 1. Moreover, since $c_1 \xrightarrow{\rho}_M c_2$, we have that c_2 is of the form $(q, \bot, w_2, \ldots, w_n)$ with $w_2 \in Stack(\mathscr{A})$. Then, we take $d_2 = (q, w_2, \ldots, w_n)$ and from the definition of $\mathscr{N}$ we have $d_1 \xrightarrow{\rho}_{\mathscr{N}} d_2$. Clearly $c_2 R d_2$.
- Case $\rho \in \Delta_{(2,2)}$ Here we have two cases to deal with depending on whether top of stack 1 of $\mathscr{N}$ is a nonterminal or from Γ.
  - Let us assume that d_1 is of the form $(q, a w_2, w_3, \cdots, w_n)$ where $a \in (\Gamma \setminus \{\bot\})$, $w_2, w_3, \ldots, w_n \in Stack(\mathscr{A})$. This implies that c_1 is of the form $(q, \bot, a w_2', w_3, \cdots, w_n)$. Let us assume that the result of firing the transition $\rho \in \Delta_{(2,2)}$ is the configuration c_2 which will be of the form $(q, \bot, u w_2', w_3, \ldots, w_n)$ where $u \in (\Gamma \setminus \{\bot\})^*$. Let d_2 be the configuration $(q, u w_2, w_3, \cdots, w_n)$ of $\mathscr{N}$. Since $c_1 R d_1$, we have $c_2 R d_2$. Moreover, from the definition of $\mathscr{N}$ we have $d_1 \xrightarrow{\rho}_{\mathscr{N}} d_2$.
  - Let us assume that d_1 is of the form $(q, X w_2, w_3, \ldots, w_n)$ where $X \in \mathrm{NT}$ is a nonterminal symbol. Then by Lemma 45, there is a configuration d_1' such that $d_1 \rightarrow^*_{\mathscr{N}} d_1'$, $c_1 R d_1'$ and the top symbol of the first stack of d_1' is a. Then, we apply the previous sub-case to d_1'.
- Case $\rho \in \Delta_{(2,3)}$ This case proceeds exactly as the previous one except that values pushed are on stack 3 instead of stack 2.
- Case $\rho \in \Delta_{(2,1)} \Delta_1^*$. Let $t \in \Delta_{(2,1)}$ such that $\rho = t\rho'$ for some ρ'. Let us assume that t is of the form $((q, \bot, \gamma, \epsilon, \ldots, \epsilon), (q', \gamma' \bot, \epsilon, \epsilon, \ldots, \epsilon))$. Then there exists a configuration $c \in \mathscr{C}(\mathscr{A})$ such that $c_1 \xrightarrow{t}_{\mathscr{A}} c$ and $c \xrightarrow{\rho'}_{\mathscr{A}} c_2$. Let us assume that c_1 and c_2 are of the form $(q, \bot, \gamma w_2, w_3, \ldots,$

w_n) and $(q'', \perp, uw_2, w_3, \ldots, w_n)$ respectively. Since $c_1 \xrightarrow{t}_{\mathscr{A}} c$, we have that $c = (q', \gamma'\perp, w_2, w_3, \ldots, w_n)$.
Without loss of generality, we will assume that d_1 does not contain a non-terminal as top of stack (otherwise by Lemma-45, we can get to a configuration with this property which can be reached from d_1, preserving R). Hence, d_1 is of the form $(q, \gamma w_2', w_3, \ldots, w_n)$. Now let $d_2 = (q'', (q', \gamma', q'')w_2', w_3, \ldots, w_n)$. Clearly, $d_1 \rightarrow_{\mathscr{N}} d_2$. Moreover, we have that $c_2 R d_2$ since $u^R \in L_{G_{\mathscr{A}}}((q', \gamma', q''))$ by Lemma 43.

□

Lemma 47. *Let $d_1, d_2 \in \mathscr{C}(\mathscr{N})$ be two configurations of $\mathscr{N}$ such that $c_N^{init} \rightarrow^*_{\mathscr{N}} d_1 \xrightarrow{t}_{\mathscr{N}} d_2$ for some $t \in \Delta'$. Then for every configuration $c_2 \in \mathscr{C}(\mathscr{A})$ such that $c_2 R d_2$, there is a configuration $c_1 \in \mathscr{C}(\mathscr{A})$ such that $c_1 R d_1$ and $c_1 \rightarrow^*_{\mathscr{A}} c_2$.*

Proof. Let $d_1, d_2 \in \mathscr{C}(\mathscr{N})$ be two configurations of $\mathscr{N}$ such that $c_{\mathscr{N}}^{init} \rightarrow^*_{\mathscr{N}} d_1 \xrightarrow{t}_{\mathscr{N}} d_2$ for some $t \in \Delta'$. Let $c_2 \in \mathscr{C}_1^{\mathscr{A}}$ such that $c_2 R d_2$.

- Case t of the form $((q, \perp, \gamma_3, \ldots, \gamma_n), (q', \perp, \alpha_3, \ldots, \alpha_n))$. This implies that d_1 and d_2 are of the form $(q, \perp, w_3, \ldots, w_n)$ and $(q', \perp, w_3', \ldots, w_n')$. Thus c_2 is of the form $(q', \perp, \perp, w_3', \ldots, w_n')$. Let t' be a transition of M of the form $((q, \perp, \perp, \gamma_3, \ldots, \gamma_n), (q', \perp, \perp, \alpha_3, \ldots, \alpha_n))$ (from the definition of $\mathscr{N}$, we know that such transition exists). Then, let c_1 be the configuration of $\mathscr{A}$ defined as follows: $(q, \perp, \perp, w_3, \ldots, w_n)$. Then it is easy to see that $c_1 R d_1$ and $c_1 \xrightarrow{t'}_{\mathscr{A}} c_2$.

- Case t of the form $((q, \perp, \gamma, \epsilon, \ldots, \epsilon), (q', \gamma'\perp, \epsilon, \ldots, \epsilon))$. This implies that d_1 and d_2 are of the form $(q, \perp, \gamma w_3, \ldots, w_n)$ and $(q', \gamma'\perp, w_3, \ldots, w_n)$. Thus c_2 is of the form $(q', \perp, \gamma'\perp, w_3, \ldots, w_n)$. Let t' be a transition of $\mathscr{A}$ of the form $((q, \perp, \perp, \gamma, \epsilon, \ldots, \epsilon), (q', \perp, \gamma'\perp, \epsilon, \ldots, \epsilon))$ (from the definition of $\mathscr{N}$, we know that such transition exists). Then, let c_1 be the configuration of $\mathscr{A}$ defined as follows: $(q, \perp, \gamma'\perp, w_3, \ldots, w_n)$. Then it is easy to see that $c_1 R d_1$ and $c_1 \xrightarrow{t'}_{\mathscr{A}} c_2$.

- Case t of the form $((q, \gamma, \epsilon, \ldots, \epsilon), (q', \epsilon, \gamma', \epsilon, \ldots, \epsilon))$. This implies that d_1 and d_2 are of the form $(q, \gamma w_2, w_3, \ldots, w_n)$ and $(q', w_2, \gamma' w_3, w_4, \ldots, w_n)$. Thus c_2 is of the form $(q', \perp, u, \gamma' w_3, w_4, \ldots, w_n)$. Let t' be a transition of $\mathscr{A}$ of the form $((q, \perp, \gamma, \epsilon, \ldots, \epsilon), (q', \perp, \epsilon, \gamma', \epsilon, \ldots, \epsilon))$ (from the definition of $\mathscr{N}$, we know that such transition exists). Then, let c_1 be the configuration of $\mathscr{A}$ defined as follows: $(q, \perp, \gamma u, w_3, w_4, \ldots, w_n)$. Then it is easy to see that $c_1 R d_1$ and $c_1 \xrightarrow{t'}_{\mathscr{A}} c_2$.

- Case t of the form $((q, \gamma, \epsilon, \ldots, \epsilon), (q', \alpha, \epsilon, \ldots, \epsilon))$. This implies that d_1 and d_2 are of the form $(q, \gamma w_2, w_3, \ldots, w_n)$ and $(q', \alpha w_2, w_3, w_4, \ldots, w_n)$. Thus c_2 is of the form $(q', \perp, \alpha u, w_3, w_4, \ldots, w_n)$. Then let $c_1 = (q', \perp, \gamma u, w_3, w_4, \ldots, w_n)$ and $t' = ((q, \perp, \gamma, \epsilon, \ldots, \epsilon), (q', \perp, \alpha, \epsilon, \ldots, \epsilon))$. Clearly t' is a transition in $\mathscr{A}$, $c_1 \xrightarrow{t'}_{\mathscr{A}} c_2$ and $c_1 R d_1$.

- Case t of the form $((q, X, \epsilon, \ldots, \epsilon), (q', w^R, \epsilon, \epsilon, \ldots, \epsilon))$ with $X \Rightarrow_{G_M} w$. This implies that d_1 and d_2 are of the form $(q, X w_2, w_3, \ldots, w_n)$ and $(q', w^R w_2, w_3, \ldots, w_n)$. Thus c_2 is of the form $(q', \perp, u, w_3, w_4, \ldots, w_n)$. Let $c_1 = c_2$. Then it is easy to see that $c_1 R d_1$ and $c_1 \rightarrow^*_{\mathscr{A}} c_2$.

- Case t of the form $((q,\gamma,\epsilon,\ldots,\epsilon),(q'',(q',\gamma',q''),\epsilon,\ldots,\epsilon))$. This implies that d_1 and d_2 are of the form $(q,\gamma w_2,w_3,\ldots,w_n)$ and $(q'',(q',\gamma',q'')w_2,w_3,\ldots,w_n)$. Thus c_2 is of the form $(q'',\perp,uu',w_3,\ldots,w_n)$ such that $u^R \in L_{G_{\mathscr{A}}}((q',\gamma',q''))$ and $(q',\perp,u',w_3,\ldots,w_n)R(q',w_2,w_3,\ldots,w_n)$. From Lemma 43, we know that there exists $\rho' \in \Delta_1^*$ such that $(q',\gamma'\perp,u',w_3,\ldots,w_n) \xrightarrow{\rho'}{}^{*}_{\mathscr{A}} (q'',\perp,u\cdot u',w_3,\ldots,w_n)$.

  Let t' be a transition of $\mathscr{A}$ of the form $((q,\perp,\gamma,\epsilon,\ldots,\epsilon),(q',\gamma'\perp,\epsilon,\ldots,\epsilon))$ (from the definition of $\mathscr{N}$, we know that such transition exists). Then, let c_1 be the configuration of $\mathscr{A}$ defined as follows: $(q,\perp,\gamma u',w_3,\ldots,w_n)$. Then it is easy to see that $c_1 R d_1$ and $c_1 \xrightarrow{t'}_{\mathscr{A}} (q',\gamma'\perp,u',w_3,\ldots,w_n) \xrightarrow{\rho'}{}^{*}_{\mathscr{A}} c_2 = (q'',\perp,uu',w_3,\ldots,w_n)$.

□

Proof of Lemma 44

Proof. $\Rightarrow$ Suppose f is reachable in $\mathscr{A}$, then there is a compuation of the form $c^{\text{init}}_{\mathscr{A}} \rightarrow^* c_1 \rightarrow^* c_2 \rightarrow^* \cdots \rightarrow^* c_n$, such that $State(c_n) = f$ and each $c_i \in (Q \times \{\perp\} \times (Stack(\mathscr{A}))^{n-1})$. Firstly note that $c^{\text{init}}_{\mathscr{A}} R c^{\text{init}}_{\mathscr{N}}$. Now using Lemma 46, we can find $d_1, d_2, \cdots d_n$ such that $c_i R d_i$ and $d_i \rightarrow^* d_{i+1}$ for all $i \in [1..n]$. From this we have that $c^{\text{init}}_{\mathscr{N}} \rightarrow^* d_1 \rightarrow^* d_2 \cdots \rightarrow^* d_n$. Further from definition of R, we have that $State(d_n) = f$.

$\Leftarrow$

Now suppose that f is reachable in $\mathscr{N}$. Then there is a computation of the form $c^{\text{init}}_{\mathscr{N}} \rightarrow d_1 \rightarrow d_2 \rightarrow \cdots \rightarrow d_n$ such that $State(d_n) = f$. Let $d_n = (f,\gamma_2,\cdots\gamma_n)$, we let $c_n = (f,\perp,\gamma_2,\cdots,\gamma_n)$, clearly $c_n R d_n$. Now using Lemma 47 we can find $c_{n-1},\cdots,c_1,c_0$ such that $c_i R d_i$ for all $i \in [1..n]$ and $c_0 R c^{\text{init}}_{\mathscr{N}}$, such that $c_0 \rightarrow^* c_1 \rightarrow^* c_2 \cdots \rightarrow^* c_n$. Further $c_0 = c^{\text{init}}_{\mathscr{A}}$ follows from the definition of R.

□

Complexity

Clearly the number of states of $\mathscr{N}$ remains the same ($|Q|$), size of the stack alphabet of $\mathscr{N}$ is bounded by $O(|Q|^2.|\Gamma|)$ and the number of transitions is bounded by $O(|Q|^3.|\Delta|)$. Since we repeat this procedure n times, the final pushdown system that we construct has state size as $|Q|$, the stack size as $O(|Q|^{2n}.|\Gamma|)$ and the transition size as $O(|Q|^{3n}.|\Delta|)$. This gives us the desired upper bound.

7.2.2 Hardness result

Lemma 48. *Given an AOMPDS $\mathscr{A} = (n,Q,\Gamma,\Delta,\gamma_0)$ and a state $q \in Q$, the problem of deciding whether q is reachable from the initial configuration is* EXPTIME-HARD

Proof. The emptiness of the intersection of a PDA with n finite automata is known to be EXPTIME-HARD ([80]). We reduce this problem to the reachability problem on an AOMPDS.

Let $\mathscr{P}$ be a pushdown automaton and $\mathscr{B}_i, 2 \le i \le n$ be finite automata. We assume that $\mathscr{B}_i$ do not contain ϵ-transitions.

We assume that the pushdown automaton $\mathscr{P}$ recognizes the context-free language L, and the $n-1$ finite state automata $\mathscr{B}_2,\ldots,\mathscr{B}_n$ recognize the regular languages $L_2,\ldots,L_n$ respectively.

The simulation proceeds as follows: The AOMPDS $\mathscr{A}$ first starts the simulation of the pushdown automaton $\mathscr{P}$ using its first stack. An ϵ-labeled transition of $\mathscr{P}$ is simulated by a transition on the first stack while the other stacks remain unchanged. A labeled transition of $\mathscr{P}$ with an input symbol a is simulated by a transition on the first stack, followed by a transition that pushes the input symbol a into the second stack. At the end of this phase, we have that the first stack of $\mathscr{A}$ is empty and that the second stack of $\mathscr{A}$ contains a word u^R such that $u \in L$. Then, $\mathscr{A}$ starts the simulation of the finite state automaton $\mathscr{B}_2$ in order to check that $u \in L_2$. A transition of the form (q,a,q') of $\mathscr{B}_2$ is simulated by a transition of $\mathscr{A}$ that moves the current state from q' to q while popping the stack symbol a from the second stack and pushing a into the third stack (this can be achieved by popping a and storing it in state space). At the end of this phase, we have that the first and second stacks of $\mathscr{A}$ are empty and that the third stack of $\mathscr{A}$ contains the word u such that $u \in L \cap L_2$. Next, $\mathscr{A}$ starts the simulation of the finite state automaton $\mathscr{B}_3$ in order to check that $u \in L_3$. A transition of the form (q,a,q') of $\mathscr{B}_3$ is simulated by a transition of $\mathscr{A}$ that moves the current state from q to q' while popping the stack symbol a from the third stack and pushing a into the fourth stack. We can see that at the end of this phase, we have that the first, second, and the third stacks of $\mathscr{A}$ are empty and that the fourth stack of $\mathscr{A}$ contains the word u^R such that $u \in L \cap L_2 \cap L_3$. The simulation go on in a similar manner as in the previous cases to check that, for every $i \in [4..n]$, we have $u \in L_i$.

□

7.3 LTL Model Checking on AOMPDS

In this section, we show that given an LTL formula, model checking it against runs of adjacent ordered multi-pushdown system is EXPTIME-COMPLETE.

Let φ be an w-regular formula built from a set of atomic propositions *Prop*, and let $\mathscr{A} = (n,Q,\Gamma,\Delta,q_0,\gamma_0)$ be an AOMPDS with a labeling function $\Lambda : Q \to 2^{Prop}$ associating to each state $q \in Q$ the set of atomic propositions that are true in it. In the following, we are interested in solving *the model checking problem* which consists of checking whether all the infinite runs starting from $c^{init}_{\mathscr{A}}$ satisfy the formula φ.

To solve this problem, we adopt an approach similar to [40] and we construct a B*ü*chi automaton $B_{\neg\varphi}$ over the alphabet 2^{Prop} accepting the negation of φ [141]. Then, we compute the product of the AOMPDS $\mathscr{A}$ and of the B*ü*chi automaton $B_{\neg\varphi}$ to obtain an AOMPDS $\mathscr{A}_{\neg\varphi}$ with a set of repeating states F. Now, it is easy to see that the original problem can be reduced to the *reachability problem* which checks whether there is an infinite run of $\mathscr{A}_{\neg\varphi}$ starting from $c^{init}_{\mathscr{A}_{\neg\varphi}}$ that visits infinitely often a state in F. We will use the following Theorem from [14] which shows how to reduce the repeated state reachability problem for OMPDSs to the reachability problem for OMPDSs.

Theorem 18 ([14])**.** *Let $M = (n,Q,\Gamma,\Delta,q_0,\gamma_0)$ be an OMPDS and q_f be a state of M. There is*

an infinite run starting from c_M^{init} that visits infinitely often the state q_f if and only if there are $i \in [1..n]$, $q \in Q$, and $\gamma \in \Gamma \setminus \{\bot\}$ such that:

- $c_M^{init} \rightarrow_M^* (q, (\bot)^{i-1}, \gamma w, w_{i+1}, \ldots, w_n)$ *for some* $w, w_{i+1}, \ldots, w_n \in \Gamma^*$.
- $(q, (\bot)^{i-1}, \gamma\bot, (\bot)^{n-i}) \xrightarrow{\rho_1}_M (q_f, w_1, w_2, \ldots, w_n) \xrightarrow{\rho_2}_M (q, (\bot)^{i-1}, \gamma w_i', w_{i+1}', \ldots, w_n')$ *for some* $w_1, \ldots, w_n, w_i', \ldots, w_n' \in (\Gamma \setminus \{\bot\})^*\bot$, $\rho_1 \in \Delta'^*$ *and* $\rho_2 \in \Delta'^+$ *where* Δ' *contains all the transitions of the form* $((q, (\bot)^{j-1}, \gamma_j, \epsilon, \ldots, \epsilon), (q, \alpha_1, \ldots, \alpha_n)) \in \Delta$ *such that* $1 \leq j \leq i$ *and* $\gamma_j \in (\Gamma \setminus \{\bot\})$.

Now, we are ready to state our results for AOMPDSs:

Theorem 19. *Let $M = (n, Q, \Gamma, \Delta, q_0, \gamma_0)$ be an AOMPDS and q_f be a state of M. Then checking whether there is an infinite run starting from c_M^{init} that visits infinitely often the state q_f can be solved in time $O(|M|)^{poly(n)}$.*

Proof. From Theorem 18, we know that checking whether there is an infinite run starting from c_M^{init} that visits infinitely often the state q_f can be reduced to checking whether there exist $i \in [1..n]$, $q \in Q$, and $\gamma \in \Gamma \setminus \{\bot\}$ such that:

1. $c_M^{init} \rightarrow_M^* (q, (\bot)^{i-1}, \gamma w, w_{i+1}, \ldots, w_n)$ for some $w, w_{i+1}, \ldots, w_n \in (\Gamma \setminus \{\bot\})^*\bot$.
2. $(q, (\bot)^{i-1}, \gamma\bot, (\bot)^{n-i}) \xrightarrow{\rho_1}_M (q_f, w_1, w_2, \ldots, w_n) \xrightarrow{\rho_2}_M (q, (\bot)^{i-1}, \gamma w_i', w_{i+1}', \ldots, w_n')$ for some $w_1, \ldots, w_n, w_i', \ldots, w_n' \in (\Gamma \setminus \{\bot\})^*$, $\rho_1 \in \Delta'^*$ and $\rho_2 \in \Delta'^+$.

Let us fix an index $i \in [1..n]$, a state q, and a stack symbol $\gamma \in \Gamma \setminus \{\bot\}$.

Checking whether $c_M^{init} \rightarrow_M^* (q, (\bot)^{i-1}, \gamma w, w_{i+1}, \ldots, w_n)$ for some $w, w_{i+1}, \ldots, w_n \in (\Gamma \setminus \{\bot\})^*$ can be easily reduced to the reachability problem of an AOMPDS M_1 (whose size is linear in M) that proceeds into two steps. In the first step, M_1 mimics the behavior of M. Then, nondeterministically, checks if the current state is q and the top most of the i-th stack is γ, and if it is the case M_1 moves its state to a special state $f \notin Q$ and starts emptying all its stacks (from i to n). Thus, we have $c_M^{init} \rightarrow_M^* (q, (\bot)^{i-1}, \gamma w, w_{i+1}, \ldots, w_n)$ for some $w, w_{i+1}, \ldots, w_n \in \Gamma^*$ if and only if M_1 can reach the configuration $(f, \bot, \ldots, \bot)$ from $c_{M_1^{init}}$.

Now, we can show in similar way that checking whether $(q, (\bot)^{i-1}, \gamma\bot, (\bot)^{n-i}) \xrightarrow{\rho_1}_M (q_f, w_1, w_2, \ldots, w_n) \xrightarrow{\rho_2}_M (q, (\bot)^{i-1}, \gamma w_i', w_{i+1}', \ldots, w_n')$ for some $w_1, \ldots, w_n, w_i', \ldots, w_n' \in \Gamma^*$, $\rho_1 \in \Delta'^*$ and $\rho_2 \in \Delta'^+$, can be reduced to the reachability problem of an AOMPDS M_2 (whose size is linear in M) that proceeds as follows: First, M_2 starts by reaching the configuration $(q, (\bot)^{i-1}, \gamma\bot, (\bot)^{n-i})$ from its initial one (using some transitions not belonging to M). Then M_2 moves its current state from q to a copy (q, false). Now M_2 can start simulating M while restricting its behavior to the set of transitions in Δ' and replacing any state q' of M by its copy state (q', false). M_2 checks if the current state is (q_f, false). At this point, for every transition Δ' of the form $((q_f, \gamma_1, \ldots, \gamma_n), (q_2, \alpha_1, \ldots, \alpha_n))$, M_2 has a transition of the form $(((q_f, \mathit{false}), \gamma_1, \ldots, \gamma_n), ((q_2, \mathit{true}), \alpha_1, \ldots, \alpha_n))$. This transition is to ensure that $\rho_2 \in \Delta'^+$ (i.e., the trace ρ_2 is not empty). After performing one of such transitions, M_2 continues the simulation of M restricted to the set of transitions in Δ' and replacing any state q' of M by its copy state (q', true). Finally, in non-deterministic way, M_2 checks if its current state is (q, true) and the topmost stack symbol of its i-stack is γ, and if it is the case, he moves to a special state $f \notin Q$ and starts emptying all

its stacks (from i to n). Then, we have $(q,(\perp)^{i-1},\gamma,(\perp)^{n-i}) \xrightarrow{\rho_1}_M (q_f,w_1,w_2,\ldots,w_n) \xrightarrow{\rho_2}_M (q,(\perp)^{i-1},\gamma w_i',w_{i+1}',\ldots,w_n')$ for some $w_1,\ldots,w_n,w_i',\ldots,w_n' \in \Gamma^*$, $\rho_1 \in \Delta'^*$ and $\rho_2 \in \Delta'^+$ if and only if the configuration $(f,\perp,\ldots,\perp)$ is reachable by M_2 from $c_{M_2^{init}}$.

Finally, we can use the constructions given in previous section to show that checking whether the configuration $(f,\perp,\ldots,\perp)$ is reachable in M_k (with $k \in \{1,2\}$) from $c_{M_k^{init}}$ can be solved in time $O(|M|)^{poly(n)}$.

□

As an immediate corollary, we obtain:

Corollary 2. *The model checking problem for the linear-time temporal logic on runs of AOMPDS is* EXPTIME-COMPLETE.

7.4 Applications of AOMPDS

7.4.1 An application to Recursive Queuing Concurrent Programs

La Torre et al. [138], study the decidability of control state reachability in networks of concurrent processes communicating via queues. Each component process may be recursive, i.e., equipped with a pushdown store, and such systems are called *recursive queuing concurrent programs* (RQCP) in [138]. Further, the state space of the entire system may be global or we may restrict each process to have its own local state space (so that the global state space is the product of the local states). In the terminology of [138] the latter are called RQCPs without shared memory.

An architecture describes the underlying topology of the network, i.e., a graph whose vertices denote the processes and edges correspond to communication channels (queues). One of the main results in [138] is a precise characterization of the architectures for which the reachability problem for RQCP's is decidable. Understandably, given the expressive power of queues and stacks, this class is very restrictive. To obtain any decidability at all, one needs the *well-queuing* assumption, which prohibits any process from dequeuing a message from any of its incoming channels as long as its stack is non-empty. They show that, even under the well-queuing assumption, the only architectures for which the reachability problem is decidable for RQCPs without shared memory are the so called *directed forest* architectures. A directed tree is a tree with an identified root and where all edges are oriented away from the root towards the leaves. A directed forest is a disjoint union of directed trees. They use a reduction to the reachability problem for bounded-phase MPDSs and obtain a double exponential decision procedure.

We now show that this problem can be reduced to the reachability problem for AOMPDS and obtain an EXPTIME upper-bound.[1] The reduction is sketched below. An EXPTIME upper-bound is also obtained via tree-width bounds [112] (Thm. 4.6).

[1] The argument in Theorem 17 can also be adapted to show EXPTIME-HARDNESS.

Theorem 20. *The control state reachability problem for RQCPs with a directed forest architecture, without shared memory and under the well-queuing assumption can be solved in* EXPTIME.

Proof. We only consider the directed tree architecture and the result for the directed forest follows quite easily from this. An observation, from [138], is that it suffices to only consider executions with the following property: if q is a child of p then p executes all its steps (and hence deposits all its messages for q) before q executes. We fix some topologically sorted order of the tree, say $p_1, p_2, \ldots, p_m$. The AOMPDS we construct only simulates those executions of the RQCP in which all moves of p_i are completed before p_{i+1} begins its execution. We call such a run of the RQCP as a *canonical run*. The number of stacks used is $2m-1$. The message alphabet is $\Gamma \times \{1,\ldots,m\} \cup \bigcup_{1\le i\le m} \Sigma_i$, where Γ is the communication message alphabet and Σ_i is the stack alphabet of process p_i. We write Γ_i to denote $\Gamma \times \{i\}$.

As we simulate a canonical run ρ of the RQCP in the order $p_1,\ldots,p_m$, the invariant we maintain is that, at the beginning of the simulation of process p_i, the contents of stack $2i-1$ is some α so that $\alpha \downarrow_{\Gamma_i}$ is the contents of the unique input channel to p_i as p_i begins its execution in ρ. Thus, we can simulate p_i's contribution to ρ, by popping from stack $2i-1$ when a value is to be consumed from the input queue. If top of stack $2i-1$ does not belong ot Γ_i, then we transfer it to stack $2i$. When p_i sends a message to any other process p_j in ρ (which must be one of its children in the tree) we simulate it by tagging the message with the process identity and pushing it on stack $2i$. Finally, as observed in [138], the stack for p_i can also be simulated on top of stack $2i-1$ since a value is dequeued only when its local stack is empty (according to the well-queuing assumption). At the end of the simulation of process p_i, we empty any contents left on stack $2i-1$ (transferring elements of $\Gamma \times \{i+1,\ldots,m\}$ to stack $2i$). Finally, we copy stack $2i$ onto stack $2i+1$ and simulate process p_{i+1} using stack $2i+1$ (thus ensuring that there is no reversal of the contents of the queues). The state space is linear in the size of the RQCP and hence we conclude that the reachability problem for RQCPs can be solved in EXPTIME using Theorem 17. □

7.4.2 An application to bounded-phase reachability

Recall that a *Phase* of a stack $i \in [1..n]$ is a computation that involves pops (and zero test) only from stack-i i.e. it is a computation of the form $\pi = c_0 \xrightarrow{t_1} c_1 \xrightarrow{t_2} \cdots$ in which $Trace(\pi) \in \Delta^{\downarrow_i}$. Where $\Delta^{\downarrow_i} = \Delta \cap (Q \times (\text{op} \setminus \bigcup_{j\neq i} \cup_{a\in\Gamma}\{\mathbf{Pop}_j(a)\} \cup \{\mathbf{Zero}_j\}) \times Q)$. We will refer to such a computation as i-run to mean that is a 1-phase computation of stack-i. Now, a run $c \xrightarrow{\rho}{}^*{}_M c'$ is a k-*phase run* if we may write $\rho = \rho_1.\rho_2.\ldots\rho_k$ with $\rho_i \in \Delta^*$, $c = c_0 \xrightarrow{\rho_1}{}^*{}_M c_1 \xrightarrow{\rho_1}{}^*{}_M c_2 \ldots \xrightarrow{\rho_k}{}^*{}_M c_k = c'$ and each $c_i \xrightarrow{\rho_{i+1}}{}^*{}_M c_{i+1}$ is a 1-phase run. We say that such a run is a *good k-phase run* if $k \le n$ and for all $1 \le i \le k$, $c_i \xrightarrow{\rho_{i+1}}{}^*_M c_{i+1}$ is an i-run in which the stacks $1,2\ldots(i-1)$ are empty in every configuration.

The (*good*) *k-phase reachability problem* is to decide for a MPDS M, a number k and a state $q \in Q$, whether there is a l-phase run (good l-phase run) $(q_0,\bot,\bot,\ldots,\gamma_0\bot) \rightarrow^*_M (q,\alpha_1,\alpha_2,\ldots,\alpha_n)$ with $\alpha_i \in (\Gamma \setminus \{\bot\})^*\bot$ for some $l \le k$. The k-phase reachability problem is shown to be 2-ETIME-Complete in [137]. We provide a reduction of this problem to the reachability problem for AOMPDSs, providing a simple proof of decidability for BPMPDSs and illustrating

the expressive power of AOMPDS. We first observe that the k-phase reachability problem can be transformed to a good k-phase reachability problem.

Lemma 49. *Let $M = (n, Q, \Gamma, \Delta, q_0, \gamma_0)$ be a simple MPDS, k an integer and $q \in Q$. Then, the k-phase reachability problem for q in M can be reduced to the good k-phase reachability problem for some q' in a simple MPDS $M' = (k+1, Q', \Gamma \cup \{\#\}, \Delta', q'_0, \gamma_0)$ where $|Q'| = O(|Q|.n^k)$ and $|\Delta'| = O(|\Delta|.n^{2k})$. Further, every run of M' is actually a good l-phase run for some $l \leq k$.*

Proof. The automaton begins by guessing a sequence $s_1, s_2, \ldots, s_k$, $1 \leq s_j \leq n$, of stacks that would be used in the k-phase run that is to be simulated. Notice that any stack s may appear more than once in this sequence (or even not at all). We inductively ensure that when this automaton begins its ith phase, the contents of stack i are exactly the contents of stack s_i at the beginning of phase i of the k-phase run of M which is being simulated.

A move of M, during phase i may push values into not only stack s_i but also the other stacks. The activity on stack s_i is simulated accurately using stack i. Further, we simply disregard the values pushed on any stack s that does not appear among $s_{i+1}, \ldots, s_k$ since these values will never be used. Finally, any value pushed on to a stack s that appears among s_{i+1}, $\ldots, s_k$ is pushed on stack j where j is the smallest number such that $s_j = s$. At the end of the simulation of phase i, if stack s_i does not appear among $s_{i+1}, \ldots, s_k$ then we simply empty its contents before switching to simulating phase $i+1$ using stack $i+1$. If stack s_i is used again and j is the least number greater than i with $s_j = s_i$ then we transfer the contents of stack i to stack j via stack $i+1$ (so that the order is not reversed).

□

Next, we show that any good k-phase reachability problem for any simple MPDS M can be reduced to the reachability problem for an AOMPDS M'. Thus, using the EXPTIME complexity of AOMPDS, we get a 2-EXPTIME algorithm for BPMPDS.

Lemma 50. *Any good k-phase reachability problem for an MPDS $M = (n, Q, \Gamma, \Delta, q_0, \gamma_0)$ can be reduced to the reachability problem for an AOMPDS $M' = (2^{n-1}, Q', \Gamma', \Delta', q'_0, \gamma'_0)$ where: $|Q'| = O(|Q|.2^{O(n)})$, $|\Delta'| = O(2^{O(k)}.|\Delta|)$ and $|\Gamma'| = O(2^{O(k)}.|\Gamma|)$.*

Proof. (sketch) Observe that in a good k-phase run, during the ith phase values are popped only from stack i, which is also the leftmost nonempty stack. Further values are pushed only on stacks numbered i or higher. To simulate such a run using an AOMPDS we should ensure that the values are pushed only on stacks i or $i+1$ (or $i-1$, but given the nature of a good k-phase run this will be unnecessary). Our strategy is to push the values meant for all the stacks other than i into stack $i+1$, after appropriately tagging them with the identity of their destination. This naive strategy has some problems. Firstly, when operating on stack i in phase i we will encounter values meant for other stacks (with a tag identifying the destination stack j, $j > i$). We simply transfer these values as and when they are encountered to stack $i+1$. The second problem is that when values tagged with stack j are eventually transferred to stack j during phase $j-1$, they may not occur in the right order. One reason for this reordering is that as we transfer contents from stack i to stack $i+1$, the values meant for a future stack j get reversed in order. This is a relatively minor issue which can be handled by inserting an additional stack, if necessary, to carry out one more reversal.

A more serious problem that results in an exponential increase in the number of stacks is the following — phase i in M may involve pushing values on stack j, $j > i$ and we instead tag it with j and push it on stack $i+1$. However, it is quite possible that inside stack i there might be values for stack j pushed during earlier phases which will be uncovered later and transferred to stack $i+1$, resulting in shuffling the values meant for stack j that are generated by i and those that are only transferred by i. To get round this we need to keep several copies of each stack, one for each stack whose phase may push values into this stack.

If the number of phases (and stacks) in M is just 1 or 2 then there is no problem as the aforementioned shuffling does not occur for trivial reasons. Suppose M has 3 stacks (and phases). Then we keep two copies of stack 3 which call say 3.1 and 3.2. We assume the order of the stacks is $1, 2, 3.2, 3.1$. Whenever a value is to be pushed onto stack 3 during phase 1, it is marked as destined for 3.1 and pushed on stack 2. During phase 2 any value destined for stack 3 is marked as destined for stack 3.2 and pushed on to the next stack (which is also 3.2). The phase corresponding to stack 3 is now broken up into two phases – first a phase on stack 3.2 and another on stack 3.1. The phase change from stack 3.2 to stack 3.1 takes place only when stack 3.2 is empty. Notice that this precisely captures the fact that any value pushed by phase 1 on stack 3 is accessible only after every value pushed by phase 2 on stack 3 has been removed. Observe that if M has 4 stacks then we will need 4 copies of stack 4 ($4.3.2, 4.3.1, 4.2, 4.1$ in that order) where the any value meant for stack 4 during a phase on stack $i \in \{1, 2, 3.2, 3.1\}$) is marked as destined for stack $4.i$. Thus, if there were n phases (and stacks) we construct end up with an AOMPDS with 2^{n-1} phases and stacks. It also turns out that the number of stacks between any stack i and its consumers (stacks of the form $j.i$, $j \in [1..n]$) is always even. Thus no additional reversal is needed.

