Parameterised Pushdown Systems with Non-Atomic Writes

M. Hague

Oxford University, Department of Computer Science
and
Laboratoire d'Informatique Gaspard-Monge, Université Paris-Est

Abstract. We consider the master/slave parameterised reachability problem for networks of pushdown systems, where communication is via a global store using only non-atomic reads and writes. We show that the control-state reachability problem is decidable. As part of the result, we provide a constructive extension of a theorem by Ehrenfeucht and Rozenberg to produce an NFA equivalent to certain kinds of CFG. Finally, we show that the non-parameterised version is undecidable.

Note, this is the long version of work appearing in FSTTCS 2011.

1 Introduction

A parameterised reachability problem is one where the system is defined in terms of a given input, usually a number n. We then ask whether there is some n such that the resulting system can reach a given state. An early result shows that this problem is undecidable, even when the system defined for each n is a finite state machine: one simply has to define the nth system to simulate a Turing machine up to n steps [2]. Thus, the Turing machine terminates iff there is some n such that the nth system reaches a halting state.

Such a result, however, is somewhat pathological. More natural parameterised problems concentrate on the replication of components. For instance, we may have a leadership election algorithm amongst several nodes. For this algorithm we would want to know, for example, whether there is some n such that, when n nodes are present, the routine fails to elect a leader. This problem walks the line between decidability and undecidability, even with finite-state components: in a ring network, when nodes can communicate to their left and right neighbours directly, Suzuki proves undecidability [32]; but, in less disciplined topologies, the problem becomes decidable [16].

In particular, the above decidability result considers the following problem: given a master process $\mathcal{U}$ and slave $\mathcal{C}$, can the master in parallel with n slaves reach a given state. Communication in this system is by anonymous pairwise synchronisation (that is, a receive request can be satisfied by *any* thread providing the matching send, rather than a uniquely identified neighbour). This problem reduces to Petri-nets, which can, for each state of $\mathcal{C}$, keep a count of the number of threads in that state. When communication is via a finite-state global store, which all threads can read from and write to (atomically), it is easy to see that decidability can be obtained by the same techniques.

These results concern finite-state machines. This is ideal for hardware or simple protocols. When the components are more sophisticated (such as threads created by a web-server), a more natural and expressive (infinite-state) program model — allowing one to accurately simulate the control flow of first-order recursive programs [20] — is given by pushdown systems (PDSs). Such systems have proved popular in the sequential setting (e.g. [8, 14, 29, 27]), with several successful implementations [6, 7, 29]. Unfortunately, when two PDSs can communicate, reachability quickly becomes undecidable [26].

In recent years, many researchers have tackled this problem, proposing many different approximations, and restrictions on topology and communication behaviour (e.g. [23, 9–11, 30, 28, 18]). A pleasantly surprising (and simple) result in this direction was provided by Kahlon [21]: the parameterised reachability problem for systems composed of n slaves $\mathcal{C}$ communicating by anonymous synchronisation is decidable. This result relies heavily on the inability of the system to restrict the number of active processes, or who they communicate with. Indeed, in the presence of a master process $\mathcal{U}$, or communication via a global store, undecidability is easily obtained.

In this work we study the problem of adding the master process and global store. To regain decidability, we only allow *non-atomic* accesses to the shared memory. We then show — by extending a little-cited theorem of Ehrenfeucht and Rozenberg [13] — that we can replace the occurrences of $\mathcal{C}$ with regular automata[1]. This requires the introduction of different techniques than those classically used. Finally, a product construction gives us our result. In addition, we show that, when n is fixed, the problem remains undecidable, for all n. For clarity, we present the single-variable case here. In the appendix we show that the techniques extend easily to the case of k shared variables.

After discussing further related work, we begin in Section 2 with the preliminaries. In Section 3 we define the systems that we study. Our main result is given is Section 4 and the accompanying undecidability proof appears in Section 5. In Section 6 we show how to obtain a constructive version of Ehrenfeucht and Rozenberg's theorem. Finally, we conclude in Section 7.

Related Work Many techniques attack parameterisation (e.g. network invariants and symmetry). Due to limited space, we only discuss PDSs here. In addition to results on parameterised PDSs, Kahlon shows decidability of concurrent PDSs communicating via nested-locks [22]. In contrast, we cannot use locks to guarantee atomicity here.

A closely related model was studied by Bouajjani *et al.* in 2005. As we do, they allow PDSs to communicate via a global store. They do not consider parameterised problems directly, but they do allow the dynamic creation of threads. By dynamically creating an arbitrary number of threads at the start of the execution, the parameterised problem can be simulated. Similarly, parameterisation can simulate thread creation by activating hitherto dormant threads. However, since Bouajjani *et al.* allow atomic read/write actions to occur, the problem they consider is undecidable; hence, they consider *context-bounded reachability.*

Context-bounded reachability is a popular technique based on the observation that many bugs can be identified within a small number of context switches [25]. This idea has been extended to *phase-bounded* systems where only one stack may be decreasing in any one phase [3, 31]. Finally, in another extension of context-bounded model-checking, Ganty *et al.* consider *bounded under-approximations* where runs are restricted by intersecting with a word of the form $a_1^* \dots a_n^*$ [15]. In contrast to this work, these techniques are only accurate up to a given bound. That is, they are sound, but not complete. Recently, La Torre *et al.* gave a sound algorithm for parameterised PDSs together with a technique that may detect completeness in the absence of recursion [34].

Several models have been defined for which model-checking can be sound and complete. For example, Bouajjani *et al.* also consider acyclic topologies [5, 11]. As well as restricting the network structure, Sen and Viswanathan [30], La Torre *et al.* [33] and later Heußner *et al.* [18], show how to obtain decidability by only allowing communications to occur when the stack satisfies certain conditions.

One of the key properties that allow parameterized problems to become decidable is that once a copy of the duplicated process has reached a given state, then any number of additional copies may also be in that state. In effect, this means that any previously seen state may be returned to at any time. This property has also been used by Delzanno *et al.* to analyse recursive ping-pong protocols [12] using *Monotonic Set-extended Prefix Rewriting.* However, unlike our setting, these systems do not have a master process.

Finally, recent work by Abdulla *et al.* considers parameterised problems with non-atomic global conditions [1]. That is, global transitions may occur when the process satisfy a global condition that is not evaluated atomically. However, the processes they consider are finite-state in general. Although a procedure is proposed when unbounded integers are allowed, this is not guaranteed to terminate.

[1] A reviewer points out that the upward-closure of a context free language has been proved regular by Atig *et al.* [5] with the same complexity, which is sufficient for our purposes. However, a constructive version of Ehrenfeucht and Rozenberg is a stronger result, and hence remains a contribution.

2 Preliminaries

We recall the definitions of finite automata and pushdown systems and their language counterparts. We also state a required result by Ehrenfeucht and Rozenberg.

Definition 1 (Non-Deterministic Finite Word Automata). *We define a* non-deterministic finite word automaton *(NFA) $\mathcal{A}$ as a tuple $(\mathcal{Q}, \Gamma, \Delta, q_0, \mathcal{F})$ where $\mathcal{Q}$ is a finite set of states, Γ is a finite alphabet, $q_0 \in \mathcal{Q}$ is an initial state, $\mathcal{F} \subseteq \mathcal{Q}$ is a set of final states, and $\Delta \subseteq \mathcal{Q} \times \Gamma \times \mathcal{Q}$ is a finite set of transitions.*

We will denote a transition (q, γ, q') using the notation $q \xrightarrow{\gamma} q'$. We call a sequence $q_1 \xrightarrow{\gamma_1} q_2 \xrightarrow{\gamma_2} \cdots \xrightarrow{\gamma_{z-1}} q_z$ a *run* of $\mathcal{A}$. It is an accepting run if $q_1 = q_0$ and $q_z \in \mathcal{F}$. The language $\mathcal{L}(\mathcal{A})$ of an NFA is the set of all words labelling an accepting run. Such a language is *regular*.

Definition 2 (Pushdown Systems). *A* pushdown system *(PDS) $\mathcal{P}$ is a tuple $(\mathcal{Q}, \Sigma, \Gamma, \Delta, q_0, \mathcal{F})$ where $\mathcal{Q}$ is a finite set of control states, Σ is a finite stack alphabet with a special bottom-of-stack symbol $\perp$, Γ is a finite output alphabet, $q_0 \in \mathcal{Q}$ is an initial state, $\mathcal{F} \subseteq \mathcal{Q}$ is a set of final states, and $\Delta \subseteq (\mathcal{Q} \times \Sigma) \times \Gamma \times (\mathcal{Q} \times \Sigma^*)$ is a finite set of transition rules.*

We will denote a transition rule $((q, a), \gamma, (q', w'))$ using the notation $(q, a) \xhookrightarrow{\gamma} (q', w')$. The bottom-of-stack symbol is neither pushed nor popped. That is, for each rule $(q, a) \xhookrightarrow{\gamma} (q', w') \in \Delta$ we have, when $a \neq \perp$, w does not contain $\perp$, and, $a = \perp$ iff $w' = w \perp$ and w does not contain $\perp$. A configuration of $\mathcal{P}$ is a tuple (q, w), where $q \in \mathcal{Q}$ is the current control state and $w \in \Sigma^*$ is the current stack contents. There exists a transition $(q, aw) \xrightarrow{\gamma} (q', w'w)$ of $\mathcal{P}$ whenever $(q, a) \xhookrightarrow{\gamma} (q', w') \in \Delta$. We call a sequence $c_0 \xrightarrow{\gamma_1} c_1 \xrightarrow{\gamma_2} \cdots \xrightarrow{\gamma_z} c_z$ a *run* of $\mathcal{P}$. It is an accepting run if $c_0 = (q_0, \perp)$ and $c_z = (q, w)$ with $q \in \mathcal{F}$. The language $\mathcal{L}(\mathcal{P})$ of a pushdown system is the set of all words labelling an accepting run. Such a language is *context-free*. Note, in some cases, we omit the output alphabet Γ. In this case, the only character is the empty character ε, with which all transitions are labelled. In general, we will omit the empty character ε when it labels a transition.

