Data Multi-Pushdown Automata*

Parosh Aziz Abdulla[1], C. Aiswarya[2], and Mohamed Faouzi Atig[3]

1 **Uppsala University, Uppsala, Sweden**
parosh@it.uu.se
2 **Chennai Mathematical Institute, Chennai, India**
aiswaryanitc@gmail.com
3 **Uppsala University, Uppsala, Sweden**
mohamed_faouzi.atig@it.uu.se

Abstract

We extend the classical model of multi-pushdown systems by considering systems that operate on a finite set of variables ranging over natural numbers. The conditions on variables are defined via gap-order constraints that allow to compare variables for equality, or to check that the gap between the values of two variables exceeds a given natural number. Furthermore, each message inside a stack is equipped with a data item representing its value. When a message is pushed to the stack, its value may be defined by a variable. When a message is popped, its value may be copied to a variable. Thus, we obtain a system that is infinite in multiple dimensions, namely we have a number of stacks that may contain an unbounded number of messages each of which is equipped with a natural number. It is well-known that the verification of any non-trivial property of multi-pushdown systems is undecidable, even for two stacks and for a finite data-domain. In this paper, we show the decidability of the reachability problem for the classes of data multi-pushdown system that admit a bounded split-width (or equivalently a bounded tree-width). As an immediate consequence, we obtain decidability for several subclasses of data multi-pushdown systems. These include systems with single stacks, restricted ordering policies on stack operations, bounded scope, bounded phase, and bounded context switches.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Pushdown Systems, Model-Checking, Gap-Order, Bounded Split-Width

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2017.38

1 Introduction

In the last few years, a lot of efforts have been devoted to the verification of *discrete* program models that have *infinite* state spaces such as Petri nets, lossy channel machines and multi-pushdown systems. In particular, multi-pushdown systems have been extensively studied as a natural model for concurrent Boolean recursive programs. Unfortunately, multi-pushdown systems are in general Turing powerful, and hence all basic decision problems are undecidable for them [31]. To overcome the undecidability barrier, several subclasses of multi-pushdown systems have been proposed (e.g., [16, 30, 10, 2, 25, 29, 26, 34, 11, 13, 12, 24, 6, 5, 33, 28]).

Bounded context-switch has been proposed in [30] as an adequate criterion for the verification of multi-pushdown systems. The idea is to restrict the analysis to executions that can be split into a given number of contexts where, in each context, pop and push operations are exclusive to one stack. The context-bounded reachability problem is NP-complete, though the state space that can be explored is still unbounded.

* This work was partially supported by DST Inspire and the Linnaeus centre of excellence UPMARC.

In [25], La Torre et al. generalize the notion of context into phase. A phase is a sequence of operations in which at most one stack can be popped, while there is no restriction on the push operations. The bounded-phase restriction considers only executions of the system that can be split into a given number of phases. In this case, the phase-bounded reachability problem is decidable in double exponential time.

Another generalization of bounded context-switch is bounded scope [26] which restricts the analysis of the multi-pushdown system to those executions in which the number of context-switches between any push operation and its corresponding pop operation is bounded by a given number. This definition extends the notion of contexts in term of coverage while being orthogonal to the notion of phase. In [26], the scope-bounded reachability problem is shown to be PSPACE-complete.

Another way to obtain decidability is to impose a linear order on stacks [16]. Stack operations are constrained in such a way that any pop operation is only allowed on the first non-empty stack. In [10], the reachability problem is shown to be 2ETIME-complete when assuming this ordering policy on stack operations. Furthermore, imposing such a restriction strictly extends the notion of phases while being orthogonal to scope-boundedness.

In [29, 21, 27], a unified technique to reason about multi-pushdown systems under such restrictions is presented. The idea is to see an execution as a graph with extra edges relating push operations and their corresponding pop operations. Then, the authors prove that the graphs generated under these restrictions have bounded split-width (or equivalently bounded tree-width). As an immediate consequence of Courcelle's theorem [20], the decidability of the reachability problem for multi-pushdown systems under these restrictions is obtained.