□

7.5 Adjacent ordered restriction

In this section we will introduce a restriction called *adjacent ordered restriction* on runs of MPDS. We then show that solving reachability problem on AOMPDS is same as solving reachability on MPDS under such a restriction. We first recall the definition of **Act** (Indicating the currently active i.e. the least nonempty stack). We defined it as, if $c \in Q \times \bot^{j-1} \times (\Gamma^+\bot) \times (\Gamma^*\bot)^{n-j}$, then $\mathbf{Act}(c) = j$ and if $c \in Q \times \{\bot\}^n$, then $\mathbf{Act}(c) = n$.

Definition 10. *Given an MPDS $M = (n, Q, \Gamma, \Delta, q_0, \gamma_0)$, for any two configurations $c, c' \in \mathscr{C}(M)$, with $\mathbf{Act}(c) = j$, the one step execution $\pi = c \xrightarrow{\tau}_M c'$ is said to be an* adjacent ordered *execution iff τ is a push, internal or a pop operation and one of the following holds.*

1. *$\tau \in (Q \times \cup_{a\in\Gamma} \mathbf{Pop}_j(a) \cup \mathbf{Int}_j \times Q) \cap \Delta$, i.e. the internal transitions (designated for stack-j) and the pop operations are allowed on least non-empty stack.*
2. *Any push operation from Δ can be performed only on least non-empty stack or its adjacent stacks.*

   b.1 If $1 < j < n$, then $\tau \in (Q \times \cup_{a\in\Gamma, k\in[j-1..j+1]} \mathbf{Push}_k(a) \times Q) \cap \Delta$
   b.2 If $j = n$ then $\tau \in (Q \times \cup_{a\in\Gamma, k\in[n-1,n]} \mathbf{Push}_k(a) \times Q) \cap \Delta$
   b.3 If $j = 1$ then $\tau \in (Q \times \cup_{a\in\Gamma, k\in[1,2]} \mathbf{Push}_k(a) \times Q) \cap \Delta$

Given any finite computation of an MPDS M, it is said to be adjacent ordered computation if every one step computation in it is adjacent ordered i.e. $\pi = c_0 \xrightarrow{\tau_1} c_1 \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_n} c_n$ is said to be adjacent ordered if for all $i \in [1..n-1]$, $c_i \xrightarrow{\tau_{i+1}} c_{i+1}$ is adjacent ordered. The definition can be extended to the infinite case in straight forward manner.

Adjacent ordered reachability problem asks whether a given configuration can be reached from the initial configuration through an adjacent ordered execution. We show below that given an MPDS M, we can construct in polynomial time an AOMPDS $\mathscr{A}$ such that the adjacent ordered reachability on M can be reduced to reachability on $\mathscr{A}$. For this purpose, we will first fix our MPDS to be $M = (n, Q, \Gamma, \Delta, q_0, \gamma_0)$. The required AOMPDS is given by $\mathscr{A} = (n, Q_{\mathscr{A}}, \Gamma_{\mathscr{A}}, \Delta_{\mathscr{A}}, s, \bar{\perp})$, where

- $Q_{\mathscr{A}} = Q \cup \{s\}$ is the finite set of states.
- $\Gamma_{\mathscr{A}} = \Gamma \cup \overrightarrow{\Gamma} \cup \overleftarrow{\Gamma} \uplus \{\bar{\perp}\}$ is the stack alphabet, where $\bar{\perp}$ is a new symbol , $\overrightarrow{\Gamma} = \{\overrightarrow{a} \mid a \in \Gamma\}$ and $\overleftarrow{\Gamma} = \{\overleftarrow{a} \mid a \in \Gamma\}$. The symbol $\bar{\perp}$ will be pushed onto the last stack (as an initial symbol) and will never be popped. The stack alphabet $\overrightarrow{\Gamma}$ and $\overleftarrow{\Gamma}$ will be used to move symbols on to right and left stacks adjacent to the least nonempty stack.
- s is the new initial state and will be used to push γ_0, the intial stack symbol of M and enter the state q_0, the initial state of M.
- The transition relation $\Delta_{\mathscr{A}}$ is described below.

a.1 From the start state s, there is a transition to push γ_0 and move to q_0 i.e. we have $((s, \perp^{n-1}, \bar{\perp}), (q_0, \perp^{n-1}, \gamma_0\bar{\perp})) \in \Delta_{\mathscr{A}}$

a.2 The following transitions are used to simulate the transitions in 1 from definition 10

1. For all $i \in [1..n]$ and $(q, \mathbf{Pop}_i(a), q') \in \Delta$ we add $((q, \perp^{i-1}, a, \epsilon^{n-i}), (q', \perp^{i-1}, \epsilon^{n-i+1})) \in \Delta_{\mathscr{A}}$
2. For all $i \in [1..n]$ and $(q, \mathbf{Int}_i, q') \in \Delta$ we add for all $a \in \Gamma \cup \bar{\perp}$, the transitions $((q, \perp^{i-1}, a, \epsilon^{n-i}), (q', \perp^{i-1}, a, \epsilon^{n-i})) \in \Delta_{\mathscr{A}}$

a.3 For all $i \in [1..n]$ and $(q, \mathbf{Push}_i(b), q') \in \Delta$, the following transitions are added to simulate the transitions in 2 from definition 10

1. For all $a \in \Gamma \cup \bar{\perp}$, the transitions $((q, \perp^{i-1}, a, \epsilon^{n-i}), (q', \perp^{i-1}, ba, \epsilon^{n-i})) \in \Delta_{\mathscr{A}}$. These transitions allows pushes on least nonempty stack.
2. If $i < n$, then for all $a \in \Gamma \cup \bar{\perp}$, the transitions $((q, \perp^{i}, a, \epsilon^{n-i-1}), (q', \perp^{i}, \overleftarrow{b}a, \epsilon^{n-i-1})) \in \Delta_{\mathscr{A}}$. These transitions pushes a symbol tagged with direction that will be transferred to lower numbered stack, using transitions from a.4.
3. If $i > 1$ then, for all $a \in \Gamma \cup \bar{\perp}$, the transitions $((q, \perp^{i-2}, a, \epsilon^{n-i+1}), (q', \perp^{i-2}, \overrightarrow{b}a, \epsilon^{n-i+1})) \in \Delta_{\mathscr{A}}$. These transitions pushes a symbol tagged with direction that will be transferred to higher numbered stack, using transitions from a.4.

a.4 We also add the following transitions to transfer the right and left symbols to its respective stack.

1. For all $i < n$, and all $\overrightarrow{a} \in \overrightarrow{\Gamma}$, we add $((q, \perp^{i-1}, \overrightarrow{a}, \epsilon^{n-i}), (q, \perp^{i-1}, \epsilon, a, \epsilon^{n-i-1})$
2. For all $i > 1$, and all $\overleftarrow{a} \in \overleftarrow{\Gamma}$, we add $((q, \perp^{i-1}, \overleftarrow{a}, \epsilon^{n-i}), (q, \perp^{i-2}, a, \epsilon^{n-i+1})$

We will now prove the following Lemma, which states that our construction preserves

reachability.

Lemma 51. *For any configuration $c = (q, \gamma_1\bot, \cdots \gamma_n\bot) \in \mathscr{C}(M)$, we have an adjacent ordered computation $c^{init}_M \rightarrow^*_M c$ iff $c^{init}_{\mathscr{A}} \rightarrow^*_{\mathscr{A}} d$, where $d = (q, \gamma_1\bot, \cdots \gamma_n\bar{\bot}\bot) \in \mathscr{C}(M)$.*

Proof. ($\Rightarrow$) We prove this direction by inducting on the length of the run. The base case is a zero length run. In this case, we use the transition a.1 to get the required run in $\mathscr{A}$.

For the inductive case, we assume a run $c^{\text{init}}_M \rightarrow^*_M c$ of length greater than one. Clearly such a run can be split as $c^{\text{init}}_M \rightarrow^*_M c' \xrightarrow{\tau} c$. Let $j = \mathbf{Act}(c')$ and $c' = (q', \gamma'_1\bot, \cdots \gamma'_n\bot)$. Inductively there is a run of the form $c^{\text{init}}_{\mathscr{A}} \rightarrow^*_{\mathscr{A}} d'$ where $d' = (q', \gamma'_1\bot, \cdots \gamma'_n\bar{\bot}\bot)$. We show how to extend such a run for each possible choice of τ that is adjacent ordered.

- Suppose τ was of the form $\tau = (q', \mathbf{Pop}_j(a), q)$ then we use the transition 1. in a.2 to extend the run.
- Suppose τ was of the form $\tau = (q', \mathbf{Int}_j, q)$ then we use the transition 2. in a.2 to extend the run.
- Suppose τ was of the form $\tau = (q', \mathbf{Push}_j(a), q)$ then we use the transition 1. in a.3 to extend the run.
- Suppose τ was of the form $\tau = (q', \mathbf{Push}_{j-1}(a), q)$ then we use the transition 2. in a.3 followed by transition in a.4 to extend the run.
- Suppose τ was of the form $\tau = (q', \mathbf{Push}_{j+1}(a), q)$ then we use the transition 3. in a.3 followed by transition in a.4 to extend the run.

($\Leftarrow$)

Firstly let $S = \{(q, \gamma_1\bot, \cdots, \gamma_n\bar{\bot}\bot) | (q, \gamma_1\bot, \cdots, \gamma_n\bot) \in \mathscr{C}(M)\}$. For this direction, we induct on the number of times a configuration from S is seen in the run. For the base case, suppose the number of times a configuration from S is seen is 1, clearly such a run is of the form $c^{\text{init}}_{\mathscr{A}} \rightarrow^*_{\mathscr{A}} (q_0, \bot^{n-1}, \gamma_0\bar{\bot}\bot)$. But note that $(q_0, \bot^{n-1}, \gamma_0\bot) = c^{\text{init}}_M$. From this we get the required run in M.

For induction case let us assume a run $c^{\text{init}}_{\mathscr{A}} \rightarrow^*_{\mathscr{A}} d$ such that $d \in S$ and the number of times the configuration from S is seen is greater than one. Then such a run can be split as $c^{\text{init}}_{\mathscr{A}} \rightarrow^*_{\mathscr{A}} d' \rightarrow^*_{\mathscr{A}} d$, where in $\pi' = d' \rightarrow^*_{\mathscr{A}} d$, there are no intermediate configurations from S. Let $d' = (q', \gamma'_1\bot, \cdots, \gamma'_n\bar{\bot}\bot)$, then from induction hypothesis, we have a run of the form $c^{\text{init}}_M \rightarrow^*_M c'$, where $c' = (q', \gamma'_1\bot, \cdots, \gamma'_n\bot)$. Let $\mathbf{Act}(c') = i$, then $\gamma'_1, \cdots, \gamma'_{i-1} = \epsilon$. By the definition of the transitions of A, the run π' is of length at most 2. We consider various possible choices of π' and show that in each case, we can correspondingly extend the run of M.

- Suppose that the transition used in π' was $((q', \bot^{i-1}, a, \epsilon^{n-i}, (q, \bot^{i-1}, \epsilon^{n-i-1})$, from 1. in a.2. Then clearly by construction, we know that there is a transition of the form $(q', \mathbf{Pop}_i(a), q) \in \Delta$. Further it is clear that firing such a transition from c' confirms to the adjacent ordered restriction. Using this we can extend the run in M to get the required run.
- The case where the transition used in π' was $((q', \bot^{i-1}, a, \epsilon^{n-i}, (q, \bot^{i-1}, a, \epsilon^{n-i})$ from 2. in a.2, is similar.
- We consider the case where the transitions used was from 2. from a.3, of the form $((q', \bot^i, a, \epsilon^{n-i-1}, (q, \bot^i, \overleftarrow{b}a, \epsilon^{n-i-1})$, followed by the transition from a.4 of the form $((q, \bot^i, \overleftarrow{b}, \epsilon^{n-i-1})$, $(q, \bot^{i-1}, b, \epsilon^{n-i}))$. Then clearly by construction, we have $(q', \mathbf{Push}_{i-1}(b), q) \in \Delta$. Further it

is clear that firing such a transition from c' confirms to the adjacent ordered restriction (since $\mathbf{Act}(c') = i$). Using this we can extend the run in M to get the required run.

– Rest of the cases are similar and easy and hence we omit the same.

□

From the above construction and the Lemma 51, the following Theorem is immediate.

Theorem 21. *Given an MPDS $M = (n, Q, \Gamma, \Delta, q_0, \gamma_0)$, we can construct an AOMPDS $\mathscr{A}$ such that for any configuration $c \in \mathscr{C}(M)$, there is an adjacent ordered computation $\pi = c^{init}_{M} \rightarrow^*_M c$ iff there is a computation $\pi' = c^{init}_{\mathscr{A}} \rightarrow^*_{\mathscr{A}} c$.*

7.6 Conclusion

In this chapter, we introduced a new restriction called adjacent ordered restriction on the executions of the MPDS. We went on to show that reachability under such a restriction is EXPTIME COMPLETE. We also showed how to solve the repeated reachability problem and hence also the problem of model checking LTL formulas over runs of the MPDs with such a restriction. We went on to show that the model introduced has many applications. Towards this, we showed how to reduce the reachability on a recursive queueing concurrent program to reachability on an AOMPDS. We also showed how to get an alternative algorithm for deciding the bounded-phase reachability through reachability on AOMPDS. Since the model of AOMPDS was formalised differently, we also proposed a restriction called the adjacent ordered restriction on the runs of the MPDS model and showed that reachability under such a restriction can be captured by the AOMPDS model.

Chapter 8

Accelerations on multi-pushdown systems

8.1 Introduction

In this chapter we will present a new kind of under-approximation technique by means of accelerating loops. The idea of accelerating loops is similar in spirit to *global model checking problem*. In global model checking problem, the aim is to compute from (a representation of) initial set of configuration I, (a representation of) the set of all reachable configurations from I (denoted $Post^*(I)$). There are many useful applications of global model checking, the most obvious one being reachability. Note that our description of global model-checking does not require that the representations of the initial set I and the reachable set $post^*(I)$ be the same. For eg. for PDSs, whether we use finite sets or regular sets for the initial set of configurations, the final set can be described effectively as a regular set. However, if both sets use the same description, then we say that the representation is stable. Stability is an useful property as it permits us to compose (and hence iterate finitely) the algorithm.

For PDSs if the initial set of configurations is a regular language then the set of reachable configurations is a computable regular language ([40, 71]). The model we are considering here is a multi-pushdown system, which we already know is Turing powerful. Hence the only hope is to achieve this by some under-approximation technique. The configuration of a MPDS can be represented as a tuple of words giving the current state and the contents of the each of the stacks. We can then represent sets of configurations by *recognizable* or *regular* languages [34]. Given a recognizable language representing the set of initial configurations, the set of configurations that may be reached via runs with at most k-context switches is also a (computable) recognizable language [124]. Thus, the global model checking problem under bounded context setting is decidable and this has many applications, including the obvious one — reachability can be decided.

Another well known technique used in the verification of infinite state systems is that of loop accelerations. It is similar in spirit to global model checking but with different applications. The idea is to consider a loop of transitions (a finite sequence of transitions that lead from a control state back to the same control state). The aim is to determine the effect of it-

erating the loop. That is, to effectively construct a representation of the set of configurations that may be reached by valid iterations of the loop. Loop accelerations turn to be a very useful (e.g., [43, 31, 39, 13, 33, 89, 38, 37, 69, 32, 109, 110, 88, 68]) in the analysis of a variety of infinite state systems.

We propose to use accelerations as an under-approximation technique in the verification of MPDSs. We take this further by proposing a technique that composes the iterations of such loops with context bounded runs to obtain a new decidable under-approximation for MPDSs. Observe that there is no bound on the number of context switches under loop iterations while a context bounded run permits unrestricted recursive behaviours, not permitted by loop iterations, thus complementing each other.

We begin by showing that both regular sets as well as rational sets of configurations are stable w.r.t. bounded context executions. Next we show that this does not extend to iterations of loops. We show that under iterations of a loop, the $post^*$ of a regular set of transitions is always rational while that of a rational set need not be rational. We then address the question of a representation that is stable w.r.t. loop accelerations. Towards this we propose a new representation for configurations called n-CSRE inspired by the CQDDs [43] and the class of bounded semilinear languages [49]. This forms a very expressive class, for eg. the 1-dimensional version is equivalent to the class of semilinear bounded languages (see [49]). We show that n-CSREs are closed under union, intersection and concatenation. Furthermore, we have the decidability of the emptiness, membership problem as well as the inclusion problem for n-CSREs. Then, we show that n-CSREs are indeed stable w.r.t iteration of loops. This result also has the pleasant feature that the construction is in polynomial time. However, n-CSREs are not stable w.r.t bounded context executions.

As a final step we introduce a joint generalization of both loop iterations and bounded context executions called bounded context-switch sets. We show that the class of languages defined by n-dimensional constrained automata (the most general class considered here and a n-dimensional version of Parikh automata) is stable w.r.t accelerations via bounded context-switch sets. Since membership is decidable for this class, we obtain a decidability of reachability under this generous class of behaviours. Observe that the class of n-dimensional constrained automata is not closed under intersection and that the inclusion problem is undecidable.

8.2 Acceleration

Given a set of configurations $C \subseteq \mathscr{C}(M)$ and a set of sequences of transitions $\Theta \subseteq \Delta^*$, the *acceleration problem* for M, with respect to C and Θ, consists in computing the set of configurations c' such that $c \xrightarrow{\sigma}_M c'$ where $c \in C$ and $\sigma \in \Theta^*$. We use $Post_{\Theta^*}(C)$ to denote the set $\{c' \mid c \xrightarrow{\sigma}_M c', c \in C, \sigma \in \Theta^*\}$. Observe that the global reachability problem is the acceleration problem with the set of initial configurations as C and $\Theta = \Delta$. We first show that global model checking under context bounded restriction can be seen as acceleration problem. Before that, we will first recall some properties of rational and regular languages (see, e.g., [34]).

8.2.1 Properties of rational languages

Recall that a n-dim language is *rational* if it is the language of some n-tape automaton [34]. A n-dim language L is *regular* if it is a finite union of products of n rational 1-dim languages. First, the class of recognisable languages, for any dimension $n \geq 1$, is closed under boolean operations. On the other hand, for every $n \geq 2$, the class of n-dim rational languages is closed under union and concatenation but not under complementation or under intersection. However, the emptiness and membership problems for rational languages are decidable and further the inclusion problem is also decidable for recognisable languages. The inclusion problem is undecidable for rational languages.

We describe some additional closure properties of rational languages that are well known and will prove useful. Rational languages are effectively closed under the permutation of indices: Let A be a n-tape automaton over $\Sigma_1, \ldots, \Sigma_n$. Given a mapping $h : [1..n] \to [1..n]$, it is possible to construct a n-tape automaton $h(A)$, linear in the size of A, such that $(w_1, ..., w_n) \in L(A)$ iff $(w_{h(1)}, \ldots, w_{h(n)}) \in L(h(A))$. Rational languages are also effectively closed under projection: Given a set of indices $\iota = \{i_1 < i_2 < \ldots i_m\} \subset [1..n]$, we can construct an automaton $\Pi_\iota(A)$, linear in size of A, such that $L(\Pi_\iota(A)) = \{(w_{i_1}, w_{i_2}, \ldots, w_{i_m}) \mid (w_1, w_2, \ldots, w_n) \in L(A)\}$. Rational languages are also closed under *composition* operation : Let A be as before and let A' be a rational language over $\Sigma'_1, \Sigma'_2, \ldots, \Sigma'_m$. Let $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$ be two indices s.t. $\Sigma'_j = \Sigma_i$. Then, it is possible to construct a $(n+m-1)$-tape automaton $A \circ_{(i,j)} A'$, whose size is $O(|A|.|A'|)$, accepting $(w_1, \ldots, w_n, w'_1, \ldots, w'_{j-1}, w'_{j+1}, \ldots, w'_m)$ iff $(w_1, \ldots, w_n) \in L(A)$ and $(w'_1, \ldots, w'_{j-1}, w_i, w'_{j+1}, \ldots, w'_m) \in L(A')$, i.e. the composition corresponding to the synchronization of the i^{th} tape of A with the j^{th} tape of A'.

8.2.2 Context-Bounding as an acceleration problem

In the following, we show that context-bounding analysis [124, 119, 106, 95] for an MPDS $M = (n, Q, \Gamma, \Delta, q_0)$ can be formulated as an acceleration problem w.r.t. the class of rational/regular configurations. Given two configuration $c, c' \in \mathscr{C}(M)$ and $k \in \mathbb{N}$, the k-context reachability problem consists in checking whether there is a sequence of transitions $\sigma \in \Delta_{i_1}^* \Delta_{i_2}^* \cdots \Delta_{i_k}^*$, with $i_1, i_2, \ldots, i_k \in [1..n]$, such that $c \xrightarrow{\sigma}{}^*{}_M c'$. The decidability of the k-context reachability problem can be seen as an immediate corollary of the decidability of the membership problem for rational languages and the following result:

Theorem 22. *Let $i \in [1..n]$. For every regular (rational) set of configurations C, the set $Post_{\Delta_i^*}(C)$ is regular (rational) and effectively constructible.*

The set $Post_{\Delta_i^*}(C)$ has been shown to be regular and effectively constructible when C is regular in [124]. In the following, we prove Theorem 22 for the case when C is rational. We write M_i for the PDS $(Q, \Gamma, \Delta_i, q_0)$ simulating the behavior of M only on the stack i. First we recall a result established in [51, 107].

Lemma 52. *It is possible to construct, in polynomial time in the size of M_i, a 4-tape finite state automaton $T(i)$, over Q, Γ, Q, Γ, such that $(q, u, q', v) \in L(T(i))$ iff $(q, u) \xrightarrow{\sigma}{}^*{}_{M_i} (q', v)$ for some sequence $\sigma \in \Delta_i^*$.*

Proof. Before going into the proof,we will recall the definition of a γ run that we introduced earlier. We say $\pi = (q,\alpha) \rightarrow^*_\gamma (q',\beta)$ iff γ is the longest prefix of stack in all the configuration that occurs in π.

Let $\mathscr{U} = \{(q,a,q') \mid (q,a) \xrightarrow{\sigma}^*_{M_i} (q',\epsilon)\,,\, a \in \Gamma\,,\, \sigma \in \Delta_i^*\}$ be set such that if $(q,a,q') \in \mathscr{U}$ then there is a run that starts at q, consumes a from stack and reaches q' without involving a zero test, similarly let $\mathscr{V} = \{(q,a,q') \mid (q,\epsilon) \xrightarrow{\sigma}^*_{M_i} (q',a)\,,\, a \in \Gamma\,,\, \sigma \in \Delta_i^*\}$ and let $\mathscr{W} = \{(q,q') \mid (q, \perp) \xrightarrow{\sigma}^*_{M_i} (q',\perp), \sigma \in \Delta_i^*\}$. Observe the set $\mathscr{U}$, $\mathscr{V}$ and $\mathscr{W}$ are effectively constructible in polynomial time by reduction to the reachability problem for pushdown systems.

Clearly if $(q,u) \rightarrow^*_{M_i} (q',v)$ then there is a configuration $(q'',w), q'' \in Q, w \in (\Gamma \setminus \perp)^* \perp$ such that $(q,u) \rightarrow^*_w (q''_1,w) \rightarrow^*_w (q''_2,w) \rightarrow^*_w (q',v)$ i.e. any run can be split into decreasing part and an increasing part and a possible zero test part in between. This also means that, $u = u'w$ and $v = v'w$ (note that w can just be $\perp$). Hence it is easy to see that $L(T(i)) = \{(q,uw,q',vw) \mid (q,u) \rightarrow (q'',\epsilon) \wedge (q'',\epsilon) \rightarrow (q',v) \wedge w \in \Gamma^+\perp\} \cup \{(q,u\perp,q',v\perp) \mid (q,u\perp) \rightarrow^* (q''_1,\perp) \wedge (q''_2,\perp) \rightarrow^* (q', v\perp) \wedge (q''_1,\perp) \rightarrow^* (q''_2,\perp)\}$.

We will now give the construction of $T(i)$. The states of $T(i)$ are $Q_{T(i)} = \{s\} \cup (Q \times (Q \cup \{\mathbf{0}\})) \cup \{e,f\}$. The initial state is s and the final state is f. The states of $T(i)$ contains two component, the first component is used to simulate the decreasing phase and the second component to simulate the increasing phase. From the initial state on reading state from the first tape, updates it in the first component of the state i.e. we add for all $q \in Q$, $(s,(q,\epsilon,\epsilon,\epsilon), (q,\mathbf{0})) \in \delta_{T(i)}$. Subsequently it starts simulating moves of decreasing phase, i.e. we add $((q,\mathbf{0}), (\epsilon,a,\epsilon,\epsilon),(q',\mathbf{0})) \in \delta_{T(i)}$ if $(q,a,q') \in \mathscr{U}$. It is easy to see that $(q,\mathbf{0}) \xrightarrow{(\epsilon,u,\epsilon,\epsilon)}_{T(i)} (q',\mathbf{0})$ is a run in $T(i)$ iff $(q,u) \rightarrow_{M_i} (q',\epsilon)$.

We add transition that guesses the intermediate point from where the automata starts simulating the increasing phase from there onwards. To this effect, we add for all $q' \in Q$, $((q,\mathbf{0}),(\epsilon,\epsilon,q',\epsilon),(q,q')) \in \delta_{T(i)}$. To simulate the increasing phase, we add for all $(q',a,q'') \in \mathscr{V}$, transition $((q,q''),(\epsilon,\epsilon,\epsilon,a),(q.q')) \in \delta_{T(i)}$. As in previous case, it is easy to see that $(q, q') \xrightarrow{(\epsilon,\epsilon,\epsilon,v)}_{T(i)} (q,q'')$ iff $(q'',\epsilon) \rightarrow_{M_i} (q',v)$. Finally we add for all $q \in Q$, $((q,q),(\epsilon,\epsilon,\epsilon,\epsilon),e) \in \delta_{T(i)}$ and for all $a \in \Gamma \setminus \{\perp\}$, $(e,(\epsilon,a,\epsilon,a),e) \in \delta_{T(i)}$ and $(e,(\epsilon,\perp,\epsilon,\perp),f) \in \delta_{T(i)}$, these transitions guesses the intermediate point (for the case where there is no zero test) and checks if the word below is same in even stacks.

We also add for all $(q,q') \in \mathscr{W}$, the transition $((q,q'),(\epsilon,\perp,\epsilon,\perp),f)$ (for the case where there is zero test). It is easy to see that if $((q,q') \xrightarrow{(\epsilon,\perp,\epsilon,\perp)} f$ then we have $(q,\perp) \rightarrow^*_{M_i} (q',\perp)$.

The correctness of the construction follows from the fact that, there is an accepting run form s, iff it one of the following forms.

$$s \overset{(q,\epsilon,\epsilon,\epsilon)}{\rightarrow} (q,\mathbf{0}) \xrightarrow{(\epsilon,u,\epsilon,\epsilon)}{}^* (q'',\mathbf{0}) \overset{(\epsilon,\epsilon,q',\epsilon)}{\rightarrow} (q'',q') \xrightarrow{(\epsilon,\epsilon,\epsilon,v)}{}^* (q'',q'') \rightarrow e \xrightarrow{(\epsilon,w\perp,\epsilon,w\perp)}{}^* f$$

or

$$s \rightarrow (q,\mathbf{0}) \xrightarrow{(\epsilon,u,\epsilon,\epsilon)}{}^* (q''_1,\mathbf{0}) \xrightarrow{(\epsilon,\epsilon,q',\epsilon)}{}^* (q''_1,q') \xrightarrow{(\epsilon,\epsilon,\epsilon,v)}{}^* (q''_1,q''_2) \xrightarrow{(\epsilon,\perp,\epsilon,\perp)} f$$

It is easy to see that the former case is true iff $(q,uw\perp) \rightarrow^*_{M_i} (q'',w\perp) \rightarrow^*_{M_i} (q',vw\perp)$ and the latter case is true iff $(q,u\perp) \rightarrow^*_{M_i} (q''_1,\perp) \rightarrow^*_{M_i} (q''_2,\perp) \rightarrow^*_{M_i} (q',v\perp)$.

□

Observe that Lemma 52 relates any possible starting configuration (q,u) with any configuration (q',v) reachable from (q,u) in M_i. Let us assume now that we are given a $(n+1)$-tape automaton $A=(P,Q,\Gamma,\ldots,\Gamma,\delta,p_0,F)$ accepting the set C. In the following, we show how to compute a $(n+1)$-tape finite state automaton A' accepting the set $Post_{\Delta_i^*}(C)$. To do that, we proceed as follows: We first compose A with $T(i)$, synchronsing the second tape of $T(i)$ (containing the stack contents at the starting configuration) with the $(i+1)$-th tape of A, to construct a $(n+4)$-tape automaton $A_1 = A \circ_{(i+1,2)} T(i)$. We also need to synchronize the starting states (i.e. the first tape of A with the first tape of $T(i)$). This can be done by intersecting A_1 with the (regular) language $\bigcup_{q\in Q}\{q\}\times(\Gamma^*)^n\times\{q\}\times Q\times\Gamma^*$. Let A_2 be the automaton resulting from the intersection operation. Then, we project away the starting control state (occurring on tapes 1 and $n+2$) and the content of the $i+1$-th tape to we obtain the $(n+1)$-tape automaton $A_3=\Pi_\iota(A_2)$ where $\iota=([1..n]\setminus\{1,i+1,n+2\})$. This is almost what is needed except that the new content of the stack i occurs at the last position instead of position $i+1$ and the control state occurs at penultimate position instead of the first position. We rearrange this using the permutation operation. We let $A'=h(A_3)$ where h is defined as follows: (1) $h(1)=n$, (2) $h(j)=j-1$ for all $j\le i$, (3) $h(i+1)=n+1$, and (4) $h(j)=j-2$ for all $j>i$.

Observe that the size of A' is polynomial in $|A|$. As an immediate consequence of this result and the fact that the membership problem for rational/regular languages can be checked in polynomial time, we can deduce that the k-context reachability problem can be decided in polynomial in the size of M and exponential in k (as in [124]).

8.2.3 Accelerating Loops: Case of regular/rational sets

In this section, we address the acceleration problem for the iterative execution of a sequence of transitions in the control graph of a MPDS $M=(n,Q,\Gamma,\Delta,s)$. More precisely, given a sequence of transitions $\theta\in\Delta^*$ and a set of configurations $C\subseteq\mathscr{C}(M)$, we are interested in characterising the set $Post_{\theta^*}(C)$. In sequel, when we consider a sequence of transitions, we will assume that there are no zero test (unless mentioned other wise). We will also assume all the sequences we consider are state wise compatible. By statewise compatible, we mean, given any sequence $(q_1,\mathrm{op}_1,q_1')(q_2,\mathrm{op}_1,q_2')\cdots(q_m,op_m,q_m')$, we have for all $i\in[1..m]$, $q_{i+1}=q_i'$. Note that in case we are accelerating loops and if there are zero tests, only finitely many configurations can be reached. In fact if we can successfully iterating such a loop arbitrary number of times, we will end up in the same configuration each time.

Computing the effect of a sequence of transitions

Let $M=(n,Q,\Gamma,\Delta)$ be an MPDS and $\sigma\in\Delta^*$ a sequence of transitions of the form $(q_0,\mathrm{op}_0,q_1)(q_1,\mathrm{op}_1,q_2)\cdots(q_{m-1},\mathrm{op}_{m-1},q_m)$. Intuitively, we associate to each stack i a pair (u_i,v_i) such that the effect of executing the sequence σ on stack i is popping the word u_i and then pushing the word v_i on to it (i.e. the stack content is transformed from u_iw to v_iw for some w). To this end, for every $i\in[1..n]$, we introduce a partial function $\mathtt{Eff}_i:(\Gamma^*\times\Gamma^*\times\Delta^*)\rightharpoonup(\Gamma^*\times\Gamma^*)$ (we will let the function map to $\perp$ when it is not defined). Roughly speaking, assuming that we have already computed the effect of a transition sequence σ on stack i to be (u,v), i.e. to pop u and push v, $\mathtt{Eff}_i(u,v,t)$ computes the effect of $\sigma.t$ on stack i. In the below

definition, given any transition of the form $\tau = (q, \mathrm{op}, a, q')$, we let $Op(\tau) = \mathrm{op}$. Given $u, v \in \Gamma^*$ and $t \in \Delta$, we define $\mathtt{Eff}_i((u,v),t)$ as follows:

- if $Op(t) = \mathbf{Pop}_i(a)$ for some $a \in \Gamma$ then
  - $\mathtt{Eff}_i((u,\epsilon),t) = (u \cdot a, \epsilon)$,
  - If $v = a \cdot v'$ for some $v' \in \Gamma^*$ then $\mathtt{Eff}_i((u,v),t) = (u,v')$,
  - Otherwise $\mathtt{Eff}_i((u,v),t) = \bot$.
- if $Op(t) = \mathbf{Push}_i(a)$ for some $a \in \Gamma$, then $\mathtt{Eff}_i((u,v),t) = (u, a \cdot v)$
- If $Op(t) = \mathbf{Int}_i$ or $t \in \Delta \setminus \Delta_i$, then $\mathtt{Eff}_i((u,v),t) = (u,v)$.

We extend the definition of $\mathtt{Eff}_i$ to sequence of transitions as expected: For every two words $u, v \in \Gamma^*$, we have

1. $\mathtt{Eff}_i((u,v),\epsilon) = (u,v)$,
2. For every $\sigma' \in \Delta^*$ and $t \in \Delta$, we have $\mathtt{Eff}_i((u,v),\sigma' \cdot t) = \mathtt{Eff}_i(\mathtt{Eff}_i((u,v),\sigma'),t)$ if $\mathtt{Eff}_i((u,v),\sigma') \neq \bot$ is defined, and $\mathtt{Eff}_i((u,v),\sigma' \cdot t) = \bot$ otherwise.

Our aim is to compute the complete effect of some sequence σ on stack i and this is given by $\mathtt{Eff}_i((\epsilon,\epsilon),\sigma)$. We shall refer to this as $\mathtt{Summ}(i,\sigma)$. The next lemma formalizes our intuition about $\mathtt{Summ}$ and characterizes precisely when a sequence of transitions σ may be executed and computes its effect on all the stacks (if it is executable).

Lemma 53. *Let $c = (p, w_1, \ldots, w_n)$ and $c' = (p', w'_1, \ldots, w'_n)$ be two configurations of M. $c \xrightarrow{\sigma}{}^* c'$ such that $\sigma \in \Delta^*$ iff for every $i \in [1..n]$, we have $w_i = u_i u'_i$ and $w'_i = v_i u'_i$ for some $u_i, v_i, u'_i \in \Gamma^*$ such that $\mathtt{Summ}(i,\sigma) = (u_i, v_i)$.*

Proof. ($\Rightarrow$)
We will prove this lemma by induction on the length of the run.
Base case $|\sigma| = 0$: For the base case we consider the zero length run. $\mathtt{Summ}(i,\epsilon)$ is defined as $\mathtt{Eff}((\epsilon,\epsilon),\epsilon) = (\epsilon,\epsilon)$. Hence holds trivially.
Length greater than 0, case $\sigma.\tau$:
Let $c \xrightarrow{\sigma} c'' \xrightarrow{\tau} c'$ with $c = (p, w_1, \ldots, w_n)$, $c'' = (p'', w''_1, \cdots, w''_n)$ and $c' = (p', w'_1, \ldots, w'_n)$. By induction, we have $\mathtt{Summ}(i,\sigma) = (u_i, v''_i)$, with $w_i = u_i u'_i$ and $w''_i = v''_i u'_i$. Firstly note that, for all $j \neq i$ we have $w''_j = w'_j$ and $\mathtt{Summ}(j,\sigma\tau) = (u_j, v''_j)$. Hence it is enough to only relate w'_i and w''_i.