We use a theorem of Ehrenfeucht and Rozenberg [13]. With respect to a context-free language $\mathcal{L}$, a *strong iterative pair* is a tuple (x, y, z, u, t) of words such that for all $i \geq 0$ we have $xy^izu^it \in \mathcal{L}$, where y and u are non-empty words. A strong iterative pair is *very degenerate* if, for all $i, j \geq 0$ we have that $xy^izu^jt \in \mathcal{L}$.

Theorem 1 ([13]). *For a given context-free language $\mathcal{L}$, if all strong iterative pairs are very degenerate, then $\mathcal{L}$ is regular.*

However, Ehrenfeucht and Rozenberg do not present a constructive algorithm for obtaining a regular automaton accepting the same language as an appropriate context-free language. Hence, we provide such an algorithm in Section 6.

3 Non-Atomic Pushdown Systems

Given an alphabet $\mathcal{G}$, let $r(\mathcal{G}) = \{ r(g) \mid g \in \mathcal{G} \}$ and $w(\mathcal{G}) = \{ w(g) \mid g \in \mathcal{G} \}$. These alphabets represent read and write actions respectively of the value g.

Definition 3 (Non-atomic Pushdown Systems). *Over a finite alphabet $\mathcal{G}$, a* non-atomic pushdown system *(naPDS) is a tuple $\mathcal{P} = (\mathcal{Q}, \Sigma, \Delta, q_0, \mathcal{G})$ where $\mathcal{Q}$ is a finite set of control-states, Σ is a finite stack alphabet with a bottom-of-stack symbol $\perp$, $q_0 \in \mathcal{Q}$ is a designated initial control state and $\Delta \subseteq (\mathcal{Q} \times \Sigma) \times (r(\mathcal{G}) \cup w(\mathcal{G}) \cup \{ \varepsilon \}) \times (\mathcal{Q} \times \Sigma^*)$.*

That is, a non-atomic pushdown system is a PDS where the output alphabet is used to signal the interaction with a global store, and there are no final states: we are interested in the behaviour of the system, rather than the language it defines.

Definition 4 (Networks of naPDSs). *A network of n* non-atomic pushdown systems *(NPDS) is a tuple $\mathcal{N} = (\mathcal{P}_1, \ldots, \mathcal{P}_n, \mathcal{G}, g_0)$ where, for all $1 \leq i \leq n$, $\mathcal{P}_i = \left(\mathcal{Q}_i, \Sigma_i, \Delta_i, q_0^i, \mathcal{G}\right)$ is a NPDS over $\mathcal{G}$ and $g_0 \in \mathcal{G}$ is the initial value of the global store.*

A configuration of an NPDS is a tuple $(q_1, w_1, \ldots, q_n, w_n, g)$ where $g \in \mathcal{G}$ and for each i, $q_i \in \mathcal{Q}_i$ and $w_i \in \Sigma_i^*$. There is a transition $(q_1, w_1, \ldots, q_n, w_n, g) \to (q'_1, w'_1, \ldots, q'_n, w'_n, g')$ whenever, for some $1 \leq i \leq n$ and all $1 \leq j \leq n$ with $i \neq j$, we have $q'_j = q_j$, $w'_j = w_j$, and

- $(q_i, w_i) \to (q'_i, w'_i)$ is a transition of $\mathcal{P}_i$ and $g' = g$; or
- $(q_i, w_i) \xrightarrow{r(g)} (q'_i, w'_i)$ is a transition of $\mathcal{P}_i$ and $g' = g$; or
- $(q_i, w_i) \xrightarrow{w(g')} (q'_i, w'_i)$ is a transition of $\mathcal{P}_i$.

A path π of $\mathcal{N}$ is a sequence of configurations $c_1 c_2 \ldots c_m$ such that, for all $1 \leq i < m$, $c_i \to c_{i+1}$. A run of $\mathcal{N}$ is a path such that $c_1 = \left(q_0^1, \perp, \ldots, q_0^n, \perp, g_0\right)$.

4 The Parameterised Reachability Problem

We define and prove decidability of the parameterised reachability problem for naPDSs. We finish with a few remarks on the extension to multiple variables, and on complexity issues.

Definition 5 (Parameterised Reachability). *For given naPDSs $\mathcal{U}$ and $\mathcal{C}$ over $\mathcal{G}$, initial store value g_0 and control state q, the parameterised reachability problem asks whether there is some n such that the NPDS $\mathcal{N}_n = \left(\mathcal{U}, \underbrace{\mathcal{C}, \ldots, \mathcal{C}}_{n}, \mathcal{G}, g_0\right)$ has a run to some configuration containing the control state q.*

In this section, we aim prove the following theorem.

Theorem 2. *The parameterised reachability problem for NPDSs is decidable.*

Without loss of generality, we can assume q is a control-state of $\mathcal{U}$ (a $\mathcal{C}$ process can write its control-state to the global store for $\mathcal{U}$ to read). The idea is to build an automaton which describes for each $g \in \mathcal{G}$ the sequences $g_1 \ldots g_m \in \mathcal{G}^*$ that need to be read by some $\mathcal{C}$ process to be able to write g to the global store. We argue using Theorem 1 that such *read languages* are regular (and construct regular automata using Lemma 5). Broadly this is because, between any two characters to be read, any number of characters may appear in the store and then be overwritten before the process reads the required character. We then combine the resulting languages with $\mathcal{U}$ to produce a context-free language that is empty iff the control-state q is reachable.

4.1 Regular Read Languages

For each $g \in \mathcal{G}$ we will define a *read-language* $\mathcal{L}_{w(g)}$ which intuitively defines the language of read actions that $\mathcal{C}$ must perform before being able to write g to the global store. Since $\mathcal{C}$ may have to write other characters to the store before g, we use the symbol # as an abstraction for these writes. The idea is that, for any run of the parameterised system, we can construct another run where each copy of $\mathcal{C}$ is responsible for a single particular write to the global store, and $\mathcal{L}_{w(g)}$ describes what $\mathcal{C}$ must do to be able to write g.

To this end, given a non-atomic pushdown system $\mathcal{P}$ we define for each $g \in \mathcal{G}$ the pushdown system $\mathcal{P}_{w(g)}$ which is $\mathcal{P}$ augmented with a new unique control-state f, and a transition $(q, a) \hookrightarrow (f, a)$ whenever $\mathcal{P}$ has a rule $(q, a) \xhookrightarrow{w(g)} (q', w)$. Furthermore, replace all $(q, a) \xhookrightarrow{w(g')} (q', w)$ rules with $(q, a) \xhookrightarrow{\#} (q', w)$ where $\# \notin \mathcal{G}$. These latter rules signify that the global store contents have been changed, and that a new value must be written before reading can continue. This implicitly assumes that $\mathcal{C}$ does not try to read the last value it has written. This can be justified since, whenever this occurs, because we are dealing with the parameterised version of the problem, we can simply add another copy of $\mathcal{C}$ to produce the required write.

We interpret f as the sole accepting control state of $\mathcal{P}_{w(g)}$ and thus $\mathcal{L}(\mathcal{P}_{w(g)})$ is the language of reads (and writes) that must occur for g to be written. We then allow any number of (ignored) read and # events[2] to occur. That is, any word in the read language contains a run of $\mathcal{C}$ with any number of additional actions that do not affect the reachability property interspersed. Let $R = \{ r(g') \mid g' \in \mathcal{G} \} \cup \{ \# \}$, we define the read language $\mathcal{L}_{w(g)} \subseteq R^*$ for $w(g)$ as

$$\mathcal{L}_{w(g)} = \left\{ R^*\gamma_1 R^* \dots R^*\gamma_z R^* \;\middle|\; \gamma_1 \dots \gamma_z \in \mathcal{L}(\mathcal{P}_{w(g)}) \right\} .$$

Note, in particular, that $\gamma_1 \dots \gamma_z \in R^*$.

Lemma 1. *For all $g \in \mathcal{G}$, $\mathcal{L}_{w(g)}$ is regular and an NFA $\mathcal{A}$ accepting $\mathcal{L}_{w(g)}$, of doubly-exponential size, can be constructed in doubly-exponential time.*

Proof. Take any strong iterative pair (x, y, z, t, u) of $\mathcal{L}_{w(g)}$. To satisfy the preconditions of Theorem 1, we observe that $xzu \in \mathcal{L}_{w(g)}$ since we have a strong iterative pair. Then, from the definition of $\mathcal{L}_{w(g)}$ we know $xR^*zR^*u \subseteq \mathcal{L}_{w(g)}$ and hence, for all i, j, $xy^izt^ju \subseteq \mathcal{L}_{w(g)}$ as required. Thus $\mathcal{L}_{w(g)}$ is regular. The construction of $\mathcal{A}$ comes from Lemma 5.

4.2 Simulating the System

We build a PDS that recognises a non-empty language iff the parameterised reachability problem has a positive solution. The intuition behind the construction of $\mathcal{P}_{sys}$ is that, if a collection of $\mathcal{C}$ processes have been able to use the output of $\mathcal{U}$ to produce a write of some g to the global store, then we may reproduce that group of processes to allow as many writes g to occur as needed. Hence, in the construction below, once $q_i \in \mathcal{F}_i$ has been reached, g_i can be written at any later time. The # character is used to prevent sequences such as $r(g)w(g')r(g)$ occurring in read languages, where no process is able to provide the required write $w(g)$ that must occur after $w(g')$. Note that, if we did not use # in the read languages, such sequences could occur because the $w(g')$ would effectively be ignored.

The construction itself is a product construct between $\mathcal{U}$ and the regular automata accepting the read languages of $\mathcal{C}$. The regular automata read from the global variable, writing # when a # action should occur. Essentially, they mimic the behaviour of an arbitrary number of $\mathcal{C}$ processes in their interaction — via the global store — with $\mathcal{U}$ and each other. The value of the global store is held in the last component of the product.