However, all these models assume a finite-state control, which means that the variables of the modelled programs are assumed to range over finite domains. Several extensions of (multi-)pushdown systems with data have been studied in the literature (see e.g., [14, 8, 1, 17, 22]). Most of these extensions concern the case of multi-pushdown systems with one stack except the work presented in [14] where an extension of multi-pushdown systems with data has been proposed. In order to obtain decidability of the reachability problem, the model requires the strong assumption of *data freshness*, and the restriction of the stack accesses to the bounded phase policy. Furthermore, the variable operations are restricted to checking (dis)equality.

In this paper, we consider an extension of multi-pushdown systems, which we call *Data Multi-Push-Down Automata* (Dmpda), that strengthens the classical model in two ways. First in addition to stacks, a Dmpda uses a finite set of variables ranging over the natural numbers. Moreover, each message inside the stack is equipped with a natural number which represents its value. Thus, we obtain a model that is possibly unbounded in multiple dimensions, namely we have a number of stacks such that each stack may contain an unbounded number of messages each of which is equipped with a natural number. The operations allowed on variables are defined by the *gap-order* constraint system [18, 32]. More precisely, Dmpda allow to compare the values of variables for equality, or to check that the gap between the values of two variables exceeds a given natural number. Also, a variable may be assigned a new arbitrary value, the value of another variable, or a value that is larger than at least a given natural number than the current value of another variable. Furthermore, a push operation may copy the value of a variable to the pushed message, and a pop operation may copy the value attached to the popped message to a variable. In this manner, the model of Dmpda subsumes two basic models, namely multi-pushdown systems (that we get by removing the variables and neglecting the values associated to the pushed messages) and the model of *integral relational automata* [18] (that we get by removing all the stacks).

Our main result is the decidability of the reachability problem for the classes of Dmpda

that admit a bounded split-width. To that aim, we solve a more general problem, namely we characterize the *reachability relation* on variables between each pair of control states. More precisely, we present an algorithm for computing a finite set of gap-order formulas whose denotations describe values of variables that allow to reach one state from another with empty stacks. The main ingredient of the algorithm is a symbolic representation, called *traces*, that encode certain transition sequences in the automaton. A trace represents a set of *partial runs*. A partial run does not record the contents of the stacks, but marks positions inside the run that correspond to matching push/pop operations. Furthermore, a partial run is not contiguous in the sense that it may contain a number of "holes". Our algorithm will characterize the relation between the variables at the points where the holes occur. In particular, a partial run with no holes corresponds to a concrete run that starts and ends with empty stacks. The definition of partial runs allows to extend naturally the notion of *split width* [21] that has been considered for the analysis of multi-pushdown systems (without data). Intuitively, a run has a bounded split width if it can be built from atomic runs by using a shuffle and a contraction operator without producing any intermediate runs with more holes than the given bound. An atomic run is one that consists either of a single transition, or a pair of matching push/pop transitions. We show that our algorithm is guaranteed to terminate for all classes of systems that generate runs with a bounded split width. As an immediate consequence, we obtain the decidability for several subclasses of multi-pushdown systems with data including the ones that restrict the ordering policy on stack operations, or bound the scope, the number of phases, or the number of context switches.

Related work. Several subclasses of multi-pushdown systems have been proposed in the literature including bounded-context [30], bounded-phase [25], bounded scope [26] and ordered multi-pushdown systems [16]. The reachability problem for these classes has been shown to be decidable under the assumption of finiteness of the set of control states. These classes are subclasses of our model Dmpda and our decidability result subsumes the decidability of the reachability problem for these models. In contrast, we do not provide any complexity results.

Split-width and tree-width[1] have been used for showing, in a unified way, the MSO decidability of several classes of multi-pushdown systems [29, 21, 27]. The method has been extended for message passing systems[6] and parameterized message passing systems[23]. However the considered models are restricted to the manipulation of variables over finite data domains while in our model, variables range over natural numbers. In fact the results presented in [29, 21, 27] are orthogonal to our result since we do not consider the model-checking problem against monadic second order logic.

Decidability of the reachability problem for pushdown systems (i.e., multi-pushdown systems with one stack) with data has been extensively studied in the literature (see e.g., [1, 17, 22, 3, 15, 19]). The closest work is pushdown systems with gap-order constraints [1], which is subsumed by our model. Furthermore, the techniques used to show the decidability of the reachability problem for pushdown systems with gap-order constrains are different from the ones used in this paper.