- Case where $Op(\tau = \mathbf{Int}_i)$ is easy since, by definition $\mathtt{Summ}(i,\sigma\tau) = (u_i, v''_i)$ and we have by nature of τ, $w''_i = w'_i$.
- Case where $Op(\tau) = \mathbf{Push}_i(a)$, by definition of $\mathtt{Summ}$, we have that $\mathtt{Summ}(i,\sigma.\tau) = (u_i, a.v''_i)$ and by nature of τ, we have $w'_i = a.w''_i$.
- Case where $Op(\tau) = \mathbf{Pop}_i(a)$, firstly by nature of τ, we have $a.w'_i = w''_i$. We have two cases to consider
  1) Where $v''_i = \epsilon$ in this case, note that $w''_i = u'_i = a u''_i$ and $w'_i = u''_i$. From this, we also have $w_i = u_i.u'_i = u_i.a.u''_i$. Also by definition we get $\mathtt{Summ}(i,\sigma.\tau) = (u_i a, \epsilon)$.
  2) Where $v_i \neq \epsilon$, in this case we have $a w'_i = w''_i = v''_i u'_i$, hence v''_i is of the form $v''_i = a v_i$ and $w'_i = v_i u_i$. Further by definition, we have that $\mathtt{Summ}(i,\sigma\tau) = (u_i, v_i)$.

($\Leftarrow$)

For all $i \in [1..n]$, let $\texttt{Summ}(i, \sigma\tau) = (u_i, v_i)$ and let $\texttt{Summ}(i, \sigma) = (u_i, v_i'')$, we will assume by induction, the existence of a run $c \xrightarrow{\sigma}{}^* c''$ such that $c = (p, w_1, \ldots, w_n)$, $c'' = (p'', w_1'', \cdots, w_n'')$ such that for all $i \in [1..n]$, we have $w_i = u_i u_i'$ and $w_i'' = v_i'' u_i'$. Further by definition, we have for all $j \neq i$, $\texttt{Summ}(j, \sigma\tau) = (u_j, v_j'')$. Now consider $\sigma\tau$. By assumption, we are given $\sigma\tau$ that is state wise compatible.

- Case where $\tau = (p'', \mathbf{Int}_i, p') \in \Delta_i$ is straight forward. From the definition of transition relation, we have $c \rightarrow^* c'' \rightarrow (p', w_1'', \cdots, w_n'')$. We have by definition, $\texttt{Summ}(i, \sigma\tau) = (u_i, v_i'')$. Hence the result follows.
- Case where $\tau = (p'', \mathbf{Push}_i(a), p')$ then $\texttt{Summ}(i, \sigma\tau) = (u_i, a v_i'')$, hence $v_i = a v_i''$. Clearly from definition of transition relation $c'' \xrightarrow{\tau} c'$ where $c' = (p', w_1'', \cdots, a w_i'', w_{i+1}'', \cdots, w_n'')$. Note that by induction, we have $w_i'' = v_i'' u_i'$, from this it is easy to see that $w_i' = a w_i'' = a v_i'' u_i'$.
- Case where $\tau = (p'', \mathbf{Pop}_i(a), p')$ and $v_i = \epsilon$ then $\texttt{Summ}(i, \sigma\tau) = (u_i.a, \epsilon)$. We have by induction hypothesis, $w_i = u_i a u_i'$ and $w_i'' = a u_i'$. Since top of stack is a, we have $c_i'' \xrightarrow{\tau} c_i'$ where $c' = (p', w_1'', \cdots, u_i', \cdots, w_n'')$.
- Case where $Op(\tau) = \mathbf{Pop}_i(a)$ and $v_i \neq \epsilon$ then , clearly $v_i = a v_i''$ and hence $w_i'' = a v_i u_i'$. Now since we have an a on top of stack, we have $c'' \rightarrow c'$, with $w_i = v_i u_i'$.

□

Now, we will characterize $\texttt{Summ}(i, \sigma^j)$ with $j \geq 1$, i.e., the effect of iterating the sequence σ j-times, in terms of $\texttt{Summ}(i, \sigma)$ for all $i \in [1..n]$. Observe that if $\texttt{Summ}(i, \sigma) = \bot$, then $\texttt{Summ}(i, \sigma^j) = \bot$ for all $j \geq 1$. Hence, let us assume that $\texttt{Summ}(i, \sigma) = (u_i, v_i)$ for some words $u_i, v_i \in \Gamma^*$. First, let us consider the case when the sequence σ can be iterated twice and compute its effect on all the stacks. Now, using the definition of $\texttt{Summ}$ it is not difficult to conclude that $\texttt{Summ}(i, \sigma\sigma)$ is defined iff either v_i is a prefix of u_i or u_i is a prefix of v_i. We can in fact say more. If the former holds we let x_i be the unique word such that $u_i = v_i x_i$ and $y_i = \epsilon$. In case of the latter we let y_i be the unique word such that $v_i = u_i y_i$ and $x_i = \epsilon$. Then, we have $\texttt{Summ}(i, \sigma\sigma) = (u_i', v_i')$ for all $i \in [1..n]$ where $u_i' = u_i x_i$ and $v_i' = v_i y_i$. We define a partial function $\texttt{Iter} : ([1..n] \times \Delta^*) \rightharpoonup (\Gamma^* \times \Gamma^*)$ such that $\texttt{Iter}(i, \sigma)$ is the pair (x_i, y_i) as defined above when $\texttt{Summ}(i, \sigma\sigma)$ is defined, and $\texttt{Iter}(i, \sigma) = \bot$ otherwise. We can now generalize this computation of $\texttt{Summ}$ to any number of iterations of σ as shown below.

Lemma 54. *Let $i \in [1..n]$. If $\texttt{Summ}(i, \sigma\sigma)$ is well-defined then $\texttt{Summ}(i, \sigma^j)$ is well-defined for all $j \geq 1$. Furthermore, $\texttt{Summ}(i, \sigma^j) = (u_i x_i^{j-1}, v_i y_i^{j-1})$ with $\texttt{Summ}(i, \sigma) = (u_i, v_i)$ and $\texttt{Iter}(i, \sigma) = (x_i, y_i)$.*

Proof. Firstly we will assume that $\texttt{Summ}(i, \theta)$ is well-defined. Given any summary sequence of transitions, $\sigma = \tau_1 \cdots \tau_m$, we define a set $\texttt{Nest}(\sigma) \subseteq [1..m]$ as follows. For any $j \in [1..m-1]$ and for any $a \in \Gamma$, if $\tau_j = (q_1, \mathbf{Push}_i(a), q_2)$ and $\tau_{j+1} = (q_3, \mathbf{Pop}_i(a), q_4)$ for some $q_1, q_2, q_3, q_4 \in Q$, then $j, j+1 \in \texttt{Nest}(\sigma)$. For any $k, j \in [1..m]$, if $k+1 \in \texttt{Nest}(\sigma)$ and $j-1 \in \texttt{Nest}(\sigma)$ and $\tau_k = (q_1, \mathbf{Push}_i(a), q_2)$ and $\tau_j = (q_3, \mathbf{Pop}_i(a), q_4)$ for some $q_1, q_2, q_3, q_4 \in Q$, then $k, j \in \texttt{Nest}(\sigma)$. Clearly $\texttt{Nest}(\sigma)$ captures all the intermediate positions of σ that are well matched. We define for all $i \in [1..m-1]$, if $i \in \texttt{Nest}(\sigma)$ or $Op(\tau_i) = \mathbf{Int}_j$ for some $j \in [1..n]$, then $\texttt{Unmatched}(i, \sigma) = \texttt{Unmatched}(i+1, \sigma)$, $\texttt{Unmatched}(i, \sigma) = \tau_i.\texttt{Unmatched}(i+1, \sigma)$ otherwise and for $i = m$, if

$i \in \texttt{Nest}(\sigma)$ then $\texttt{Unmatched}(m,\sigma) = \epsilon$, $\texttt{Unmatched}(i,\sigma) = \tau_i$ otherwise. Clearly $\texttt{Unmatched}(1, \sigma)$ gives us set of transitions in σ that are not well matched. Given a transition τ,we will use $\Gamma(\tau)$ to return the stack alphabet of the operation.

The following lemma states that in any sequence of transition, $\texttt{Summ}(i,\theta)$ is well defined iff barring the transitions that are well-matched, all transitions that pop elements from stack occur before ones that push into the stack. This is directly follows from the definition of Summ.

Lemma 55. *Given a sequence of transitions θ that are state wise compatible,* $\texttt{Summ}(i,\theta)$ *is well defined iff* $\texttt{Unmatched}(1,\theta\downarrow_{\Delta_i}) \in (Q \times \cup_{a\in\Gamma}\mathbf{Pop}_i(a) \times Q)^*.(Q \times \cup_{a\in\Gamma}\mathbf{Push}_i(a) \times Q)^*$. *Let* $\texttt{Unmatched}(1,\theta\downarrow_{\Delta_i}) = \alpha_i.\beta_i$ *where* $\alpha_i \in (Q \times \cup_{a\in\Gamma}\mathbf{Pop}_i(a) \times Q)^*$ *and* $\beta_i \in (Q \times \cup_{a\in\Gamma}\mathbf{Push}_i(a) \times Q)^*$. *Infact we can easily show that* $\texttt{Summ}(i,\theta) = (\Gamma(\alpha_i), \Gamma(\beta_i)^R)$

Proof. ($\Leftarrow$) We will induct on the length of the transition sequence θ. If $|\theta| = 0$ then there is nothing to prove. For the induction case, we will assume that $\theta = \sigma.\tau$, let $\texttt{Unmatched}(1, \sigma\downarrow_{\Delta_i}) = \alpha\beta \in (Q \times \cup_{a\in\Gamma}\mathbf{Pop}_i(a) \times Q)^*.(Q \times \cup_{a\in\Gamma}\mathbf{Push}_i(a) \times Q)^*$. We will further assume that $\alpha = \tau_{j_1}\tau_{j_2}\cdots\tau_{j_n}$ and $\beta = \tau'_{j_1}\tau'_{j_2}\cdots\tau'_{j_m}$. From the induction hypothesis, we have $\texttt{Summ}(i,\sigma) = (u_i, v_1)$, where $u_i = \Gamma(\alpha)$ and $v_i = \Gamma(\beta)^R$.

- case there $Op(\tau) = \mathbf{Push}_i$ or $Op(\tau) = \mathbf{Int}_i$ is simple and straight forward.
- case where $Op(\tau) = \mathbf{Pop}_i$,
  - If $\beta = \epsilon$, then clearly $\texttt{Unmatched}(1,\sigma\tau\downarrow_{\Delta_i}) = \alpha\tau$ such that $\alpha\tau \in (Q \times \cup_{a\in\Gamma}\mathbf{Pop}_i(a) \times Q)^*$, from this we know $v_i = \epsilon$. Now by definition, $\texttt{Summ}(i,\sigma\tau) = \Gamma(\alpha)\Gamma(\tau)$.
  - If β is not ϵ then clearly τ matches with τ'_{j_m} (otherwise $\texttt{Unmatched}(1,\sigma\tau\downarrow_{\Delta_i})$ will not be in the required form), hence $\texttt{Unmatched}(1,\sigma\tau\downarrow_{\Delta_i}) = \alpha\beta'$ where $\beta' = \tau^i_{j_1}\tau^i_{j_2}\cdots\tau^i_{j_{m-1}}$. Now clearly $v_i = av'_i$, where $v'_i = \Gamma(\beta')$, from this and definition of Summ, we have $\texttt{Summ}(i,\sigma\tau) = (u_i, v'_i)$

  ($\Rightarrow$)

Again we will show by induction on length of transition sequence θ. If $|\theta| = 0$ then there is nothing to prove. For the induction case, we will assume that $\theta = \sigma.\tau$, Let $\texttt{Summ}(i,\sigma\tau) = (u_i, v_i)$ and $\texttt{Summ}(i,\sigma) = (u'_i, v'_i)$.

- case where $Op(\tau) = \mathbf{Push}_i$ or $Op(\tau) = \mathbf{Int}_i$ is simple and straight forward.
- case where $Op(\tau) = \mathbf{Pop}_i$ and $v'_i = \epsilon$, then we have $v_i = \epsilon$ and $u_i = u'_i\Gamma(\tau)$. By induction, we have $\texttt{Unmatched}(1,\sigma\downarrow_{\Delta_i}) = \alpha$ such that $\alpha \in (Q \times \cup_{a\in\Gamma}\mathbf{Pop}_i(a) \times Q)^*$ and $\Gamma(\alpha) = u'_i$. Now from this and the definition of Unmatched, the result follows.
- case where $Op(\tau) = \mathbf{Pop}_i$ and $v'_i \neq \epsilon$, then we have $u'_i = u_i$ and $\Gamma(\tau)v_i = v'_i$. By induction, we have $\texttt{Unmatched}(1,\sigma\downarrow_{\Delta_i}) = \alpha\beta$ such that $\Gamma(\alpha) = u_i$ and $\Gamma(\beta)^R = v'_i$. From this it is easy to infer that $\beta = \beta'\tau'$, where $\tau' \in Q \times \mathbf{Push}(\Gamma(\tau)) \times Q$. From this we know that τ', τ are well matched. Now from this and the definition of Unmatched, the result follows.

□

It is also easy to see that $\texttt{Summ}(i,\theta.\theta)$ is well defined iff $\texttt{Unmatched}(1,\theta.\theta\downarrow_{\Delta_i}) \in (Q \times \cup_{a\in\Gamma}\mathbf{Pop}_i(a) \times Q)^*.(Q \times \cup_{a\in\Gamma}\mathbf{Push}_i(a) \times Q)^*$. But $\texttt{Unmatched}(1,\theta.\theta\downarrow_{\Delta_i}) = \texttt{Unmatched}(1, \alpha_i.\beta_i.\alpha_i.\beta_i)$. Hence we have that either $\Gamma(\alpha_i)$ is prefix of $\Gamma(\beta_i)^R$ or $\Gamma(\beta_i)^R$ is a prefix of $\Gamma(\alpha_i)$.

If $\Gamma(\alpha_i)$ is prefix of $\Gamma(\beta_i)$ then $\alpha_i.\beta_i.\alpha_i.\beta_i = \alpha_i.\beta_i^1.\beta_i^2.\alpha_i.\beta_i$ where $\Gamma(\beta_i^2)^R = \Gamma(\alpha_i)$. clearly $\beta_i^2.\alpha_i$ is well matched hence we have $\texttt{Unmatched}(1, \alpha_i.\beta_i.\alpha_i.\beta_i) = \alpha_i.\beta_i^1.\beta_i$ and $\texttt{Summ}(i, \theta.\theta) = (u_i, v_i.x_i)$, where $x_i = \Gamma(\beta_i^1)^R$. In general, it is easy to see that $(\alpha_i.\beta_i)^j = (\alpha_i.\beta_i^1.\beta_i^2)^{j-1}.\alpha_i.\beta_i$ with $\texttt{Unmatched}(1, (\alpha_i.\beta_i)^j) = (\alpha_i.(\beta_i^1)^{j-1}.\beta_i)$ and $\texttt{Summ}(i, \theta^j) = (u_i, v_i.x_i^{j-1})$

Similarly, If $\Gamma(\beta_i)$ is prefix of $\Gamma(\alpha_i)$ then $\alpha_i.\beta_i.\alpha_i.\beta_i = \alpha_i.\beta_i.\alpha_i^1.\alpha_i^2.\beta_i$ where $\Gamma(\beta_i)^R = \Gamma(\alpha_i^1)$ and $\beta_i.\alpha_i^1$ is well matched. Hence $\texttt{Unmatched}(1, \alpha_i.\beta_i.\alpha_i.\beta_i) = \alpha_i.\alpha_i^2.\beta_i$, in general we have $(\alpha_i.\beta_i)^j = \alpha_i.\beta_i.(\alpha_i^1.\alpha_i^2.\beta_i)^{j-1}$. Hence the result follows.

□

Acceleration of regular/rational sets of configurations by loops

In the following, we first show that the class of regular (resp. rational) sets of configurations is not closed under $Post_{\theta^*}$. Then, we show that the image by $Post_{\theta^*}$ of any regular set of configurations is a rational one.

Theorem 23. *There is is an MPDS $M = (n, Q, \Gamma, \Delta)$, a regular (resp. rational) set of its configurations C and a transition sequence $\theta \in \Delta^*$ such that the set of configurations $Post_{\theta^*}(C)$ is not regular (resp. rational).*

Proof. Let us first show that the set of regular languages is not closed wrt. acceleration of simple sequence of transitions. To do that, consider an MPDS $M = (2, \{q, q'\}, \{a, b\}, \Delta)$ where Δ only contains the following two transitions $(q, \mathbf{Push}_1(a), q')$ and $(q', \mathbf{Push}_2(b), q)$. Let $\theta = (q, \mathbf{Push}_1(a), q')(q', \mathbf{Push}_2(b), q)$. It is easy to see that $Post_{\theta^*}(\{(q, \epsilon, \epsilon)\}) = \{(q, a^i, b^i) \mid i \in \mathbb{N}\}$ which is not regular.

Now let us show the non-closure of rational languages. Consider an MPDS $M = (2, \{q, q'\}, \{a, b\}, \Delta)$ where Δ only contains the following two rules $(q, \mathbf{Push}_1(b), q')$ and $(q', \mathbf{Pop}_2(a), q)$. Let $\theta = (q, \mathbf{Push}_1(b), q')(q', \mathbf{Pop}_2(a), q)$ and $C = \{(q, a^i, a^i) \mid i \in \mathbb{N}\}$. Observe that C is a rational set of configurations. It is easy to see that $Post_{\theta^*}(C) = \{(q, b^j a^i, a^{i-j}) \mid 0 \leq j \leq i\}$ which is not rational. □

However, whenever C is a regular set of configurations the set $Post_{\theta^*}(C)$ has a simple description. In what follows we fix a MPDS $M = (n, Q, \Gamma, \Delta)$.

Theorem 24. *For every regular set of configurations C and transition sequence $\theta \in \Delta^*$, the set $Post_{\theta^*}(C)$ is rational and effectively constructible.*

Proof. Let θ be a sequence of transitions of the form $(q_0, op_0, q_0')(q_1, op_1, q_1') \cdots (q_m, op_m, q_m')$. Since $Post_{\theta^*}(C_1 \cup C_2) = Post_{\theta^*}(C_1) \cup Post_{\theta^*}(C_2)$, we can assume w.l.o.g that C is of the form $\{q\} \times L_1 \times \cdots \times L_n$ where each L_j is an 1-dim rational language over Γ accepted by a finite state automaton A_j for all $j \in [1..n]$. The proof proceeds by cases.

Case 1: Let us assume $q_i' \neq q_{i+1}$ for some $i \in [1..m-1]$ or $q_0 \neq q$. In this case the sequence of transitions cannot be executed and hence $Post_{\theta^*}(C) = C$.

Case 2: Let us assume $q_0 \neq q_m'$, $q_0 = q$ and $q_i' = q_{i+1}$ for all $i \in [1..m-1]$. In this case, the sequence of transitions can not be iterated more than once and so we have $Post_{\theta^*}(C) =$

$Post_\theta(C) \cup C$. We now examine the set $Post_\theta(C)$. First, let us assume that $\texttt{Summ}(i,\theta) = \bot$ for some $i \in [1..n]$. Then $Post_\theta(C) = \emptyset$ and hence $Post_{\theta^*}(C) = C$.

Let us assume now that $\texttt{Summ}(i,\theta) = (u_i, v_i)$ is well-defined for all $i \in [1..n]$. We can apply Lemma 53, to show that $Post_\theta(C) = \{q'_m\} \times L'_1 \times \cdots \times L'_n$ where for every $i \in [1..n]$, $L'_i = \{w'_i \mid \exists w_i \in \Gamma^*.\ w'_i = v_i.w_i \wedge u_i w_i \in L_i\}$. It is easy to see that L'_i is an 1-dim rational language and can be accepted by an automaton A'_i whose size is polynomial in the size of A_i and the length of θ.

Case 3: Let us assume $q_0 = q'_m$, $q_0 = q$ and $q'_i = q_i$ for all $i \in [1..m]$. In this case, the sequence of transitions forms a loop in the control flow graph of M and hence the sequence may possibly be iterated. Observe that if the function $\texttt{Summ}(i,\theta) = \bot$ for some $i \in [1..n]$, then $Post_{\theta^*}(C) = C$. Hence, let us assume that $\texttt{Summ}(i,\theta) = (u_i, v_i)$ for all $i \in [1..n]$ so that it is well-defined for each i.

Lemma 54 suggests that we should examine when $\texttt{Summ}(i,\theta\theta)$ is defined for all i. Indeed, if $\texttt{Summ}(i,\theta\theta)$ is undefined for some $i \in [1..n]$, then $Post_{\theta^*}(C) = Post_\theta(C) \cup C$ (which can be computed as shown in the previous case). So, let us further assume that $\texttt{Summ}(i,\theta\theta)$ is well-defined for all $i \in [1..n]$. Hence, the function $\texttt{Iter}(i,\sigma)$ is also well-defined. Let us assume that $\texttt{Iter}(i,\sigma) = (x_i, y_i)$

Now, we can combine Lemma 54 with Lemma 53 to give a characterization of when a sequence θ is iterable and its effect.

Lemma 56. *Let $j \geq 1$ and $c = (p, w_1, \ldots, w_n)$ and $c' = (p', w'_1, \ldots, w'_n)$ be two configurations of M. $c \xrightarrow{\theta^j} c'$ iff for every $i \in [1..n]$, we have $w_i = u_i x_i^{j-1} w''_i$ and $w'_i = v_i y_i^{j-1} w''_i$ for some $w''_i \in \Gamma^*$ with u_i, v_i, x_i and y_is are defined as above.*

Proof. The proof follows directly from lemma-54 and lemma-53. We will prove this inducting on j

base case $j = 1$: Base case directly follows from lemma-53 which states, $c \xrightarrow{\theta} c'$ with $w_i = u_i w''_i$ and $w'_i = v_i w''_i$ iff $\texttt{Summ}(i,\theta) = (u_i, v_i)$.

case $j \geq 1$: By lemma-54 if $\texttt{Summ}(i,\theta\theta)$ is defined then $\texttt{Summ}(i,\theta^j) = (u_i x_i^{j-1}, v_i y_i^{j-1})$ is defined for all $j > 1$, where $(x_i, y_i) = \texttt{Iter}(i,\theta)$. Now applying lemma-53, we get that $c \xrightarrow{\theta^j} c'$ iff for every $i \in [1..n]$, we have $w_i = u_i x_i^{j-1} w''_i$ and $w'_i = v_i y_i^{j-1} w''_i$ such that $\texttt{Summ}(i,\theta^j) = (u_i x_i^{j-1}, v_i y_i^{j-1})$.

□

With this lemma in place, let L be the $(2n+1)$-dim language defined as the set containing exactly the words of the form

$$(q, u_1 x_1^{j-1} w_1, v_1 y_1^{j-1} w_1, u_2 x_2^{j-1} w_2, v_2 y_2^{j-1} w_2, \ldots, u_n x_n^{j-1} w_n, v_n y_n^{j-1} w_n)$$

with $j \geq 1$ and $w_i \in \Gamma^*$ and where u_i, v_i, x_i and y_is are defined as above. Observe that each element of L relates a pair of configurations such that from the first we can execute the sequence θ a finite number of times to reach the second. The starting configuration is given by the first and all the even numbered positions, while the ending configuration is given by all the odd numbered positions (including the first). As a matter of fact elements of L relates exactly all such pairs in this manner. This language L is rational and we can easily compute

an $(2n+1)$-tape automaton A whose size is polynomial in the size of θ and polynomial in the size of M. To compute an $(n+1)$-tape automaton A' accepting $Post_{\theta^+}(C)$, we proceed as follows: First, we define the regular language $L' = \{q\} \times L_1 \times \Gamma^* \times \cdots \times L_n \times \Gamma^*$. Then, we compute an $(2n+1)$-tape automaton A'' accepting precisely the language resulting of the intersection of the regular language L' and L. This allows us to restrict the starting configurations to be precisely those from C. The size A'' is exponential in the number of stacks and polynomial in the size of θ, and the finite state automata $A_1, \ldots, A_n$. Finally, we need to project away the tapes concerning the starting stack configurations. We let then $A' = \Pi_\iota(A'')$ with $\iota = \{2i+1 \mid i \in [0..n]\}$. We note that this step does not result in any blow up and thus the size of A' is exponential in the number of stacks and polynomial in the size of θ and $A_1, \ldots, A_n$.

Since $Post_{\theta^*}(C) = C \cup Post_{\theta^+}(C)$ and the class of rational / regular languages is closed under union, this completes the proof of Theorem 24. □

8.2.4 Constrained Simple Regular Expressions

We now introduce the class of (1 dimensional) Constrained Simple Regular Expressions (CSRE) and prove some closure properties. CSRE definable languages form an expressive class equivalent to the bounded semi-linear languages defined in [49] and the class of languages accepted by 1-CQDD introduced in [43]. To deal with configuration sets of MPDS we need n dimensional CSREs and so we lift these results to that setting. We then show that the CSRE definable sets of configurations form a stable collection under acceleration by loops. As in previous section, we will assume that the loops do not have zero tests. However, this class is not stable w.r.t. bounded context runs. We begin by recalling some basics about *Presburger arithmetic*.

Presburger arithmetic

Presburger arithmetic is the first-order theory of natural numbers with addition, subtraction and order. We recall briefly its definition. Let $\mathcal{V}$ be a set of variables. We use $x, y, \ldots$ to denote variables in $\mathcal{V}$. The set of terms in Presburger arithmetic is defined as follows: $t ::= 0 \mid 1 \mid x \mid t - t \mid t + t$. The set of formulae of the Presburger arithmetic is defined to be $\varphi ::= t \leq t \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\varphi$.

We use the standard abbreviations: $\varphi_1 \wedge \varphi_2 = \neg(\varphi_1 \vee \varphi_2)$, $\varphi_1 \Rightarrow \varphi_2 = \neg\varphi_1 \wedge \varphi_2$, and $\forall x.\varphi = \neg\exists x.\neg\varphi$. The notions of free and bound variables, and quantifier-free formula are as usual. An *existential* Presburger formula is one of the form $\exists x_1 \exists x_2 \ldots \exists x_n.\varphi$ where φ is a quantifier-free formula. We shall often write positive boolean combinations of existential Presburger formulas in place of an existential Presburger formula. Clearly, by an appropriate renaming of the quantified variables, any such formula can be converted into an equivalent existential Presburger formula. We write $FreeVar(\varphi) \subseteq \mathcal{V}$ to denote the set of free variables of φ. Given a function μ from $FreeVar(\varphi)$ to $\mathbb{N}$, the meaning of μ *satisfies* φ is as usual and we write $\mu \models \varphi$ to denote this. We write $\varphi(x_1, x_2, \ldots, x_k)$ to denote a Presburger formula φ whose free variables are (contained in) $x_1, \ldots, x_k$. Such a formula naturally *defines* a subset of $\mathbb{N}^k$ given by $\{(i_1,$

$i_2, \ldots, i_k) \mid \mu \models \varphi(x_1, x_2, \ldots, x_k)$ where $\mu(x_j) = i_j, 1 \leq j \leq k\}$. We say that a subset S of $\mathbb{N}^k$ is definable in Presburger arithmetic if there is a formula φ that defines it.

Constrained Simple Regular Expression (CSRE)

A Constrained Simple Regular Expression (CSRE) e over an alphabet Σ is defined as a tuple of the form $e = (w_1, \ldots, w_m, \varphi(x_1, x_2, \ldots, x_m))$ where $w_1, \ldots, w_m$ is a non-empty sequence of words over Σ, and φ is an existential Presburger formula. The language defined by the CSRE e, denoted by $L(e)$, is the set of words of the form $w_1^{i_1} w_2^{i_2} \cdots w_m^{i_m}$ such that φ holds for the function μ defined by $\mu(x_j) = i_j$ for all $j \in [1..m]$. The size of e is defined by $|e| = |w_1 \cdots w_m| + |\varphi|$. CSREs define the same class of languages as CQDDs [43] (see [49]), however they have a much simpler presentation avoiding automata altogether and as we shall see quite amenable to a number of operations. We will now present a lemma that will be useful for proving some properties of CSRE.

Lemma 57. *Given two sequences of words $v_1, \ldots, v_\ell$ and $u_1, \ldots, u_k$ over Γ, it is possible to construct, in polynomial time in $|v_1 v_2 \ldots v_\ell u_1 u_2 \ldots u_k|$, an existential Presburger formula $\varphi(x_1, x_2, \ldots, x_\ell, y_1, y_2, \ldots, y_k)$ such that $\mu \models \varphi$ iff $v_1^{\mu(x_1)} \cdots v_\ell^{\mu(x_\ell)} = u_1^{\mu(y_1)} \cdots u_k^{\mu(y_k)}$.*

Proof. To prove this lemma, we we will reduce the problem to computing an existential Presburger formula recognizing the Parikh image of a pushdown automaton $\mathscr{P} = (Q, \Sigma, \Gamma, \Delta, q_0, F)$. Then, we use the following Proposition 25 to show that such an existential Presburger formula can be constructed in polynomial time.

Proposition 25. *[142] Given a pushdown automaton $\mathscr{P} = (Q, \Sigma, \Gamma, \Delta, q_0, F)$, it is possible to construct, in polynomial time in the size of $\mathscr{P}$, an existential Presburger formula ϕ such that FreeVar$(\phi) = \Sigma$ and Parikh$(L(\mathscr{P})) = \{\mu \mid \mu \models \phi\}$.*

The pushdown automaton $\mathscr{P}$ is defined as follows. The set of states Q is defined as the set $\{q_0, q_1, \ldots, q_\ell, q_{\ell+1}, q_{\ell+2} \ldots, q_{\ell+k}\}$ with $F = \{q_{\ell+k}\}$. The input alphabet Σ is defined by the set of variables $x_1, x_2, \ldots, x_\ell$ and $y_1, y_2, \ldots, y_k$. The pushdown works in phases: In the first phase, the pushdown automaton is at the state q_0 and can push the reverse of the word v_1 (denoted v_1^R) into its stack while outputting x_1 (i.e., $(q_0, \mathbf{Push}(v_1^R), x_1, q_0) \in \Delta$). In nondeterministic manner the pushdown can decide to move to the simulation of the second phase by moving its state from q_0 to q_1 (i.e., $(q_0, \mathbf{Int}, \epsilon, , q_1) \in \Delta$).

In a phase i, with $1 < i \leq \ell$, the pushdown automaton $\mathscr{P}$ is at the state q_{i-1} and and can push the reverse of the word v_i (denoted v_i^R) into its stack while outputting x_i (i.e., $(q_{i-1}$, $\mathbf{Push}(v_i^R), x_i, q_{i-1}) \in \Delta$). In nondeterministic manner the pushdown can decide to move to the simulation of the second phase by moving its state from q_{i-1} to q_i (i.e., $(q_{i-1}, \mathbf{Int}, \epsilon, q_i) \in \Delta$).

In the phase $\ell + 1$, the pushdown automaton is at the state q_ℓ and can start popping the reverse of the word u_k (denoted u_k^R) from its stack while outputting y_k (i.e., $(q_\ell, \mathbf{Pop}(u_k^R), x_i$, $q_\ell) \in \Delta$). In nondeterministic manner the pushdown can decide to move to the simulation of the next phase $\ell + 2$ by moving its state from q_ℓ to $q_{\ell+1}$ (i.e., $(q_\ell, \mathbf{Int}, \epsilon, q_{\ell+1}) \in \Delta$).

In a phase i, with $\ell+1 < i \leq \ell+k$, the pushdown automaton $\mathscr{P}$ is at the state q_{i-1} and and can pop the reverse of the word $u_{k+\ell+1-i}$ (denoted $u^R_{k+\ell-i}$) from its stack while outputting $y_{k+\ell+1-i}$ (i.e., $(q_{i-1}, \mathbf{Pop}(u^R_{k+\ell+1-i}), y_{k+\ell+1-i}, q_{i-1}) \in \Delta$). In nondeterministic manner the pushdown can decide to move to the simulation of the next phase by moving its state from q_{i-1} to q_i (i.e., $(q_{i-1}, \mathbf{Int}, \epsilon, q_i) \in \Delta$).

Finally, the set Δ is defined as the smallest set satisfying the above condition.

Let φ be the formula recognizing the Parikh image of the pushdown automaton $\mathscr{P}$ constructed as stated in Proposition 25. Then it is easy to see that $\mu \models \varphi$ iff $v_1^{\mu(x_1)} \cdots v_\ell^{\mu(x_\ell)} = u_1^{\mu(y_1)} \cdots u_k^{\mu(y_k)}$.

□

Next, we present some closure and decidability results for the class of CSRE definable languages. These results can be also deduced from [49] since CSREs define bounded semilinear languages.

Lemma 58. *The class of languages defined by CSREs is closed under intersection, union and concatenation. The emptiness, membership and inclusion problems for CSREs are decidable.*

Proof. Let us assume two CSRE $e_1 = (v_1, v_2, \ldots, v_\ell, \varphi_1(x_1, x_2, \ldots, x_\ell))$ and $e_2 = (u_1, u_2, \ldots, u_k, \varphi_2(y_1, y_2, \ldots, y_k))$. We assume w.l.o.g. that $x_i \neq y_j$ for all i and j.

- [**Concatenation**] Let $e_1 \cdot e_2$ be the CSRE expression defined by the tuple $(v_1, v_2, \ldots, v_\ell, u_1, \ldots, u_k, \varphi(x_1, \ldots, x_\ell, y_1, \ldots, y_k))$ where $\varphi = \varphi_1(x_1, \ldots, x_\ell) \wedge \varphi_2(y_1, \ldots, y_k)$. It is easy to see that $L(e_1 \cdot e_2) = L(e_1) \cdot L(e_2)$ and that the size of $e_1 \cdot e_2$ is linear in $|e_1| + |e_2|$.
- [**Union**] Let $e_1 + e_2$ be the CSRE expression defined by the tuple $(v_1, v_2, \ldots, v_\ell, u_1, \ldots, u_k, \varphi(x_1, \ldots, x_\ell, y_1, \ldots, y_k))$ such that $\varphi = (\varphi_1(x_1, \ldots, x_\ell) \wedge \wedge_{1 \leq j \leq k} y_j = 0) \vee (\varphi_2(y_1, \ldots, y_k) \wedge \wedge_{1 \leq i \leq \ell} x_i = 0)$ It is easy to see that $L(e_1 + e_2) = L(e_1) \cup L(e_2)$ and that the size of $e_1 + e_2$ is linear in $|e_1| + |e_2|$.
- [**Intersection**] We show there is a CSRE accepting $L(e_1) \cap L(e_2)$. Let $e_1 = (v_1, \ldots, v_\ell, \varphi_1(x_1, \ldots, x_k))$ and $e_2 = (u_1, \ldots, u_k, \varphi_2(y_1, \ldots, y_k))$ with $x_i \neq y_j$ for i, j. Let $\varphi_3(x_1, \ldots, x_\ell, y_1, \ldots, y_k)$ be the existential Presburger formula obtained by the application of Lemma 57 to the two sequences of words $v_1, \ldots, v_\ell$ and $u_1, \ldots, u_k$. Then the CSRE expression $e_1 \cap e_2$ defined to be $(v_1, \ldots, v_\ell, \varphi(x_1, \ldots, x_\ell))$ with $\varphi = \exists y_1 . \exists y_2 . \ldots . \exists y_n . \varphi_3(x_1, \ldots, x_\ell, y_1, \ldots, y_k) \wedge \varphi_1(x_1, \ldots, x_\ell) \wedge \varphi_2(y_1, \ldots, y_k)$ defines the intersection of the languages defined by e_1 and e_2 and that the size of $e_1 \cap e_2$ is linear in $|e_1| + |e_2|$.

  The emptiness and membership can be easily shown to be decidable and NP-complete. A lower bound for these two problem can be obtained by a straightforward reduction from the satisfiability problem of the class of existential Presburger formula.