Definition 6 ($\mathcal{P}_{sys}$). *Given an naPDS $\mathcal{U} = (\mathcal{Q}_\mathcal{U}, \Sigma, \Delta_\mathcal{U}, q_0^\mathcal{U}, \mathcal{G})$ with initial store value g_0, a control-state $f \in \mathcal{Q}_\mathcal{U}$, and, for each $g \in \mathcal{G}$, a regular automaton*

$$\mathcal{A}_{w(g)} = \left(\mathcal{Q}_{w(g)}, R, \Delta_{w(g)}, \mathcal{F}_{w(g)}, q_0^{w(g)}\right) ,$$

we define the PDS $\mathcal{P}_{sys} = (\mathcal{Q}, \Sigma, \Delta, q_0, \mathcal{F})$ where, if $\mathcal{G} = \{ g_0, \dots, g_m \}$, then

- $\mathcal{Q} = \mathcal{Q}_\mathcal{U} \times \mathcal{Q}_{w(g_0)} \times \cdots \times \mathcal{Q}_{w(g_m)} \times (\mathcal{G} \cup \{ \# \})$,
- $q_0 = \left(q_0^\mathcal{U}, q_0^{w(g_0)}, \dots, q_0^{w(g_m)}, g_0\right)$,
- $\mathcal{F} = \{ f \} \times \mathcal{Q}_{w(g_0)} \times \cdots \times \mathcal{Q}_{w(g_m)} \times (\mathcal{G} \cup \{ \# \})$,

and Δ is the smallest set containing all $(q, a) \hookrightarrow (q', w)$ where $q = (q_\mathcal{U}, q_0, \dots, q_m, g)$ and,

- $q' = (q'_\mathcal{U}, q_0, \dots, q_m, g)$ *and* $(q_\mathcal{U}, a) \hookrightarrow (q'_\mathcal{U}, w) \in \Delta_\mathcal{U}$, *or*
- $q' = (q'_\mathcal{U}, q_0, \dots, q_m, g)$ *and* $(q_\mathcal{U}, a) \xhookrightarrow{r(g)} (q'_\mathcal{U}, w) \in \Delta_\mathcal{U}$, *or*

[2] Extra # events will not allow spurious runs, as they only add extra behaviours that may cause the system to become stuck. This is because # is never read by a process.

– $q' = (q'_{\mathcal{U}}, q_0, \ldots, q_m, g')$ *and* $(q_{\mathcal{U}}, a) \xrightarrow{w(g')} (q'_{\mathcal{U}}, w) \in \Delta_{\mathcal{U}}$, *or*
– $q' = (q_{\mathcal{U}}, q_0, \ldots, q'_i, \ldots, q_m, g)$ *and* $q_i \xrightarrow{r(g)} q'_i \in \Delta_i$, $q_i \notin \mathcal{F}_i$ *and* $w = a$, *or*
– $q' = (q_{\mathcal{U}}, q_0, \ldots, q'_i, \ldots, q_m, \#)$ *and* $q_i \xrightarrow{\#} q'_i \in \Delta_i$, $q_i \notin \mathcal{F}_i$ *and* $w = a$, *or*
– $q' = (q_{\mathcal{U}}, q_0, \ldots, q_m, g_i)$, $q_i \in \mathcal{F}_i$ *and* $w = a$.

The last transition in the above definition — which corresponds to some copy of $\mathcal{C}$ writing g_i to the global store — can be applied any number of times; each application corresponds to a different copy of $\mathcal{C}$, and, since we are considering the parameterised problem, we can choose as many copies of $\mathcal{C}$ as are required.

Lemma 2. *The PDS $\mathcal{P}_{sys}$ has a run to some control-state in $\mathcal{F}$ iff the parameterised reachability problem for $\mathcal{U}$, $\mathcal{C}$, $\mathcal{G}$, g_0 and q has a positive solution.*

The full proof of correctness is given in the appendix. To construct a run reaching q from an accepting run of $\mathcal{P}_{sys}$ we first observe that $\mathcal{U}$ is modelled directly. We then add a copy of $\mathcal{C}$ for every individual write to the global component of $\mathcal{P}_{sys}$. These slaves are able to read from/write to the global component finally enabling them to perform their designated write. This is because (a part of) the changes to the global store is in the read language of the required write.

Concerning the counter-directional, we architecturalise an accepting run of $\mathcal{P}_{sys}$ from a run of the parameterised system reaching q. To this end, we observe again that we can simulate $\mathcal{U}$ directly. To simulate the slaves, we take, for every character $g \in \mathcal{G}$ written to the store, the copy of $\mathcal{C}$ responsible for its first write. From this we get runs of the $\mathcal{A}_{w(g)}$ that can be interleaved with the simulation of $\mathcal{U}$ and each other to create the required accepting run, where additional writes of each g are possible by virtue of $\mathcal{A}_{w(g)}$ having reached an accepting state (hence we require no further simulation for these writes).

Example Let $\mathcal{U}$ perform the actions $r(1)r(2)w(ok)r(f)$ and $\mathcal{C}$ run either $w(1)r(ok)w(go)$ or $w(2)r(go)w(f)$. Let $\mathcal{L}_1, \ldots, \mathcal{L}_4$ denote the following read languages.

$$\mathcal{L}_{w(1)} = \mathcal{L}_{w(2)} = R^* \quad \mathcal{L}_{w(go)} = R^* \# R^* r(ok) R^* \quad \mathcal{L}_{w(f)} = R^* \# R^* r(go) R^*$$

Take two slaves $\mathcal{C}_1$ and $\mathcal{C}_2$ and the run (the subscript denotes the active process):

$$w(1)_{\mathcal{C}_1} r(1)_{\mathcal{U}} w(2)_{\mathcal{C}_2} r(2)_{\mathcal{U}} w(ok)_{\mathcal{U}} r(ok)_{\mathcal{C}_1} w(go)_{\mathcal{C}_1} r(go)_{\mathcal{C}_2} w(f)_{\mathcal{C}_2} r(f)_{\mathcal{U}} \ .$$

This can be simulated by the following actions on the global component of $\mathcal{P}_{sys}$:

$$w(\#)_{\mathcal{L}_3} w(1)_{\mathcal{L}_1} r(1)_{\mathcal{U}} w(\#)_{\mathcal{L}_4} w(2)_{\mathcal{L}_2} r(2)_{\mathcal{U}} w(ok)_{\mathcal{U}} r(ok)_{\mathcal{L}_3} w(go)_{\mathcal{L}_3} r(go)_{\mathcal{L}_4} w(f)_{\mathcal{L}_4} r(f)_{\mathcal{U}} \ .$$

Note, we have scheduled the $w(\#)$ actions immediately before the write they correspond to.

4.3 Complexity and Multiple Stores

We obtain for each $g \in \mathcal{G}$ an automaton $\mathcal{A}_{w(g)}$ of size $\mathcal{O}\left(2^{2^{f(n)}}\right)$ in $\mathcal{O}\left(2^{2^{f(n)}}\right)$ time for some polynomial f (using Lemma 5) where n is the size of the problem description. The pushdown system $\mathcal{P}_{sys}$, then, has $\mathcal{O}\left(2^{2^{f'(n)}}\right)$ many control states for a polynomial f'. It is well known that reachability/emptiness for PDSs is polynomial in the size of the system (e.g. Bouajjani *et al.* [8]), and hence the entire algorithm takes doubly-exponential time. For the lower bound, one can reduce from SAT to obtain an NP-hardness result (as shown in the appendix). Further work is needed to pinpoint the complexity precisely.

The algorithm presented above only applies to a single shared variable. A more natural model has multiple shared variables. We may allow k variables with the addition of k global components $\mathcal{G}_1, \ldots, \mathcal{G}_k$. The main change required is the use of symbols $\#_1, \ldots, \#_k$ rather than simply $\#$ and to build $\mathcal{P}_{sys}$ to be sensitive to which store is being written to (or erased with some $\#_i$). This does not increase the complexity since $n = |\mathcal{G}_1| + \cdots + |\mathcal{G}_k|$ in the above analysis and the cost of the k-product of variables does not exceed the cost of the product of the $\mathcal{A}_{w(g)}$. We give the full details in the appendix. Note that, using the global stores, we can easily encode a PSPACE Turing machine using $\mathcal{U}$, without stack, and an empty $\mathcal{C}$. Hence the problem for multiple variables is at least PSPACE-hard.

5 Non-parameterized Reachability

We consider the reachability problem when the number of processes n is fixed. In the case when $1 \leq n \leq 2$, undecidability is clear: even with non-atomic read/writes, the two processes can organise themselves to overcome non-atomicity. When $n > 2$, it becomes harder to co-ordinate the copies of $\mathcal{C}$. A simple trick recovers undecidability. More formally, then:

Definition 7 (Non-parameterized Reachability). *For given n and naPDSs $\mathcal{U}$ and $\mathcal{C}$ over $\mathcal{G}$, initial store value g_0 and control state q, the non-parameterised reachability problem asks whether the NPDS $\mathcal{N}_n = \left(\mathcal{U}, \underbrace{\mathcal{C}, \ldots, \mathcal{C}}_{n}, \mathcal{G}, g_0\right)$ has a run to some configuration containing the control state q.*

Theorem 3. *The non-parameterized reachability problem is undecidable when $n \geq 1$. When $n > 1$, the result holds even when $\mathcal{U}$ is null.*

Proof. We reduce from the undecidability of the emptiness of the intersection of two context-free languages. First fix some $n \geq 2$ and two pushdown systems $\mathcal{P}_1$, $\mathcal{P}_2$ accepting the two languages $\mathcal{L}_1$ and $\mathcal{L}_2$.