Extensions of multi-pushdown systems with data have been studied in [14]. This work uses the strong assumption of freshness of data, and bounded phase restriction on stack accesses. In contrast, we do not assume the freshness of data and our results can be applied to several subclass of multi-pushdown systems.

[1] Split-width and tree-width are not identical, but one is bounded if and only if the other is. Further the bounds are related linearly [21].

In [8] the split-width technique is lifted to analyze timed multi-pushdown systems. Timed systems give rise to an infinite data domain. However, reachability in this case can be reduced to MSO model checking of untimed systems with finite propositional labelling indicating timing constraints, since realizability of a word with timing constraints can be expressed in MSO [7]. The crux of the decidability proof in all these cases is the use of tree-automata. In contrast, the reachability problem of DMPDA under bounded split-width does not reduce to the Boolean case. Furthermore, our algorithm uses a fix-point computation which terminates, thanks to well quasi-ordering of gap-order formulas.

2 Preliminaries

Let $\mathbb{N}$ denote the set of natural numbers. For sets A and B, we use $f : A \to B$ to denote that f is a function from A to B. We use $f[a \leftarrow a']$ to denote the function f' such that $f'(a) = a'$, and $f'(x) = f(x)$ if $x \neq a$. For $A' \subseteq A$, we use $f \odot A'$ to denote the restriction of f to A'. For sets A_1 and A_2 with $A_1 \cap A_2 = \varnothing$, and functions $f_1 : A_1 \to B$ and $f_2 : A_2 \to B$, we use $f_1 \cup f_2 : A_1 \cup A_2 \to B$ to denote the function g such that $g(a) = f_1(a)$ if $a \in A_1$ and $g(a) = f_2(a)$ if $a \in A_2$. For a finite set A, we use $|A|$ to denote the size of A.

For a set A, we use A^* to denote the set of finite words over A. We use ϵ to denote the empty word. For $w_1, w_2 \in A^*$, we use $w_1 \cdot w_2$ to denote the concatenation of w_1 and w_2.

Consider a set A and a total ordering $\preceq$ on A. We use $a_1 \prec a_2$ to denote that $a_1 \preceq a_2$ and $a_1 \neq a_2$. We use $\prec$ to denote the induced immediate successor relation, i.e., $a_1 \prec a_2$ iff $a_1 \prec a_2$ and there is no a_3 such that $a_1 \prec a_3 \prec a_2$.

3 Model

In this section, we introduce *Data Multi-Pushdown Automata* (DMPDA). A DMPDA operates on *multiple* unbounded stacks each of which allows pushing (appending) and popping (removing) messages in a last-in-first-out manner. In addition to the stacks, a DMPDA uses a finite set of variables ranging over the natural numbers.

The allowed operations on variables are defined by the *gap-order* constraint system [18, 32]. More precisely, the model allows non-deterministic value assignment, copying the value of one variable to another, and assignment of a value v to some variable such that v is larger than at least a given natural number than the current value of another variable. The transitions may be conditioned by tests that compare the values of two variables for equality, or that give the smallest allowed gap between two variables. In addition to carrying a name (taken from a finite alphabet), each message inside a stack is equipped by a natural number that represents its "value". A *push* operation may copy the value of a variable to the pushed message, and a *pop* operation may copy the value of the popped message to a variable. Notice that DMPDA extend the classical model of Push-Down Automata in three ways, namely they allow

(i) multiple stacks,
(ii) numerical variables, and
(iii) an infinite (numerical) stack alphabet.

Syntax

In the rest of the paper, we assume a finite set of variables $\mathbb{X}$, a finite set of stacks Σ, and a finite stack alphabet Γ. A DMPDA $\mathcal{A}$ is a tuple $\langle Q, \Delta \rangle$ where Q is a finite set of states, and Δ is a finite set of transitions. A transition $t \in \Delta$ is a triple $\langle q_1, op, q_2 \rangle$ where $q_1, q_2 \in Q$ are states, and op is an operation of one of the following forms:

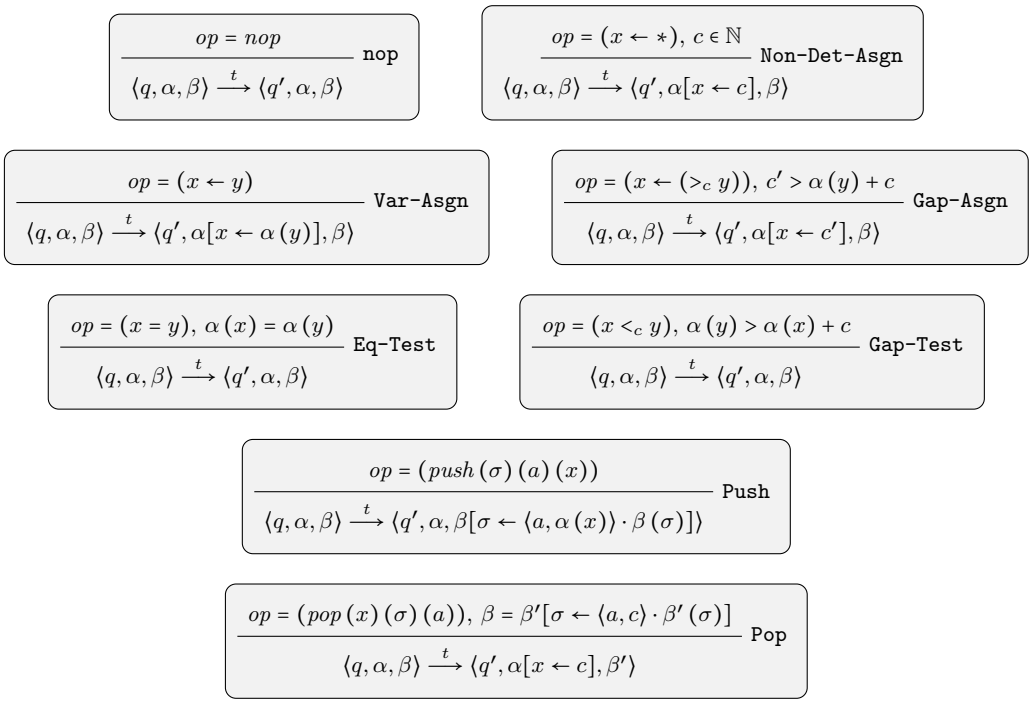

Figure 1 Inference rules defining the relation $\xrightarrow{t}$, where $t = \langle q, op, q'\rangle$.

(i) *nop* is the *empty* operation that does not change the values of the variables or the contents of the stacks.
(ii) $x \leftarrow *$ assigns non-deterministically an arbitrary value in $\mathbb{N}$ to the variable x.
(iii) $x \leftarrow y$ copies the value of variable y to x.
(iv) $x \leftarrow (>_c y)$ assigns non-deterministically to x a value that exceeds the current value of y by c (so the new value of x is $> y + c$).
(v) $x = y$ checks whether the value of x is equal to the value of y.
(vi) $x <_c y$ checks whether the gap between the values of y and x is larger than c.
(vii) $push(\sigma)(a)(x)$ pushes the symbol $a \in \Gamma$ to the stack $\sigma \in \Sigma$ and assigns to it the value of the variable x.
(viii) $pop(x)(\sigma)(a)$ pops the symbol $a \in \Gamma$ (if a is the top-most symbol at the stack $\sigma \in \Sigma$) and assigns its value to the variable x.

We define the *source* $\mathtt{src}(t) := q_1$ and the *target* $\mathtt{tgt}(t) := q_2$.

We define $\Delta^{\mathtt{intern}}$ to be the set of *internal* transitions, i.e., those that do not perform push or pop operations. We define $\Delta^{\mathtt{push}}_{\sigma,a}$ to be the set of transitions whose operations are of the form $push(\sigma)(a)(x)$ for some $x \in \mathbb{X}$. We define $\Delta^{\mathtt{push}}_{\sigma} := \cup_{a\in\Gamma}\Delta^{\mathtt{push}}_{\sigma,a}$. We define $\Delta^{\mathtt{pop}}_{\sigma,a}$ and $\Delta^{\mathtt{pop}}_{\sigma}$ analogously.