- [**Inclusion**] Let us now show that the inclusion problem is also decidable. The lemma is an immediate corollary of the following three results: (1) for any CSRE expression e, it is possible to effectively construct a bounded Parikh automaton accepting $L(e)$ (see [49] for the bounded Parikh automaton definition), (2) for any bounded Parikh automaton, it is possible to construct a bounded deterministic automaton recognizing the same language

(see Theorem 11, Corollary 12 and Section 5.3 in [49] for the bounded Parikh automaton definition), and (3) the class of deterministic Parikh automata is closed under intersection and complement and its emptiness problem is decidable (see [48]).

□

From Lemma 56 it is clear that in order to compute the effect of the iteration of a sequence θ on the content of stack i, one has to *left-quotient* the content of stack i by the sequence $u_i x_i^{j-1}$ and then add the sequence $v_i y_i^{j-1}$ (on the left). With this in mind we now examine left-quotients of languages defined by CSREs w.r.t. iterations of a given word. First we state a technical lemma.

Lemma 59. *Let e be a CSRE over an alphabet Σ and $w \in \Sigma^*$ be a word. Then, we can construct, in polynomial time in $|w| + |e|$, a CSRE $e' = (w, u_1, u_2, \ldots, u_k, \varphi(y, y_1, y_2, \ldots, y_k))$ such that for every $i \in \mathbb{N}$, $L(e_i) = \{w' \mid w^i w' \in L(e)\}$ where $e_i = (\epsilon, u_1, u_2, \ldots, u_k, (y = i \wedge \varphi(y, y_1, y_2, \ldots, y_k)))$.*

Proof. Let us assume that the CSRE e is of the form $(w_1, w_2, \ldots, w_n, \phi(x_1, x_2, \ldots, x_n))$. Prefixes of the form w^j may end in the middle of an occurance of some w_i and so, in order to compute left-quotients, it is useful to expand this CSRE into union of a set of more elaborate ones, taking into account the position of such splits. For every $i \in [1..n]$ and $u, v \in \Sigma^*$ such that $w_i = uv$, we define the CSRE $sp(i, u, v)$ as follows:

$$(w_1, \ldots, w_i, u, v, w_i, w_{i+1}, \ldots, w_n, \phi \wedge \psi_{(i,u,v)}(x_1, \ldots, x'_i, x_u, x_v, x''_i, x_{i+1}, \ldots, x_n))$$

where

$$\psi_{i,u,v} = (x_i = x'_i + x''_i + x_u + x_v) \wedge ((x_u = x_v = 0) \vee (x_u = x_v = 1))$$

We note that the size of $\psi_{i,u,v}$ is constant. It is easy to see that $L(e) = \bigcup_{i \in [1..n]} \bigcup_{w_i = uv} L(sp(i, u, v))$.

Next, we use Lemma 57 to obtain a Presburger formula $\varphi'_{(i,u,v)}(y, x_1, \ldots, x_i, x_u)$ such that $\mu \models \varphi_{(i,u,v)}(y, x_1, \ldots, x_i, x_u)$ iff $w_1^{\mu(x_1)} w_2^{\mu(x_2)} \ldots w_{i-1}^{\mu(x_{i-1})} w_i^{\mu(x_i)} u^{\mu(x_u)} = w^{\mu(y)}$. We modify the formula to also insist that $x_u = 1$, and refer to the resulting formula as $\varphi_{(i,u,v)}$. The size of $\varphi_{(i,u,v)}$ is polynomial in $|w_1 \ldots w_i u| + |w|$. Now, we combine the CSRE $sp(i, u, v)$ with the formula $\varphi_{(i,u,v)}$ as follows: Let $sp'(i, u, v)$ be the following CSRE defined

$$(w, v, w_i, w_{i+1}, \ldots, w_n, \exists x_1 \exists x_2 \ldots \exists x_i \exists x_u.\ \phi \wedge \psi_{(i,u,v)} \wedge \varphi_{(i,u,v)})$$

The free variables of the formula in $sp'(i, u, v)$, in order, are $y, x_v, x''_i, x_{i+1}, \ldots, x_n$ ($n - i + 3$ in all).

Notice that this CSRE $sp'(i, u, v)$ defines the language consisting of words of the form $w^{\mu(y)} v w_i^{\mu(x''_i)} w_{i+1}^{\mu(x_{i+1})} \ldots w_n^{\mu(x_n)}$ such that there are $\mu(x_1), \mu(x_2) \ldots \mu(x_i), \mu(x_u) \in \mathbb{N}$ such that

1) $w^{\mu(y)} = w_1^{\mu(x_1)} w_2^{\mu(x_2)} \ldots w_i^{\mu(x_i)} u$ and

2) $w_1^{\mu(x_1)} w_2^{\mu(x_2)} \ldots w_i^{\mu(x_i)} u.v.w_i^{\mu(x''_i)} w_{i+1}^{\mu(x_{i+1})} \ldots w_n^{\mu(x_n)} \in L(sp(i, u, v))$. Further, the sum of the length of the words in $sp'(i, u, v)$ is linear in the sum of the lengths of the words in e plus the $|w|$, the number of variables increases at most by a constant, and the size of the formula is $|\phi|$ plus the size of $|\varphi_{i,u,v}| + |\psi_{i,u,v}|$ (it increases additively by a polynomial in $|w_1 \ldots w_i u| + |w|$.)

Next we show how to combine the various CSRE's $sp'(i,u,v)$, $1 \le i \le n$, $uv = w_i$ into a single CSRE with the desired property. We describe this for the case of two such CSREs and the generalization to a collection is similar.

Given $sp'_{(i,u,v)}$ and $sp'_{(i',u',v;)}$ we first relabel all the variables in the Presburger formulas of the two CSREs to be disjoint leaving only the variable y as it is. So, let their respective free variables be $y, p_1, p_2, \ldots p_{n-i+2}$ and $y, q_1, q_2, \ldots, q_{n-i'+2}$. The resulting CSRE is then given by

$$(w, v, w_i, w_{i+1}, \ldots, w_n, v', w_{i'}, \ldots, w_n, \chi(y, p_1, p_2, \ldots, p_{n-i+2}, q_1, q_2, \ldots, q_{n-i'+2}))$$

where χ is

$$(\phi \wedge \psi_{i,u,v} \wedge \varphi_{(i,u,v)} \wedge (\bigwedge_{j=1}^{n-i'+2} q_j = 0)) \vee (\phi \wedge \psi_{i',u',v'} \wedge \varphi_{(i',u',v')} \wedge (\bigwedge_{j=1}^{n-i+2} p_j = 0))$$
$$= \phi \wedge ((\psi_{i,u,v} \wedge \varphi_{(i,u,v)} \wedge (\bigwedge_{j=1}^{n-i'+2} q_j = 0)) \vee (\psi_{i',u',v'} \wedge \varphi_{(i',u',v')} \wedge (\bigwedge_{j=1}^{n-i+2} p_j = 0))$$

This describes the language $L(sp'_{(i,u,v)}) \cup L(sp'_{(i',u',v')})$.

We can easily generalize this to combine all the $sp'_{(i,u,v)}$'s using a disjoint set of variables (except y) for each of them in the similar manner. Let the sum of the length of the words w_1, $\ldots$, w_n in e be K. The resulting expression:

1. Has a variable y whose value may be set to the number of iterations of w by which we desire to left quotient.
2. Whose sum of length of the words is polynomial in $K + |w|$
3. Whose number of variables increases by a polynomial in $K + |w|$
4. Whose formula has increased in length w.r.t. to ϕ by an addition whose size is polynomial in $K + |w|$.

This completes the proof of the lemma.

□

The key point about the above lemma is that the left-quotient of $L(e)$ w.r.t w^i, for some $i \in \mathbb{N}$, can be precisely identified as $L(e_i)$. Thus, the CSRE $(\epsilon, u_1, u_2, \ldots, u_k, \varphi(y, y_1, y_2, \ldots, y_k))$ defines the left-quotient of $L(e)$ w.r.t $\{w^i \mid i \in \mathbb{N}\}$, giving us the following corollary.

Corollary 3. *Let e be a CSRE over an alphabet* Σ *and* $w \in \Sigma^*$ *be a word. Then, we can construct, in polynomial time in* $|w| + |e|$*, a CSRE* e' *such that* $L(e') = \{w' \mid \exists i \in \mathbb{N}.\ w^i w' \in L(e)\}$.

Multi-Dimensional Constrained Simple Regular Expression

Let $n \ge 1$. An n-dim CSRE e over an alphabet Σ is a of tuple of the form $((u_1, \ldots, u_{k_1}), (u_{k_1+1}, \ldots, u_{k_2}), \ldots, (u_{k_{n-1}+1}, \ldots, u_{k_n}), \varphi(x_1, \ldots, x_{k_n}))$ where: (1) $1 \le k_1 < k_2 < \cdots < k_n$ and (2) for every $i \in [1..k_n]$, u_i is a word over Σ. An n-dim CSRE $\mathbf{e}$ accepts the n-dim language, denoted by $L(e)$, consisting of the n-dim words of the form $(u_1^{i_1} \cdots u_{k_1}^{i_{k_1}}, \cdots, u_{k_{n-1}+1}^{i_{k_{n-1}+1}} \cdots u_{k_n}^{i_{k_n}})$ such that φ holds for the function μ defined by $\mu(x_j) = i_j$ for all $j \in [1..k_n]$. In order to simply the notations, we

sometimes write e as follows $(\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_n}, \varphi(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}))$ where $\mathbf{u}_i = (u_{k_{i-1}+1}, \ldots, u_{k_i})$ and $\mathbf{x}_i = (x_{k_{i-1}+1}, \ldots, x_{k_i})$ for all $i \in [1..n]$. In the following, we show that the class of n-languages accepted by n-dim CSREs enjoys the same closure properties as the class of CSREs. Furthermore we have the same decidability results.

Lemma 60. *Let $n \geq 1$. The class of n-languages defined by n-CSREs is closed under intersection, union and concatenation. The emptiness problem , membership problem as well as inclusion problem are decidable for n-dim CSREs.*

Proof. The proofs are similar to the ones of Lemma 58. Consider two n-CSREs $\mathbf{e_1} = ((u_1, \ldots, u_{\ell_1}), \ldots, (u_{\ell_{n-1}+1}, \ldots, u_{\ell_n}), \varphi_1(x_1, \ldots, x_{\ell_n}))$ and $\mathbf{e_2} = ((v_1, \ldots, v_{k_1}), \ldots, (v_{k_{n-1}+1}, \ldots, v_{k_n}), \varphi_2(y_1, \ldots, y_{k_n}))$ with $x_i \neq y_j$ for i, j. We will show that there is an CSRE accepting the language $L(\mathbf{e_1}) \cdot L(\mathbf{e_2})$. Let $\mathbf{e_1} \cdot \mathbf{e_2}$ be the CSRE expression defined by the tuple $((u_1, \ldots, u_{\ell_1}, v_1, \ldots, v_{k_1}), \ldots, (u_{\ell_{n-1}+1}, \ldots, u_{\ell_n}, u_{k_{n-1}+1}, \ldots, u_{k_n}), \varphi(x_1, \ldots, x_{k_1}, y_1, \ldots y_{\ell_1}, x_{k_1+1}, \ldots, x_{k_2}, y_{\ell_1+1}, \ldots, y_{\ell_2}, \ldots, x_{k_n}, y_{\ell_{n-1}+1}, \ldots, y_{\ell_n}))$ where $\varphi = \varphi_1(x_1, \ldots, x_{k_n}) \wedge \varphi_2(y_1, \ldots, y_{\ell_n})$. It is easy to see that $L(\mathbf{e_1} \cdot \mathbf{e_2}) = L(\mathbf{e_1}) \cdot L(\mathbf{e_2})$.

We will show that it is possible to reduce emptiness, membership and inclusion problems to their corresponding ones for CSRE. The idea is to encode the n-dim CSRE as a CSRE using a special symbol $\sharp$ (not appearing in the alphabet) to separate all the different tapes. To that aim, we define the function $Encode$ that maps any n-dim CSRE to an CSRE. Let $\mathbf{e} = ((u_1, \ldots, u_{k_1}), \ldots, (u_{k_{n-1}+1}, \ldots, u_{k_n}), \varphi(x_1, \ldots, x_{k_n}))$ be an n-dim CSRE over the alphabet Σ. We assume that $\sharp \notin \Sigma$. Then we define $Encode(\mathbf{e})$ to be the CSRE $(u_1, \ldots, u_{k_1}, \sharp, \ldots, \sharp, u_{k_{n-1}+1}, \ldots, u_{k_n}, \varphi'(x_1, \ldots, x_{k_1}, y_1, x_{k_1+1}, \ldots, x_{k_2}, y_2, \ldots, y_{n-1}, x_{k_{n-1}+1}, \ldots, x_{k_n}))$ where $\varphi' = \varphi(x_1, \ldots, x_{k_n}) \wedge \wedge_{1 \leq i \leq n-1} y_i = 1$. It is easy to see that, for any n-dim word $(w_1, \ldots, w_n)$ and any n-dim CSREs $\mathbf{e}$ we have $(w_1, \ldots, w_n) \in L(\mathbf{e_1})$ iff $w_1 \sharp \cdots \sharp w_n \in L(Encode(\mathbf{e_1}))$. Hence, $L(\mathbf{e_1}) = \emptyset$ iff $L(Encode(\mathbf{e_1})) = \emptyset$ and $L(\mathbf{e_1}) \subseteq L(\mathbf{e_2})$ iff $L(Encode(\mathbf{e_1})) \subseteq L(Encode(\mathbf{e_2}))$. Thus, the emptiness, membership and containment problems are decidable. □

Next we extend Lemma 59 to the case of n-dim CSREs — n-dim CSREs are closed under left quotienting by simultaneous iterations of a tuple of words $w_i, 1 \leq i \leq n$, one for each component. Even more, this can be achieved by constructing an n-CSRE in which the number iterations may be set parametrically. The construction is also in polynomial time. Firstly recall that we use $\mathbf{u}[i \leftarrow w]$ to denote the n-dim word $(u_1, u_2, \ldots, u_{i-1}, w, u_{i+1}, \ldots, u_n)$.

Lemma 61. *Let $n \geq 1$. Let e be a n-dim CSRE over an alphabet Σ and $\mathbf{w} = (w_1, \ldots, w_n), w_i \in \Sigma^*$. Then, we can construct, in polynomial time in $|e| + \sum_i |w_i|$, an n-dim CSRE $e[\mathbf{w}] = (\mathbf{u}_1, \ldots, \mathbf{u}_n, \varphi(\mathbf{x}_1, \ldots, \mathbf{x}_n))$ such that $\mathbf{u}_i[1] = w_i$, for $1 \leq i \leq n$ and for every $j \in \mathbb{N}$, $L(e[\mathbf{w}, j]) = \{\mathbf{v} | \mathbf{v}[i \leftarrow w_i^j \mathbf{v}[i]] \in L(e)\}$, where $e[\mathbf{w}, j] = (\mathbf{u}_1[1 \leftarrow \epsilon], \ldots, \mathbf{u}_n[1 \leftarrow \epsilon], (\bigwedge_{1 \leq i \leq n} \mathbf{x}_i[1] = j \wedge \varphi(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)))$.*

Proof. First suppose, we only quotient the 1st component by iterations of a w_1. We can do this by simply following the proof of Lemma 59 almost identically. In the resulting n-CSRE, the increase in size is according to the items 2, 3 and 4 listed at the end of that proof. The increase is additive, increasing by a polynomial in the sum of the lengths of the words in $\mathbf{u_1}$ and $|w_1|$, and provides a variable y_1 (as indicated in item 1) to control the number of iterations of w_1 by which the first component is to be quotiented.

Repeating this now for the second component, again following the same line as in the proof of Lemma 59 will result in an additive increase in size by a polynomial in the sum of the lengths of the words in $\mathbf{u_2}$ and $|w_2|$ and provide a variable y_2 to control the iterations of w_2 and so on. After n steps on is left with a CSRE which has only grown polynomially in $|\mathbf{u}_1|+\ldots|\mathbf{u}_n|+|w_1|+\ldots+|w_n|$.

We can then existentially quantify the $y_2 \ldots y_n$, add a conjunct of the form $y_1 = y_2 \wedge y_1 = y_3 \ldots y_1 = y_n$ and drop the empty words corresponding to the positions referred to by $y_2, \ldots, y_n$ to arrive at the desired formula. □

We now have all the technical ingredients necessary to study the stability of sets of configurations defined by n-dim CSREs. We say that a set C of configurations of the MPDS M is CSRE representable if there is a function f that maps any state $q \in Q$ of M to an n-dim CSRE e_q such that $(q, w_1, \ldots, w_n) \in C$ iff $(w_1, \ldots, w_n) \in L(f(q))$.

Acceleration of CSRE representable set of configurations

Let $M = (n, Q, \Gamma, \Delta)$ be an MPDS. We now examine the sets $Post_{\Delta_i^*}(C)$ and $Post_{\theta^*}(C)$ where Δ_i is a set of transitions on the i-th stack of M and $\theta \in \Delta^*$ where C is a CSRE representable set of configurations.

Theorem 26. *For every transition sequence $\theta \in \Delta^*$, the class of CSRE representable sets of configurations is effectively closed under $Post_{\theta^*}$. Further post set can be computed in time polynomial in the size of θ and $|M|$.*

Proof. Let θ be a sequence of transitions of the form $(q_0, op_0, q_0')(q_1, op_1, q_1') \cdots (q_m, op_m, q_m')$ and C be a CSRE representable set of configurations. Since $Post_{\theta^*}(C_1 \cup C_2) = Post_{\theta^*}(C_1) \cup Post_{\theta^*}(C_2)$, we can assume w.l.o.g that C consists of configurations of the form $(q, w_1, \ldots, w_n)$ for some fixed $q \in Q$. Let f be a function from Q to n-dim CSREs such that $L(f(p)) = \{(w_1, \ldots, w_n) \mid (q, w_1, \ldots, w_n) \in C\}$ if $p = q$ and $L(f(p)) = \emptyset$ otherwise. Next, we assume that $f(q) = (\mathbf{u}_1, \ldots, \mathbf{u_n}, \varphi(\mathbf{x}_1, \ldots, \mathbf{x}_n))$. The proof proceeds by cases.

Case 1: Let us assume $q_i' \neq q_i$ for some $i \in [1..m]$ or $q_0 \neq q$. In this case the sequence of transitions cannot be executed and hence $Post_{\theta^*}(C) = C$.

Case 2: Let us assume $q_0 \neq q_m'$, $q_0 = q$ and $q_i' = q_i$ for all $i \in [1..m]$. In this case, the sequence of transitions cannot be iterated more than once and so we have $Post_{\theta^*}(C) = Post_\theta(C) \cup C$. We now examine the set $Post_\theta(C)$. First, let us assume that $\mathtt{Summ}(i, \theta) = \perp$ for some $i \in [1..n]$. Then it is easy to see that $Post_\theta(C) = \emptyset$ and hence $Post_{\theta^*}(C) = C$.

Let us assume now that $\mathtt{Summ}(i, \theta) = (u_i, v_i)$ is well-defined for all $i \in [1..n]$. We can construct a n-CSRE e' such that $(q_m', w_1, \ldots, w_n) \in Post_{\theta^*}(C)$ iff $(w_1, \ldots, w_n) \in L(e')$ in two steps: Let $e_1 = f(q)[(u_1, u_2, \cdots, u_n), 1]$. This left quotients component i by u_i as required and the size of e_1 is polynomial in the size of θ, M and $f(q)$. Let us assume that e_1 is of the form $((\epsilon, w_2, \ldots, w_{\ell_1}), \ldots, (\epsilon, w_{\ell_{n-1}+2}, \ldots, w_{\ell_n}), \varphi''(x_1, \ldots, x_{\ell_n}))$. Next, we simultaneously add the content v_i to stack i, $1 \leq i \leq n$ as follows: Let the n-CSRE e' be $((v_1, \epsilon, w_2, \ldots, w_{\ell_1}), \ldots, (v_n, \epsilon, w_{\ell_{n-1}+2}, \ldots, w_{\ell_n}), \varphi'(y_1, x_1, \ldots, x_{k_1}, \ldots, y_n, x_{\ell_{n-1}+1}, \ldots, x_{\ell_n}))$ where $\varphi' = \varphi'' \wedge \bigwedge_{1 \leq h \leq n} y_i = 1$.

It is easy to see that $Post_{\theta^*}(C)$ is CSRE representable by the function f' defined as follows $f'(q) = f(q)$, $f'(q'_m) = e'$, and $\mathrm{L}(f'(p)) = \emptyset$ for all $p \notin \{q, q'_m\}$. Observe that the construction of $Post_{\theta^*}(C)$ is done in polynomial time in the sizes of θ, M and $f(q)$.

Case 3: Let us assume $q_0 = q'_m$, $q_0 = q$ and $q'_i = q_i$ for all $i \in [1..m]$. In this case, the sequence of transitions forms a loop in the control flow graph of M and hence the sequence may possibly be iterated. Observe that if the function $\mathtt{Summ}(i,\theta) = \perp$ for some $i \in [1..n]$, then $Post_{\theta^*}(C) = C$. Hence, let us assume that $\mathtt{Summ}(i,\theta) = (u_i, v_i)$ for all $i \in [1..n]$ so that it is well-defined for each i. Lemma 54 suggests that we should examine when $\mathtt{Summ}(i,\theta\theta)$ is defined for all i. Indeed, if $\mathtt{Summ}(i,\theta\theta)$ is undefined for some $i \in [1..n]$, then $Post_{\theta^*}(C) = Post_{\theta}(C) \cup C$ (which can be computed as shown in the previous case). So, let us further assume that $\mathtt{Summ}(i, \theta\theta)$ is well-defined for all $i \in [1..n]$. Hence, the function $\mathtt{Iter}(i,\sigma)$ is also well-defined. Let us assume that $\mathtt{Iter}(i,\sigma) = (x_i, y_i)$

We then construct a n-CSRE e' such that $(q'_m, w_1, \ldots, w_n) \in Post_{\theta^+}(C)$ iff $(w_1, \ldots, w_n) \in L(e')$ in a sequence of steps: First we construct the n-CSRE expression $e_1 = f(p)[(u_1, u_2, \cdots, u_n), 1]$. Let us assume that e_1 is of the form $((\epsilon, w_2, \ldots, w_{\ell_1}), \ldots, (\epsilon, w_{\ell_{n-1}+2}, \ldots, w_{\ell_n}), \varphi_1(x_1, \ldots, x_{\ell_n}))$. Observe that the size of e_1 is polynomial in the sizes of θ, M and $f(q)$ and it simultaneously left quotients component i by u_i. Now, we must simultaneously left-quotient the ith component by x_i^j, for a fixed j and then follow this by adding simultaneously y_i^j to component i (for the same j) and then add simultaneously v_i to component i ($1 \le i \le n$). To achieve this we begin by applying Lemma 61 to e_1 to construct the n-CSRE expression $e_2 = e_1[(x_1, \ldots, x_n)]$. Observe that the size of e_2 is also polynomial in the sizes of θ, M and $f(q)$. Let us assume that e_2 is of the form $((\epsilon, w'_2, \ldots, w'_{j_1}), \ldots, (\epsilon, w'_{j_{n-1}+2}, \ldots, w'_{j_n}), \varphi_2(z_1, \ldots, z_{j_n}))$. We now exploit the parametrized nature of $e_1[(x_1, \ldots, x_n)]$ stated in Lemma 61. We let $e' = ((v_1, y_1, \epsilon, w'_2, \ldots, w'_{j_1}), \ldots, (v_n, y_n, \epsilon, w'_{j_{n-1}+2}, \ldots, w'_{j_n}), \varphi'(t_1, t'_1, z_1, \ldots, z_{k_1}, \ldots, t_n, t'_n, z_{\ell_{n-1}+1}, \ldots, z_{j_n}))$ where $\varphi' = \varphi_2 \wedge \bigwedge_{1 \le h \le n} t_i = 1 \wedge (z_1 = z_{j_1+1} = \cdots = z_{z_{j_{n-1}+1}} = t'_1 = t'_2 = \cdots = t'_n)$.

Finally, it is easy to see that $Post_{\theta^*}(C)$ is CSRE representable by the function f' such that $L(f'(q)) = L(f(q)) \cup L(e')$, and $\mathrm{L}(f'(p)) = \emptyset$ for all $p \notin \{q\}$. Observe that the size of $f'(q)$ is still polynomial in the sizes of θ, M and $f(q)$. □

Unfortunately, CSRE representable sets are not stable w.r.t. bounded context runs.

8.3 Acceleration of Bounded-Context-Switch Sets

In the following, we introduce the class of *constrained* rational languages (as an extension of constrained (or Parikh) automata languages to the settings of multi-dimensional words [91, 48]). We show that the class of constrained rational languages is stable with respect to bounded-context runs and simple loops. Even better, we show that the class of constrained rational languages is stable with respect to bounded-context-switch sets, a generalization of loops and contexts. The following section is structured as follows: First, we give the formal definition of the class of constrained rational language and state some of its properties. Then, we present the class of bounded-context-switch. Finally, we show that the class of constrained rational languages is stable with respect to acceleration by bounded-context-switch sets.

8.3.1 Constrained Rational Languages

A constrained automaton is a finite-state automaton augmented with a semi-linear set to filter (or restrict) the accepting runs. We assume that this semi-linear set is described by an existential Presburger formula. In the following, we extend this model to multi-dimensional words. Let $n \geq 1$ and $\Sigma_1, \ldots, \Sigma_n$ be n finite alphabets. Formally, a n-tape constrained finite-state automaton over $\Sigma_1, \ldots, \Sigma_n$ is defined as $\mathscr{C} = (A, \varphi)$ where $A = (Q, \Sigma_1, \ldots, \Sigma_n, \delta, q_0, F)$ is a n-tape finite-state automaton and φ is an existential Presburger formula such that $\mathit{FreeVar}(\varphi) = \delta$. Furthermore, we assume w.l.o.g. that if $(q, \mathbf{u}, q')$ is in δ then $|\mathbf{u}[1] \cdot \mathbf{u}[2] \cdots \mathbf{u}[n]| \leq 1$. The language of $\mathscr{C}$, denoted by $L(\mathscr{C})$, is the set of n-dim words $\mathbf{w}$ for which there is an accepting run π of A over w such that $\mathit{Parikh}(\pi) \models \varphi$. A n-dim language is *constrained* rational if it is the language of some n-tape constrained automaton. Let us state some properties about the class of constrained rational languages. These properties can be inferred from the properties of rational languages [34] and Parikh/constrained automata [91, 48, 146].

Lemma 62. *The class of constrained rational languages is closed under union and concatenation but not under intersection. The emptiness and membership problems are decidable while the emptiness of intersection problem is undecidable.*

Proof. We will now prove the closures w.r.t. constrained rational languages.

- **Union:** Let $\mathscr{A} = (A, \varphi_1)$ and $\mathscr{B} = (B, \varphi_2)$ be two constrained rational automata such that $A = (Q^A, \Sigma_1, \cdots, \Sigma_n, \delta^A, q_0^A, F^A)$ and $B = (Q^B, \Sigma_1, \cdots, \Sigma_n, \delta^B, q_0^B, F^B)$. Without loss of generality, we will assume that $Q^A \cap Q^B = \emptyset$ and hence $\delta^A \cap \delta^B = \emptyset$. We construct the constrained rational automaton $\mathscr{C} = (C, \varphi)$ such that $L(\mathscr{C}) = L(\mathscr{A}) \cup L(\mathscr{B})$ as follows: $C = (Q^A \cup Q^B \cup \{s_0\}, \Sigma_1, \cdots, \Sigma_n, \delta^C, s_0, F^A \cup F^B)$ where $\delta^C = \delta^A \cup \delta^B \cup \{(s_0, \epsilon^n, q_0^A), (s_0, \epsilon^n, q_0^B)\}$ and $\varphi = \varphi_1 \vee \varphi_2$. The, it is easy to see that $L(\mathscr{C}) = L(\mathscr{B}) \cup L(\mathscr{A})$. Note that the overall size of $\mathscr{C}$ is linear in size of $\mathscr{A}$ and $\mathscr{B}$.
- **Concatenation:** Let $\mathscr{A} = (A, \varphi_1)$ and $\mathscr{B} = (B, \varphi_2)$ be two constrained rational automata such that $A = (Q^A, \Sigma_1, \cdots, \Sigma_n, \delta^A, q_0^A, F^A)$ and $B = (Q^B, \Sigma_1, \cdots, \Sigma_n, \delta^B, q_0^B, F^B)$. Without loss of generality, we will assume that $Q^A \cap Q^B = \emptyset$ and hence $\delta^A \cap \delta^B = \emptyset$. We construct the constrained rational automaton $\mathscr{C} = (C, \varphi)$ such that $L(\mathscr{C}) = L(\mathscr{A}) \cdot L(\mathscr{B})$ as follows: $C = (Q^A \cup Q^B \cup, \Sigma_1, \cdots, \Sigma_n, \delta^C, q_0^A, F^B)$, $\delta^C = \delta^A \cup \delta^B \cup \{(q, \epsilon^n, q_0^B) \mid q \in F^A\}$ and $\varphi = \varphi_1 \wedge \varphi_2$. The correctness of the construction follows immediately from the construction. Note that the over all size of $\mathscr{C}$ is linear in size of $\mathscr{A}$ and $\mathscr{B}$.
- **Emptiness:** Emptiness follows directly from the fact that n-tape constrained automaton $\mathscr{A}$ over $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ can be seen as 1-tape constrained automaton $\mathscr{B}$ over the alphabet $(\Sigma_1 \cup \{\epsilon\}) \times \cdots \times (\Sigma_n \cup \{\epsilon\})$ for which the emptiness problem is well-known to be decidable [91, 48]).
- **Membership:** This follows immediately from the decidability of the emptiness problem and the closure of the class of constrained languages wrt. intersection with regular languages (see Lemma 63). In fact any multi-dimensional language consisting of finite words is regular.

- **Undecidability of emptiness of intersection:** This directly follows from the fact that emptiness of intersection of rational language is undecidable.

□

We can extend the permutation, projection and composition operations to the context of constrained rational languages in the straightforward manner. We also show the same closure properties as in the case of rational languages.

Lemma 63. *The class of constrained rational languages is closed under permutation, projection, composition and intersection with regular languages.*

Proof. We will now prove the claim of the above lemma:

- **Permutation:** Given a constrained rational automaton $\mathscr{A} = (A, \varphi)$ with $A = (Q^A, \Sigma_1, \cdots, \Sigma_n, \delta^A, q_0^A, F^A)$, and a permutation $h : [1..n] \mapsto [1..n]$, we can construct another constrained rational automaton $\mathscr{B} = (B, \varphi')$ with $B = (Q^A, \Sigma_{h(1)}, \cdots, \Sigma_{h(n)}, \delta^B, q_0^A, F^A)$ such that $L(\mathscr{B}) = \{(w_{h(1)}, w_{h(2)}, \cdots, w_{h(n)}) \mid (w_1, w_2, \cdots, w_n) \in L(\mathscr{A})\}$. The states, initial state and final states of out B automaton are the same as that of A. Along with the transition relation δ^B, we will also construct below a bijective function $g : \delta^A \mapsto \delta^B$ mapping the transitions of A to that of B. This will be useful for constructing the Presburger formula. For any $\tau = (q, (a_1, \cdots, a_n), q') \in \delta^A$, we add $\tau' = (q, (a_{h(1)}, \cdots, a_{h(n)}), q') \in \delta^B$ and let $g(\tau) = \tau'$. Let us assume that $\varphi(t_1, t_2, \ldots, t_m)$ where $t_1, t_2, \ldots, t_m \in \delta^A$ are the free variables of φ. Then the formula φ' as $\varphi(t_1/g(t_1), t_2/g(t_2), \ldots, t_m/g(t_m))$. It is then easy to see that $L(\mathscr{B}) = \{(w_{h(1)}, w_{h(2)}, \cdots, w_{h(n)}) \mid (w_1, w_2, \cdots, w_n) \in L(\mathscr{A})\}$. Note that the over all size of $\mathscr{B}$ is linear in size of $\mathscr{A}$
- **Projection:** Given a constrained rational automaton $\mathscr{A} = (A, \varphi)$ with $A = (Q^A, \Sigma_1, \cdots, \Sigma_n, \delta^A, q_0^A, F^A)$ and a set of indices $\iota = \{i_1 < i_2 < \ldots i_m\} \subset \{1, \ldots, n\}$, we can construct another constrained rational automaton $\mathscr{B} = (B, \varphi')$ with $B = (Q^B, \Sigma_2, \cdots, \Sigma_n, \delta^B, q_0^B, F^B)$ such that $L(\mathscr{B}) = \{(w_{i_1}, w_{i_2}, \cdots, w_{i_m}) \mid (w_1, w_1, \cdots, w_n) \in L(\mathscr{A})\}$. The states of B are given by $Q^B = Q^A \cup \{p_\tau \mid \tau \in \delta^A\}$, i.e. we add $|\delta^A|$ new states to the automaton along with the states of A. The initial state of B is the same as the initial state of A i.e. $q_0^B = q_0^A$. Similarly the final state of B is the same as the final states of A i.e. $F^A = F^B$. Along with the transition relation, we will also describe a function $g : \delta^A \mapsto \delta^B$ that will help us to construct the Presburger formula. For any $\tau = (q, (a_1, \cdots, a_n), q') \in \delta^A$, we add $\tau_1 = (q, (a_{i_1}, \cdots, a_{i_m}), p_\tau) \in \delta^B$ and $\tau_2 = (p_\tau, (\epsilon, \cdots, \epsilon), q') \in \delta^B$. We let $g(\tau) = \tau_1$. Now the Presburger formula φ' is defined as $\varphi' = \varphi(\tau_1/g(\tau_1), \tau_2/g(\tau_2), \ldots, \tau_\ell/g(\tau_\ell))$, where $\tau_1 \cdots \tau_\ell$ are the transitions of A. Note that the complexity of such an construction is at most polynomial. The number states of B is the sum $|\delta^A|$ and $|A|$. Furthermore, we have that δ^A can be in the order of $Q^2 \times \Sigma$ where $\Sigma = \bigcup_{i \in [1..n]} \Sigma_i$.
- **Composition:** Let $\mathscr{A} = (A, \varphi_1)$ and $\mathscr{B} = (B, \varphi_2)$ be two constrained rational automata such that $A = (Q^A, \Sigma_1, \cdots, \Sigma_n, \delta^A, q_0^A, F^A)$ and $B = (Q^B, \Sigma_1, \Sigma'_2, \cdots, \Sigma'_m, \delta^B, q_0^B, F^B)$. Then, we can construct a constrained rational automaton $\mathscr{C} = (C, \varphi)$ such that $C = (Q^C, \Sigma_1, \cdots, \Sigma_n, \Sigma'_2, \cdots, \Sigma'_m, \delta^C, q_0^C, F^C)$ and $L(\mathscr{C}) = \{(w_1, \cdots, w_n, u_2, \cdots, u_m) \mid (w_1, \cdots, w_n) \in L(\mathscr{A}) \wedge (w_1, u_2, \cdots, u_m) \in L(\mathscr{B})\}$. Observe that we assume that the composition between $\mathscr{A}$ and $\mathscr{B}$ is done over the first tape since the class of constrained rational languages is closed under permutation.