We define $\mathcal{C}$ to be the disjunction of $\mathcal{C}_1, \ldots, \mathcal{C}_n$. That is, $\mathcal{C}$ makes a non-deterministic choice of which $\mathcal{C}_i$ to run ($1 \leq i \leq n$). Let $1, \ldots, n, f, !$ be characters not in the alphabet of $\mathcal{L}_1$ and $\mathcal{L}_2$. The process $\mathcal{C}_1$ will execute, for each $\gamma_1 \ldots \gamma_z \in \mathcal{L}_1$, a sequence

$$w(1)r(n)w(\gamma_1)r(!)w(\gamma_2)r(!) \ldots w(\gamma_z)r(!)w(f) \ .$$

It is straightforward to build $\mathcal{C}_1$ from $\mathcal{P}_1$. Similarly, the process $\mathcal{C}_2$ will execute, for each $a_1 \ldots a_m \in \mathcal{L}_2$, a sequence

$$r(1)w(2)r(\gamma_1)w(!)r(\gamma_2)w(!) \ldots r(\gamma_z)w(!)r(f)$$

and move to a fresh control-state q_f. It is straightforward to build $\mathcal{C}_2$ from $\mathcal{P}_2$. The remaining processes for $3 \leq i \leq n$ simply perform the sequence $r(i-1)w(i)$.

The control-state q_f can be reached iff the intersection of $\mathcal{L}_1$ and $\mathcal{L}_2$ is non-empty. To see this, first consider a word witnessing the non-emptiness of the intersection. There is immediately a run of $\mathcal{N}_n$ reaching q_f where each ith $\mathcal{C}$ process behaves as $\mathcal{C}_i$.

In the other direction, take a run of $\mathcal{N}_n$ reaching q_f. First, observe that for each $1 \leq i \leq n$ there must be some copy of $\mathcal{C}$ running $\mathcal{C}_i$. This is because, otherwise, there is some i not written to the global store, and hence all $i' \geq i$, including n, are not written. Then $\mathcal{C}_1$ can never write f and $\mathcal{C}_2$ can never move to q_f. Finally, take the sequence $a_1 \ldots a_m$ written by $\mathcal{C}_1$ (and read by $\mathcal{C}_2$). This word witnesses non-emptiness as required.

In the case when $n = 1$, we simply have $\mathcal{U}$ run $\mathcal{C}_1$ and $\mathcal{C}$ run $\mathcal{C}_2$.

6 Making Ehrenfeucht and Rozenberg Constructive

We show how to make Theorem 1 constructive. To prove regularity, Ehrenfeucht and Rozenberg assign to each word a set of types $\theta(w)$, and prove that, if $\theta(w) = \theta(w')$, then $w \sim w'$ in the sense of Myhill and Nerode [19]. We first show how to decide $\theta(w) = \theta(w')$, and then show how to build the automaton. For the sake of brevity, we will assume familiarity with context-free grammars (CFGs) and their related concepts [19].

For our purposes, we consider a context-free grammar (in Chomsky normal form) G to be a collection of rules of the form $A \to BC$ or $A \to a$, where A, B and C are *non-terminals* and a is a *terminal* in Γ. There is also a designated *start* non-terminal S. A word w is in $\mathcal{L}(G)$ if there is a *derivation-tree* with root labelled by S such that an internal node labelled by A has left- and right-children labelled by B and C when we have $A \to BC$ in the grammar and a leaf node is labelled by a when it has parent labelled by A (with one child) and $A \to a$ is in the grammar. Furthermore w is the *yield* of the tree; that is, w labels the leaves. Note, all nodes must be labelled according to the scheme just described. One can also consider the derivation of w in terms of *rewrites* from S, where the parent-child relationship in the tree gives the requires rewriting steps.

6.1 Preliminaries

We first recall some relevant definitions from Ehrenfeucht and Rozenberg. We write $\#_a(w)$ to mean the number of occurrences of the character a in the word w.

Definition 8 (Type of a Word). *Let Γ be an alphabet and let $x, w \in \Gamma^*$, We say that w is of type x, or that x is a type of w (denoted $\tau(x, w)$) if*

1. *for every $a \in \Gamma$, $\#_a(x) \leq 1$, and*
2. *there exists a homomorphism h such that*
   (a) *for every $a \in \Gamma$, $h(a) \in a \cup a\Gamma^*a$, and*
   (b) *$h(x) = w$.*

If x satisfies the above, we also say that x is a type in Γ^*.

Given a CFG G in Chomsky normal form, we assume a derivation tree T of G is a labelled tree where all internal nodes are labelled with the non-terminal represented by the node, and all leaf nodes are labelled by their corresponding characters in Γ. Given a derivation tree T, Ehrenfeucht and Rozenberg define a marked tree $\overline{T}$ with an expanded set of non-terminals and terminals. Simultaneously, we will define the spine of a marked tree. Intuitively, we take a path in the tree and mark it with the productions of G that have been used and the directions taken.

Given an alphabet of terminals and non-terminals Σ and a derivation tree T, define the alphabet $\overline{\Sigma} = \{ (A, B, C, k) \mid k \in \{ 1, 2 \} \wedge A \to BC \in G \} \cup \{ (A, a) \mid A \to a \in G \}$. This is the marking alphabet of G.

Definition 9 (Spine of a Derivation Tree). *Let T be a derivation tree in G and let $\rho = v_0 \dots v_s$ be a path in T where $s \geq 1$, v_0 is the root of T, v_s is a leaf of T and $\ell(v_0), \dots, \ell(v_s)$ are the labels corresponding to nodes of ρ. Now for each node v_j, $0 \leq j \leq s$, change its label to $\overline{\ell}(v_j)$ as follows:*

1. *if $A \to BC$ is the production used to rewrite the node j (hence $\ell(v_j) = A$) and v_j has a direct descendant to the left of ρ, then $\ell(v_j)$ is changed to $\overline{\ell}(v_j) = (A, B, C, 1)$,*
2. *if $A \to BC$ is the production used to rewrite the node j and v_j has a direct descendant to the right of ρ, then $\ell(v_j)$ is changed to $\overline{\ell}(v_j) = (A, B, C, 2)$,*
3. *if $A \to a$ is the production used to rewrite the node j then $\ell(v_j)$ is changed to $\overline{\ell}(v_j) = (A, a)$,*
4. *$\overline{\ell}(v_s) = \ell(v_s)$.*

The resulting tree is called the marked ρ-version *of T and denoted by $\overline{T}(\rho)$. The word $\overline{\ell}(v_0) \dots \overline{\ell}(v_s)$ is referred to as the* spine *of $\overline{T}(\rho)$ and denoted by $Spine\big(\overline{T}(\rho)\big)$.*

We write $\delta(w, z)$ whenever there exists some u such that the word wu has a derivation tree T in G with a path ρ ending on the last character of w and with $Spine\big(\overline{T}(\rho)\big) = z$. Then, we have $\theta(w) = \{ x \mid \delta(w, z) \wedge \tau(x, z) \}$. Intuitively, this is the *spine-type* of w.

Finally, Ehrenfeucht and Rozenberg show that, whenever all strong iterative pairs of G are very degenerate, then $\theta(w) = \theta(w')$ implies $w \sim w'$. Since there are a finite number of types x, we have regularity by Myhill and Nerode.

6.2 Building the Automaton

We show how to make the above result constructive. The first step is to decide $\theta(w) = \theta(w')$ for given w and w'. To do this, from G and some type x, we build G_x which generates all w such that $\delta(w, z)$ holds for some z of type x. Thus $x \in \theta(w)$ iff $w \in \mathcal{L}(G_x)$.

First note that there is a simple (polynomial) regular automaton $\mathcal{A}_x$ recognising, for $x = a_1 \dots a_s$ the language

$$\left(a_1 \cup a_1 \overline{\Sigma}^* a_1\right) \dots \left(a_s \cup a_s \overline{\Sigma}^* a_s\right)$$

and $z \in \mathcal{L}(\mathcal{A}_x)$ iff z is of type x. The idea is to build this automaton into the productions of G to obtain G_x such that all characters to the left (inclusive) of the path chosen by $\mathcal{A}_x$ are kept, while all those to the right are erased.

Definition 10 (G_x). *For a given word type x and CFG G, the grammar G_x has the following production rules:*

- *all productions in G,*
- *$A_q \to B_{q'} C_\varepsilon$ for each $A \to BC \in G$ and $q \xrightarrow{(A,B,C,1)} q'$ in $\mathcal{A}_x$,*
- *$A_q \to B C_{q'}$ for each $A \to BC \in G$ and $q \xrightarrow{(A,B,C,2)} q'$ in $\mathcal{A}_x$,*
- *$A_q \to a$ for each $A \to a \in G$ and $q \xrightarrow{(A,a)} q'$ in $\mathcal{A}_x$ where q' is a final state,*
- *$A_\varepsilon \to B_\varepsilon C_\varepsilon$ for each $A \to BC \in G$,*
- *$A_\varepsilon \to \varepsilon$ for each $A \to a \in G$.*

The initial non-terminal is S_{q_0} where S is the initial non-terminal of G and q_0 is the initial state of $\mathcal{A}_x$.

The correctness of G_x is straightforward and hence relegated to the appendix.

Lemma 3. *For all w, we have $w \in \mathcal{L}(G_x)$ iff $x \in \theta(w)$.*

Lemma 4 (Deciding $\theta(w) = \theta(w')$). *For given w and w', we can decide $\theta(w) = \theta(w')$ in $\mathcal{O}\left(2^{f(n)}\right)$ time for some polynomial f where n is the size of G.*

Proof. For a given alphabet $\overline{\Sigma}$, there are $\sum_{r=1}^{m} r!$ types where $m = \left|\overline{\Sigma}\right|$. Since m is polynomial in n, there are $\mathcal{O}\left(2^{f(n)}\right)$ word types. Hence, we simply check $w \in \mathcal{L}(G_x)$ and $w' \in \mathcal{L}(G_x)$ for each type x. This is polynomial for each x, giving $\mathcal{O}\left(2^{f(n)}\right)$ in total.

From this, we can construct, following Myhill and Nerode, the required automaton, using a kind of fixed point construction beginning with an automaton containing the state q_ε from which the equivalence class associated to the empty word will be accepted.