Semantics

A DMPDA induces a transition system as follows. A *configuration* $\mathbf{c}$ is a triple $\langle q,\alpha,\beta\rangle$ where $q \in Q$ is a state, $\alpha : \mathbb{X} \to \mathbb{N}$ defines the values of the variables, and $\beta : \Sigma \to (\Gamma \times \mathbb{N})^*$ defines, for each stack $\sigma \in \Sigma$, its content $\beta(\sigma)$. The content of a stack is a word whose elements are of the form $\langle a, c\rangle$ where a is a symbol and c is its value. In particular, we define β_ϵ such that $\beta_\epsilon(\sigma) = \epsilon$ for all $\sigma \in \Sigma$. We say that $\mathbf{c}$ is *plain* if $\beta = \beta_\epsilon$.

We define the transition relation $\longrightarrow := \cup_{t \in \Delta} \xrightarrow{t}$ on the set of configurations, where $\xrightarrow{t}$ describes the effect of the transition t. The semantics of the transition relation is presented through the inference rules of Fig. 1, explained below one by one.

- *nop*. The values of the variables and the stack contents are not changed.
- $x \leftarrow *$. The value of the variable x is changed non-deterministically to some natural number. The values of the other variables and the stack contents are not changed.
- $x \leftarrow y$. The value of the variable y is copied to the variable x. The values of the other variables and the stack contents are not changed.
- $x \leftarrow (>_c y)$. The variable x is assigned non-deterministically a value that exceeds the value of y by c. The values of the other variables and the stack contents are not changed.
- $x = y$ (resp. $x <_c y$). The transition is only enabled if the value of y is equal to the value of x (resp. larger than the value of x by more than c). The values of the variables and the stack contents are not changed.

- $push\,(\sigma)\,(a)\,(x)$. The symbol a is pushed onto the stack σ with a value equal to that of x.
- $pop\,(x)\,(\sigma)\,(a)$. The symbol a is popped from the stack σ (if it is the top-most symbol of σ), and its value is copied to the variable x.

We use $\xrightarrow{*}$ to denote the reflexive transitive closure of $\longrightarrow$. A *run* π is an alternating sequence $\mathbf{c}_0 t_1 \mathbf{c}_1 \cdots \mathbf{c}_{n-1} t_n \mathbf{c}_n$ of configurations and transitions such that $\mathbf{c}_{i-1} \xrightarrow{t_i} \mathbf{c}_i$ for all $i : 1 \leq i \leq n$. We say that π is *plain* if $\mathbf{c}_0$ and $\mathbf{c}_n$ are plain. For configurations $\mathbf{c}$ and $\mathbf{c}'$, we write $\mathbf{c} \xrightarrow{\pi} \mathbf{c}'$ to denote that there is a run π of the above form such that $\mathbf{c}_0 = \mathbf{c}$ and $\mathbf{c}_n = \mathbf{c}'$. Notice that $\mathbf{c} \xrightarrow{*} \mathbf{c}'$ iff $\mathbf{c} \xrightarrow{\pi} \mathbf{c}'$ for some run π.

Reachability

In the *Reachability Problem*, we are given two plain configurations $\mathbf{c}_1$ and $\mathbf{c}_2$, and are asked whether $\mathbf{c}_1 \xrightarrow{*} \mathbf{c}_2$. In order to solve the reachability problem we will study *reachability relations*. For states $q, q' \in Q$, we define $\mathbb{R}\,(q, q') := \left\{ \langle \alpha, \alpha' \rangle \mid \langle q, \alpha, \beta_\epsilon \rangle \xrightarrow{*} \langle q', \alpha', \beta_\epsilon \rangle \right\}$. Intuitively, we summarize the values of the variables that allow us to move from q to q', starting and ending with empty stacks. More precisely, $\mathbb{R}\,(q, q')$ contains all pairs $\langle \alpha, \alpha' \rangle$ such that we can start from a configuration where the state is q, the values of the variables are given by α, and the stacks are empty to another configurations where the state is q', the values of the variables are given by α', and the stacks are empty again.