The states of the automata C are given by $Q^C = Q^A \times Q^B$, the initial state is given by $q_0^C = (q_0^A, q_0^B)$ and the set of final states is given by $F^C = F^A \times F^B$. We will define the transition relation δ^C along with two partial functions $g : \delta_C \mapsto \delta_A$ and $h : \delta_C \mapsto \delta_B$. For every $q, q' \in Q^A$, p, $p' \in Q^B$ and $a_1 \in \Sigma_1$, if $\tau_1 = (q, (a_1, \cdots, a_n), q') \in \delta^A$ and $\tau_2 = (p, (a_1, b_2, \cdots, b_m), p') \in \delta^B$ then we add $\tau' = ((q,p), (a_1, \cdots, a_n, b_2, \cdots, b_m), (q', p')) \in \delta^C$. We also let $g(\tau') = \tau_1$ and $h(\tau') = \tau_2$. For any $\tau = (q, (\epsilon, a_2, \cdots, a_n), q') \in \delta^A$, we add $\tau' = ((q,p), (\epsilon, a_2, \cdots, a_n, \epsilon^{m-1}), (q', p)) \in \delta^C$ and we let $g(\tau') = \tau$. Similarly for any $\tau = (q, (\epsilon, b_2, \cdots, b_m), q') \in \delta^B$, we add $\tau' = ((q,p), (\epsilon^n, b_2, \cdots, b_n), (q, p')) \in \delta^C$ and we also let $h(\tau') = \tau$. For any $\tau \in \delta_A$, we will refer to $g^{-1}(\tau)$ to mean $g^{-1}(\tau) = \{\tau' \mid \tau' \in \delta_C \wedge g(\tau') = \tau\}$. We use similar notation for $h^{-1}(\tau)$ with $\tau \in \delta_B$.
Now the Presburger formula φ is defined as $\varphi = \varphi_1((\tau / \sum_{\tau' \in g^{-1}(\tau)} \tau')_{\tau \in \delta^A}) \wedge \varphi_1((\tau / \sum_{\tau' \in h^{-1}(\tau)} \tau')_{\tau \in \delta^B})$.

- **Intersection with regular:** Given a constrained rational automata $\mathscr{A} = (A, \varphi')$ with $A = (Q^A, \Sigma_1, \cdots, \Sigma_n, \delta^A, q_0^A, F^A)$ and a regular n-dim language L. Since the intersection operator is distributive over the union operator and the closure of the class of rational languages under union, we can assume w.l.o.g. that L is of the form $L(A_1) \times L(A_2) \times \cdots \times L(A_n)$ where for every $i \in [1..n]$, $A_i = (Q_i, \Sigma_i, \delta_i, q_i^{init}, F_i)$ is a finite state automaton. Let $A'_i = (Q_i, \Sigma_1, \ldots, \Sigma_n, \delta'_i, q_i^{init}, F_i)$ be the n-tape automaton such that (1) $\delta'_i \subseteq \{\epsilon\}^{i-1} \times \Sigma_i \cup \{\epsilon\}^{n-i}$ and (2) $(q, (\epsilon^{i-1}, a, \epsilon^{n-i}), q') \in \delta'_i$ iff $(q, a, q') \in \delta_i$. Then, it is easy to set that $L \cap L(\mathscr{A}) = L(((((\mathscr{A} \circ_{(1,1)} (A'_1, true)) \circ_{(2,2)} (A'_2, true)) \circ_{(3,3)} (A'_3, true)) \cdots \circ_{(n,n)} A'_n))$. Since the class of constrained the class of constrained automata is closed under composition, we obtain that it is possible to construct a constrained rational automaton $\mathscr{A}'$ such that $L(\mathscr{A}') = L \cap L(\mathscr{A})$. Furthermore, it is easy to see that the size of A' is exponential in $|A_1| + |A_2| + \cdots + |A_n|$ and polynomial in the size of $\mathscr{A}$. Observe that in fact the complexity is polynomial in $(|A_1| + |A_2| + \cdots + |A_n|)^n$ and the size of $\mathscr{A}$.

□

The complexity of permutation, projection, composition is at most polynomial in size of input automata whereas the intersection with regular languages is at most exponential in the size of the description of the regular language and polynomial in the size of constrained rational automaton.

Acceleration of Bounded-Context-Switch Sets

Let $M = (n, Q, \Gamma, \Delta)$ be an MPDS. A *bounded-context-switch* set over M is defined by a tuple $\Lambda = (\tau_0, \tau_1, \ldots, \tau_{2m})$ with $m \in \mathbb{N}$ where (1) for every $i \in [0..m]$, we have $\tau_{2i} \subseteq \Delta_{i_j}$ for some $i_j \in [1..n]$ with $i_0 = i_{2m}$, and (2) for every $i \in [0..(m-1)]$, $|\tau_{2i+1}| = 1$. The size of Λ is defined as the sum of the sizes of the finite sets τ_j for all $j \in [0..2m]$. The set of sequences of transitions recognized by Λ, denoted by $L(\Lambda)$, is $\tau_0^* \tau_1 \tau_2^* \cdots \tau_{2m}^*$. Observe that when $m = 0$ and $\tau_0 = \Delta_i$ for some $i \in [1..n]$, $L(\Lambda)$ corresponds to a context associated to the stack i. And whenever $\tau_{2i} = \emptyset$ for all $i \in [0..m]$, $L(\Lambda)$ is a sequence of transitions. Thus, bounded-context-switch sets generalize both loops and contexts. Observe that dropping one of τ_{2i+1} from the definition of Λ will allow the simulation of unbounded unrestricted context-switch sequences and hence leads to the undecidability of the simple reachability problem. Next, we state our main theorem:

Theorem 27. *Let M be an MPDS and $\Lambda = (\tau_0, \tau_1, \ldots, \tau_{2m})$ be a bounded-context-switch set over M. For every constrained rational set of configurations C, $Post_{L(\Lambda)^*}(C)$ is a constrained rational set and effectively constructible.*

The rest of this section is dedicated to the proof of Theorem 27. First we prove an extension of Lemma 52 that shows that in addition to computing pairs of the form (q, u, q', u) such that there is a run π from (q, u) to (q', u') one may in addition keep track of the number iterations of $L(\Lambda)$ seen along π.

Lemma 64. *Let $\mathscr{P} = (Q, \Sigma, \Gamma, \delta, q_0)$ be an PDS and $\Lambda = (\tau_0, \tau_1, \ldots, \tau_{2m})$ be a bounded-context-switch set over $\mathscr{P}$ such that $\tau_j \subseteq \delta$. Let $\sharp$ be a special symbol not included in Γ. Then it is possible to construct, in exponential time in the sizes of $\mathscr{P}$ and Λ, an 5-tape finite-state automaton $T = (Q_T, Q, \Gamma, Q, \Gamma, \{\sharp\}, \delta_\pi, q_0, F_T)$ such that $(q, u, q', v, \sharp^m) \in L(T)$ iff $(q, u) \xrightarrow{\pi}_{\mathscr{P}} (q', v)$ for some sequence $\pi \in (L(\Lambda))^m$. Furthermore, the size of T is exponential in the sizes of $\mathscr{P}$ and Λ.*

Proof. The proof of this lemma is based on the combination of the proof of Lemma 52 with the fact the Parikh images of context-free languages can be effectively realised as a regular languages. The aim is to build a rational automata T such that $L(T) = \{(q, u, q', v, \sharp^m) \mid (q, u) \xrightarrow{\pi}{}^*_{\mathscr{P}} (q' v) \wedge \pi \in (L(\Lambda))^m\}$. It is easy to see that for any two configurations (q, u) and (q', v) in $\mathscr{P}$, if $(q, u) \xrightarrow{\Lambda^m}{}^* (q' v)$ then clearly such a run can be split as $(q, u'w) \xrightarrow{\sigma_1}{}^*{}_w (q'', w) \xrightarrow{\sigma_2}{}^*{}_w (q', v'w)$ for some $w \in (\Gamma \setminus \bot)^* \bot$, $\sigma_1.\sigma_2 = \Lambda^m$ and some u', v' such that $u = u'w$, $v = v'w$. Let $u' = a_1 \cdots a_\ell$ and $v' = b_1 \cdots b_m$ then clearly we can further split the run into increasing and decreasing phase. If $w \neq \bot$, then we can split it as

$$(q, a_1 \cdots a_n w) \xrightarrow{\sigma_1}{}^*{}_w (q_1, a_2 \cdots a_n w) \xrightarrow{\sigma_2}{}^*{}_w \cdots \rightarrow^*{}_w (q'', w) \rightarrow^*{}_w (q'_1, b_n w) \rightarrow^*{}_w \cdots \xrightarrow{\sigma_n}{}^*{}_w (q, v'w)$$

Such that $\sigma_1 \cdots \sigma_n = \Lambda^m$. Or if $w = \bot$ then we can split it as

$$(q, a_1 \cdots a_n \bot) \xrightarrow{\sigma_1}{}^*{}_\bot (q_1, a_2 \cdots a_n \bot) \xrightarrow{\sigma_2}{}^*{}_\bot \cdots \rightarrow^*{}_\bot (q''_1, \bot) \rightarrow_\bot (q''_2, \bot) \rightarrow^*{}_\bot (q'_1, b_n \bot) \rightarrow^*{}_\bot \cdots \xrightarrow{\sigma_n}{}^*{}_\bot (q, v \bot)$$

Note that these intermediate sequences σ_i need not fall at the Λ boundary. Given a run (say $(q, a_1 \cdots a_n w) \xrightarrow{\sigma_1}{}^*{}_w (q_1, a_2 \cdots a_n w)$), the number of times Λ sequences are executed and completed in this part of run is regular (since this can be captured as parikh image of a pushdown system, which is known to be regular [62]). However only this information is not sufficient to concatenate sequence of such runs. We also need to keep track of the position inside Λ where the run ended.

For this purpose, we will construct a new pushdown automaton $\mathbf{P} = (S, \{\sharp\}, \Gamma, \Delta, p_0, S)$ that has embedded in it the information of which position in Λ it is executing. The states of such a pushdown automaton are $S = Q \times [0..m]$. The initial and final state of this pushdown is not important at this point of time. In the newly constructed pushdown system, the input alphabets (of the original system) are ignored. Further each time a context switch transition of the form $\tau \in \tau_{2i-1}, i \in [1..m]$ is performed, the current position inside Λ is updated. The pushdown system only includes the transitions from the bounded-context-switch set. The transition relation Δ is defined as follows. We will along with the definition of transition also define a mapping $g : \Delta \mapsto \delta \cup \{\epsilon\}$, which will later be used in the proof.

1. for all $\tau = (p, \mathbf{Pop}_i(a), b, p') \in \tau_{2i}, i \in [0..m]$, $\tau' = ((p,i), \mathbf{Pop}(a), \epsilon, (p', i)) \in \Delta$ and $g(\tau') = \tau$
2. for $\tau = (p, \mathbf{Pop}_i(a), b, p') \in \tau_{2i-1}, i \in [1..m]$, $\tau' = ((p,i), \mathbf{Pop}(a), \epsilon, (p', i+1)) \in \Delta$ and $g(\tau') = \tau$
3. for all $\tau = (p, \mathbf{Push}_i(a), b, p') \in \tau_{2i}, i \in [0..m]$, $\tau' = ((p,i), \mathbf{Push}(a), \epsilon, (p', i)) \in \Delta$ and $g(\tau') = \tau$
4. for $\tau = (p, \mathbf{Push}_i(a), b, p') \in \tau_{2i-1}, i \in [1..m]$, $\tau' = ((p,i), \mathbf{Push}(a), \epsilon, (p', i+1)) \in \Delta$ and $g(\tau') = \tau$
5. for all $\tau = (p, \mathbf{Int}_i(a), b, p') \in \tau_{2i}, i \in [0..m]$, $\tau' = ((p,i), \mathbf{Int}, \epsilon, (p', i)) \in \Delta$ and $g(\tau') = \tau$
6. for $\tau = (p, \mathbf{Int}_i(a), b, p') \in \tau_{2i-1}, i \in [1..m]$, $\tau' = ((p,i), \mathbf{Int}, \epsilon, (p', i+1)) \in \Delta$ and $g(\tau') = \tau$
7. for all $\tau = (p, \mathbf{Zero}_i(a), b, p') \in \tau_{2i}, i \in [0..m]$, $\tau' = ((p,i), \mathbf{Zero}, \epsilon, (p', i)) \in \Delta$ and $g(\tau') = \tau$
8. for $\tau = (p, \mathbf{Zero}_i(a), b, p') \in \tau_{2i-1}, i \in [1..m]$, $\tau' = ((p,i), \mathbf{Zero}, \epsilon, (p', i+1)) \in \Delta$ and $g(\tau') = \tau$
9. We also add $\tau = ((p,m), \mathbf{Int}, \sharp, (p,0)) \in \Delta$ and we let $g(\tau) = \epsilon$

We extend the function g in the straightforward manner to sequence of transitions. It is easy to see that, $\pi = ((p,i),u) \xrightarrow{\sigma}_{\mathbf{P}} ((p',j),v)$ iff $(p,u) \xrightarrow{g(\sigma)}_{\mathscr{P}} (p,v)$, further $\Sigma(\sigma) = \sharp^k$ if $g(\sigma) \in \tau_{2i}^* \tau_{2i+1} \tau_{2i+2}^* \tau_{2i+3} \cdots \tau_{2m}^* \Lambda^k \tau_0^* \tau_1 \cdots \tau_{2j}^*$, for some $k \in \mathbb{N}$. We will call a sequence of the form $\tau_{2i}^* \tau_{2i+1} \tau_{2i+2}^* \tau_{2i+3} \cdots \tau_{2m}^* \Lambda^k \tau_0^* \tau_1 \cdots \tau_{2j}^*$ as $(2i, \Lambda^k, 2j)$

For any $i, j \in [0..m]$, let $L^-((p,i),a,(p',j)) = \{\Sigma(\sigma) \mid ((p,i),a) \xrightarrow{\sigma}_{\mathbf{P}} ((p',j),\epsilon)\}$. The above language recognises $\Sigma(\sigma) = \sharp^k$ if there is an execution sequence that starts at state p and position τ_{2i} in Λ, executes rest of sequence of Λ, iterates Λ^{k-1}, executes the Λ sequence up to τ_{2j}, reaches state p' and in the process removes the letter a from the stack. Clearly such a language is effectively regular and at most exponential in size of $\mathbf{P}$. This follows from the fact that Parikh image of pushdown automata are effectively regular [63]. We will assume the finite state automaton recognizing such a language to be $B^-((p,i),a,(p',j))$.

Similarly let $L^+((p,i),a,(p',j)) = \{\Sigma(\sigma) \mid ((p,i),\epsilon) \xrightarrow{\sigma}_{\mathbf{P}} ((p',j),a)\}$. Let $L^=((p,i),(p',j)) = \{\Sigma(\sigma) \mid ((p,i),\bot) \xrightarrow{\sigma}_{\mathbf{P}} ((p',j),\bot)\}$. These languages are also regular for the same reason as above. We will assume the finite state automata recognizing the Parikh image of $L^+((p,i),a,(p',j))$ to be $B^+((p,i),a,(p',j))$ and we will assume the finite state automata recognising the Parikh image of $L^=((p,i),(p',j))$ to be $B^=((p,i),(p',j))$.

We now show how to construct the rational automata T. Towards this, we will introduce some notations. For any $?x \in \{+,-,=\}$, we will refer to states of $B^{?x}$ automata as $State(B^{?x})$. Similarly, we will refer to initial, final states and transitions as $Initial(B^{?x})$, $Final(B^{?x})$ and $\Delta(B^{?x})$ respectively. We will further assume that the states of such automata are distinct. We will simply use $St^=$,St^-, and St^+ to refer to set of all states of all possible $B^=$, B^- and B^+ automata respectively.

The idea behind construction of rational automata T is as follows. We need to ensure that $L(T) = \{(q,u,q',v,\sharp^m) \mid (q,u) \xrightarrow{\pi}_{\mathscr{P}} (q',v) \wedge \pi \in (L(\Lambda))^m\}$. The construction of T amounts to guessing these intermediate points of the decreasing, zero and increasing phase, while keeping track of the Parikh image of the languages generated during these runs on the 5^{th} tape.

Now the states of T is given by $Q^T = ((Q \times [0..m]) \cup St^-) \times ((Q \times [0..m]) \cup St^+ \cup \{\bot\}) \cup St^= \cup \{s, f, e\}$. It contains the states to recognise the decreasing slope, increasing slope, optional zero phase and a final state. The first component of the state is used for the decreasing phase and the second component of the state is used to simulate the increasing phase. When the decreasing phase is active, the second component is $\bot$. The initial state is s and set of final states is $F_T = \{f\}$.

We add $(s,(q,\epsilon,\epsilon,\epsilon,\epsilon),((q,0),\bot)) \in \delta_T$ to start the simulation of the decreasing phase, starting from q. We add for all $i,j \in [0..m]$, $a \in \Gamma \setminus \{\bot\}$ and $q,q' \in Q$, we add $(((q,i),\bot),(\epsilon,a,\epsilon,\epsilon,\epsilon),(\mathit{Initial}(B^-((q,i),a,(q',j)),\bot))) \in \delta_T$, this guesses the points of the decreasing phase that removes a from the stack. Similarly we add $((\mathit{Final}(B^-((q,i),a,(q',j))),\bot),(\epsilon,\epsilon,\epsilon,\epsilon,\epsilon),((q',j),\bot)) \in \delta_T$, this ensures that on completing the simulation of automaton, the control returns back. The above transitions have the following effect. Towards simulating the automata, we add for all $(p,a,p') \in \Delta(B^-((q,i),a,(q',j)))$, the transition $((p,\bot),(\epsilon,\epsilon,\epsilon,\epsilon,a),(p',\bot)) \in \Delta'$.

The effect of these set of transitionsan be described as, from the initial state s on reading a state q on tape-1, we start the decreasing phase by entering the state $((q,0),\bot)$. From any state of the form $((q,i),\bot)$, on reading a letter a from tape-2, it guesses the resulting state (q',j) of a run that ends up consuming a from stack and simulates the automata $B^-((q,i),a,(q',j))$. We return to (q',j) after completing to simulate the automata. It is easy to see that $s \xrightarrow{(q,\epsilon^4))}{}^* ((q,0),\bot) \xrightarrow{(\epsilon,u,\epsilon,\epsilon,\sharp^m)}{}^* ((q',j),\bot)$ iff $((q,0),u) \xrightarrow{\sigma}{}^*_{\mathbf{P}} ((q',j),\epsilon)$, with $\sigma \in (2i,\Lambda^{m-1},2j)$.

Similarly we add set of transitions to deal with the increasing slope. We add $(((q,i),\bot),(\epsilon,\epsilon,q',\epsilon,\epsilon),((q,i),(q',m))) \in \delta_T$. This transition starts the simulation of increasing slope. We add for all $k \in [0..m]$, $i,j \in [0..m]$, $a \in \Gamma$ and $p,q,q' \in Q$, the transition $(((p,k),(q,i)),(\epsilon,\epsilon,\epsilon,a,\epsilon),((p,k),\mathit{Initial}(B^+((q',j),a,(q,i))))) \in \delta_T$. Similarly we add $(((p,k),\mathit{Final}(B^+((q',j),a,(q,i)))),(\epsilon,\epsilon,\epsilon,\epsilon,\epsilon),((p,k),(q',j))) \in \delta_T$. Towards simulating the automata, we add for all $(p,\#,p') \in \Delta(B^+((q,i),a,(q',j)))$, the transition $(((q,k),p),(\epsilon,\epsilon,\epsilon,\epsilon,\#),((q,k),p')) \in \Delta'$. We also add $((q,k),(q,k),\epsilon^5,e) \in \Delta'$ and $(e,(\epsilon,a,\epsilon,a,\epsilon),e) \in \Delta'$ and $(e,(\epsilon,\bot,\epsilon,\bot,\epsilon),f) \in \Delta'$. It is easy to see that $((q',j),\bot) \xrightarrow{(\epsilon,\epsilon,q,\epsilon,\epsilon)}{}^* ((q',j),(q,m)) \xrightarrow{(\epsilon,\epsilon,\epsilon,\nu,\#^m)}{}^* ((q',j),(q',j)) \rightarrow^* (e) \xrightarrow{(\epsilon,w,\epsilon,w,\epsilon)}{}^* (e) \xrightarrow{(\epsilon,\bot,\epsilon,\bot,\epsilon)}{}^* (f)$ iff $((q',w\bot) \xrightarrow{\sigma}{}^* (q,v w\bot)$ with $\sigma \in (2i,\Lambda^{m-1},2j)$.

To simulate the zero tests, we add the following transitions $(((q',j),(q,i)),(\epsilon,\bot,\epsilon,\bot,\epsilon),\mathit{Initial}(B^=((q',j),(q,i)))) \in \Delta'$ for all $q,q' \in Q$, $i,j \in [0..m]$ and also the transitions $((\mathit{Final}(B^=((q',j),(q,i)))),(\epsilon^5),f) \in \Delta'$. It is easy to see that if $((q',j),(q,i)) \xrightarrow{(\epsilon,\bot,\epsilon,\bot,\#^n)}{}^* f$ iff $(q',\bot) \xrightarrow{\sigma}{}^* (q,\bot)$, with $\sigma \in (2i,\Lambda^{m-1},2j)$.

Correctness of such a construction can be got by simply following the argument similar to that in lemma 52 □

This allows us to prove Lemma 65, leading to the proof of Theorem 27.

From an MPDS M, one can construct a PDS M_i for each stack i, which simulates the moves of M on the ith stack while guessing, non-deterministically, the effect of the moves corresponding to the other stacks. Clearly, any run of M can be decomposed in to a tuple of runs, one per M_i. However, because of the special structure of $L(\Lambda)$, a converse of this statement is true for runs of the form $L(\Lambda)^*$. Any tuple of runs, one from each M_i, which agree on the number of iterations of $L(\Lambda)$ seen along the run, can be composed together to give a run M. We formalize these arguments now.

Let $i \in [1..n]$. For each transition $t = (q,op,q') \in \Delta$, we represent the effect of the transition t on the stack i by the transition $t|_i$ defined as follows: $t|_i = t$ if $t \in \Delta_i$, and $t|_i = (q,\mathbf{Int}_i,q')$ otherwise. We extend this operation to sets of transitions as follows: For a set $T \subseteq \Delta$, $T|_i = \{t|_i \mid t \in T\}$.

Let $M_i = (1,Q,\Gamma,\Delta_i \cup \{\tau_0,\tau_1,\ldots,\tau_{2m}\}|_i)$ be a PDS simulating the i-th stack while taking into account the effect of the other stack operations. We define also $\Lambda|_i$ to be the bounded

context/switches set defined by the tuple $(\tau_0|_i, \tau_1|_i, \ldots, \tau_{2m}|_i)$. Let T_i be the 5-tape finite state automaton resulting from the application of Lemma 64 to the PDA M_i and the bounded-context-switch set $\Lambda|_i$. Then synchronising the multi-tape automata T_i on the number of occurrences of the special symbol $\sharp$ provides a relation between any possible starting configuration $(q, u_1, \ldots, u_n)$ with any configuration $(q', v_1, \ldots, v_n)$ reachable from $(q, u_1, \ldots, u_n)$ of M by firing a sequence of transitions in $L(\Lambda)^*$.

Lemma 65. *Let $m \in \mathbb{N}$. Then, $(q, u_1, \ldots, u_n) \xrightarrow{\pi}_{\mathcal{M}} (q', v_1, \ldots, v_n)$ for some sequence $\pi \in (L(\Lambda))^m$ if and only if for every $i \in [1..n]$, $(q, u_i, q', v_i, \sharp^m) \in L(T_i)$.*

Proof. ($\Rightarrow$) Let $(q, u_1, \ldots, u_n) \xrightarrow{\pi}_{\mathcal{M}} (q', v_1, \ldots, v_n)$ be a run in M such that $\pi \in (L(\Lambda))^m$. By definition, $\pi|_i$ is clearly a valid sequence of moves in M_i ($|_i$ is extended to a sequence in straight forward manner) such that $\pi|_i \in (L(\Lambda|_i))^m$. We can easily prove by induction on length of the run that if $(q, u_1, \ldots, u_n) \xrightarrow{\pi}_{\mathcal{M}} (q', v_1, \ldots, v_n)$ then $(q, u_i) \xrightarrow{\pi|_i} (q', v_i)$. By definition of $L(T_i)$, this means that $(q, u_i, q', v_i, \sharp^m) \in L(T_i)$.

($\Leftarrow$)

The idea for the other direction is as follows. If $(q, u_i, q', v_i, \sharp^m) \in L(T_i)$, then by definition there are individual runs in pushdown systems such that this run is in $Ł(\Lambda|_i)^m$. Clearly $\Lambda|_i$ contains some real transitions and some projected transitions. Note that the real transitions are also part of our multi-pushdown system M. If we can find a shuffle of all the real transition sequences of each of these runs such that the ordering among the sequence obey the original ordering in the individual run, they are state-wise compatible and they form a word in $L(\Lambda)^m$, then it is easy to see that such a transition sequence is also a valid transition sequence in the multi-pushdown system M. We will attempt below to find one such valid sequence.

Let $j \in [1..n]$ be the index of the stack such that $\tau_0, \tau_{2m} \subseteq \Delta_j$. In the following, we will prove a stronger claim than the one stated by Lemma 65. In fact we will prove that if we can reach a particular state in a thread, we can reach it in all other thread and we can also reach it by iterating Λ in MPDS , that is for every $i \in [1..n]$, $(q, u_i, q_i, v_i, \sharp^m) \in L(T_i)$ then $(q, u_i, q_j, v_i, \sharp^m) \in L(T_i)$ for all $i \in [1..n]$, and $(q, u_1, \ldots, u_n) \xrightarrow{\pi}_{\mathcal{M}} (q_j, v_1, \ldots, v_n)$ for some sequence $\pi \in (L(\Lambda))^m$.

The proof is done by induction on m.

Base Cases: The base case is when $m = 0$. This means that $(q, u_i) = (q_i, v_i)$ for all $i \in [1..n]$ and hence $(q, u_1, \ldots, u_n) \xrightarrow{\pi}_{\mathcal{M}} (q_j, v_1, \ldots, v_n)$ for $\pi = \epsilon$. Furthermore it is easy to see that $(q, u_i, q_j, v_i, \sharp^m) \in L(T_i)$ holds for all $i \in [1..n]$.

Let us assume that $m = 1$. Let us assume that $(q, u_i, q_i, v_i, \sharp) \in L(T_i)$ for all $i \in [1..n]$. From Lemma 64, this implies that for every $i \in [1..n]$, $(q, u_i) \xrightarrow{\pi_i}_{M_i} (q_i, v_i)$ with $\sigma \in (L(\Lambda))$. Then we can rewrite the run $(q, u_i) \xrightarrow{\pi_i}_{M_i} (q_i, v_i)$ as follows $(q, u_i) \xrightarrow{\alpha_{(i,0)}}_{M_i} (p_{(i,1)}, u_{(i,1)}) \xrightarrow{\alpha_{(i,1)}}_{M_i} (p_{(i,2)}, u_{(i,2)}) \xrightarrow{\alpha_{(i,2)}}_{M_i} \cdots \xrightarrow{\alpha_{(i,2m-1)}}_{M_i} (p_{(i,2m)}, u_{(i,2m)}) \xrightarrow{\alpha_{(i,2m)}}_{M_i} (q_i, v_i)$ where $\alpha_{(i,\ell)} \in \tau_\ell^*$ for all $\ell \in [0..2m]$ such that $\alpha_{(i,2r-1)} = \tau_{2r-1}$ for all $r \in [1..m]$. Then, it is easy to see that for every $i \neq j$, $(p_{(i,2m)}, u_{(i,2m)}) \xrightarrow{\alpha_{(j,2m)}|_i}_{M_i} (q_j, v_i)$ with $v_i = u_{(i,2m)}$, this follows from definition of $\Lambda|_i$. This implies that $(q, u_i) \xrightarrow{\alpha_{(i,0)} \cdot \alpha_{(i,1)} \cdots \alpha_{(i,2m-1)} \cdot \alpha_{(j,2m)}|_i}_{M_i} (q_j, v_i)$. Hence, $(q, u_i, q_j, v_i, \sharp) \in L(T_i)$ for all $i \in [1..n]$ and so we have proved the first part of the claim.

Let $s \in [0..2m]$, i_s be the index of the active stack (i.e., $\tau_s \subseteq \Delta_{i_s}$). Then we can show that for every $s \in [1..2m-1]$, $(p_{(1,s)}, u_{(1,s)}, u_{(2,s)}, \cdots, u_{(n,s)}) \xrightarrow{\alpha_{(i_s,s)}}_M (p_{(1,s+1)}, u_{(1,s+1)}, u_{(2,s+1)}, \cdots, u_{(n,s+1)})$. Furthermore, we can show $(q, u_1, u_2, \ldots, u_n) \xrightarrow{\alpha_{(j,0)}}_M (p_{(1,1)}, u_{(1,1)}, u_{(2,1)}, \cdots, u_{(n,1)})$ and $(p_{(1,2m)}, u_{(1,2m)}, u_{(2,2m)}, \cdots, u_{(n,2m)}) \xrightarrow{\alpha_{(j,2m)}}_M (q_j, v_1, v_2, \ldots, v_n)$. Now combining all these runs we get that $(q, u_1, u_2, \ldots, u_n) \xrightarrow{\alpha_{(j,0)}\alpha_{(i_1,1)}\cdots\alpha_{i_{2m-1},2m-1}\alpha_{(j,2m)}} (q_j, v_1, v_2, \ldots, v_n)$. This complete the proof for the base case.

Induction Step: Let us assume now that if for every $i \in [1..n]$, $(q, u_i, q'_i, v_i, \sharp^m) \in L(T_i)$ then $(q, u_i, q'_j, v_i, \sharp^m) \in L(T_i)$ for all $i \in [1..n]$, and $(q, u_1, \ldots, u_n) \xrightarrow{\pi}_{\mathscr{M}} (q_j, v_1, \ldots, v_n)$ for some sequence $\pi \in (L(\Lambda))^m$.

Let us show that if for every $i \in [1..n]$, $(q, u_i, q_i, v_i, \sharp^{m+1}) \in L(T_i)$ then $(q, u_i, q_j, v_i, \sharp^{m+1}) \in L(T_i)$ for all $i \in [1..n]$, and $(q, u_1, \ldots, u_n) \xrightarrow{\pi}_{\mathscr{M}} (q_j, v_1, \ldots, v_n)$ for some sequence $\pi \in (L(\Lambda))^{m+1}$.

Since $(q, u_i, q_i, v_i, \sharp^{m+1}) \in L(T_i)$ for all $i \in [1..n]$, this implies that $(q, u_i) \xrightarrow{\pi_i}_{M_i} (q_i, v_i)$ for some $\pi_i \in L(\Lambda|_i)^{m+1}$. We can then split the run $(q, u_i) \xrightarrow{\pi_i}_{M_i} (q_i, v_i)$ as follows: $(q, u_i) \xrightarrow{\pi'_i}_{M_i} (q'_i, w_i) \xrightarrow{\pi''}_{M_i} (q_i, v_i)$ such that $\pi'_i \in L(\Lambda|_i)^m$ and $\pi''_i \in L(\Lambda|_i)$.

Since for every $i \in [1..n]$, $(q, u_i) \xrightarrow{\pi'_i}_{M_i} (q'_i, w_i)$ such that $\pi'_i \in L(\Lambda|_i)^m$, we have $(q, u_i, q'_i, w_i, \sharp^m) \in L(T_i)$ from Lemma 64. Now we can apply the induction hypothesis to these runs and we get that $(q, u_i, q'_j, w_i, \sharp^m) \in L(T_i)$ for all $i \in [1..n]$, and $(q, u_1, \ldots, u_n) \xrightarrow{\sigma}_{\mathscr{M}} (q'_j, w_1, \ldots, w_n)$ for some sequence $\pi \in (L(\Lambda))^m$.

On the other hand, since $\tau_0 \subseteq \Delta_j$, we can show that $(q'_j, w_i) \xrightarrow{\pi'''}_{M_i} (q_i, v_i)$ such that $\pi''_i \in L(\Lambda|_i)^m$. This implies that $(q'_j, w_i, q_i, v_i, \sharp) \in L(T_i)$ from Lemma 64. Now we can apply the induction hypothesis to these runs and we get that $(q'_j, w_i, q_j, v_i, \sharp) \in L(T_i)$ for all $i \in [1..n]$, and $(q'_j, w_1, \ldots, w_n) \xrightarrow{\sigma'}_{\mathscr{M}} (q_j, v_1, \ldots, v_n)$ for some sequence $\pi \in (L(\Lambda))^m$.

Since $(q, u_i, q'_j, w_i, \sharp^m) \in L(T_i)$ and $(q'_j, w_i, q_j, v_i, \sharp) \in L(T_i)$ for all $i \in [1..n]$, we get that $(q, u_i) \xrightarrow{\pi'_i}_{M_i} (q'_j, w_i)$ and $(q'_j, w_i) \xrightarrow{\pi'''}_{M_i} (q_j, v_i)$. Combining these two runs we get $(q, u_i) \xrightarrow{\pi' \cdot \pi'''}_{M_i} (q_j, v_i)$ for all $j \in [1..n]$.

Now, we can also combine the following two runs $(q, u_1, \ldots, u_n) \xrightarrow{\sigma}_{\mathscr{M}} (q'_j, w_1, \ldots, w_n)$ and $(q'_j, w_1, \ldots, w_n) \xrightarrow{\sigma'}_{\mathscr{M}} (q_j, v_1, \ldots, v_n)$, we get $(q, u_1, \ldots, u_n) \xrightarrow{\sigma \cdot \sigma'}_{\mathscr{M}} (q_j, v_1, \ldots, v_n)$. □

□

Remark: *The ideas used in the proof of Lemma 64 can be used to compute more details such as the number of occurrences of every transition (instead of number of iterations of $L(\Lambda)$). However, without the special structure of $L(\Lambda)$, this does not seem to lead to a corresponding generalization of Lemma 65.*

Now, we are ready to prove Theorem 27. Let us assume now that we are given a $(n+1)$-tape constrained finite-state automaton $\mathscr{C} = (A, \phi)$ where $A = (P, Q, \Gamma, \ldots, \Gamma, \delta, p_0, F)$ and $L(\mathscr{C}) = C$. In the following, we show how to compute a $(n+1)$-tape constrained finite state automaton $\mathscr{C}'$ accepting the set $Post_{(L(\Lambda))^*}(C)$. To do that, we proceed as follows: We first compose C with the synchronized automaton (T_1, true), synchronizing the second tape of T_1 (containing the stack contents at the starting configuration of the M_1) with the second tape of A, to construct

a $(n+5)$-tape constrained automaton $\mathscr{C}_1 = \mathscr{C} \circ_{(2,2)} (T_1, \mathsf{true})$. We then need to synchronize the starting states (i.e., the first tape of A with the first tape of T_1). This can be done by intersecting $\mathscr{C}_1$ with the (regular) language $\bigcup_{q \in Q} \{q\} \times (\Gamma^*)^n \times \{q\} \times Q \times \Gamma^* \times (\{\sharp\})^*$. Let $\mathscr{C}'_1$ be the $(n+5)$-tapes resulting of this intersection. Then, we project away the starting control state occurring on the $n+2$-tape and the content of the second tape to obtain the $(n+3)$-tape constrained automaton $\mathscr{C}''_1 = \Pi_\iota(\mathscr{C}'_1)$ where $\iota = ([1..n+5] \setminus \{2, n+2\})$.

Then, we need to compose $\mathscr{C}''_1$ with the constrained automaton (T_2, true), synchronizing the second tape of T_2 (containing the stack contents at the starting configuration of the M_2) with the second tape of $\mathscr{C}''_1$, to construct a $(n+7)$-tape constrained automaton $\mathscr{C}_2 = \mathscr{C}''_1 \circ_{(2,2)} (T_2, \mathsf{true})$. We then need to synchronize the starting states (i.e., the first tape of $\mathscr{C}''_1$ with the first tape of T_2). This can be done by intersecting $\mathscr{C}_2$ with the (regular) language $\bigcup_{q \in Q} \{q\} \times (\Gamma^*)^{n-1} \times Q \times \Gamma^* \times (\{\sharp\})^* \times \{q\} \times Q \times \Gamma^* \times (\{\sharp\})^*$. Let $\mathscr{C}'_2$ be the $(n+7)$-tapes resulting of this intersection. Then, we project away the starting control state occurring on the $n+4$-tape and the content of the second tape to obtain the $(n+5)$-tape constrained automaton $\mathscr{C}''_2 = \Pi_{\iota'}(\mathscr{C}'_2)$ where $\iota' = ([1..n+6] \setminus \{2, n+4\})$.