Lemma 5. *For a CFG G such that all strong iterative pairs are very degenerate, we can build an NFA $\mathcal{A}$ of $\mathcal{O}\left(2^{2^{f(n)}}\right)$ size in the same amount of time, where n is the size of G.*

Proof. Let G be a CFG such that all strong iterative pairs are degenerate. We build an NFA $\mathcal{A}$ such that $\mathcal{L}(G) = \mathcal{L}(\mathcal{A})$ by the following worklist algorithm.

1. Let the worklist contain only ε (the empty word) and $\mathcal{A}$ have the initial state q_ε.
2. Take a word w from the worklist.
3. If $w \in \mathcal{L}(G)$, make q_w a final state.
4. For each $a \in \Gamma$
   (a) if there is no state $q_{w'}$ such that $\theta(wa) = \theta(w')$, add q_{wa} to $\mathcal{A}$ and add wa to the worklist,
   (b) take $q_{w'}$ in $\mathcal{A}$ such that $\theta(wa) = \theta(w')$,
   (c) add the transition $q_w \xrightarrow{a} q_{w'}$ to $\mathcal{A}$.
5. If the worklist is not empty, go to point 2, else, return $\mathcal{A}$.

Since this follows the Myhill-Nerode construction, using $\theta(w) = \theta(w')$ as a proxy for $w \sim w'$, we have that the algorithm terminates and is correct. Hence, with the observation that there are $\mathcal{O}\left(2^{2^{f(n)}}\right)$ different values of the sets $\theta(w)$, we have the lemma.

7 Conclusions and Future Work

In this work, we have studied the parameterised master/slave reachability problem for pushdown systems with a global store. This provides an extension of work by Kahlon which did not allow a master process, and communication was via anonymous synchronisation; however, this is obtained at the expense of atomic accesses to global variables. Our algorithm introduces new techniques to pushdown system analysis.

An initial inspiration for this work was the study of weak-memory models, which do not guarantee that — in a multi-threaded environment — memory accesses are *sequentially consistent.* In general, if atomic read/writes are permitted, the verification problem is harder (for example, Atig *et al.* relate the finite-state case to lossy channel machines [4]); hence, we removed atomicity as a natural first step. It is not clear how to extend our algorithm to accommodate weak-memory models and it remains an interesting avenue of future work.

Another concern is the complexity gap between the upper and lower bounds. We conjecture that the upper bound can be improved, although we may require a new approach, since the complexity comes from the construction of regular read languages. A related question is whether we can improve the size of the automata $\mathcal{A}_{w(g)}$. Since a PDS of size n can recognise the language $\{ a^{2^n} \}$, we have a read language requiring an exponential number of a characters; hence, the $\mathcal{A}_{w(g)}$ must be at least exponential in the worst case. It is worth noting that Meyer and Fischer give a language whose *deterministic* regular automaton is doubly-exponential in the size of the corresponding deterministic PDS [24]. However, in the appendix, we provide an example showing that this language is not very degenerate. If the PDS is not deterministic, Meyer and Fischer prove there is no bound, in general, on the relationship in sizes.

Finally, we may also consider applications to recursive ping-pong protocols in the spirit of Delzanno *et al.* [12].

Acknowledgments Nous remercions Jade Alglave pour plusieurs discussions qui ont amorcées ce travail. This work was funded by EPSRC grant EP/F036361/1. We also thank the anonymous reviewers and Ahmed Bouajjani for their helpful remarks.

References

1. P. A. Abdulla, N. B. Henda, G. Delzanno, and A. Rezine. Handling parameterized systems with non-atomic global conditions. In *Proceedings of the 9th international conference on Verification, model checking, and abstract interpretation*, VMCAI'08, pages 22–36, Berlin, Heidelberg, 2008. Springer-Verlag.
2. K. Apt and D. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters (IPL)*, 1986.
3. M. F. Atig, B. Bollig, and P. Habermehl. Emptiness of multi-pushdown automata is 2etime-complete. In *Developments in Language Theory*, pages 121–133, 2008.
4. M. F. Atig, A. Bouajjani, S. Burckhardt, and M. Musuvathi. On the verification problem for weak memory models. In *POPL*, pages 7–18, 2010.
5. M. F. Atig, A. Bouajjani, and T. Touili. On the reachability analysis of acyclic networks of pushdown systems. In *CONCUR*, pages 356–371, 2008.
6. T. Ball and S. K. Rajamani. Bebop: A symbolic model checker for boolean programs. In *Proceedings of the 7th International SPIN Workshop on SPIN Model Checking and Software Verification*, pages 113–130, London, UK, 2000. Springer-Verlag.
7. T. Ball and S. K. Rajamani. The SLAM project: Debugging system software via static analysis. In *Conference Record of POPL'02: The 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 1–3, Portland, Oregon, January 16–18, 2002.
8. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *International Conference on Concurrency Theory*, pages 135–150, 1997.
9. A. Bouajjani, J. Esparza, S. Schwoon, and J. Strejcek. Reachability analysis of multithreaded software with asynchronous communication. In *FSTTCS*, pages 348–359, 2005.
10. A. Bouajjani, J. Esparza, and T. Touili. A generic approach to the static analysis of concurrent programs with procedures. *SIGPLAN Not.*, 38(1):62–73, 2003.

11. A. Bouajjani, M. Müller-Olm, and T. Touili. Regular symbolic analysis of dynamic networks of pushdown systems. *CONCUR 2005 - Concurrency Theory*, pages 473–487, 2005.
12. G. Delzanno, J. Esparza, and J. Srba. Monotonic set-extended prefix rewriting and verification of recursive ping-pong protocols. In *Proceedings of ATVA 2006*, Beijing, China, 2006.
13. A. Ehrenfeucht and G. Rozenberg. Strong iterative pairs and the regularity of context-free languages. *ITA*, 19(1):43–56, 1985.
14. J. Esparza, A. Kučera, and S. Schwoon. Model-checking LTL with regular valuations for pushdown systems. In *Proc. of TACS 2001*, number 2215 in Lecture Notes in Computer Science, pages 306–339, 2001.
15. P. Ganty, R. Majumdar, and B. Monmege. Bounded underapproximations. In *CAV*, pages 600–614, 2010.
16. S. German and A. P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39:675–735, 1992.
17. M. Hague and A. W. Lin. Model checking recursive programs with numeric data types. In *CAV*, pages 743–759, 2011.
18. A. Heußner, J. Leroux, A. Muscholl, and G. Sutre. Reachability analysis of communicating pushdown systems. In *Proc. 13th Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS'10), Paphos, Cyprus, Mar. 2010*, volume 6014 of *Lecture Notes in Computer Science*, pages 267–281. Springer, 2010.
19. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.
20. N. D. Jones and S. S. Muchnick. Even simple programs are hard to analyze. *J. ACM*, 24:338–350, April 1977.
21. V. Kahlon. Parameterization as abstraction: A tractable approach to the dataflow analysis of concurrent programs. In *LICS*, pages 181–192, 2008.
22. V. Kahlon, F. Ivancic, and A. Gupta. Reasoning about threads communicating via locks. In *CAV*, pages 505–518, 2005.
23. R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems.* PhD thesis, TU-München, 1998.
24. A. R. Meyer and M. J. Fischer. Economy of description by automata, grammars, and formal systems. In *IEEE Symposium on Foundations of Computer Science*, pages 188–191, 1971.
25. S. Qadeer. The case for context-bounded verification of concurrent programs. In *Proceedings of the 15th international workshop on Model Checking Software*, SPIN '08, pages 3–6, Berlin, Heidelberg, 2008. Springer-Verlag.
26. G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *Transactions on Programming Languages and Systems (TOPLAS)*, 2000.
27. T. Reps, S. Schwoon, S. Jha, and D. Melski. Weighted pushdown systems and their application to interprocedural dataflow analysis. *Sci. Comput. Program.*, 58(1-2):206–263, 2005.
28. S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS*, pages 93–107, 2005.
29. S. Schwoon. *Model-checking Pushdown Systems.* PhD thesis, Technical University of Munich, 2002.
30. K. Sen and M. Viswanathan. Model checking multithreaded programs with asynchronous atomic methods. In *CAV*, pages 300–314, 2006.
31. A. Seth. Global reachability in bounded phase multi-stack pushdown systems. In *CAV*, pages 615–628, 2010.
32. I. Suzuki. Proving properties of a ring of finite-state machines. *Inf. Process. Lett.*, 28:213–214, July 1988.
33. S. La Torre, P. Madhusudan, and G. Parlato. Context-bounded analysis of concurrent queue systems. In *Proceedings of the Theory and practice of software, 14th international conference on Tools and algorithms for the construction and analysis of systems*, TACAS'08/ETAPS'08, pages 299–314, Berlin, Heidelberg, 2008. Springer-Verlag.
34. S. La Torre, P. Madhusudan, and G. Parlato. Model-checking parameterized concurrent programs using linear interfaces. In *CAV*, pages 629–644, 2010.

A Proofs for Section 4

The proof of Lemma 2 is split into the following two lemmas.

Lemma 6. *The PDS $\mathcal{P}_{sys}$ has a run to some control-state in $\mathcal{F}$, then the parameterised reachability problem for $\mathcal{U}$, $\mathcal{C}$, $\mathcal{G}$, g_0 and q has a positive solution.*

Proof. Take an accepting run of $\mathcal{P}_{sys}$. We can extract a number of sequences from this run. First, let $G = g_G^1, \ldots, g_G^z$ be the sequence of values written to the global (last) component of $\mathcal{P}_{sys}$'s control-state. Note, $g_G^1 = g_0$. Then, for each $g \in \mathcal{G}$ that is written to the global component, let R_g be the sequence of read and # events that took $\mathcal{A}_{w(g)}$ from $q_0^{w(g)}$ to a state in $\mathcal{F}_{w(g)}$. Since this is accepted by the read language of g, there is a subword $r(g^1), \ldots, r(g^x)$ of R_g and sequences of writes $W_0, \ldots, W_x$ such that $W_0 r(g^1) W_1 \ldots r(g^x) W_x w(g)$ is a run of $\mathcal{C}$ (with internal transitions hidden).