4 Gap-Order Formulas

Fix a set $\mathbb{X}$ of variables ranging over $\mathbb{N}$. An *atomic gap-order formula* over $\mathbb{X}$ is either of the form $x = y$ or of the form $x <_c y$ where $x, y \in \mathbb{X}$ and $c \in \mathbb{N}$. A *gap-order formula* ϕ over $\mathbb{X}$ is a conjunction of atomic constraints over $\mathbb{X}$. Sometimes, we represent ϕ as a set (containing all its conjuncts). For a function $\mathtt{Val} : \mathbb{X} \to \mathbb{N}$, we write (as expected) $\mathtt{Val} \vDash \phi$ to denote that $\mathtt{Val}$ satisfies ϕ. We will also consider existentially quantified formulas of the form $\exists \mathbb{Y}.\ \phi$ where ϕ is a gap-order formula over $\mathbb{X}$, and $\mathbb{Y} \subseteq \mathbb{X}$. For $\mathtt{Val} : \mathbb{X} - \mathbb{Y} \to \mathbb{N}$, we write $\mathtt{Val} \vDash \exists \mathbb{Y}.\ \phi$ to denote that there is a mapping $\mathtt{Val}' : \mathbb{Y} \to \mathbb{N}$ such that $\mathtt{Val} \cup \mathtt{Val}' \vDash \phi$. For a (quantified) gap-order formula ϕ, we define its denotation $[\![\phi]\!] := \{\mathtt{Val} \mid \mathtt{Val} \vDash \phi\}$.

A gap-order formula ϕ over $\mathbb{X}$ is in *normal form* if it satisfies the following conditions:

1. If $(x <_{c_1} y) \in \phi$ and $(y <_{c_2} z) \in \phi$ then $(x <_{c_3} z) \in \phi$ for some c_3 with $c_1 + c_2 < c_3$.
2. If $(x <_c y) \in \phi$ and $(y = z) \in \phi$ (or $(z = y) \in \phi$) then $(x <_c z) \in \phi$.

3. If $(x <_c y) \in \phi$ and $(x = z) \in \phi$ (or $(z = x) \in \phi$) then $(z <_c y) \in \phi$.
4. If $(x = y) \in \phi$ and $(y = z) \in \phi$ then $(x = z) \in \phi$ or $(z = x) \in \phi$.
5. For each $x, y \in \mathbb{X}$, there is at most one conjunct in ϕ containing both x and y.

▸ **Lemma 1** ([4, 1]). *For each gap-order formula ϕ, we can effectively compute a gap-order formula ϕ' such that ϕ' is in normal form and $[\![\phi']\!] = [\![\phi]\!]$.*

We obtain ϕ' from ϕ by repeatedly adding conjuncts to maintain properties 1-4 and removing conjuncts which violate property 5 (for instance, if we have both $(x <_{c_1} y) \in \phi$ and $(x <_{c_2} y) \in \phi$, with $c_1 \leq c_2$, then we can remove the former conjunct.) Normalization can be used to check consistency: the formula is consistent iff no inequalities of the form $x <_c x$ are generated.

Furthermore, we can use normalization to perform quantifier elimination as follows. For sets of variables $\mathbb{Y} \subseteq \mathbb{X}$ and a gap-order formula ϕ, let $\phi \ominus \mathbb{Y}$ to be the gap-order formula we get from ϕ by eliminating all conjuncts in which a variable $x \in \mathbb{Y}$ occurs.

▸ **Lemma 2** ([4, 1]). *Suppose that ϕ is consistent and in normal form. Assume that $\mathtt{Val} \models \phi \ominus \mathbb{Y}$. Then there is a $\mathtt{Val}' : \mathbb{Y} \to \mathbb{N}$ such that $\mathtt{Val} \cup \mathtt{Val}' \models \phi$*

From Lemma 2 we get the following corollary.

▸ **Corollary 3.** *Suppose that ϕ is consistent and in normal form. Then $[\![\exists \mathbb{Y}.\ \phi]\!] = [\![\phi \ominus \mathbb{Y}]\!]$.*

We write $\phi_1 \sqsubseteq \phi_2$ to denote that $[\![\phi_2]\!] \subseteq [\![\phi_1]\!]$. We can check $\phi_1 \sqsubseteq \phi_2$ as follows. By Lemma 1 we can assume that ϕ_1 and ϕ_2 are in normal form. Then the following conditions should be satisfied: (1) If $(x <_{c_1} y) \in \phi_1$ then $(x <_{c_2} y) \in \phi_2$ for some $c_2 \geq c_1$, and (2) if $(x = y) \in \phi_1$ then $(x = y) \in \phi_2$ or $(y = x) \in \phi_2$.