This procedure is then repeated for all the constrained automata (T_i, true), with $i \in [3..n]$, to obtain at the end the $(3n+1)$-tape constrained automaton $\mathscr{C}''_n$. We can also project away the starting state stored at the first tape from $\mathscr{C}''_n$ since it is not any more needed. So, let $\mathscr{G}$ be $(3n)$-tape constrained automaton such that $\mathscr{G} = \Pi_{[2..3n]}(\mathscr{C}''_n)$.

Now, we need to synchronize the automata (T_i, true) on their final states stored respectively at the tapes $3(i-1)+1$, with $i \in [1..n]$, of $\mathscr{G}$. To do that we intersect $\mathscr{G}$ with the (regular) language $\bigcup_{q \in Q} \{q\} \times \Gamma^* \times (\{\sharp\})^* \times \{q\} \times \Gamma^* \times (\{\sharp\})^* \times \cdots \times \{q\} \times \Gamma^* \times (\{\sharp\})^*$. Let $\mathscr{G}'$ be the $(3n)$-tapes resulting of this intersection. We can then project away the copies of the final control states and only keep its first occurrence to obtain $(2n+1)$-tape constrained automaton $\mathscr{G}''$ defined as follows: $\mathscr{G}'' = \Pi_{\iota''}(\mathscr{G}')$ where $\iota'' = ([1..3n] \setminus \{3i+1 \mid i \in [1..n-1]\})$. Let us assume that $\mathscr{G}''$ is of the form (A', ϕ') where $A' = (P', Q, \Gamma, \{\sharp\}, \Gamma, \sharp, \ldots, \Gamma, \{\sharp\}, \delta', p'_0, F')$. Furthermore, let us assume that for every $i \in [1..n]$, δ'_i is the subset of δ' containing only the transitions of the form $(p, \mathbf{v}, p') \in \delta'$ such that $\mathbf{v}[2i+1] = \sharp$. Observe that by definition, we have $\mathbf{v}[j] = \epsilon$ for all $j \neq 2i+1$.

From Lemma 65, we need to ensure the same number of the special letters $\sharp$ in all the tapes $\{2i+1 | i \in [1..n]\}$, we need to augment the formula ϕ' with additional constraints. Let $\mathscr{G}''' = (A', \phi'')$ where $\phi'' = \phi' \wedge (\sum_{t_1 \in \delta'_1} t_1 = \sum_{t_2 \in \delta'_2} t_2 = \cdots = \sum_{t_n \in \delta'_n} t_n)$. Finally, the $n+1$-tape constrained finite state automaton $\mathscr{C}'$ can be constructed from $\mathscr{G}'''$ by projecting away the tapes with symbol $\sharp$ i.e. the tapes $\{2i+1 | i \in [1..n]\}$. Hence, $\mathscr{C}' = \Pi_{\iota'''}(\mathscr{G}''')$ where $\iota''' = ([1..n] \setminus \{2i+1 | i \in [1..n]\})$. This completes the proof of Theorem 27.

8.4 Conclusion

In this chapter, we first showed that rational sets of configurations are stable w.r.t. bounded context executions.

- We showed that under iterations of a loop of a regular set of transitions is always rational while that of a rational set need not be rational.
- We then introduced a new representation for configurations called n-CSRE. We went on to show that n-CSREs are indeed stable w.r.t iteration of loops.

- We introduced a joint generalization of both loop iterations and bounded context executions called bounded context-switch sets. We showed that the class of languages defined by n-dimensional constrained automata is stable w.r.t accelerations via bounded context-switch sets.

Chapter 9

Parity games on MPDS

9.1 Introduction

In this chapter, we consider the problem of solving the parity games over the multi-pushdown systems with bounded-phase restriction. Informally, a parity game is a two player game (say player-0 and player-1), played on a graph. The vertices of the graph are partitioned into player-0 and player-1 positions. Further each node in the graph is assigned a rank from a finite set of natural numbers. The game starts with a token placed in a node designated as the initial node. The play proceeds in rounds and the player of each round is determined by the player of the node, in which the token is placed. During each round, the player moves the token from the current node to one of its adjacent nodes. The winner of the game is determined by the minimum rank visited infinitely often in the play. Now given any initial position in the graph, one can ask if a particular player can play in such a way that he win all plays starting from that position, irrespective of how the other player responds. Then we say that the player wins from his position and has a winning strategy, formal definitions are provided in the next section. Now given a parity game and a position in the game, one can ask if there is a winning strategy for a particular player.

Parity games were originally introduced in [116] and subsequently studied as a winning condition of games in [59]. Close connections of parity games with μ-calculus [58, 135] makes this problem interesting and important. The parity games on finite state systems is well studied [86, 36, 87] and is known to be in UP $\cap$ co-UP. Solving parity games on infinite structures has also been a topic of interest [50, 143]. Parity games on pushdown systems were studied in [143] and was shown to be decidable. In [47, 93], it was shown that set of all winning positions in a parity game on pushdown systems is regular.

In this chapter, we are interested in solving the parity game over a multi-pushdown system with bounded-phase restriction. This problem was first studied by A. Seth in [133]. He showed how to obtain a NON-ELEMENTARY decision procedure to solve the problem. Here, we show a seemingly simpler and inductive argument for the same problem. For this, we observe that any sub-game involving only one phase is essentially a game on a pushdown system. Using this crucial fact, we then show how to reduce a parity games over a multi-pushdown systems with k- phase restriction to a parity games over MPDS with $k-1$ phase

restriction, thus reducing the problem to pushdown games. The complexity of our construction is also NON-ELEMENTARY.

We next show how to reduce the problem of checking satisfiability of a first order formula, with ordering relation $FO(<)$ over natural numbers to solving a parity games over MPDS with bounded-phase restriction. The satisfiability problem of $FO(<)$ over natural numbers is known to be NON-ELEMENTARY-COMPLETE [136]. This reduction shows that the high complexity required to solve parity games on MPDS with bounded-phase restriction cannot be avoided. With this, we settle the question posed by A.Seth [133], on whether such a high complexity for solving parity games over MPDS with bounded scope is really required.

9.2 Parity Games

Definition 11. *A Parity game is defined over game graph $\mathcal{G} = (V, E, \tau, \sigma)$ where*

- *V is a (possibly infinite) set of nodes.*
- *$E \subseteq V \times V$ is a set of edges.*
- *$\tau : V \mapsto [0,1]$ is a function that defines ownership of the node.*
- *$\sigma : V \mapsto [1..m]$ for some $m \in \mathbb{N}$ is a ranking function that assigns a rank to each node.*

For any node $s \in V$, we define $E(s) = \{s' \mid (s,s') \in E\}$. We say a π is a finite play of $\mathcal{G}$ iff $\pi = s_1 s_2 \cdots s_n$ such that for all $i \in [1 \ldots n-1]$, $(s_i, s_{i+1}) \in E$ and $E(s_n) = \emptyset$. π is said to be infinite play of $\mathcal{G}$ iff $\pi = s_1 s_1 s_2 \cdots$ such that for all $i \in \mathbb{N}$, we have $(s_i, s_{i+1}) \in E$. We will assume w.l.o.g. that graphs we deal with do not have any dead end nodes (i.e. there is no s such that $E(s) = \emptyset$) and hence that all our plays are infinite.

For any infinite play $\pi = s_0 s_1 s_2 \cdots$, we let S^∞_π to be the set of all nodes that appear infinitely often in the play π. We define $Parity(\pi) = min(inf(\pi)) \ mod\ 2$, where $inf(\pi) = \{\sigma(s) \mid s \in S^\infty_\pi\}$ i.e. it is the parity of minimum rank that is seen infinitely often along the run. An infinite play π is winning for player-0 iff $Parity(\pi)$ is 0, otherwise it is winning for player-1.

For any $i \in [0,1]$, we let $V_i = \{s \mid s \in V \wedge \tau(s) = i\}$ i.e. it is the set of positions owned by player-i. A strategy function f for player-0 is defined as $f : V^* V_0 \mapsto 2^V \setminus \emptyset$. An infinite play $\pi = v_0 v_1 v_2 \cdots$ is said to be confirming to a strategy function f iff for any prefix of the play $\pi' = v_0 \cdots v_i \in V^* V_0$, $v_{i+1} \in f(\pi')$. An strategy function f is said to be winning for player-0 from any node s, if the set of all possible plays π starting from the node s, such that it is confirming to the strategy function f are winning for player-0. The strategy function for player-1 is defined analogously. We say a node s is winning for player-0 (or player-1) iff there is a strategy function that is winning for player-0 (or player-1) from that position.

A strategy function f of player-i is said to be a memoryless strategy or positional strategy if it is of the form $f : V_i \mapsto 2^V \setminus \emptyset$, i.e. it only depends on single node. Any given play $\pi = v_0 v_1 \cdots$ is said to be confirming to the memoryless strategy function f of player-i, if for all nodes $v_j \in V_i$ ($j \in \mathbb{N}$), we have $v_{j+1} \in f(v_j)$. We can now define the memoryless winning strategy function analogous to the previous case.

A natural question in this setting is whether for any position s, one of the two players has a winning strategy from that position (determinacy) and if so whether the strategy is memoryless (memoryless determinacy). The determinacy of parity games follows from a very general

result due to Martin's determinacy theorem which establishes the determinacy for a much wider class of games.

Memoryless determinacy theorem for parity games [59] establishes that we not only have determinacy, we also have that the winning player has a memoryless winning strategy.

Theorem 28. *[59] Given a parity game $G = (V, E, \tau, \sigma)$, there is a partition of nodes V, $V = W_0 \uplus W_1$ and memoryless strategy functions σ_0, σ_1 such that σ_i is winning for player-i from each positions in W_i.*

This still leaves the interesting question of how to determine the winning sets and the winning strategies in specific games. While this is easy for finite graphs, extensions to infinite graphs is difficult. We refer the readers to [73] for a detailed survey on infinite games in general.

9.2.1 Some useful results on parity games

In this section, we prove a couple of facts about parity games in general, that will be useful later in the chapter. The following Lemma states that if there is a mapping from one game graph to another such that any move in the former can be simulated in the latter, and if such a simulation preserves the player and the rank at each position of the play, then the winning positions are also preserved by the mapping.

Lemma 66. *Let $G = (V_G, E_G, \tau_G, \sigma_G)$ and $H = (V_H, E_H, \tau_H, \sigma_H)$ be games graphs and let $\mathbf{F} : V_G \longrightarrow V_H$ be any function such that for any position $x \in V_G$*

1. *$\sigma_G(x) = \sigma_H(\mathbf{F}(x))$, the function is rank preserving.*
2. *$\tau_G(x) = \tau_H(\mathbf{F}(x))$ i.e. x and $\mathbf{F}(x)$ belongs to the same player i.*
3. *If $x \to x'$ then $\mathbf{F}(x) \to \mathbf{F}(x')$.*
4. *If $\mathbf{F}(x) \to y$ then there exists x' such that $x \to x'$ and $\mathbf{F}(x') = y$.*

Then, any position x is winning for player 0 (player-1) in G if and only if $\mathbf{F}(x)$ is winning for player 0 (resp player-1) in H.

Proof. ($\Leftarrow$) Firstly we will assume that player-0 is winning from the node $\mathbf{F}(x)$ in H (and hence has a strategy function h that is winning from node $\mathbf{F}(x)$) and show how to construct a strategy function g for player-0 in G, which is winning from x. For any node $v \in V_{G_0}$, we let $g(v) = \{v' \mid \mathbf{F}(v') \in h(\mathbf{F}(v))\}$. Using item 4, we can conclude that $g(v) \neq \emptyset$. Consider any infinite play $\pi = x v_1 v_2 \cdots$ that is confirming to strategy function g, we will show that such a play is winning for player-0. For this, we will consider the play $\mathbf{F}(\pi) = \mathbf{F}(x)\,\mathbf{F}(v_1)\,\mathbf{F}(v_2)\cdots$ and show that such a play is confirming to the strategy function h. Consider any node from the play $\mathbf{F}(v_i) \in V_{H_0}$, we need to show that $\mathbf{F}(v_{i+1}) \in h(\mathbf{F}(v_i))$. But notice that $v_{i+1} \in g(v_i)$ since π is conforming to g. By definition $g(v_i) = \{v' \mid \mathbf{F}(v') \in h(\mathbf{F}(v_i))\}$ and hence $\mathbf{F}(v_{i+1}) \in h(\mathbf{F}(v_i))$. Hence we have that for every infinitely play in G such that it is conforming to strategy function g, there is an infinite play in H that is confirming to h with an identical sequence of ranks. Since h is a winning strategy, $Parity(F(\pi)) = 0$ and so $Parity(\pi) = 0$. Thus, we have that g is winning for player-0. For the other direction, since parity games are determined via memoryless strategies, it is enough to prove that if player-1 is winning from a node $\mathbf{F}(x)$ in H,

then player-1 is winning from x in G. But this follows from similar argument as above. Hence we have the result.

□

Given a game graph $\mathcal{G} = (V, E, \tau, \sigma)$, $U \subseteq V$ is said to be a trap of $\mathcal{G}$ iff $E \cap U \times (V \setminus U) = \emptyset$. i.e. once the game enters U, there is no way for it to exit. The following Lemma states that given any parity game graph, the game graph obtained by fusing all the winning positions of player-0 (and that of player-1), into one node, preserves the winning positions.

Lemma 67. *Let $G = (V_G, E_G, \tau_G, \sigma_G)$ be a parity game and let $V_H \subseteq V_G$ be a trap of G. Suppose V_{H0} and V_{H1} are the winning positions for player 0 and 1 in the subgame V_H. Then, consider the game graph $G' = (V_{G'}, E_{G'}, \tau_{G'}, \sigma_{G'})$ constructed as follows:*

1. *Delete the subgame V_H*
2. *Add two new positions q_w and q_l.*
3. *If $s \rightarrow t$ is an edge in E with $s \notin V_H$ and $t \in V_{H0}$ then add an edge from s to q_w.*
4. *If $s \rightarrow t$ is an edge in E with $s \notin V_H$ and $t \in V_{H1}$ then add an edge from s to q_l.*
5. *Add edges from q_w to q_w and q_l to q_l.*
6. *For all $v \in V_G \setminus V_H$, we let $\tau_{G'}(v) = \tau_G(v)$, $\tau_{G'}(q_w) = 0$, $\tau_{G'}(q_l) = 1$.*
7. *For all $v \in V_G \setminus V_H$, $\sigma_{G'}(v) = \sigma(v)$ and $\sigma_{G'}(q_w) = 0$, $\sigma_{G'}(q_l) = 1$.*

Then, any position in V_G that is not in V_H is winning for any player in G if and only if it is winning for that player in the game G'.

Proof. To prove the above Lemma , we will first prove Lemma 68 which states that given any game $G = (V, E, \tau, \sigma)$ and set of vertices $V_H \subseteq V$ which is a trap, any position x in game G is winning for player-0 iff it is winning in a modified game $G' = (V, E, \tau, \sigma')$ where the ranking function for vertices in the trap that is winning for player-i is replaced by i ($i \in \{0, 1\}$) and remains unchanged for rest of the vertices.

Lemma 68. *Let $G = (V, E, \tau, \sigma)$ be a parity game. Let V_H be set of positions that is a trap, let $V_{H0} \subseteq V_H$ be set of positions that are winning for player-0 and V_{H1} be set of positions that are winning for player-1 in the subgame on V_H, then any position $x \in V$, is winning for player-0 in G iff it is winning in a game $G' = ((V, E, \tau), \sigma')$ where σ' is given by, for all $v \in V \setminus V_H$, $\sigma'(v) = \sigma(v)$, for all $v \in V_{H0}$, $\sigma'(v) = 0$ and for all $v \in V_{H1}$, $\sigma'(v) = 1$.*

Proof. ($\Rightarrow$)

Let g be any strategy funtion of G which is winning for player-0 from node x. We will show that the same strategy function is also winning in G'. Assume any play π, starting at x and which confirms to strategy function g. Note that π is also a valid play in G', confirming to g. Supposing π is play that does not involve vertices from V_H then there is nothing to prove, since the ranking function is the same for nodes not in V_H. So we will assume that π involves nodes from V_H.

Since π is winning for player-0 and by assumption involves vertices from V_H, note that V_H is a trap, hence there is a finite prefix of π after which all the vertices visited are from V_H, in-fact V_{H_0} (since π is winning for player-0 and nodes in V_{H1} are winning for player-1 in the

subgame on V_H). Hence the minimum parity of nodes visited infinitely often for the play π in G' is 0 and hence winning for player-0.

($\Leftarrow$)

For this direction, from determinacy of parity games, we only need to prove that for any node x, if player-1 is winning from it in G, he is also winning from the same node in G'. But such a proof is very similar to the case above.

□

Next we will prove Lemma 69 which states that any position x in a game G is winning for player-0 iff it is winning in a modified game G' where the vertices that are winning for player-0 in any trap are reassigned as player-0 positions and vertices that are winning for player-1 are reassigned as player-1 positions.

Lemma 69. *Let $G = (V, E, \tau, \sigma)$ be any parity game, let $V_H = V_{H0} \uplus V_{H1}$ be a trap, where V_{H0} are winning positions of player-0 and V_{H1} are winning positions of player-1 in V_H. Any position x is winning in G iff if it is winning in a similar game $G' = (V, E, \tau', \sigma)$ where $\tau'(u) = \tau(u)$ if $u \in V \setminus V_H$, $\tau'(u) = 0$ if $u \in V_{H0}$ and $\tau'(u) = 1$ if $u \in V_{H1}$*

Proof. ($\Rightarrow$) Let g be any strategy funtion of G which is winning for player-0 from node x. We will show how to construct a strategy function g' for player-0 in G' such that he can win any play starting at node x.

- For any $v \in V \setminus V_H$ such that $\tau(v) = 0$, we let $g'(v) = g(v)$.
- For any $v \in V_{H_0}$ such that $\tau(v) = 0$, we let $g'(v) = g(v)$.
- For any $v \in V_{H_0}$ such that $\tau(v) = 1$, we let $g'(v) = v'$ for some $v' \in E(v)$.

It is easy to see that such a strategy function is winning for player-0 from node x. If the play never enters V_H, it is winning since g was originally winning. If it ever enters V_{H_0}, notice that g was winning for any arbitrary choice of player-1, from this we can conclude that g' is winning.

($\Leftarrow$) The other direction is obtained via the determinacy and the symmetric argument for strategies for player-1. □

With these two Lemmas in place, the proof becomes simpler. Now let $G_1 = (V_G, E_G, \tau_{G_1}, \sigma_G)$ be the game obtained from G, by applying Lemma 68 and let $G_2 = (V_G, E_G, \tau_{G_1}, \sigma_{G_2})$ be a game obtained from G_1, by applying Lemma 69. Now it is easy to see that we can define a mapping $\mathbf{F} : V_{G_2} \mapsto V_{G'}$ which has the properties required by Lemma-66. The mapping $\mathbf{F} : V_{G_2} \mapsto V_{G'}$ is given by

- For any $v \in V_{G_2} \setminus V_H$, we let $\mathbf{F}(v) = v$
- For any $v \in V_{H_0}$, we let $\mathbf{F}(v) = q_w$
- For any $v \in V_{H_1}$, we let $\mathbf{F}(v) = q_l$

From this it is easy to see that for any $x \in V_G \setminus V_H$ is winning in G iff it is winning in G'.

□

9.2.2 Parity games on pushdown system

Parity games on finite state system have been well studied, from the time they were first introduced in [116]. Extensions of parity games to finitely described infinite structures have also been studied in past. The key step in this direction has been the result of I. Walukiewicz [143], showing how winners can be determined in pushdown games, i.e. games played on configuration graphs generated by a pushdown system. Since parity games has close connections with model checking mu-calculus, such a result also provides an algorithm to model check mu-calculus formulas over pushdown systems. We define parity game on pushdown systems below.

Definition 12 (Parity game on PDS)**.** *Given a pushdown system $P = (Q, \Gamma, \Delta, q_0)$ and mappings $\tau : Q \mapsto [0,1]$ and $\sigma : Q \mapsto [1..m]$, parity game on PDS is simply a parity game played on the game graph $\mathcal{G} = (\mathcal{C}(P), \rightarrow, \tau, \sigma)$, where τ and σ are extended to configurations as follows. For any configuration of the form $c = (q, \gamma)$, $\tau(c) = \tau(q)$ and $\sigma(c) = \sigma(q)$.*

In [47, 132], T. Cachat and O. Serre independently proved that the set of all winning positions of a particular player in a parity game played on a pushdown system is effectively regular. This can also be obtained using tree automata techniques as shown in [93].

Theorem 29. *[47, 132] The set of all winning positions for player 0 (or player 1) in a pushdown game can be described by an effectively constructible, exponential sized finite state automaton over the alphabet $\Gamma \cup Q$ which accepts a word $wq \in \Gamma^* Q$ if and only if the configuration (q, w) is winning for player 0 (or player 1).*

We will use this finite representation in our construction for bounded-phase games.

9.3 Bounded phase parity games on MPDS

For purpose of defining the *bounded-phase* parity games, we will first enhance the configurations of a multi-pushdown system with the information about number of phases remaining and the identity of current stack.

Definition 13 (Bounded-phase parity games)**.** *Given a multi-pushdown system $M = (n, Q, \Gamma, \Delta, q_0)$ and a constant k, we define the set of enhanced configurations of M as $\mathcal{E}^k(M)$ as $\mathcal{C} \times [0..n] \times [1..k]$. Such an enhanced configuration, apart from containing configuration of the multi-pushdown system, also records the currently active stack and number of remaining phases. We will omit the k and simply refer to it as $\mathcal{E}(M)$ when ever k is clear from the context. At beginning of any computation, we let the (penultimate) current stack component of $\mathcal{E}(M)$ to be 0, indicating that none of the stacks are active. From such a position, the stack gets active on the very first pop or zero test. We define the transition relation $\rightsquigarrow$ as follows. Given any two configurations $(c, i, j), (c', i', j') \in \mathcal{E}(M)$, we say $(c, i, j) \rightsquigarrow (c', i', j')$ iff $c \xrightarrow{\tau} c'$ and one of the following holds.*

- *If $\tau = (q, \mathbf{Pop}_l, q')$ or $\tau = (q, \mathbf{Zero}_l, q')$ for some $l \in [1..n]$ and $i = 0$ then $j = j' = k$ and $i' = l$.*
- *if $\tau = (q, \mathbf{Push}_k(a), q')$ or $\tau = (q, \mathbf{Int}_k, q')$ for some $k \in [1 \ldots n]$ or $\tau = (q, \mathbf{Pop}_i, q')$ or $\tau = (q, \mathbf{Zero}_i, q')$ then $i' = i$, $j' = j$*

- *if* $\tau = (q, \mathbf{Pop}_l(a), q')$ *or* $\tau = (q, \mathbf{Zero}_l, q')$ *for some* $l \neq i$ *and* $j > 1$ *then* $i' = l, j' = j - 1$

Let $\tau : Q \mapsto [0,1]$ *be a map that designates each state to a player,* $\sigma : Q \mapsto [1..m]$ *be a map that assigns rank to each state and* k *be any natural number, then a* k*-bounded-phase parity game is a parity game played on the game graph* $\mathcal{G} = (\mathcal{E}(M), \rightsquigarrow, \tau, \sigma)$*, where* τ, σ *are extended to configurations as follows. For any* $(c, i, j) \in \mathcal{E}(M)$*, we let* $\tau((c, i, j)) = \tau(\text{State}(c))$ *and* $\sigma((c, i, j)) = \sigma(\text{State}(c))$*. We will denote such a game as* $\mathcal{G} = (k, M, \tau, \sigma)$

Given a bounded-phase parity game $\mathcal{G} = (k, M, \tau, \sigma)$ and a node $s \in \mathcal{E}(M)$, we would like to determine whether there is a strategy function g that is winning for player-0 from the node s.

9.4 Decidability of bounded phase parity games

Our construction for solving the k bounded-phase parity game proceeds inductively on the value of k. The intuitive idea is to first show that if the game is a single phase game, then the game graph of such a game actually corresponds to just the positions of a pushdown game and by Theorem 29 we know the set of winning positions are recognisable.

Secondly observe that that the positions in the game graph are stratified in the following sense – if (c', i', k') is reachable from (c, i, k) then $k' \leq k$. From this, we know that if the game were to enter the last phase, it will continue to remain in that phase. Hence any position in the last phase corresponds to a position of a pushdown game, which is known to be recognisable. Using this information, we will go onto show how to reduce the k bounded-phase game to a $k-1$ bounded-phase game.

9.4.1 Decidability of 1-phase game

In an 1-phase game, the configurations can be of the form $(c, i, 1)$ with $i \neq 0$ or of the form $(c, 0, 1)$. We will show in each of the cases that the set of positions winning for player-0 is a recognisable set (i.e. it can be effectively determined). For the sub-game involving only configurations of the form $(c, i, 1)$, we will show that such positions correspond to positions of a pushdown game. Now using the fact that set of all positions winning for player-0 in pushdown game is a recognisable set, we will show that nodes that are winning for player-0 in sub-game involving configurations of the form $(c, i, 1)$ is also a recognisable set. For the case involving configurations of the form $(c, 0, 1)$, we will reduce such a sub-game to a parity game involving only finitely many states.We formalise the details below.

Lemma 70. *Consider any bounded-phase parity game* $\mathcal{G} = (1, M, \tau, \sigma)$*, where* $M = (n, Q, \Gamma, \Delta, q_0)$*. We can effectively determine the set of all positions of the form* $\mathcal{E}^1(M)$*,* $i \in [0..n]$ *, that are winning for player* 0*.*

Proof. The nodes in $\mathcal{E}^1(M)$ are either of the form $(c, 0, 1)$ or of the form $(c, i, 1)$ for some $i \neq 0$. We will first consider the nodes of the form $(c, i, 1)$ and show that the winner can be determined. The general idea of the proof is to first construct a pushdown system for each $i \in [1..n]$, from the given multi-pushdown system M. Such a pushdown system will simulate the moves of stack-i by using its own stack for any operations on stack i, and ignoring

the pushes on other stacks. The pushdown system (corresponding to stack-i) is defined as, $P_i = (Q, \Gamma, \delta_i, q_0)$, where δ_i is defined as follows.

- For every $\tau = (q, \mathbf{Pop}_i(a), q') \in \Delta$, we add $\tau' = (q, \mathbf{Pop}(a), q') \in \delta_i$. We add similar transitions for $\tau = (q, \mathbf{Zero}_i(a), q') \in \Delta$, $\tau = (q, \mathbf{Push}_i(b), q') \in \Delta$ and $\tau = (q, \mathbf{Int}_i, q') \in \Delta$
- For $j \neq i$ and for every $\tau = (q, \mathbf{Push}_j(b), q') \in \Delta$ and $\tau = (q, \mathbf{Int}_j, q') \in \Delta$, we add $\tau' = (q, \mathbf{Int}, q') \in \delta_i$.

The winning positions of each player of the sub-game with configurations of the form $(c, i, 1)$ with $i \neq 0$, can be captured using the pushdown game $\mathscr{H} = (\mathscr{C}(P_i), \rightarrow, \tau, \sigma)$. Let the function $\mathbf{F} : \mathscr{E}^1(M) \mapsto \mathscr{C}(P_i)$ be given by

$$\mathbf{F}(((q, \gamma_1, \gamma_2, \cdots, \gamma_n), i, 1)) \quad = \quad ((q, \gamma_i))$$

The function $\mathbf{F}$ simply disregards content of stacks other than i and keeps stack i intact. Following set of claims shows that such a mapping will preserve the properties required by Lemma 66.

Claim 11. *For any $v \in \mathscr{E}^1(M)$, we have $\tau(v) = \tau(\mathbf{F}(v))$ and $\sigma(v) = \sigma(\mathbf{F}(v))$.*

Proof. Straight forward from the fact that $\mathbf{F}$ preserves the state. □

Claim 12. *For any $u = (c, i, 1), v = (c', i, 1) \in \mathscr{E}(M)$, if $(u \rightsquigarrow v)$ then we have $\mathbf{F}(u) \rightarrow \mathbf{F}(v)$*

Proof. Since we have assumed $i \neq 0$, by definition the allowed operations does not include the transitions of the form $Q \times (\bigcup_{a \in \Gamma} \mathbf{Pop}_j(a) \cup \mathbf{Zero}_j) \times Q$ for some $j \neq i$. Let us assume that $c = (q, \gamma_1, \cdots, \gamma_n)$ and $c' = (q', \gamma'_1, \cdots, \gamma'_n)$, then $\mathbf{F}(c) = (q, \gamma_i)$ and $\mathbf{F}(c') = (q', \gamma'_i)$

- Suppose the transition used was of the form $(q, \mathbf{Push}_j(a), q') \in \Delta$, for some $j \neq i$, then we have that $\gamma'_j = a\gamma_j$ and for all $l \neq j$, we have $\gamma'_l = \gamma_l$ (more specifically $\gamma_i = \gamma'_i$). Further by construction, we also have the transition $(q, \mathbf{Int}, q') \in \delta_i$. From this we get $(q, \gamma_i) \rightarrow (q', \gamma'_i)$
- Suppose the transition used was of the form $(q, \mathbf{Push}_i(a), q') \in \Delta$, then we have that $\gamma'_i = a\gamma_i$ and for all $j \neq i$, we have $\gamma'_j = \gamma_j$. Further by construction, we also have the transition $(q, \mathbf{Push}(a), q') \in \delta_i$. From this we get $(q, \gamma_i) \rightarrow (q', \gamma'_i)$
- Suppose the transition used was of the form $(q, \mathbf{Pop}_i(a), q') \in \Delta$, then clearly for all $j \neq i$, $\gamma'_j = \gamma_j$ and $\gamma_i = a\gamma'_i$. By definition we have a transition $(q, \mathbf{Pop}(a), q') \in \delta_i$ and so we have $(q, \gamma_i) \rightarrow (q', \gamma'_i)$.
- We will omit the other cases, since they are similar to one of the cases mentioned above.

□

Claim 13. *Suppose for some $v \in \mathscr{E}(M)$, we have $\mathbf{F}(v) \rightarrow d$, then there is an $u \in \mathscr{E}(M)$ such that $\mathbf{F}(u) = d$ and $v \rightsquigarrow u$*

Proof. Let us assume that $v = (q, \gamma_1, \cdots, \gamma_n)$ then clearly $\mathbf{F}(v) = (q, \gamma_i)$. Let $d = (q', \gamma'_i)$.

- Suppose $\tau = (q, \mathbf{Pop}(a), q') \in \delta_i$ was the transition used in $\mathbf{F}(v) \rightarrow d$. Clearly $\gamma_i = a\gamma'_i$. By construction, τ was added in first place due to existence of some transition $\tau' = (q, \mathbf{Pop}_i(a), q') \in \Delta$. We will let $u = ((q', \gamma_1, \cdots, \gamma_{i-1}, \gamma'_i, \cdots, \gamma_n), i, 1)$. Clearly $v \rightsquigarrow u$ and $\mathbf{F}(u) = d$.

- The case where $\tau = (q, \mathbf{Zero}, q') \in \delta_i$ or $\tau = (q, \mathbf{Push}(a), q') \in \delta_i$ was the transition used in $\mathbf{F}(v) \rightarrow d$ is similar to case mentioned above.
- Suppose $\tau = (q, \mathbf{Int}, q') \in \delta_i$ was the transition used in $\mathbf{F}(v) \rightarrow d$. Note that we have such a transition in δ_i due to presence of a transition of the form $\tau' \in (\bigcup_{j \neq i} Q \times (\{\mathbf{Push}_j(a) \mid a \in \Gamma\} \cup \{\mathbf{Int}_j\}) \times Q) \cup (Q \times \{\mathbf{Int}_i\} \times Q)$. We will assume that the transition τ was added due to existence of $\tau' = (q, \mathbf{Push}_j(a), q') \in \Delta$, rest of the cases are straight forward. In this case, we will let $\gamma'_j = a\gamma_j$ and for all $l \neq j$, we will let $\gamma_l = \gamma'_l$. It is easy to see that $v \rightsquigarrow u$ and $\mathbf{F}(u) = d$.

□

Thus using Lemma 66, the position $(c, i, 1)$ in our subgame is winning for a player-i if and only if $\mathbf{F}((c, i, 1))$ is winning for player-i in the pushdown game $(\mathscr{C}(P_i), \rightarrow, \tau, \sigma)$. Thus, the set of all winning positions of a 1-phase game involving stack-i is given by $\mathscr{S} = \{(c, i, 1) \mid \mathbf{F}((c, i, 1)) \in \mathscr{R}_{P_i}\}$ where $\mathscr{R}_{P_i}$ is the set of winning positions in the game $(\mathscr{C}(P_i), \rightarrow, \tau, \sigma)$. It is easy to see that $\mathscr{S}$ is recognisable set since $\mathscr{R}_{P_i}$ is recognisable by Theorem 29.

Finally we consider positions of the form $(c, 0, 1)$. Any configuration (c', i, k') reached from configuration $(c, 0, 1)$ must necessarily have $k' = 1$. Further, if the game ever enters a position with $i \neq 0$, from the above, we may immediately determine the winner of the game from thereon (Since we already know how to compute set of all winning positions of a 1-phase game involving stack-i). This allows us to formulate a finite state game whose solution determines the winning positions of the form $(c, 0, 1)$. Note that the game can remain in a position of the form $(c, 0, 1)$ iff the transitions involve only push moves or internal moves. The moment a pop move is made, the stack is fixed and the game enters a configuration of the form $(c, i, 1)$, for some $i \in [1..n]$.

Let $B_i = (Q_{B_i}, \Gamma \cup Q, s_i, \delta^{B_i}, F_i)$ be the deterministic finite state automaton that accepts a word of the form $\perp w^R q$ (where $(q, w\perp)$ is a configuration of P_i) iff it belongs to winning positions of game $(\mathscr{C}(P_i), \rightarrow, \tau, \sigma)$. Such an automata is guaranteed by Theorem 29, we note that the size of such an automata is exponential in the size of the pushdown system. The finite state game we have in mind is one which instead of keeping track of the contents of each stack i, only keeps track of the top of stack symbol and the state reached by B_i on reading the contents of that stack. We plan to do this only for the push and internal moves and hence it is indeed feasible. Any pop or zero test moves would commit to a stack (in other words move to a configuration of the form $(c, i, 1)$ for $i \neq 0$), in which case we may immediately determine the winner using the state of B_i. The details are formalised below.

The state space of the finite state game H is $(Q \times \Gamma^n \times Q_{B_1} \times Q_{B_2} \cdots Q_{B_n}) \cup \{q_w, q_l\}$, we will refer to this as $V(H)$. The state q_w is entered on determining that the game will be won by player 0 and q_l if it is determined that the game will be lost by player 0 (or equivalently won by player 1). The edges $\rightarrow_H$ of the game graph are given as follows:

1. $q_w \rightarrow q_w$
2. $q_l \rightarrow q_l$
3. For all $i \in [1..n]$, we have if $(q, \mathbf{Push}_i(b), q') \in \Delta$, then we have $(q, a_1, \cdots, a_n, p_1, \cdots p_n) \rightarrow (q', a_1, \cdots, b, \cdots, a_n, p_1, \cdots, \delta^{B_i}(p_i, a_i), \cdots, p_n)$, for all $a_1, a_2, \cdots, a_n \in \Gamma$ and $p_i \in Q_{B_i}$.
4. For all $i \in [1..n]$, we have if $(q, \mathbf{Int}_i, q') \in \Delta$ then $(q, a_1, \cdots, a_n, p_1, \cdots p_n) \rightarrow (q', a_1, \cdots, a_n, p_1, \cdots, p_n)$. This handles the case of internal moves.