Furthermore, let $\#^i$ be a sequence of # characters the same length as W_i. Notice, we can fix a sub-sequence $G_g = \#^0 g^1 \cdots g^x \#^x g$ of R_g corresponding to a run of $\mathcal{C}$ in the sense that, # characters represent some write action, the g^h for all $1 \leq h \leq x$ are read events of g^h, and g is a write of g. Similarly, $\mathcal{U}$ has a sub-sequence $G_{\mathcal{U}}$ leading to q. This sequence is mapped on to G as follows. The sequence G partitions the run of $\mathcal{P}_{sys}$ into contiguous sections with each g_G^i beginning a new section. Since G_g is a sub-sequence of R_g which is in turn a sub-sequence of the run of $\mathcal{P}_{sys}$, there is a natural mapping of elements of G_g to the transitions in the run of $\mathcal{P}_{sys}$. Each character is mapped to the element of G that begins the section the transition occurs in. Similarly, $\mathcal{U}$ has a sequence $G_{\mathcal{U}}$ leading to q.

We create the NPDS which has a unique process $\mathcal{C}$ for each g_G^i in G that is not # and is not written by $\mathcal{U}$ (that is, a process for each individual write). We build the run in z segments: one for each g_G^i. In each segment, all processes whose sub-sequence G_g or $G_{\mathcal{U}}$ maps a character onto g_G^i will be scheduled to make the corresponding transitions. These can be scheduled in any order, except the process running first in the segment must be the process responsible for writing g_G^i. When $g_G^i = \#$, the process will not write # to the store, but some other character. Since no process reads # this is safe.

Observe that there may be some g_G^i that are not written by any process. In this case $g_G^i = \#$ (since we allowed # to occur at any time) and, because no process reads #, the corresponding segment is merely ε.

Lemma 7. *If the parameterised reachability problem for $\mathcal{U}$, $\mathcal{C}$, $\mathcal{G}$, g_0 and q has a positive solution, then $\mathcal{P}_{sys}$ has a run to some control-state in $\mathcal{F}$.*

Proof. Take a run $C = c_0 c_1 \ldots c_z$ of the NPDS with n copies of $\mathcal{C}$ that reaches q. From this, we build an accepting run π of $\mathcal{P}_{sys}$. The initial configuration of π is $\left(q_0^{\mathcal{U}}, q_0^{w(g_0)}, \ldots, q_0^{w(g_m)}, g_0\right)$. Assume we have a run π_i corresponding to the run of the NPDS up to c_i. This run will have the property that the first component (the control-state of $\mathcal{U}$) of the last configuration in π_i will match the control-state of $\mathcal{U}$ in c_i. Hence, π_z will be the required accepting run.

Take the first write of $w(g)$ of each $g \in \mathcal{G}$ that is written by some copy of $\mathcal{C}$. Take the run of $\mathcal{C}$ that produced the write which is a sequence of reads and writes $W_0 R_1 W_1 \ldots R_x W_x$ (with internal moves omitted). Let $\#_j$ be a sequence of # characters with the same length as W_j. There is as accepting run $q_0^{w(g)} \xrightarrow{\gamma_1} q_1^{w(g)} \xrightarrow{\gamma_2} \cdots \xrightarrow{\gamma_y} q_y^{w(g)}$ of $\mathcal{A}_{w(g)}$ where $\#_0 R_1 \#_1 R_1 \ldots R_x \#_x = \gamma_1 \ldots \gamma_y$. Furthermore, $\gamma_1 \ldots \gamma_y$ can be mapped onto a sub-word of the sequence of actions taken on the global component up to the first write of g.

Let $\left(q_{i_{\mathcal{U}}}^{\mathcal{U}}, q_{i_{g_0}}^{w(g_0)}, \ldots, q_{i_{g_m}}^{w(g_m)}, g^i\right)$ be the final configuration of π_i. We extend π_i with the following transitions, in order of appearance.

- For all g such that we have a maximal path $q_{i_g}^{w(g)} \xrightarrow{r(g)} \cdots \xrightarrow{r(g)} q_{i_g+1}^{w(g)}$, make the transitions to $q_{i_g+1}^{w(g)}$. (That is, read g as many times as possible.)

– If the transition between c_i and c_{i+1} is a move of $\mathcal{U}$, then simulate the move directly.
– If the transition is a write move $w(g)$ by a copy of $\mathcal{C}$ which is not responsible for the first write of g, but is responsible for for the first write of some other g', then advance $q_{i_{g'}}^{w(g')} \xrightarrow{\#} q_{i_{g'}}^{w(g')}$, setting the global component to # as required. Note that the transition from $q_{i_{g'}}^{w(g')}$ must be a # move since it is a write move of $\mathcal{C}$ and all preceding reads and writes have been simulated.
– Further to the above, if it is a write of g by some $\mathcal{C}$, we know that $q_{i_g}^{w(g)}$ is an accepting state of $\mathcal{A}_{w(g)}$. This is because we have been simulating the sequence $W_0 R_1 W_1 \dots R_x W_x$ with the accepting run $\#_0 R_1 \#_1 \dots R_x \#_x$. Hence we can (and do) perform the write of g to the global component.
– Other types of transitions have no further updates to π_i. In particular, if the transition is a read move by some copy of $\mathcal{C}$ we do not add any transitions (these moves are taken care of more eagerly above).

This completes the construction of π_i, and thus π_y gives us a required accepting run of $\mathcal{P}_{sys}$.

B Non-Atomic Pushdown Systems with Multiple Variables

B.1 Model Definition

Definition 11 (Non-atomic Pushdown Systems with Multiple-Variables). *Over a partitioned finite alphabet $\mathcal{G} = \mathcal{G}_1 \uplus \dots \uplus \mathcal{G}_k$, a* non-atomic pushdown system *(naPDS) is a tuple $\mathcal{P} = (\mathcal{Q}, \Sigma, \Delta, q_0, \mathcal{G}_1, \dots, \mathcal{G}_k)$ where $\mathcal{Q}$ is a finite set of control-states, Σ is a finite stack alphabet, $q_0 \in \mathcal{Q}$ is a designated initial control state and $\Delta \subseteq (\mathcal{Q} \times \Sigma) \times (r(\mathcal{G}) \cup w(\mathcal{G}) \cup \{ \varepsilon \}) \times (\mathcal{Q} \times \Sigma^*)$.*

Definition 12 (Networks of naPDSs with Multiple Variables). *A network of* n non-atomic pushdown systems *(NPDS) is a tuple $\mathcal{N} = \left(\mathcal{P}_1, \dots, \mathcal{P}_n, \mathcal{G}_1, \dots, \mathcal{G}_k, g_0^1, \dots, g_0^k\right)$ where, for all $1 \leq i \leq n$, $\mathcal{P}_i = \left(\mathcal{Q}_i, \Sigma_i, \Delta_i, q_0^i, \mathcal{G}_1, \dots, \mathcal{G}_k\right)$ is a NPDS over $\mathcal{G}_1, \dots, \mathcal{G}_k$ and for all $1 \leq i \leq k$, $g_0^i \in \mathcal{G}_i$ is the initial value of the ith global store.*

A configuration of an NPDS is a tuple $(q_1, w_1, \dots, q_n, w_n, g_1, \dots, g_k)$ where $g_i \in \mathcal{G}_i$ for each $1 \leq i \leq k$, and for each $1 \leq i \leq n$, $q_i \in \mathcal{Q}_i$ and $w_i \in \Sigma_i^*$. We have a transition

$$(q_1, w_1, \dots, q_n, w_n, g_1, \dots, g_k) \to (q_1', w_1', \dots, q_n', w_n', g_1', \dots, g_k')$$

whenever, for some $1 \leq i \leq n$ and all $1 \leq j \leq n$ with $i \neq j$ we have $q_j' = q_j$, $w_j' = w_j$, and

– $(q_i, w_i) \to (q_i', w_i')$ is a transition of $\mathcal{P}_i$ and for all $1 \leq l \leq k$, $g_l' = g_l$; or
– $(q_i, w_i) \xrightarrow{r(g_l)} (q_i', w_i')$ is a transition of $\mathcal{P}_i$ for some $1 \leq l \leq k$ and for all $1 \leq l' \leq k$, $g_{l'}' = g_{l'}$; or
– $(q_i, w_i) \xrightarrow{w(g_l')} (q_i', w_i')$ is a transition of $\mathcal{P}_i$ for some $1 \leq l \leq k$ and for all $1 \leq l' \leq k$ such that $l' \neq l$, $g_{l'}' = g_{l'}$.

A path π of $\mathcal{N}$ is a sequence of configurations $c_1 c_2 \dots c_z$ such that, for all $1 \leq i < z$, $c_i \to c_{i+1}$. A run of $\mathcal{N}$ is a path such that $c_1 = \left(q_0^1, \bot, \dots, q_0^n, \bot, g_0^1, \dots, g_0^k\right)$.

B.2 Reachability Analysis

In this section, we aim prove the following theorem.

Theorem 4. *The parameterised reachability problem for NPDSs with multiple variables is decidable.*

Again, we assume q is a control-state of $\mathcal{U}$. The idea is the same as the single variable case, except for some minor adjustments to handle the extra variables.

Regular Read Languages Given a non-atomic pushdown system $\mathcal{P}$ we define for each $g \in \mathcal{G}$ the pushdown system $\mathcal{P}_{w(g)}$ which is $\mathcal{P}$ augmented with a new unique control-state f, and a transition $(q, a) \hookrightarrow (f, a)$ whenever $\mathcal{P}$ has a rule $(q, a) \xhookrightarrow{w(g)} (q', w)$. Furthermore, replace all $(q, a) \xhookrightarrow{w(g')} (q', w)$ rules with $(q, a) \xhookrightarrow{\#_i} (q', w)$ where $\#_i \notin \mathcal{G}_1 \cup \dots \cup \mathcal{G}_k$ and $g' \in \mathcal{G}_i$.