5 Traces

We introduce a data structure, called *traces*, that encode *partial runs*. Roughly speaking, a partial run is a run with a number of "holes" inserted. Thus, a partial run consists of a sequence of *segments* each of which is a sequence of consecutive transitions. The source of one transition in a segment is identical to the target of the preceding transition. Furthermore, each push operation in the partial run is matched by a pop operation, and vice versa, such that the push/pop operations respect the stack semantics (i.e., we do not allow push/pop transitions that are "pending" in a partial run). First, we define the set of atomic traces, and describe two operations that allow to build new traces from existing ones. Then, we define an entailment relation on traces. Finally, we use traces to define the notion of *split width*. In the rest of the section, we fix a DMPDA $\mathcal{A} = \langle Q, \Delta \rangle$.

5.1 Definition

A trace τ is a tuple $\langle \mathbb{I}, \preceq, \mathtt{src}, \mathtt{tgt}, E, \phi \rangle$ defined as follows.

- $\mathbb{I}$ is a finite (index) set. Each index will be used to represent the summary of a segment. The summary is given by the starting and the end states of each segment, i.e., the source of the first transition and the target of the last transition in the segment, and by a relation on the values of the variables before and after performing the different segments. For each variable $x \in \mathbb{X}$ and index $\mathtt{i} \in \mathbb{I}$, we will introduce two new variables $x^{\mathtt{i}}_{\mathtt{s}}$ and $x^{\mathtt{i}}_{\mathtt{t}}$ representing the *source* and *target* values of x, i.e., the value of x at the start and at the end of the segment represented by the index $\mathtt{i}$. We define the set $\mathbb{X}^{\mathtt{i}} := \{x^{\mathtt{i}}_{\mathtt{s}} | (x \in \mathbb{X}) \wedge (\mathtt{i} \in \mathbb{I})\} \cup \{x^{\mathtt{i}}_{\mathtt{t}} | (x \in \mathbb{X}) \wedge (\mathtt{i} \in \mathbb{I})\}$, and define $\mathbb{X}^{\mathbb{I}} := \cup_{\mathtt{i} \in \mathbb{I}} \mathbb{X}^{\mathtt{i}}$.

- $\preceq$ is a total ordering on $\mathbb{I}$ that gives the order in which the segments represented by the indices are performed. We let $\prec$ be the induced immediate successor relation (cf. Section 2).
- $\mathtt{src} : \mathbb{I} \to Q$ maps each index to a state representing the source of the corresponding segment, i.e., the state from which the segment starts (the source of the first transition in the segment). Analogously, $\mathtt{tgt} : \mathbb{I} \to Q$ defines the target of the segment, i.e., the state at the end of the segment (the target of the last transition in the segment).
- $E : \mathbb{I} \times \mathbb{I} \to 2^{\Sigma}$ is a function representing an "edge relation" between the indices. For indices $\mathtt{i}_1, \mathtt{i}_2$ the value of $E(\mathtt{i}_1, \mathtt{i}_2)$ gives the set of stacks such that there is a push operation in the segment represented by $\mathtt{i}_1$ whose corresponding pop operation is performed in the segment represented by $\mathtt{i}_2$. We impose two conditions on E. First, we require that $E(\mathtt{i}_1, \mathtt{i}_2) \neq \varnothing$ only if $\mathtt{i}_1 \prec \mathtt{i}_2$ since a pop operation can only occur after the corresponding push operation (and furthermore, we do not record push operations whose pop operations lie in the same segment). Second, we require that, for all stacks $\sigma \in \Sigma$, there are no indices $\mathtt{i}_1 \prec \mathtt{i}_2 \prec \mathtt{i}_3 \prec \mathtt{i}_4$ such that $\sigma \in E(\mathtt{i}_1, \mathtt{i}_3) \cap E(\mathtt{i}_2, \mathtt{i}_4)$. This condition ensures that we are consistent with the stack semantics since there is no overlap between two pairs of push/pop operations on the same stack.
- ϕ is a gap-order formula over the set $\mathbb{X}^{\mathbb{I}}$, that defines the relation on values of the variables at the start and the end of the different segments.

We will equate traces that are equivalent modulo the renaming of the indices. For a trace $\tau = \langle \mathbb{I}, \preceq, \mathtt{src}, \mathtt{tgt}, E, \phi \rangle$, we define its *degree* $\#\tau = |\mathbb{I}|$, i.e., it is the size of the index set.