5. If $(q, \mathbf{Pop}_i(a_i), q') \in \Delta$ then if $\delta^{B_i}(p_i, q') \in F_i$, we have $(q, a_1, \cdots, a_n, p_1, \cdots, p_n) \rightarrow q_w$ else if $\delta^{B_i}(p_i, q') \notin F_i$, we have $(q, a_1, \cdots, a_n, p_1, \cdots, p_n) \rightarrow q_l$
6. If $(q, \mathbf{Zero}_i, q') \in \Delta$ then, if $\delta^{B_i}(s_i, \bot.q') \in F_i$, we have $(q, a_1, \cdots, a_{i-1}, \bot, a_{i+1} \cdots, a_n, p_1, \cdots, p_{i-1}, s_i, p_{i+1}, p_n) \rightarrow q_w$ else if $\delta^{B_i}(s_i, \bot.q') \notin F_i$, we have $(q, a_1, \cdots, a_{i-1}, \bot, a_{i+1} \cdots, a_n, p_1, \cdots, p_{i-1}, s_i, p_{i+1}, \cdots, p_n) \rightarrow q_l$

Now consider the ranking function σ' that assigns 0 to q_w, 1 to q_l, i.e. $\sigma(q_w) = 1$ and $\sigma(q_l) = 0$ and for all other positions of the form $c = (q, a_1, \cdots, a_n, p_1, \cdots, p_n)$, we have $\sigma'(c) = \sigma(c)$. Similarly, consider τ' that assigns $\tau'(q_w) = 0$ and $\tau'(q_l) = 1$. Further $\tau'(c) = \tau(c)$ for any $c = (q, a_1, \cdots, a_n, p_1, \cdots, p_n)$, as in above case. We claim that nodes in the subgame involving configurations of the form $(c, 0, 1)$ can be reduced to the finite state parity game given by $H = (V(H), \rightarrow_H, \sigma', \tau')$.

The idea now is to provide a mapping from positions of the form $(c, 0, 1)$ in G to positions in H. For this purpose, we wish to first eliminate from G, using Lemma 67, any positions of the form $(c, i, 1)$ for $i \neq 0$. Note that, the set of all position $S = \{(c, i, 1) \mid (c, i, 1) \in \mathscr{E}(M), i \neq 0\}$ is a trap in G. Further let $W \subseteq S$ be the set of winning positions for player-0 and let $L \subseteq S$ be the set of winning positions for player-1. Now consider the game graph G' obtained by deleting S from G, adding two new vertices p_{win}, p_{lose} replacing all edges to W by edges to p_{win} and edges to L by edges to p_{lose}. Then by application of Lemma 67 a position in $\mathscr{E}(M) \setminus S$ is winning for any player iff it is winning in G'. Observe that the set $\mathscr{E}(M) \setminus S$ is exactly $\{(c, 0, 1) | c \in \mathscr{C}(M)\}$

Our aim is now to use Lemma 66 to determine the winning position of players in G' using the finite game graph H. Towards this, we will now provide a mapping $\mathbf{F}$ from positions of G' to positions in H as follows.

- $\mathbf{F}((q, a_1\gamma_1, a_2\gamma_2, \cdots, a_n\gamma_n), 0, 1)) = ((q, a_1, \cdots, a_l, \delta_1^P(x_1, \gamma_1^R), \delta_2^P(x_2, \gamma_2^R), \cdots, \delta_n^P(x_n, \gamma_n^R))$
- $\mathbf{F}(p_{\text{win}}) = q_w$ and $\mathbf{F}(p_{\text{lose}}) = q_l$.

Lemma 71. *A position $(c, 0, 1)$ is winning for player-i in G' if and only if $\mathbf{F}((c, 0, 1))$ is winning for player-i in H.*

Proof. Proof follows directly from the following set of simple to see claims and Lemma-66.

Claim 14. *For any $(c, 0, 1), (c', 0, 1) \in \mathscr{E}(M)$. if $(c, 0, 1) \xrightarrow{\tau}_{G'} (c', 0, 1)$, then $\mathbf{F}(c, 0, 1) \rightarrow_H \mathbf{F}(c', 0, 1)$. Further if $(c, 0, 1) \rightarrow p_{win}$ ($(c, 0, 1) \rightarrow p_{lose}$), we have $\mathbf{F}(c, 0, 1) \rightarrow_H q_w$ ($\mathbf{F}(c, 0, 1) \rightarrow_H q_l$))*

Proof. We fix $c = (q, a_1\gamma_1, \cdots, a_n\gamma_n)$ for some $q \in Q, a_1\gamma_1, \cdots, a_n\gamma_n \in \Gamma^* \bot$.

Suppose $(c, 0, 1) \xrightarrow{\tau}_{G'} (c', 0, 1)$, where c' is of the form $c' = (q', \gamma_1', \cdots, \gamma_n')$ for some $q' \in Q, \gamma_1'$, $\cdots, \gamma_n' \in \Gamma^* \bot$. From definition of $\mathbf{F}$ we have $\mathbf{F}(c, 0, 1) = ((q, a_1, \cdots a_n, p_1, \cdots, p_n)$, where $p_i = \delta_i^P(x_i, \gamma_i^R)$. Clearly τ cannot be a transition of type $Q \times \bigcup_{i \in [1..n]} \{\mathbf{Pop}_i(a) \mid a \in \Gamma\} \cup \{\mathbf{Zero}_i\} \times Q$ (i.e. it cannot be any transition that commits the phase to a stack).

Let us suppose that $\tau = (q, \mathbf{Push}_i(a), q') \in \Delta$. Then clearly $\gamma_i' = aa_i\gamma_i$ and for $j \neq i$ $\gamma_j' = \gamma_j$. By construction, we have $\tau' = ((q, a_1, \cdots, a_n, p_1, \cdots, p_n)$, $(q', a_1, \cdots, a_{i-1}, a, \cdots, a_n, p_1$, $\cdots, \delta_i(p_i, a_i), \cdots, p_n) \in \rightarrow_H$. From this we have $\mathbf{F}(c, 0, 1) \rightarrow_H \mathbf{F}(c', 0, 1) = ((q', a_1, \cdots, a, a_{i+1}$, $\cdots a_n, p_1, \cdots, p_i', p_{i+1}, \cdots, p_n)$.

The case for τ being of type $\mathbf{Int}_i$ is similar to the above case.

Now suppose $(c,0,1) \rightarrow_{G'} p_{\text{win}}$, then clearly $(c,0,1) \xrightarrow{\tau}_G (c',i,1)$, for some $(c',i,1) \in W$. Let $c' = (q',\gamma'_1,\cdots,\gamma'_n)$ for some $q' \in Q, \gamma'_1,\cdots,\gamma'_n \in \Gamma^* \perp$. Suppose τ was a pop operation $\mathbf{Pop}_i(a_i)$, then $\gamma'_i = \gamma_i$, since $(c',i,1) \in W$, we have $\delta_i(p_i,q') = q_w$, hence we have $\mathbf{F}(c,0,1) \rightarrow_H q_w$. The case where τ is a zero test or $(c',i,1) \in L$ are similar. □

Claim 15. *For any $(c,0,1) \in \mathscr{E}(M)$, if $\mathbf{F}(c,0,1) \rightarrow_H d$ for some $d \in V(H) \setminus \{q_w, q_l\}$, then we have some $(c',0,1) \in \mathscr{E}(M)$ such that $\mathbf{F}(c',0,1) = d$ and $(c,0,1) \xrightarrow{\tau} (c',0,1)$. Further if $d = q_w$ ($d = q_l$) then we have $(c,0,1) \rightarrow_{G'} p_{win}$ ($(c,0,1) \rightarrow_{G'} p_{lose}$).*

Proof. We will first fix $c = (q,\gamma_1,\cdots,\gamma_n)$ then $\mathbf{F}(c,0,1) = (q,a_1,\cdots,a_n,p_1,\cdots p_n) \rightarrow (q',a_1,\cdots, b,\cdots,a_n,p_1,\cdots,\delta_i^P(p_i,a_i),\cdots,p_n)$, where $a_i = \text{Top}(\gamma_i)$, $p_i = \delta_i^P(x_i,\gamma_i^R)$.

Suppose that the transition used for the move $\mathbf{F}(c,0,1) \rightarrow_H d$ be any transition added in 3 (other case of transition being one added in 4 is similar). We know that this was added due to existence of a transition $(q,\mathbf{Push}_i(b),q') \in \Delta_i$. Let $c' = (q',\gamma'_1,\cdots,b\gamma'_i,\cdots\gamma'_n)$, clearly $(c,0,1) \rightarrow (c',0,1)$ and $\mathbf{F}(c',0,1) = d$.

If $\mathbf{F}(c,0,1) \rightarrow_H q_w$, then clearly the corresponding transition used to add such a transition was either a pop $\mathbf{Pop}_i$ or a zero $\mathbf{Zero}_i$ move. Let us suppose that the transition was $(q,\mathbf{Zero}_i, q') \in \Delta_i$ (the other pop case is similar). Clearly such a move fixes a stack i, let $c' = (q',\gamma_1,\cdots,\gamma'_{i-1},\perp,\cdots,\gamma_n)$. Clearly we have $(c,0,1) \rightsquigarrow (c',i,1)$. But such a position is a trap, further it is clear that $(c',i,1) \in W(i)$ (since $\mathbf{F}((c,0,1) \rightsquigarrow q_w)$. By definition of G', we have $(c,0,1) \rightsquigarrow'_M p_{\text{win}}$. □

Claim 16. *The function $\mathbf{F}$ preserves player position and rank.*

Proof. Notice that functions τ and σ depend only on states, the proof is immediate from this fact.

□

From this, proof of 71 follows.

□

In addition note that the set of positions of the form $(c,0,1)$ that are winning for player 0 are precisely those in $S_{\text{Win}} = \{w \mid f(w) \text{ is winning for player-0}\}$ and this clearly is a recognizable set. This completes the proof of Lemma 70.

□

9.4.2 Decidability of k phase game

Next, using similar ideas to the ones elaborated above, we show that we can reduce the problem of determining the winning positions of a k bounded-phase MPDS game to determining winning positions of a different $k-1$ bounded-phase MPDS game .

The idea is to use the fact that 1-phase sub-game of a k-phase game is determined and reduce the k-phase bounded-phase MPDS game to a $k-1$ phase MPDS game. Notice that after execution of $k-1$ phases, what remains is a 1-phase sub-game. In this 1-phase sub-game, the stack contents of all other stacks (exclusing the currently active stack) are irrelevant and hence it can easily be simulated by a pushdown automata. Let $\mathscr{K} = \{(c,i,1) \mid (c,i,1) \in$

$\mathscr{E}^k \wedge i \in [1..n]\}$. Recall the pushdown automata P_i constructed in Lemma 70. As in the case of Lemma 70, we can provide a mapping $\mathbf{F}$ from the sub-game involving positions from $\mathscr{K}$ to positions in the game $\mathscr{H} = (\mathscr{C}(P_i), \rightarrow, \tau, \sigma)$, such that $\mathbf{F}$ satisfies the properties of Lemma 66 (as a matter of fact, the game graph K is isomorphic to the trap consisting of positions of the form $(c, i, 1), i \neq 0$ in the game graph of a 1-phase parity game). From this, we get the following Lemma which states that the set of winning positions of a 1 phase sub-game can be effectively determined using the set of winning positions of the pushdown system P_i.

Lemma 72. *$s \in \mathscr{K}$ is winning for player-0 iff $\mathbf{F}(s)$ is winning for player-0 in the pushdown game $\mathscr{H} = (\mathscr{C}(P_i), \rightarrow, \tau, \sigma)$*

Now to handle the case of k-phase game, we first invoke Theorem 29 to obtain $B_i = (Q_{B_i}, \Gamma \cup Q, s_i, \delta^{B_i}, F_i)$ that recognises the winning positions of the pushdown system P_i. Suppose at the end of $k-1$ phase, we know the state that the automata B_i reaches on reading stack i, then we can at the beginning of phase k determine whether player-0 is winning from that position or not. The case for 1 phase game was easy since we had only pushes to contend with (and hence it was possible to simulate B_i using only the state space). However, in case of a $k-1$ phase game, we need to also handle pop operations. Hence it is not possible to simulate B_i automata by just keeping it in state space. The informal idea is to keep the B_i automata as part of state space and simulate it on each push onto the stack-i. In addition, on each push, along with the stack symbol, we also store in the stack, the state of B_i that was reached before the current push. Now each time a pop operation is performed, we can retrieve the correct state of the B_i automata. The details are formalised below.

Let (k, M, τ, σ) be a bounded-phase game with $M = (n, Q, \Gamma, \Delta, q_0)$ with $k > 1$. We define the new MPDS as $M(k) = (n, Q_{M(k)}, \Gamma_{M(k)}, \Delta', q_0^{M(k)})$, where

- $Q_{M(k)} = Q \times Q_{B_1} \times \cdots \times Q_{B_n} \times \Gamma^n \times [0..n] \times [2..k] \cup \{q_w, q_l\}$
- $\Gamma_{M(k)} = \bigcup_{i \in [1..n]} (\Gamma \times Q_{B_i}) \cup \{\perp\}$
- $q_0^{M(k)} = (q_0, s_1, \cdots, s_n, \perp^n, 0, k)$

The transition relation Δ' is defined as follows

1. if $(q, \mathbf{Push}_i(b), q') \in \Delta$ then we have for all $i \in [1..n]$, $p_i \in Q_{B_i}$, $m \in [0..n]$, $l \in [2..k]$ and $a_i \in \Gamma$, $((q, p_1, \cdots, p_n, a_1, \cdots, a_n, m, l), \mathbf{Push}_i(a_i, p_i), (q', p_1, \cdots, p_{i-1}, \delta_i^P(p_i, a_i), \cdots, p_n, a_1, \cdots, a_{i-1}, b, a_{i+1}, \cdots, a_n, m, l)) \in \Delta'$. We always store the top of stack in the state space. Every time we push (say b), the previous top of stack in the state (a_i) is pushed into the actual stack and the top of stack in the state is replaced with the current push. Further the component corresponding to B_i automata in the state is also updated.
2. if $(q, \mathbf{Int}_i, q') \in \Delta$ then we have for all $i \in [1..n]$, $p_i \in Q_{B_i}$ and $a_i \in \Gamma$, $((q, p_1, \cdots, p_n, a_1, \cdots, a_n, m, l), \mathbf{Int}_i, (q', p_1, \cdots, p_n, a_1, \cdots, a_n, m, l)) \in \Delta'$.
3. For each $(q, \mathbf{Pop}_j(a_j), q') \in \Delta$ we add the following transitions.
   - $((q, p_1, \cdots, p_n, a_1, \cdots, a_n, 0, k), \mathbf{Pop}_j(b_j, p'_j), (q', p_1, \cdots, p_{j-1}, p'_j, p_{j+1}, \cdots, p_n, a_1, \cdots, a_{j-1}, b_j, a_{j+1}, \cdots, a_n, j, k)) \in \Delta'$, for all $b_j \in \Gamma$. This transition corresponds to the case where no pop or zero test operation were performed previously (currently active stack remains zero), in this case the currently active stack is updated with j and the phase component is left unchanged.

- $((q,p_1,\cdots,p_n,a_1,\cdots,a_n,j,l),\mathbf{Pop}_j(b_j,p'_j),(q',p_1,\cdots,p_{j-1},p'_j,p_{j+1},\cdots,p_n,a_1,\cdots,a_{j-1},b_j,a_{j+1},\cdots,a_n,j,l)) \in \Delta'$. The pop happens in the currently active stack and hence there is no change in phase.
- For any $l > 2, i \neq j$, $((q,p_1,\cdots,p_n,a_1,\cdots,a_n,i,l),\mathbf{Pop}_j(b_j,p'_j),(q',p_1,\cdots,p_{j-1},p'_j,p_{j+1},\cdots,p_n,a_1,\cdots,a_{j-1},b_j,a_{j+1},\cdots,a_n,j,l-1)) \in \Delta'$. This transition corresponding to pop from stack-j when the currently active stack is i. The phase number and the stack number are adjusted accordingly.
- For any $i \neq j$ and $\delta(p'_j,q') \in F_j$, $((q,p_1,\cdots,p_n,a_1,\cdots,a_n,i,2),\mathbf{Int},q_w) \in \Delta'$. During the last phase, instead of entering the phase, we goto q_w if the entered configuration is winning for player-0
- For any $i \neq j$ and $\delta(p'_j,q') \notin F_j$, $((q,p_1,\cdots,p_n,a_1,\cdots,a_n,i,2),\mathbf{Int},q_l) \in \Delta'$. During the last phase, instead of entering the phase, we goto q_l if the entered configuration is winning for player-1

4. For each $(q,\mathbf{Zero}_j,q') \in \Delta$ we add the following transitions.
   - $((q,p_1,\cdots,p_n,a_1,\cdots,a_{j-1},\perp,\cdots,a_n,0,k),\mathbf{Zero}_j,(q',p_1,\cdots,p_{j-1},\cdots,p_n,a_1,\cdots,a_{j-1},\perp,\cdots,a_n,j,k)) \in \Delta'$.
   - $((q,p_1,\cdots,p_n,a_1,\cdots,a_{j-1},\perp,\cdots,a_n,j,l),\mathbf{Zero}_j,(q',p_1,\cdots,p_{j-1},\cdots,p_n,a_1,\cdots,a_{j-1},\perp,\cdots,a_n,j,l)) \in \Delta'$, for all $l \in [2..k]$.
   - For all $l > 2$ and $i \neq j$, $((q,p_1,\cdots,p_n,a_1,\cdots,\perp,a_{j-1},\cdots,a_n,i,l),\mathbf{Zero}_j,(q',p_1,\cdots,p_{j-1},\cdots,p_n,a_1,\cdots,a_{j-1},\perp,\cdots,a_n,j,l-1)) \in \Delta'$.
   - For any $i \neq j$ and $\delta(s_j,q') \in F_j$, $((q,p_1,\cdots,p_n,a_1,\cdots,a_{j-1},\perp,\cdots,a_n,i,2),\mathbf{Int},q_w) \in \Delta'$.
   - For any $i \neq j$ and $\delta(s_j,q') \in F_j$, $((q,p_1,\cdots,p_n,a_1,\cdots,a_{j-1},\perp,\cdots,a_n,i,2),\mathbf{Int},q_w) \in \Delta'$.
5. We further add $(q_l,\mathbf{Int},q_l)$ and $(q_w,\mathbf{Int},q_w)$ to the transitions

Observe that any run of this system may involve at most $k-1$ phases, as every change of phase results in a reduction in the last component (of the state) and after $k-1$ reductions, leads to one of the states q_w or q_l (where only skip moves are enabled).

For correctness of the construction, we first define a ranking function σ' as follows. $\sigma'(q_w) = 0$, $\sigma'(q_l) = 1$ and for all other states $s = (q,p_1,\cdots,p_n,a_1,\cdots,a_n) \in Q_{M(k)}$, we let $\sigma'(s) = \sigma(q)$. Similarly we define τ' as $\tau'(q_w) = 0$, $\tau'(q_l) = 1$ and for all other states $s = (q,p_1,\cdots,p_n,a_1,\cdots,a_n) \in Q_{M(k)}$, we let $\tau'(s) = \tau(q)$ and we show that we may associate positions of the form (c,i,k) in the bounded-phase game on (k,M,τ,σ) with positions of the form $(d,i,k-1)$ in the bounded-phase game on $(k-1,M(k),\tau',\sigma')$ that preserves the winner.

For a sequence $w = a_n a_{n-1}\ldots a_1 a_0 \in (\Gamma \setminus \{\perp\})^+\perp$ and $1 \leq j \leq l$, let $\rho_j(w) = (a_{n-1},p_{n-1})\ldots(a_2,p_2)(a_1,p_1)(a_0,p_0)\perp$ (we let $\rho_j(\perp) = \perp$) where $p_0 = s_j$ and $p_i = \delta^{B_j}(p_{i-1},a_{i-1})$ for all $i \in [1..n]$. Further, let $\boldsymbol{\delta_j}(w) = \delta^{B_j}(p_{n-1},a_{n-1})$ (we let $\boldsymbol{\delta_j}(\perp) = p_0$). We now define the map $\mathbf{F}$ from the k-bounded-phase parity game on (k,A,τ,σ) to the $k-1$-bounded-phase parity game on the game $(k-1,A(k),\tau,\sigma)$ as follows:

- $\mathbf{F}((q,\gamma_1,\cdots,\gamma_n),i,j) = (((q,\boldsymbol{\delta_1}(\gamma_1),\cdots,\boldsymbol{\delta_n}(\gamma_n),\mathrm{Top}(\gamma_1),\cdots,\mathrm{Top}(\gamma_l),i,j),\rho_1(\gamma_1),\cdots,\rho_l(\gamma_n)),i,j)$, if $j > 1$. Where Top is a function that returns top of the stack.
- $\mathbf{F}((q,\gamma_1,\cdots,\gamma_n),i,1) = q_w$ if $(q,\gamma_1,\cdots,\gamma_n)$ is winning for player 0.
- $\mathbf{F}((q,\gamma_1,\cdots,\gamma_n),i,1) = q_l$ if $(q,\gamma_1,\cdots,\gamma_n)$ is losing for player 0.

Lemma 73. *The following holds*

1. *The map* $\mathbf{F}$ *preserves ownership and rank of all positions* (c,i,j) *with* $j>1$.
2. *For any configuration* (c,i,j) *if* $(c,i,j)\rightarrow(c',i',j')$ *with* $j'>1$ *then* $\mathbf{F}((c,i,j))\rightarrow\mathbf{F}((c',i',j'))$
3. *For any configuration* (c,i,j) *if* $\mathbf{F}(c,i,j)\rightarrow d$ *for any* $d\notin\{q_w,q_l\}$ *then there is* (c',i',j') *with* $(c,i,j)\rightarrow(c',i',j')$, $j'>1$ *and* $\mathbf{F}(c',i',j')=d$.
4. *if* $(c,i,2)\rightarrow(c',i',1)$, *then* $\mathbf{F}(c,i,2)\rightarrow q_w$ *iff* $(c',i',1)$ *is a winning position.*
5. *if* $(c,i,2)\rightarrow(c',i',1)$, *then* $\mathbf{F}(c,i,2)\rightarrow q_l$ *iff* $(c',i',1)$ *is a losing position.*

Hence we can effectively determine the set of all positions that are winning for player 0, *in the bounded-phase parity game* (k,M,τ,σ) .

Proof. This proof is very similar to proof of Lemma 70, hence we will omit it.

□

Note that we have successfully reduced a k-bounded-phase game to a $k-1$-bounded-phase game. However note that each such reduction is exponential in the size of previous system. Since we do as many such reductions as the number of phases, the overall complexity will be a tower of exponents (size of tower being the number of phases). Hence the overall reduction is a NON-ELEMENTARY. The question now is whether such a NON-ELEMENTARY construction can be avoided. We will in sequel show a lower bound that suggests that such a complexity cannot be avoided.

9.5 Lower bounds for bounded phase parity games

We show that the satisfiability of a first order formula with ordering relation over natural numbers, can be reformulated as a bounded-phase parity game over *MPDS*. We first briefly recall the first order theory of natural numbers with ordering relation ($FO(<)$).

Let $\mathcal{V}$ be countably infinite set of variables, we will use $x,y,z,x_1,x_2\cdots$ to refer to the variables in $\mathcal{V}$. The set of terms in $FO(<)$ is defined as $t:=x\mid t<t\mid t=t$. The set of formulas is defined to be $\Psi:=t\mid\neg t\mid\Psi\vee\Psi\mid\Psi\wedge\Psi\mid\forall x\Psi\mid\exists x\Psi$. The notion of free, bound (quantified) variable are defined as usual. We write $FreeVar(\Psi)\subseteq\mathcal{V}$ to denote the set of all free variables (unquantified variables) of Ψ.

Given any formula Ψ over variables $\mathcal{V}$, we define a valuation function as $\mu:\mathcal{V}\mapsto\mathbb{N}$. The notion of a valuation function satisfying a formula is inductively defined below.

- $\mu\models(x<y)$ iff $\mu(x)<\mu(y)$
- $\mu\models(x=y)$ iff $\mu(x)=\mu(y)$
- $\mu\models(\neg\Psi')$, iff $\mu\not\models\Psi'$
- $\Psi=\Psi_1\vee\Psi_2$ iff $\mu\models\Psi_1$ or $\mu\models\Psi_2$
- $\Psi=\Psi_1\wedge\Psi_2$ iff $\mu\models\Psi_1$ and $\mu\models\Psi_2$
- $\Psi=\exists x.\Psi_1$ iff there is a $m\in\mathbb{N}$ such that $\mu[x\leftarrow m]\models\Psi_1$, where $\mu[x\leftarrow m]$ is a new valuation function μ', such that for $y\neq x$, $\mu'(y)=\mu(y)$ and $\mu'(x)=m$.
- $\Psi=\forall x.\Psi_1$ iff for all $m\in\mathbb{N}$, we have $\mu[x\leftarrow m]\models\Psi_1$

Given any formula Ψ and a valuation function μ, we call μ a model of Ψ, iff $\mu \models \Psi$. A formula with no free variables is called a sentence. A sentence is said to satisfied iff there is some valuation function that satisfies it.

Note that negation is defined only on the atomic formulas. However, given any formula Ψ, we inductively define $\mathsf{Dual}(\Psi)$ as follows.

- $\mathsf{Dual}(x = y) = \neg(x = y)$
- $\mathsf{Dual}(x < y) = \neg(x < y)$
- $\mathsf{Dual}(\Psi_1 \wedge \Psi_2) = \mathsf{Dual}(\Psi_1) \vee \mathsf{Dual}(\Psi_2)$
- $\mathsf{Dual}(\Psi_1 \vee \Psi_2) = \mathsf{Dual}(\Psi_1) \wedge \mathsf{Dual}(\Psi_2)$
- $\mathsf{Dual}(\exists x.\Psi) = \forall x.\mathsf{Dual}(\Psi)$
- $\mathsf{Dual}(\forall x.\Psi) = \exists x.\mathsf{Dual}(\Psi)$

It is easy to see that for any given formula Ψ and a model μ, $\mu \models \Psi$ iff $\mu \not\models \mathsf{Dual}(\Psi)$.

Given a formula Ψ and its model μ we define a linearisation of μ w.r.t. Ψ to be a word of the form $x_1 a^{j_1} x_2 a^{j_2} \cdots x_n a^{j_n} \perp$, where $\{x_1, \cdots, x_n\} = \mathit{FreeVar}(\Psi)$ and for each $k \in [1..n]$, $j_k + j_{k-1} \cdots j_n = \mu(x_{i_k})$.

Similarly, for any set of variables $\mathcal{V}$, we say a string $(\alpha = x_n a^{i_n} x_{n-1} a^{i_{n-1}} \cdots x_1 a^{i_1}) \in (\mathcal{V} \cup a)^*$ is a valuation string if for all $l, k \in [1..n]$, we have $l \neq k \implies x_l \neq x_k$ (i.e. each x_i appears at most once). Firstly, given any valuation string $\alpha = x_n a^{i_n} x_{n-1} a^{i_{n-1}} \cdots x_1 a^{i_1}$ and a set of variable $\mathcal{V}$, we define $\mu_\alpha^{\mathcal{V}}$ as, for any $j \in [1..n]$, $\mu_\alpha^{\mathcal{V}}(x_j) = a^{i_j} + a^{i_{j-1}} + \cdots + a^{i_1}$, i.e. it maps the variables x_j, to a value equal to number of a's appearing before it in α. For any $x \in \mathcal{V}$ such that x does not appear in α, we let $\mu_\alpha(x) = 0$.

Given any formula Ψ, we use $Cl(\Psi)$ to indicate the set of all formulas obtained by closing the formula Ψ over subformulas. Note that even if Ψ is a sentence, elements of $Cl(\Psi)$ can have free variables. We now show that satisfiability of first order formula over $(N, <)$ (known to have non-elementary complexity [136]) can be reduced to parity games over bounded-phase MPDS. With out loss of generality, we will also assume that each variable in the formula occurs at most once.

Let Ψ be the given formula and let μ be an initial valuation. We are interested in knowing if $\mu \models \Psi$. The informal idea is to construct an MPDS, in which the state space contains the subformulas of the given formula Ψ (i.e. $Cl(\Psi)$), along with some intermediary states. The MPDS starts with the linearisation of the initial valuation in its stack and the formula Ψ. At any point in the game, the *MPDS* maintains the unprocessed part of the formula $\phi \in Cl(\Psi)$ as part of its state space and the linear encoding (linearisation) of the current valuation μ (w.r.t. ϕ) in its stack .

There are two parts to the game depending on whether the unprocessed part begins with a quantifier or not. If the unprocessed part of formula begins with a quantifier $\forall$, then player-1 strips off the quantifier and assigns a valuation to the corresponding variable by modifying the stack. If it begins with a $\exists$ quantifier then the valuation is provided by player-0. For this, if the valuation that the player wishes to provide is less than the some variables already in the stack, the elements are moved to stack-2 till the appropriate position is found, the variable is placed in this position and the elements from stack-2 are moved back onto stack-1. If the valuation that the player wishes to provide is greater than all the variables present in the stack,

extra a's are appended and the variable is placed.

If the outer most operator is $\wedge$, then the player-1 chooses a subformula and the game proceeds. If the outer most operator is $\vee$, then the player-0 selects a subformula. The game proceeds till the unprocessed part is an atomic formula, in which case it can easily be verified based on the valuations in the stack.

We will formally describe the construction of the required *MPDS* $M_\Psi = (2, Q, \Gamma = \{a, \bot\} \cup FreeVar(\Psi), \Delta, q_0)$. We will describe the construction in two parts. The first part describes the moves till we reach an atomic formula. It contains the following set of states $Cl(\Psi) \cup Cl(\Psi) \times \{\text{lt}, \text{gt}, m_{1,2}, m_{2,1}\} \cup Cl(\Psi) \times \{m_{1,2}, m_{2,1}\} \times (\{a\} \times FreeVar(\Psi))$. The transition relation Δ is defined as follows. We will used $?x$ to denote either of $\exists x$ or $\forall x$.

a.1 For all $\psi_1 \wedge \psi_2 \in Cl(\Phi)$, the transitions $(\psi_1 \wedge \psi_2, \mathbf{Int}, \psi_1), (\psi_1 \wedge \psi_2, \mathbf{Int}, \psi_2) \in \Delta$.
a.2 For all $\psi_1 \vee \psi_2 \in Cl(\Phi)$, the transitions $(\psi_1 \vee \psi_2, \mathbf{Int}, \psi_1)$ and $(\psi_1 \vee \psi_2, \mathbf{Int}, \psi_2) \in \Delta$
a.3 For all $?x.\psi \in Cl(\Phi)$, we add $(?x.\psi, \mathbf{Int}, (?x.\psi, \text{lt}))$ and $(?x.\psi, \mathbf{Int}, (?x.\psi, \text{gt})) \in \Delta$, this transition enables guessing whether the current variable x needs to be inserted in between the existing variable (valuation falls below the current maximum) or needs to be inserted on top (is greater than the current maximum).
a.4 We also add $((?x.\psi, \text{gt}), \mathbf{Push}_1(a), (?x.\psi, \text{gt})) \in \Delta$ (pushes a into stack-1 to increase possible valuation for x)
a.5 We also add $((?x.\psi, \text{gt}), \mathbf{Push}_1(x), \psi) \in \Delta$ (Marks position of x and shift to the sub-formula).
a.6 We add $((?x.\psi, \text{lt}), \mathbf{Int}, (?x.\psi, m_{1,2})) \in \Delta$ (Begin moving from stack-1 to 2).
a.7 We add $((?x.\psi, m_{1,2}), \mathbf{Pop}_1(a), (?x.\psi, m_{1,2}, a)) \in \Delta$ and $((?x.\psi, m_{1,2}, a), \mathbf{Push}_2(a), (?x.\psi, m_{1,2})) \in \Delta$, $\forall a \in \Gamma \setminus \{\bot\}$ (moves values from stack-1 to 2)
a.8 Similarly we add $((?x.\psi, m_{2,1}), \mathbf{Pop}_2(a), (?x.\psi, m_{2,1}, a)) \in \Delta$ and $((?x.\psi, m_{2,1}, a), \mathbf{Push}_1(a), (?x.\psi, m_{2,1})) \in \Delta$, $\forall a \in \Gamma \setminus \{\bot\}$ (moves values from stack-2 to 1)
a.9 We add $((?x.\psi, m_{1,2}), \mathbf{Push}_1(x), (?x.\psi, m_{2,1})) \in \Delta$ (Begin moving from stack-2 back to 2)
a.10 We add $((?x.\psi, m_{2,1}), \mathbf{Zero}_2, \psi) \in \Delta$ (Move to the next sub-formula) .

In the second part, we describe the state space starting at a state of the form $(x = y)$ or $(x < y)$ that determines winner of the game. It contains the following set of states $\{x = y, x < y, a_y \mid x, y \in V\} \cup \{T, F\}$. The transitions are described below.

b.1 $(x = y, \mathbf{Pop}_1(z), x = y) \in \Delta$, for all $z \in V \setminus \{x, y\} \cup \{a\}$, pop all elements other than x, y.
b.2 $(x = y, \mathbf{Pop}_1(z), z') \in \Delta$, for $z \in \{x, y\}, z' \in \{x, y\} \setminus \{z\}$, as soon as one of $\{x, y\}$ is seen (say x) goto a state expecting to see the other variable (y if we saw x previously).
b.3 For $x \in V$, we add $(x, \mathbf{Pop}_1(a), F) \in \Delta$, if we see an a when we are expecting a variable in $x \in V$, we goto the losing state F.
b.4 For $x, y \in V$, $y \neq x$ we add $(x, \mathbf{Pop}_1(y), x) \in \Delta$, if we see a variable other than x, we skip.
b.5 For $x \in V$, we add $(x, \mathbf{Pop}_1(x), T) \in \Delta$, if we see a variable x, we goto winning state.
b.6 The set transitions needed for $\neg(x = y)$ are similar
b.7 $(x < y, \mathbf{Pop}_1(z), x < y) \in \Delta$ for all $z \in V \setminus \{x, y\} \cup \{a\}$, pop all elements other than x, y.
b.8 $(x < y, \mathbf{Pop}_1(x), F) \in \Delta$, if you find x before y, clearly $x > y$ in the valuation, goto losing state.
b.9 $(x < y, \mathbf{Pop}_1(y), a_x) \in \Delta$, if you find y before x, we need to verify if both values are not equal.

b.10 $(a_x, \mathbf{Pop}_1(a), T) \in \Delta$, if you find a before x, we goto state T.
b.11 $(a_x, \mathbf{Pop}_1(x), F) \in \Delta$, if you find x before a, we goto state F.
b.12 For all $z \in V, z \neq x$, we add $(a_x, \mathbf{Pop}_1(z), a_x) \in \Delta$ to skip rest of the variables.
b.13 The set transitions needed for $\neg(x < y)$ are similar.
b.14 We also add $(T, \mathbf{Int}, T)$ and $(F, \mathbf{Int}, F)$ to Δ.

We will now consider the bounded-phase parity game given by $(|\Psi|, \mathscr{C}(M_\Psi), \tau, \sigma)$ where $\sigma : Q \mapsto \{0,1\}$ and $\tau : Q \mapsto \{0,1\}$ are defined as:

- We let $\sigma(T) = 0$ and $\sigma(F) = 1$.
- We let $\sigma((\exists x.\Psi', gt)) = 1$ and $\sigma((\forall x.\Psi', gt)) = 0$. This will ensure that either of the player cannot simply win by just pushing elements onto the stack, for all other $q \in Q$, we let $\sigma(q) = 0$.
- For any state s such that its subformula component is of the form, $\forall x.\Psi'$ or $\Psi_1 \wedge \Psi_2$, we let $\tau(s) = 1$ (player-1 position). Otherwise, $\tau(s) = 0$ i.e. we let all other states to be player-0 position.

Notice that along any positions in the game, where the state is only a subformula from $Cl(\Psi)$, the stack content of the first stack α is a valuation string. This is easy to see since by nature of the formula we have assumed that along any path, we can never encounter the same variable twice. Clearly such a $\mu^{\mathcal{V}}_\alpha$ function is a valuation function. We show in Lemma 74, that along positions in game graph where the state is only a subformula from $Cl(\Psi)$, the valuation function constructed out of the content of stack 1 is actually a model of the subformula iff player-0 has a winning strategy from that position.