Again, we interpret f as the sole accepting control state of $\mathcal{P}_{w(g)}$ giving the read language $\mathcal{L}_{w(g)}$ for $w(g)$ defined as

$$\mathcal{L}_{w(g)} = \left\{ \; R^*\gamma_1 R^* \dots R^*\gamma_z R^* \;\middle|\; \gamma_1 \dots \gamma_z \in \mathcal{L}\big(\mathcal{P}_{w(g)}\big) \; \right\}$$

where $R = \{ \; r(g') \mid g' \in \mathcal{G} \; \} \cup \{ \; \#_1, \dots, \#_k \; \}$.

Lemma 8. *For all $g \in \mathcal{G}$, $\mathcal{L}_{w(g)}$ is regular and an NFA $\mathcal{A}$ accepting $\mathcal{L}_{w(g)}$, of doubly-exponential size, can be constructed in doubly-exponential time.*

Proof. Identical to the single variable case.

Simulating the System We build a PDS that recognises a non-empty language iff the parameterised reachability problem has a positive solution. The intuition behind the construction of $\mathcal{P}_{sys}$ is the same as the single variable case, except minor adjustments are needed to handle the interaction with multiple variables.

Definition 13 ($\mathcal{P}_{sys}$). *Given an naPDS $\mathcal{U} = \left(\mathcal{Q}_\mathcal{U}, \Sigma, \Delta_\mathcal{U}, q_0^\mathcal{U}, \mathcal{G}_1, \dots, \mathcal{G}_k\right)$ over $\mathcal{G} = \mathcal{G}_1 \uplus \dots \uplus \mathcal{G}_k$ with initial values $g_0^1, \dots, g_0^k$, a control-state $f \in \mathcal{Q}_\mathcal{U}$, and, for each $g \in \mathcal{G}$, a regular automaton $\mathcal{A}_{w(g)} = \left(\mathcal{Q}_{w(g)}, R, \Delta_{w(g)}, \mathcal{F}_{w(g)}, q_0^{w(g)}\right)$, we define the PDS $\mathcal{P}_{sys} = (\mathcal{Q}, \Sigma, \Delta, q_0, \mathcal{F})$ where*

- *we let, for all i, $\mathcal{G}_i = \{ \; g_0^i, \dots, g_{m_i}^i \; \}$ and,*
- *let $\boldsymbol{\mathcal{Q}} = \mathcal{Q}_{w\left(g_0^1\right)} \times \dots \times \mathcal{Q}_{w\left(g_{m_1}^1\right)} \times \dots \times \mathcal{Q}_{w\left(g_0^k\right)} \times \dots \times \mathcal{Q}_{w\left(g_{m_k}^k\right)}$, then*
- *$\mathcal{Q} = \mathcal{Q}_\mathcal{U} \times \boldsymbol{\mathcal{Q}} \times (\mathcal{G}_1 \cup \{ \; \#_1 \; \}) \times \dots \times (\mathcal{G}_k \cup \{ \; \#_k \; \})$,*
- *$q_0 = \left(q_0^\mathcal{U}, q_0^{w\left(g_0^1\right)}, \dots, q_0^{w\left(g_{m_k}^k\right)}, g_0^1, \dots, g_0^k\right)$,*
- *$\mathcal{F} = \{ \; f \; \} \times \boldsymbol{\mathcal{Q}} \times (\mathcal{G}_1 \cup \{ \; \#_1 \; \}) \times \dots \times (\mathcal{G}_k \cup \{ \; \#_k \; \})$,*

and Δ is the smallest set containing all $(q, a) \hookrightarrow (q', w)$ where $q = \left(q_\mathcal{U}, q_0^1, \dots, q_{m_k}^k, g_1, \dots, g_k\right)$ and,

- *$q' = \left(q'_\mathcal{U}, q_0^1, \dots, q_{m_k}^k, g_1, \dots, g_k\right)$ and $(q_\mathcal{U}, a) \hookrightarrow (q'_\mathcal{U}, w) \in \Delta_\mathcal{U}$, or*
- *$q' = \left(q'_\mathcal{U}, q_0^1, \dots, q_{m_k}^k, g_1, \dots, g_k\right)$ and $(q_\mathcal{U}, a) \xhookrightarrow{r(g_i)} (q'_\mathcal{U}, w) \in \Delta_\mathcal{U}$ for some i, or*
- *$q' = \left(q'_\mathcal{U}, q_0^1, \dots, q_{m_k}^k, g_1, \dots, g'_i, \dots, g_k\right)$ and $(q_\mathcal{U}, a) \xhookrightarrow{w(g'_i)} (q'_\mathcal{U}, w) \in \Delta_\mathcal{U}$ for some $g'_i \in \mathcal{G}_i$, or*
- *$q' = \left(q_\mathcal{U}, q_0^1, \dots, p_j^i, \dots, q_{m_k}^k, g_1, \dots, g_k\right)$ and $q_j^i \xrightarrow{r(g_l)} p_j^i \in \Delta_j^i$ for some l, $q_j^i \notin \mathcal{F}_j^i$ and $w = a$, or*
- *$q' = \left(q_\mathcal{U}, q_0^1, \dots, p_j^i, \dots, q_{m_k}^k, g_1, \dots, \#_l, \dots, g_k\right)$ and $q_j^i \xrightarrow{\#_l} p_j^i \in \Delta_j^i$, $q_j^i \notin \mathcal{F}_j^i$ and $w = a$, or*
- *$q' = \left(q_\mathcal{U}, q_0^1, \dots, q_{m_k}^k, g_1, \dots, g_j^i, \dots, g_k\right)$, $q_j^i \in \mathcal{F}_j^i$ and $w = a$.*

We have the following property.

Lemma 9. *The PDS $\mathcal{P}_{sys}$ has a run to some control-state in $\mathcal{F}$ iff the parameterised reachability problem for $\mathcal{U}$, $\mathcal{C}$, $\mathcal{G}_1, \dots, \mathcal{G}_k$, $g_0^1, \dots, g_0^k$ and q has a positive solution.*

We prove this property in the following lemmas, and conclude that the parameterised reachability problem with multiple variables is decidable.

Lemma 10. *The PDS $\mathcal{P}_{sys}$ has a run to some control-state in $\mathcal{F}$, then the parameterised reachability problem for $\mathcal{U}$, $\mathcal{C}$, $\mathcal{G}_1, \dots, \mathcal{G}_k$, $g_0^1, \dots, g_0^k$ and q has a positive solution.*

Proof. Take an accepting run of $\mathcal{P}_{sys}$. We can extract a number of sequences from this run. First, let $G = \boldsymbol{g}^1, \ldots, \boldsymbol{g}^z$ be the sequence of updates to the global (last k) components of $\mathcal{P}_{sys}$'s control-state. That is, $g^1 = \left(g_0^1, \ldots, g_0^k\right)$, and $\boldsymbol{g}^{i+1}$ is generated from $\boldsymbol{g}^i$ by the next change to a global component. Then, for each g that is written to a global component, let R_g be the sequence of read and $\#_1, \ldots, \#_k$ events that took $\mathcal{A}_{w(g)}$ from $q_0^{w(g)}$ to a state in $\mathcal{F}_{w(g)}$. Since this is accepted by the read language of g, there is a subword $r\left(g^1\right), \ldots, r(g^x)$ of R_g and sequences of writes $W_0, \ldots, W_x$ such that $W_0 r\left(g^1\right) W_1 \ldots r(g^x) W_x w(g)$ is a run of $\mathcal{C}$ (with internal transitions hidden).

Furthermore, let $\#^i$ be a sequence of actions derived from W_i by replacing each write to a variable j with the character $\#_j$. We can fix a sub-sequence $G_g = \#^0 g^1 \cdots g^x \#^x g$ of R_g corresponding to the run of $\mathcal{C}$ above. This sequence is mapped on to G as follows. The sequence G partitions the run of $\mathcal{P}_{sys}$ into contiguous sections with each $\boldsymbol{g}^i$ beginning a new section. Since G_g is a sub-sequence of R_g which is in turn a sub-sequence of the run of $\mathcal{P}_{sys}$, there is a natural mapping of elements of G_g to the transitions in the run of $\mathcal{P}_{sys}$. Each character is mapped to the element of G that begins the section the transition occurs in. Similarly, $\mathcal{U}$ has a sequence $G_{\mathcal{U}}$ leading to q.

We create the NPDS which has a unique process $\mathcal{C}$ for each $\boldsymbol{g}^i$ in G that is not a $\#_j$ event for some j and is not written by $\mathcal{U}$ (that is, a process for each individual write). We build the run in z segments: one for each $\boldsymbol{g}^i$. In each segment, all processes whose sub-sequence G^g (when the update given by $\boldsymbol{g}^i$ is a write of the character g) or $G_{\mathcal{U}}$ maps a character onto $\boldsymbol{g}^i$ will be scheduled to make the corresponding transitions (including internal transitions). These can be scheduled in any order, except the process running first in the segment must be the process responsible for writing g. When $\boldsymbol{g}^i$ is a write of $\#_j$, the process will not write $\#_j$ to the jth component of the store, but some other character. Since no process reads $\#_j$ this is safe.

Observe that there may be some updates $\boldsymbol{g}^i$ that are not written by any process. In this case the update is the write of some $\#_j$ (since we allowed $\#_j$ to occur at any time) and, because no process reads $\#_j$, the corresponding segment is merely ε.

Lemma 11. *If the parameterised reachability problem for $\mathcal{U}$, $\mathcal{C}$, $\mathcal{G}_1, \ldots, \mathcal{G}_k$, $g_0^1, \ldots, g_0^k$ and q has a positive solution, then $\mathcal{P}_{sys}$ has a run to some control-state in $\mathcal{F}$.*

Proof. Take a run $C = c_0 c_1 \ldots c_z$ of the NPDS with n copies of $\mathcal{C}$ that reaches q. From this, we build an accepting run π of $\mathcal{P}_{sys}$. The initial configuration of π is

$$\left(q_0^{\mathcal{U}}, q_0^{w\left(g_0^1\right)}, \ldots, q_0^{w\left(g_{m_k}^k\right)}, g_0^1, \ldots, g_0^k\right) .$$

Assume we have a run π_i corresponding to the run of the NPDS up to c_i. This run will have the property that the first component (the control-state of $\mathcal{U}$) of the last configuration in π_i will match the control-state of $\mathcal{U}$ in c_i. Hence, π_z will be the required accepting run.

Take the first write of $w(g)$ for each $g \in \mathcal{G}$ that is written by some copy of $\mathcal{C}$. Take the run of $\mathcal{C}$ that produced the write which is a sequence of reads and writes $W_0 R_1 W_1 \ldots R_x W_x$ (with internal moves omitted). Let $\#^j$ be a sequence of $\#_1, \ldots, \#_k$ characters derived from W_j as in the proof of Lemma 10. There is an accepting run of $\mathcal{A}_{w(g)}$

$$q_0^{w(g)} \xrightarrow{\gamma_1} q_1^{w(g)} \xrightarrow{\gamma_2} \cdots \xrightarrow{\gamma_y} q_y^{w(g)}$$

where $\#^0 R_1 \#^1 R_1 \ldots R_x \#^x = \gamma_1 \ldots \gamma_y$. Furthermore, $\gamma_1 \ldots \gamma_y$ can be mapped onto a sub-word of the sequence of actions taken on the global components up to the first write of g.

Let $\left(q_{i_{\mathcal{U}}}^{\mathcal{U}}, q_{i_{g_0^1}}^{w\left(g_0^1\right)}, \ldots, q_{i_{g_{m_k}^k}}^{w\left(g_{m_k}^k\right)}, g_1, \ldots, g_k\right)$ be the final configuration of π_i. We extend π_i with the following transitions, in order of appearance.

- For all g such that we have a maximal path $q_{i_g}^{w(g)} \xrightarrow{r\left(g^1\right)} \cdots \xrightarrow{r(g^y)} q_{i_g+1}^{w(g)}$ where g^j for $1 \leq j \leq y$ are characters in $\{\ g_1, \ldots, g_k\ \}$, make the transitions to $q_{i_g+1}^{w(g)}$. (That is, read the current global store as many times as possible.)

– If the transition between c_i and c_{i+1} is a move of $\mathcal{U}$, then simulate the move directly.
– If the transition is a write move $w(g)$ for some $g \in \mathcal{G}$ by a copy of $\mathcal{C}$ which is not responsible for the first write of g, but is responsible for for the first write of some other g', then advance $q_{i_{g'}}^{w(g')} \xrightarrow{\#_j} q_{i_{g'}}^{w(g')}$, setting the jth global component to $\#_j$ as required. Note that the transition from $q_{i_{g'}}^{w(g')}$ must be a $\#_j$ move since it is a write move to the jth component of $\mathcal{C}$ and all preceding reads and writes have been simulated.
– Further to the above, if it is a write of g by some $\mathcal{C}$, we know that $q_{i_g}^{w(g)}$ is an accepting state of $\mathcal{A}_{w(g)}$. This is because we have been simulating the sequence $W_0R_1W_1 \dots R_xW_x$ with the accepting run $\#_0R_1\#_1 \dots R_x\#_x$. Hence we can (and do) perform the write of g to the global component.
– Other types of transitions have no further updates to π_i. In particular, if the transition is a read move by some copy of $\mathcal{C}$ we do not add any transitions (these moves are taken care of more eagerly above).

This completes the construction of π_i, and thus π_y gives us a required accepting run of $\mathcal{P}_{sys}$.

C Complexity Lower Bounds

Theorem 5. *The parameterised reachability problem for NPDSs with a single global store is NP-hard, even when the stacks are removed.*

Proof. We reduce from SAT. The encoding is as follows: $\mathcal{U}$ first guesses an assignment to the variables $x_1, \dots, x_n$ (say). He does this by writing 1_i or 0_i to the global store for each $1 \leq i \leq n$. The $\mathcal{C}$ process has n branches. Along the ith branch it reads, and remembers in its control state, the value of x_i written by $\mathcal{U}$. Then, whenever a symbol $?_i$ can be read from the global store, $\mathcal{C}$ reads it and writes 1_i or 0_i as appropriate.

Then, also in its control state, $\mathcal{U}$ evaluates the boolean formula. When it needs to obtain the value of x_i. it writes $?_i$ to the global store and waits for a copy of $\mathcal{C}$ to return the answer. A unique control state is reached if the formula evaluates to true. Hence, the defined parameterised reachability instance reaches this control state iff the formula can be satisfied.

It is not immediately obvious how to evaluate the formula in the control state. The technique is the same as in Hague and Lin [17]. To evaluate a non-atomic formula, we store it as a tree in the control state. Evaluation uses a kind of tree automaton (the run of which is encoded into the state space). The tree automaton navigates the tree in left most, depth first order. First it moves down to the left most leaf. This will be an atomic proposition. The proposition is evaluated using the technique above and the value is passed up to the parent. When first returning to a parent node, it is marked as seen. If the node is a disjunction, and the value returned is 1, then the automaton returns to the parent, also carrying the 1, otherwise it moves down into the right subtree. The automaton eventually returns from this tree with a value. Since the node is marked, it detects that it has fully evaluated the disjunction and returns the value to the parent. Evaluation is analogous for conjunction. Finally, a value is returned from the root.

The evaluation above only introduces a polynomial number of control states. Because the tree is navigated in left most depth first order, there are a linear number of different markings (if the right hand subtree is not visited, we can simply mark all of the nodes in this subtree without affecting the execution). Then, to keep track of the automaton, we attach the state of the automaton to the node of the tree it is at. This is only polynomial since there is only one node marked by the automaton state at a time.

D Proofs for Section Section 6

Lemma 3. *For all w, we have $w \in \mathcal{L}(G_x)$ iff $x \in \theta(w)$.*

Proof. First, assume $w \in \mathcal{L}(G_x)$. We show $x \in \theta(w)$. Take the derivation tree of w in G_x. By definition, this tree has a path marked by a run of $\mathcal{A}_x$, such that w is derived to the left (inclusive) of the path, and the empty word is derived to the right. By replacing all non-terminals A_q and A_ε with their corresponding non-terminals in G, and adjusting the applied production rules accordingly, we obtain a derivation tree of some word wu containing a spine of type x. Hence $x \in \theta(w)$, as required.

In the other direction, consider the derivation tree T of wu with a spine of type x that witnesses $x \in \theta(w)$. The spine induces an accepting run of $\mathcal{A}_x$. Thus, we build a derivation tree of G_x where all non-terminals and productions to the left of the spine are the same, all non-terminals and productions along the spine are annotated with the run of $\mathcal{A}_x$ and all non-terminals and productions to the right are replaced by their empty equivalent, e.g. A_ε. This induces a derivation tree of w in G_x as required.

E Lower Bounds on Automata Size

We mentioned in the conclusion the problem of whether the doubly-exponential size of the NFA built from a very degenerate context-free language must be doubly-exponential in the worst case. We have been unable to obtain this lower bound. Since, in Section 6, we construct a deterministic finite-automaton, one may ask whether a result of Meyer and Fischer [24] — that there is a deterministic PDS accepting a language whose corresponding deterministic finite automaton is doubly-exponential — can provide a lower bound in the deterministic case. Unfortunately, we provide a counter-example below. The language I_n given by Meyer and Fischer is described as follows[3]

> "I_n consists of words in $\{ 0, 1, a_1, \ldots, a_n \}^* \{ 0, 1 \}^{n-1}$ accepted by a deterministic push-down store machine which operates as follows:
> 1. Copy the input onto the store until input a_1 is encountered. If a_1 does not occur, reject the input.
> 2. Set $i = 2$.
> 3. If the next input is zero, pop the store until the first occurrence of a_i. If the next input is a one, pop the store to the second occurrence of a_i. If any other input is encountered, or the occurrences of a_i are not found, reject the input.
> 4. Increment i by one.
> 5. If $i \leq n$, repeat step 3.
> 6. If the digit on top of the store is 1 and there are no more input symbols, accept the input. Otherwise reject the input."

Intuitively, the input up to a_1 is interpreted as representing a binary tree in post-fix notation (although the PDS cannot enforce this with a small number of states, hence even "malformed" trees are accepted). After a_1, we see a sequence of 0s and 1s tracing a path in the tree. If this path ends on a node labelled by a 1, then we accept. Since there are doubly-exponential trees of depth n labelled at the leaves by 0 and 1, we get that the corresponding deterministic finite automaton must by doubly-exponential.

However, let $n = 3$ and consider the strong iterative pair

$$(x, y, z, t, u) = (0a_3 1a_3 a_2,\ 0a_3 0a_3 a_2,\ \varepsilon,\ 0a_3 1a_3 a_2,\ 0a_3 0a_3 a_2 a_1 10) \ .$$

In the following, we underline the part of the input identified by the suffix 10. For $i = 0$ we have $xy^i zt^i u = 0a_3\underline{1}a_3 a_2\ 0a_3 0a_3 a_2 a_1 10$ and for $i > 0$ we have

$$xy^i zt^i u = \ldots tu = \ldots 0a_3\underline{1}a_3 a_2\ 0a_3 0a_3 a_2 a_1 10 \ .$$

[3] In the original definition, the word finishes with $\{ 0, 1 \}^n$, though we believe this to be a mistake. After this correction, the size of the finite automaton is $2^{2^{n-1}}$. One could, of course, make other corrections to preserve the 2^{2^n} claimed.

In both cases one can verify membership in I_n.

However, consider $i = 1$ and $j = 0$. Then $xy^i zt^j u = \dots yu = \dots 0a_3\underline{0}a_3a_2\ 0a_30a_3a_2a_110$, which is not in I_n. Essentially, the sub-tree given by y violates the acceptance condition. When an occurrence of y necessitated an occurrence of t, the automaton would never read into y. However, when y and t are disconnected, y may not be "protected" by t.