Lemma 74. *Give any configuration $c \in \mathscr{C}(M)$ which is of the form $(\Psi, \alpha\bot, \bot)$ where $\alpha = x_n a^{i_n} x_{n-1} a^{i_{n-1}} \cdots x_1 a^{i_1} \in (V\Gamma^*)^*$ is a valuation string containing all the free variables of Ψ, then $\mu^{\mathcal{V}}_\alpha \models \Psi$ iff player-0 has a bounded-phase winning strategy from c.*

Proof. ($\Leftarrow$) We will assume that player-0 has a winning strategy from $(\Psi, \alpha\bot, \bot)$ and show $\mu^{\mathcal{V}}_\alpha \models \Psi$, we will prove this by induction on structure of the formula. We let f to be any winning strategy function for player-0.

- case when Ψ is of the form $x = y$ is easy to see. Since the play is winning for player-0, it is clear that the play eventually reaches the node T. By construction, reaching winning state T is possible from $(x = y, \alpha\bot, \bot)$ if only if x, y are seen adjacently, which implies $\mu^{\mathcal{V}}_\alpha(x) = \mu^{\mathcal{V}}_\alpha(y)$. Case where the atomic formula is of the form $\neg(x = y)$ is similar.
- case when Ψ is of the form $x < y$ is also easy to see. By construction, reaching winning state T is possible in this case only if x are seen first and is separated from y by an a, which implies $\mu^{\mathcal{V}}_\alpha(x) < \mu^{\mathcal{V}}_\alpha(y)$. Case where the atomic formula is of the form $\neg(x < y)$ is similar.
- In case where $\Psi = \Psi_1 \vee \Psi_2$, the configuration $c = (\Psi, \alpha\bot, \bot)$ belongs to player-0. There are two possible ways for player 0 to continue his game i.e. $(\Psi_1, \alpha\bot, \bot)$ or $(\Psi_2, \alpha\bot, \bot)$. Let $f(\Psi, \alpha\bot, \bot) = (\Psi_1, \alpha\bot, \bot)$ (strategy corresponding to player-0 for this node). Clearly by induction we have $\mu^{\mathcal{V}}_\alpha \models \Psi_1$. Hence we also have $\mu^{\mathcal{V}}_\alpha \models \Psi$.
- In case where $\Psi = \Psi_1 \wedge \Psi_2$, the configuration $c = (\Psi, \alpha\bot, \bot)$ belongs to player-1, further since node $(\Psi, \alpha\bot, \bot)$ is winning for player-0, from induction we can deduce that $\mu^{\mathcal{V}}_\alpha \models \Psi_1$ and $\mu^{\mathcal{V}}_\alpha \models \Psi_2$ are satisfiable. Hence we also have $\mu^{\mathcal{V}}_\alpha \models \Psi$.

- In case where $\Psi = \exists x.\Psi'$. We know that $c = (\Psi, \alpha\bot, \bot)$ is winning for player-0. It is easy to see that there is by construction, a node $c' = (\Psi', \alpha'\bot, \bot)$ such that $c \rightarrow^* c'$ and by induction $\mu^{\mathcal{V}}_{\alpha'} \models \Psi'$. From this and the definition of satisfiability of a valuation function, it is easy to see that $\mu^{\mathcal{V}}_{\alpha} \models \Psi$.
- In case where $\Psi = \forall x.\Psi'$. We know that $c = (\Psi, \alpha\bot, \bot)$ is winning for player-0. From nature of construction, for every value m, possible for x, player-1 can reach a configuration c' from c such that $c' = (\Psi', \alpha'\bot, \bot)$ such that $\mu^{\mathcal{V}}_{\alpha'}(x) = m$. Futher by induction, for each of these c', $\mu^{\mathcal{V}}_{\alpha'} \models \Psi'$. From this it is easy to see that $\mu^{\mathcal{V}}_{\alpha} \models \Psi$
  ($\Leftarrow$)
  For this direction, if player-0 is not winning from any position $c = (\Psi, \alpha\bot, \bot)$, then by determinacy, player-1 is winning from it. Using arguments similar to that above, we now show that if player 1 is winning from any node $c = (\Psi', \alpha'\bot, \bot)$ then, $\mu^{\mathcal{V}}_{\alpha'} \models \mathsf{Dual}(\Psi)$.

  Lemma 75. *Given any configuration $c \in \mathscr{C}(M)$ which is of the form $(\Psi, \alpha\bot, \bot)$ where $\alpha = x_n a^{i_n} x_{n-1} a^{i_{n-1}} \cdots x_1 a^{i_1} \in (V\Gamma^*)^*$ is a valuation string containing all the free variables of Ψ, then if player-*1 *has a bounded-phase winning strategy from c, then $\mu^{\mathcal{V}}_{\alpha} \models \mathsf{Dual}(\Psi)$.*

  Proof. We will assume that player-1 has a winning strategy from $(\Psi, \alpha\bot, \bot)$ and show $\mu^{\mathcal{V}}_{\alpha} \models \mathsf{Dual}(\Psi)$, we will prove this by induction on structure of the formula. We let f to be any winning strategy function for player-1.

  – case when Ψ is of the form $(x = y)$ is easy to see. $\mathsf{Dual}((x = y)) = \neg(x = y)$. Since the play is winning for player-1, it is clear that the play eventually reaches the node F. By construction, reaching winning state F is possible from $(x = y, \alpha\bot, \bot)$ if only if x, y are not seen adjacently, which implies $\mu^{\mathcal{V}}_{\alpha}(x) \neq \mu^{\mathcal{V}}_{\alpha}(y)$. Case where the atomic formula is of the form $\neg(x = y)$ is similar.
  – case when Ψ is of the form $x < y$ is also easy to see. Firstly $\mathsf{Dual}((x < y)) = \neg(x < y)$. By construction, reaching winning state F is possible in this case only if y are seen first and then x, which implies $\mu^{\mathcal{V}}_{\alpha}(y) \leq \mu^{\mathcal{V}}_{\alpha}(x)$. Case where the atomic formula is of the form $\neg(x < y)$ is similar.
  – In case where $\Psi = \Psi_1 \vee \Psi_2$, the configuration $c = (\Psi, \alpha\bot, \bot)$ belongs to player-0. There are two possible ways for player 0 to continue his game i.e. $(\Psi_1, \alpha\bot, \bot)$ or $(\Psi_2, \alpha\bot, \bot)$. Since the node is winning for player-1, by induction we have $\mu^{\mathcal{V}}_{\alpha} \models \mathsf{Dual}(\Psi_1)$ and $\mu^{\mathcal{V}}_{\alpha} \models \mathsf{Dual}(\Psi_2)$. Notice that $\mathsf{Dual}(\Psi) = \mathsf{Dual}(\Psi_1) \wedge \mathsf{Dual}(\Psi_2)$. Hence $\mu^{\mathcal{V}}_{\alpha} \models \mathsf{Dual}(\Psi)$.
  – In case where $\Psi = \Psi_1 \wedge \Psi_2$, $c = (\Psi, \alpha\bot, \bot)$ is a player-1 position. Since it is winning for player-1. Without loss of generality, let $c' = (\Psi_1, \alpha\bot, \bot)$ be the node reachable from c using the strategy function, by induction $\mu^{\mathcal{V}}_{\alpha} \models \mathsf{Dual}(\Psi_1)$. Since $\mathsf{Dual}(\Psi) = \mathsf{Dual}(\Psi_1) \vee \mathsf{Dual}(\Psi_2)$, it follows that $\mu^{\mathcal{V}}_{\alpha} \models \mathsf{Dual}(\Psi)$.
  – In case where $\Psi = \exists x.\Psi'$. We know that $c = (\Psi, \alpha\bot, \bot)$ is winning for player-1. From nature of construction, for every value m, possible for x, player-0 can reach a configuration c' from c such that $c' = (\Psi', \alpha'\bot, \bot)$ such that $\mu^{\mathcal{V}}_{\alpha'}(x) = m$. Futher by induction, for each of these c', $\mu^{\mathcal{V}}_{\alpha'} \models \mathsf{Dual}(\Psi')$. From this it is easy to see that $\mu^{\mathcal{V}}_{\alpha} \models \mathsf{Dual}(\Psi)$
  – In case where $\Psi = \forall x.\Psi'$, the position $c = (\Psi, \alpha\bot, \bot)$ is a player-1 position. Since it it winning for player-1, using the strategy function, we can find a c' such that $c \rightarrow^* c' = (\Psi', \alpha'\bot, \bot)$ from where player-1 is winning. By induction we have $\mu^{\mathcal{V}}_{\alpha} \models \mathsf{Dual}(\Psi')$. Since

$\mathsf{Dual}(\Psi) = \exists \mathsf{Dual}(\Psi')$, the result follows.

□

Now, using Lemma 75 and the fact that $\mu_\alpha^\mathcal{Y} \models \mathsf{Dual}(\Psi)$ iff $\mu_\alpha^\mathcal{Y} \not\models \Psi$ this we get $\mu_\alpha^\mathcal{Y} \not\models \Psi$.

□

Now the following corollary is easy to see.

Corollary 4. *For any setence Ψ, Ψ is satisfiable iff $(\Psi, \bot, \bot)$ is winning for player 0 in the game $(|\Psi|, \mathscr{C}(M_\Psi), \tau, \sigma)$. Thus deciding bounded-phase games has a* NON-ELEMENTARY *lower bound.*

9.6 Conclusion

In this chapter, we considered the problem of parity games over multi-pushdown systems with bounded-phase restriction. We showed an inductive NON-ELEMENTARY procedure to solve this problem. We also showed hardness for this problem by reducing the satisfiability of $FO(<)$ over natural numbers to a bounded-phase parity game.

Chapter 10

Discussion

In this thesis we have studied a number problems related to automata theoretic models of concurrent recursive programs. We conclude with a short discussion on some directions to extend the work reported here.

In chapter 3, we introduced a model called shared-memory concurrent pushdown system. We showed that even when only two 1 counter systems are communicating via 1-bit shared memory, the reachability problem is undecidable. We then introduced a new restriction called the bounded-stage restriction. We showed that when there are two pushdowns and one counter, the reachability under bounded stage restriction is undecidable. We then showed that when at most one pushdown system is involved, the problem is decidable. In this setting the case where there are only 2 pushdown systems involved is still open. This seems to be a hard problem. Another possible direction to extend this work would be to reason about omega regular properties of such systems under the bounded-stage restriction, or check for the existence of a bounded-stage non-terminating ultimately periodic computation.

In chapter 4, we showed how to obtain a polynomial sized finite representation for downward and upward closure of the language of a counter automata. We also showed how to obtain a sub-exponential sized finite representation for the Parikh image abstraction of the language of a counter automata. The question of whether such a sub-exponential sized representation is optimal is still open. Also, language theoretic problems on 1 counter automata seems to be not as well studied as CFLs or regular languages and a lot of work remains to be done there.

In chapter 6, we showed how to solve the problem of model checking LTL formulas over scope-bounded computations of an MPDS. The current lower bound for this problem depends only on the size of the LTL formula. One possible extension to this work would be to reason about exact complexity for the decision procedure when the LTL formula is fixed. The global model checking on multi-pushdown systems with bounded-scope restriction also remains to be investigated..

In chapter 7 we described the AOMPDS model and its applications. It remains to be seen if there is a simple argument that shows the decidability of reachability for OMPDS via a reduction to AOMPDSs.

In chapter 9, we gave a NON-ELEMENTARY procedure to solve the parity game on multi-

pushdown system with bounded-phase restriction. We also showed a matching lower bound for the same. One question that arises is what happens if you consider a weaker restriction or a weaker logic. In [19], we showed that even if we consider the bounded-context restriction and an EF fragment of CTL, the model checking problem is still NON-ELEMENTARY HARD . One can then ask if there is any weaker branching time fragment (for e.g. the EG fragment) for which the problem is tractable.

Bibliography

[1] Parosh Aziz Abdulla. Well (and better) quasi-ordered transition systems. *Bulletin of Symbolic Logic*, 16(4):457–515, 2010.

[2] Parosh Aziz Abdulla. Regular model checking. *STTT*, 14(2):109–118, 2012.

[3] Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Jonathan Cederberg. Analysis of message passing programs using SMT-solvers. In *ATVA'13*, volume 8172 of *LNCS*, pages 272–286, 2013.

[4] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Roland Meyer, and Mehdi Seyed Salehi. What's decidable about availability languages? In *FSTTCS'15*, volume 45 of *LIPIcs*, pages 192–205, 2015.

[5] Parosh Aziz Abdulla, Ahmed Bouajjani, Jonathan Cederberg, Frédéric Haziza, and Ahmed Rezine. Monotonic abstraction for programs with dynamic memory heaps. In *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7-14, 2008, Proceedings*, pages 341–354, 2008.

[6] Parosh Aziz Abdulla, Ahmed Bouajjani, and Bengt Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. In *Computer Aided Verification, 10th International Conference, CAV'98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, pages 305–318, 1998.

[7] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. In *Proceedings of the Eighth Annual Symposium on Logic in Computer Science (LICS '93), Montreal, Canada, June 19-23, 1993*, pages 160–170, 1993.

[8] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Inf. Comput.*, 127(2):91–101, 1996.

[9] Parosh Aziz Abdulla and Aletta Nylén. Better is better than well: On efficient verification of infinite-state systems. In *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000*, pages 132–140, 2000.

[10] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, pages 209–229, 1992.

[11] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[12] Rajeev Alur and Pavol Černý. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *POPL'11*, pages 599–610, 2011.

[13] Aurore Annichini, Eugene Asarin, and Ahmed Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In E. Allen Emerson and A. Prasad Sistla, editors, *CAV*, volume 1855 of *LNCS*, pages 419–434. Springer, 2000.

[14] Mohamed Faouzi Atig. Global model checking of ordered multi-pushdown systems. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, pages 216–227, 2010.

[15] Mohamed Faouzi Atig. *VE ÌĄRIFICATION DE PROGRAMMES CONCURRENTS: DE ÌĄCIDABILITE ÌĄ ET COMPLEXITE ÌĄ*. PhD thesis, UNIVERSITE ÌĄ PARIS DIDEROT (PARIS 7), 2010.

[16] Mohamed Faouzi Atig, Benedikt Bollig, and Peter Habermehl. Emptiness of multi-pushdown automata is 2etime-complete. In *Developments in Language Theory, 12th International Conference, DLT 2008, Kyoto, Japan, September 16-19, 2008. Proceedings*, pages 121–133, 2008.

[17] Mohamed Faouzi Atig, Ahmed Bouajjani, Michael Emmi, and Akash Lal. Detecting fair non-termination in multithreaded programs. In *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, pages 210–226, 2012.

[18] Mohamed Faouzi Atig, Ahmed Bouajjani, K. Narayan Kumar, and Prakash Saivasan. Linear-time model-checking for multithreaded programs under scope-bounding. In *Automated Technology for Verification and Analysis - 10th International Symposium, ATVA 2012, Thiruvananthapuram, India, October 3-6, 2012. Proceedings*, pages 152–166, 2012.

[19] Mohamed Faouzi Atig, Ahmed Bouajjani, K. Narayan Kumar, and Prakash Saivasan. Model checking branching-time properties of multi-pushdown systems is hard. *CoRR*, abs/1205.6928, 2012.

[20] Mohamed Faouzi Atig, Ahmed Bouajjani, K. Narayan Kumar, and Prakash Saivasan. On bounded reachability analysis of shared memory systems. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, pages 611–623, 2014.

[21] Mohamed Faouzi Atig, Ahmed Bouajjani, and Shaz Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. In *Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009,*

Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings, pages 107–123, 2009.

[22] Mohamed Faouzi Atig, Ahmed Bouajjani, and Tayssir Touili. Analyzing asynchronous programs with preemption. In *FSTTCS'08*, volume 2 of *LIPIcs*, pages 37–48, 2008.

[23] Mohamed Faouzi Atig, Ahmed Bouajjani, and Tayssir Touili. On the reachability analysis of acyclic networks of pushdown systems. In *CONCUR 2008 - Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19-22, 2008. Proceedings*, pages 356–371, 2008.

[24] Mohamed Faouzi Atig, Ahmed Bouajjani, and Tayssir Touili. On the reachability analysis of acyclic networks of pushdown systems. In *CONCUR*, volume 5201 of *Lecture Notes in Computer Science*, pages 356–371. Springer, 2008.

[25] Mohamed Faouzi Atig, K. Narayan Kumar, and Prakash Saivasan. Adjacent ordered multi-pushdown systems. In *Developments in Language Theory - 17th International Conference, DLT 2013, Marne-la-Vallée, France, June 18-21, 2013. Proceedings*, pages 58–69, 2013.

[26] Mohamed Faouzi Atig, K. Narayan Kumar, and Prakash Saivasan. Adjacent ordered multi-pushdown systems. *Int. J. Found. Comput. Sci.*, 25(8):1083–1096, 2014.

[27] Mohamed Faouzi Atig and Tayssir Touili. Verifying parallel programs with dynamic communication structures. In *Implementation and Application of Automata, 14th International Conference, CIAA 2009, Sydney, Australia, July 14-17, 2009. Proceedings*, pages 145–154, 2009.

[28] Georg Bachmeier, Michael Luttenberger, and Maximilian Schlund. Finite automata for the sub- and superword closure of CFLs: Descriptional and computational complexity. In *LATA'15*, volume 8977 of *LNCS*, pages 473–485, 2015.

[29] Thomas Ball, Rupak Majumdar, Todd D. Millstein, and Sriram K. Rajamani. Automatic predicate abstraction of C programs. In *Proceedings of the 2001 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Snowbird, Utah, USA, June 20-22, 2001*, pages 203–213, 2001.

[30] Thomas Ball and Sriram K. Rajamani. Bebop: A symbolic model checker for boolean programs. In *SPIN Model Checking and Software Verification, 7th International SPIN Workshop, Stanford, CA, USA, August 30 - September 1, 2000, Proceedings*, pages 113–130, 2000.

[31] Sébastien Bardin, Alain Finkel, Jérôme Leroux, and Laure Petrucci. FAST: acceleration from theory to practice. *STTT*, 10(5):401–424, 2008.

[32] Sébastien Bardin, Alain Finkel, Jérôme Leroux, and Philippe Schnoebelen. Flat acceleration in symbolic model checking. In *ATVA*, volume 3707 of *LNCS*, pages 474–488. Springer, 2005.

[33] Béatrice Bérard and Laurent Fribourg. Reachability analysis of (timed) petri nets using real arithmetic. In *CONCUR*, volume 1664 of *LNCS*, pages 178–193. Springer, 1999.

[34] J. Berstel. Transductions and context-free langages. TeubnerStudienbucher Informatik, 1979.

[35] Dirk Beyer, Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. The software model checker blast. *STTT*, 9(5-6):505–525, 2007.

[36] Henrik Björklund, Sven Sandberg, and Sergei G. Vorobyov. A discrete subexponential algorithm for parity games. In *STACS 2003, 20th Annual Symposium on Theoretical Aspects of Computer Science, Berlin, Germany, February 27 - March 1, 2003, Proceedings*, pages 663–674, 2003.

[37] Bernard Boigelot. On iterating linear transformations over recognizable sets of integers. *Theor. Comput. Sci.*, 309(1-3):413–468, 2003.

[38] Bernard Boigelot. Domain-specific regular acceleration. *STTT*, 14(2):193–206, 2012.

[39] Bernard Boigelot and Pierre Wolper. Symbolic verification with periodic sets. In *CAV*, volume 818 of *LNCS*, pages 55–67. Springer, 1994.

[40] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, volume 1243 of *LNCS*, pages 135–150. Springer, 1997.

[41] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR '97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997, Proceedings*, Lecture Notes in Computer Science, pages 135–150. Springer, 1997.

[42] Ahmed Bouajjani and Peter Habermehl. Symbolic reachability analysis of FIFO channel systems with nonregular sets of configurations (extended abstract). In *Automata, Languages and Programming, 24th International Colloquium, ICALP'97, Bologna, Italy, 7-11 July 1997, Proceedings*, pages 560–570, 1997.

[43] Ahmed Bouajjani and Peter Habermehl. Symbolic reachability analysis of fifo-channel systems with nonregular sets of configurations. *Theor. Comput. Sci.*, 221(1-2):211–250, 1999.

[44] Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular model checking. In *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, pages 403–418, 2000.

[45] Luca Breveglieri, Alessandra Cherubini, Claudio Citrini, and Stefano Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996.

[46] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10ˆ20 states and beyond. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90), Philadelphia, Pennsylvania, USA, June 4-7, 1990*, pages 428–439, 1990.

[47] Thierry Cachat. Uniform solution of parity games on prefix-recognizable graphs. *Electr. Notes Theor. Comput. Sci.*, 68(6):71–84, 2002.

[48] Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. On the expressiveness of parikh automata and related models. In *NCMA*, volume 282 of *books@ocg.at*, pages 103–119. Austrian Computer Society, 2011.

[49] Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Bounded parikh automata. *Int. J. Found. Comput. Sci.*, 23(8):1691–1710, 2012.

[50] Arnaud Carayol and Stefan Wöhrle. The caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science, 23rd Conference, Mumbai, India, December 15-17, 2003, Proceedings*, pages 112–123, 2003.

[51] Didier Caucal. On the regular structure of prefix rewriting. *Theor. Comput. Sci.*, 106(1):61–86, 1992.

[52] Pierre Chambart and Ph. Schnoebelen. Post embedding problem is not primitive recursive, with applications to channel systems. In *FSTTCS*, volume 4855 of *Lecture Notes in Computer Science*, pages 265–276. Springer, 2007.

[53] Edmund M. Clarke. The birth of model checking. In *25 Years of Model Checking - History, Achievements, Perspectives*, pages 1–26, 2008.

[54] Bruno Courcelle. On constructing obstruction sets of words. *Bulletin of the EATCS*, 44:178–186, 1991.

[55] Bruno Courcelle. On constructing obstruction sets of words. *Bulletin of the EATCS*, 44:178–186, 1991.

[56] Aiswarya Cyriac, Paul Gastin, and K. Narayan Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings*, pages 547–561, 2012.

[57] E. Allen Emerson. The beginning of model checking: A personal perspective. In *25 Years of Model Checking - History, Achievements, Perspectives*, pages 27–45, 2008.

[58] E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs (extended abstract). In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 328–337, 1988.

[59] E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 368–377, 1991.

[60] Javier Esparza and Pierre Ganty. Complexity of pattern-based verification for multithreaded programs. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 499–510, 2011.

[61] Javier Esparza and Pierre Ganty. Complexity of pattern-based verification for multithreaded programs. In *POPL'11*, pages 499–510, 2011.

[62] Javier Esparza, Pierre Ganty, Stefan Kiefer, and Michael Luttenberger. Parikh's theorem: A simple and direct automaton construction. *Inf. Process. Lett.*, 111(12):614–619, 2011.

[63] Javier Esparza, Pierre Ganty, Stefan Kiefer, and Michael Luttenberger. Parikh's theorem: A simple and direct automaton construction. *Inf. Process. Lett.*, 111(12):614–619, 2011.

[64] Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, pages 124–140, 2013.

[65] Javier Esparza, Pierre Ganty, and Tomás Poch. Pattern-based verification for multithreaded programs. *ACM Trans. Program. Lang. Syst.*, 36(3):9:1–9:29, 2014.

[66] Javier Esparza and Stefan Schwoon. A bdd-based model checker for recursive programs. In *Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings*, pages 324–336, 2001.

[67] M. Faouzi Atig, D. Chistikov, P. Hofman, K Narayan Kumar, P. Saivasan, and G. Zetzsche. Complexity of regular abstractions of one-counter languages. *ArXiv e-prints*, February 2016.

[68] Alain Finkel. A generalization of the procedure of karp and miller to well structured transition systems. In *ICALP*, volume 267 of *LNCS*, pages 499–508. Springer, 1987.

[69] Alain Finkel and Jérôme Leroux. How to compose presburger-accelerations: Applications to broadcast protocols. In *FSTTCS*, volume 2556 of *LNCS*, pages 145–156. Springer, 2002.

[70] Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.

[71] Alain Finkel, Bernard Willems, and Pierre Wolper. A direct symbolic approach to model checking pushdown systems. *Electr. Notes Theor. Comput. Sci.*, 9:27–37, 1997.

[72] Pierre Ganty and Rupak Majumdar. Algorithmic verification of asynchronous programs. *ACM Trans. Program. Lang. Syst.*, 34(1):6:1–6:48, May 2012.

[73] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[74] Jim Gray. Why do computers stop and what can be done about it?, 1985.

[75] Hermann Gruber, Markus Holzer, and Martin Kutrib. More on the size of higman-haines sets: Effective constructions. In *MCU'07*, volume 4664 of *LNCS*, pages 193–204, 2007.

[76] Peter Habermehl, Roland Meyer, and Harro Wimmel. The downward-closure of Petri net languages. In *ICALP'10*, volume 6199 of *LNCS*, pages 466–477, 2010.

[77] Matthew Hague. Parameterised pushdown systems with non-atomic writes. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011, December 12-14, 2011, Mumbai, India*, pages 457–468, 2011.

[78] Matthew Hague, Jonathan Kochems, and C.-H. Luke Ong. Unboundedness and downward closures of higher-order pushdown automata. In *POPL'16*, pages 151–163, 2016.

[79] Matthew Hague and Anthony Widjaja Lin. Synchronisation- and reversal-bounded analysis of multithreaded programs with counters. In *CAV'12*, volume 7358 of *LNCS*, pages 260–276, 2012.

[80] A. Heußner, J. Leroux, A. Muscholl, and G. Sutre. Reachability analysis of communicating pushdown systems. In *FOSSACS*, volume 6014 of *LNCS*, pages 267–281. Springer, 2010.

[81] G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc. (3)*, 2(7), 1952.

[82] Gerard J. Holzmann. The model checker SPIN. *IEEE Trans. Software Eng.*, 23(5):279–295, 1997.

[83] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation, Second Edition*. Addison-Wesley, 2000.

[84] Oscar H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25(1):116–133, 1978.

[85] Oscar H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25(1):116–133, 1978.

[86] Marcin Jurdzinski. Deciding the winner in parity games is in UP \cap co-up. *Inf. Process. Lett.*, 68(3):119–124, 1998.

[87] Marcin Jurdzinski, Mike Paterson, and Uri Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM J. Comput.*, 38(4):1519–1532, 2008.

[88] Richard M. Karp and Raymond E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969.

[89] Wayne Kelly, William Pugh, Evan Rosser, and Tatiana Shpeisman. Transitive closure of infinite graphs and its applications. *Journal of Parallel Programming*, 24(6):579–598, 1996.

[90] Yonit Kesten, Oded Maler, Monica Marcus, Amir Pnueli, and Elad Shahar. Symbolic model checking with rich assertional languages. *Theor. Comput. Sci.*, 256(1-2):93–112, 2001.

[91] Felix Klaedtke and Harald Rueß. Monadic second-order logics with cardinalities. In *ICALP*, volume 2719 of *LNCS*, pages 681–696. Springer, 2003.

[92] Eryk Kopczynski and Anthony Widjaja To. Parikh images of grammars: Complexity and applications. In *LICS'10*, pages 80–89, 2010.

[93] Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. An automata-theoretic approach to infinite-state systems. In *Time for Verification, Essays in Memory of Amir Pnueli*, pages 202–259, 2010.

[94] Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Context-bounded analysis of concurrent queue systems. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pages 299–314, 2008.

[95] Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Reducing context-bounded concurrent reachability to sequential reachability. In *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, pages 477–492, 2009.

[96] Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Model-checking parameterized concurrent programs using linear interfaces. In *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, pages 629–644, 2010.

[97] Salvatore La Torre, Parthasarathy Madhusudan, and Gennaro Parlato. A robust class of context-sensitive languages. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings*, pages 161–170, 2007.

[98] Salvatore La Torre, Parthasarathy Madhusudan, and Gennaro Parlato. A robust class of context-sensitive languages. In *LICS*, pages 161–170. IEEE Computer Society, 2007.

[99] Salvatore La Torre, Anca Muscholl, and Igor Walukiewicz. Safety of parametrized asynchronous shared-memory systems is almost always decidable. In *CONCUR'15*, volume 42 of *LIPIcs*, pages 72–84, 2015.

[100] Salvatore La Torre and Margherita Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In *CONCUR 2011 - Concurrency Theory - 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings*, pages 203–218, 2011.

[101] Salvatore La Torre and Margherita Napoli. A temporal logic for multi-threaded programs. In *Theoretical Computer Science - 7th IFIP TC 1/WG 2.2 International Conference, TCS 2012, Amsterdam, The Netherlands, September 26-28, 2012. Proceedings*, pages 225–239, 2012.

[102] Salvatore La Torre, Margherita Napoli, and Gennaro Parlato. Scope-bounded pushdown languages. In *Developments in Language Theory - 18th International Conference, DLT 2014, Ekaterinburg, Russia, August 26-29, 2014. Proceedings*, pages 116–128, 2014.

[103] Salvatore La Torre and Gennaro Parlato. Scope-bounded multistack pushdown systems: Fixed-point, sequentialization, and tree-width. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, pages 173–184, 2012.

[104] Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. Intruder deduction for *AC*-like equational theories with homomorphisms. In *RTA'05*, pages 308–322, 2005.

[105] Akash Lal and Thomas W. Reps. Reducing concurrent analysis under a context bound to sequential analysis. In *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7-14, 2008, Proceedings*, pages 37–51, 2008.

[106] Akash Lal and Thomas W. Reps. Reducing concurrent analysis under a context bound to sequential analysis. *FMSD*, 35(1):73–97, 2009.

[107] Akash Lal, Tayssir Touili, Nicholas Kidd, and Thomas W. Reps. Interprocedural analysis of concurrent programs under a context bound. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *LNCS*, pages 282–298. Springer, 2008.

[108] Leslie Lamport. ``sometime" is sometimes ``not never" - on the temporal logic of programs. In *Conference Record of the Seventh Annual ACM Symposium on Principles of Programming Languages, Las Vegas, Nevada, USA, January 1980*, pages 174–185, 1980.

[109] Jérôme Leroux. Acceleration for petri nets. In *ATVA*, volume 8172 of *LNCS*, pages 1–4. Springer, 2013.

[110] Jérôme Leroux and Grégoire Sutre. Flat counter automata almost everywhere! In *ATVA*, volume 3707 of *LNCS*, pages 489–503. Springer, 2005.

[111] Zhenyue Long, Georgel Calin, Rupak Majumdar, and Roland Meyer. Language-theoretic abstraction refinement. In *FASE'12*, volume 7212 of *LNCS*, pages 362–376, 2012.

[112] P. Madhusudan and G. Parlato. The tree width of auxiliary storage. In *POPL*, pages 283–294. ACM, 2011.

[113] Roman Manevich, Shmuel Sagiv, Ganesan Ramalingam, and John Field. Partially disjunctive heap abstraction. In *Static Analysis, 11th International Symposium, SAS 2004, Verona, Italy, August 26-28, 2004, Proceedings*, pages 265–279, 2004.

[114] Zohar Manna and Amir Pnueli. Verification of parameterized programs. In *in Specification and Validation Methods*, pages 167–230. University Press, 1995.

[115] K. L. McMillan. *Symbolic Model Checking: an approach to the state explosion problem*. PhD thesis, Carnegie Mellon University, 1992.

[116] Andrzej Wlodzimierz Mostowski. Regular expressions for infinite trees and a standard form of automata. In *Computation Theory - Fifth Symposium, Zaborów, Poland, December 3-8, 1984, Proceedings*, pages 157–168, 1984.

[117] Madanlal Musuvathi, David Y. W. Park, Andy Chou, Dawson R. Engler, and David L. Dill. CMC: A pragmatic approach to model checking real code. In *5th Symposium on Operating System Design and Implementation (OSDI 2002), Boston, Massachusetts, USA, December 9-11, 2002*, 2002.

[118] Madanlal Musuvathi and Shaz Qadeer. Iterative context bounding for systematic testing of multithreaded programs. In *Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation, San Diego, California, USA, June 10-13, 2007*, pages 446–455, 2007.

[119] Madanlal Musuvathi and Shaz Qadeer. Iterative context bounding for systematic testing of multithreaded programs. In *PLDI*, pages 446–455. ACM, 2007.

[120] Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. An approach to the description and analysis of hybrid systems. In *Hybrid Systems*, pages 149–178, 1992.

[121] Rohit J. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, October 1966.

[122] A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE, 1977.

[123] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57, 1977.

[124] S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.

[125] Shaz Qadeer and Jakob Rehof. Context-bounded model checking of concurrent software. In *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, pages 93–107, 2005.

[126] G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM Trans. Program. Lang. Syst.*, 22(2):416–430, 2000.

[127] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*, pages 55–74, 2002.

[128] Shmuel Sagiv, Thomas W. Reps, and Reinhard Wilhelm. Parametric shape analysis via 3-valued logic. In *POPL '99, Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, TX, USA, January 20-22, 1999*, pages 105–118, 1999.

[129] Sylvain Schmitz and Philippe Schnoebelen. The power of well-structured systems. In *CONCUR 2013 - Concurrency Theory - 24th International Conference, CONCUR 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings*, pages 5–24, 2013.

[130] Koushik Sen and Mahesh Viswanathan. Model checking multithreaded programs with asynchronous atomic methods. In *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, pages 300–314, 2006.

[131] Koushik Sen and Mahesh Viswanathan. Model checking multithreaded programs with asynchronous atomic methods. In *CAV'14*, volume 4144 of *LNCS*, pages 300–314, 2006.

[132] Olivier Serre. Note on winning positions on pushdown games with [omega]-regular conditions. *Inf. Process. Lett.*, 85(6):285–291, 2003.

[133] Anil Seth. Games on multi-stack pushdown systems. In *Logical Foundations of Computer Science, International Symposium, LFCS 2009, Deerfield Beach, FL, USA, January 3-6, 2009. Proceedings*, pages 395–408, 2009.

[134] Anil Seth. Global reachability in bounded phase multi-stack pushdown systems. In *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, pages 615–628, 2010.

[135] Colin Stirling. Lokal model checking games. In *CONCUR '95: Concurrency Theory, 6th International Conference, Philadelphia, PA, USA, August 21-24, 1995, Proceedings*, pages 1–11, 1995.

[136] Larry J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory and Logic*. PhD thesis, MIT, Cambridge, Massasuchets, USA, 1974.

[137] S. La Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS*, pages 161–170. IEEE Computer Society, 2007.

[138] S. La Torre, P. Madhusudan, and G. Parlato. Context-bounded analysis of concurrent queue systems. In *TACAS*, volume 4963 of *LNCS*, pages 299–314. Springer, 2008.

[139] Jan van Leeuwen. Effective constructions in well-partially-ordered free monoids. *Discrete Mathematics*, 21(3):237–252, 1978.

[140] Moshe Y. Vardi. Alternating automata and program verification. In *Computer Science Today*, volume 1000 of *LNCS*, pages 471–485. Springer, 1995.

[141] Moshe Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344. IEEE Computer Society, 1986.

[142] Kumar Neeraj Verma, Helmut Seidl, and Thomas Schwentick. On the complexity of equational horn clauses. In *CADE-20*, volume 3632 of *LNCS*, pages 337–352, 2005.

[143] Igor Walukiewicz. Pushdown processes: Games and model checking. In *Computer Aided Verification, 8th International Conference, CAV '96, New Brunswick, NJ, USA, July 31 - August 3, 1996, Proceedings*, pages 62–74, 1996.

[144] Pierre Wolper and Bernard Boigelot. Verifying systems with infinite but regular state spaces. In *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, pages 88–97, 1998.

[145] Pierre Wolper and Patrice Godefroid. Partial-order methods for temporal verification. In *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, pages 233–246, 1993.

[146] Karianto Wong. Parikh automata with pushdown stack. Diploma thesis, RWTH Aachen, 2004.

[147] Georg Zetzsche. An approach to computing downward closures. In *ICALP'15*, volume 9135 of *LNCS*, pages 440–451, 2015.

[148] Georg Zetzsche. Computing downward closures for stacked counter automata. In *STACS'15*, volume 30 of *LIPIcs*, pages 743–756, 2015.