5.2 Atomic Traces

Atomic traces are built using the set of transitions. We will define two types of atomic traces, namely those induced by single internal transitions, and those that are induced by pairs of matching push/pop transitions.

Internal Transitions

Let $t = \langle q_1, op, q_2 \rangle \in \Delta^{\mathtt{intern}}$ be an internal transition. We will build a trace with a single index corresponding to a single segment which contains only one transition, namely t. Formally, we define $\mathtt{MkTrace}(t) := \langle \mathbb{I}, \preceq, \mathtt{src}, \mathtt{tgt}, E, \phi \rangle$ where

- $\mathbb{I} = \{\mathtt{i}\}$, i.e., the set of indices is a singleton.
- $\preceq$ is trivial since the index set contains only one node.
- $\mathtt{src}(\mathtt{i}) = q_1$ and $\mathtt{tgt}(\mathtt{i}) = q_2$, i.e., we label the single index with the source and target states of t.
- $E = \varnothing$ reflecting the fact that the operation performed by t does not affect the stacks.
- ϕ consists of all conjuncts of the following forms:
  - if $op = nop$ or $op = (x = y)$ or $op = (x <_c y)$ then $x^{\mathtt{i}}_{\mathtt{s}} = x^{\mathtt{i}}_{\mathtt{t}}$ for all $x \in \mathbb{X}$, i.e., the values of the variables are not changed during t.
  - if $op = (x = y)$ then $x^{\mathtt{i}}_{\mathtt{s}} = y^{\mathtt{i}}_{\mathtt{s}}$, and if $op = (x <_c y)$ then $y^{\mathtt{i}}_{\mathtt{s}} > x^{\mathtt{i}}_{\mathtt{s}} + c$. The values of the variables should satisfy the condition of the transition.
  - If $op = (x \leftarrow *)$ or $op = (x \leftarrow y)$ or $op = (x \leftarrow (>_c y))$ then $z^{\mathtt{i}}_{\mathtt{s}} = z^{\mathtt{i}}_{\mathtt{t}}$ for all $z \in \mathbb{X} - \{x\}$, i.e., the values of the variables different from x are not changed during t.
  - If $op = (x \leftarrow y)$ then $x^{\mathtt{i}}_{\mathtt{t}} = y^{\mathtt{i}}_{\mathtt{s}}$.
  - If $op = (x \leftarrow (>_c y))$ then $x^{\mathtt{i}}_{\mathtt{t}} > y^{\mathtt{i}}_{\mathtt{s}} + c$.

Stack Transitions

Consider two transitions $t_1 = \langle q_1, op_1, q_2\rangle \in \Delta^{\texttt{push}}_{\sigma,a}$, $t_2 = \langle q_3, op_2, q_4\rangle \in \Delta^{\texttt{pop}}_{\sigma,a}$ where $op_1 = push(\sigma)(a)(x)$ and $op_2 = pop(y)(\sigma)(a)$. Notice that the two transitions push/pop the same symbol a to/from the same stack σ. We will build a trace with two indices corresponding to two segments containing t_1 and t_2, respectively. Formally, we define $\texttt{MkTrace}(t_1, t_2) := \langle \mathbb{I}, \preceq, \texttt{src}, \texttt{tgt}, E, \phi\rangle$, where:

- $\mathbb{I} = \{\mathtt{i}_1, \mathtt{i}_2\}$ i.e., the index set contains two elements. We use the indices $\mathtt{i}_1$ and $\mathtt{i}_2$ to represent two segments each containing a single transition, namely t_1 and t_2 respectively.
- $\mathtt{i}_1 \preceq \mathtt{i}_2$. We require that $\mathtt{i}_1$ is ordered before $\mathtt{i}_2$. This reflects the fact that a pop transition occurs after the matching push transition.
- $E(\mathtt{i}_1, \mathtt{i}_2) = \{\sigma\}$, i.e., we add an edge between $\mathtt{i}_1$ to $\mathtt{i}_2$ labeled with σ corresponding to the matching push/pop operations on σ performed by t_1 resp. t_2.
- $\texttt{src}(\mathtt{i}_1) = q_1$, $\texttt{tgt}(\mathtt{i}_1) = q_2$, $\texttt{src}(\mathtt{i}_2) = q_3$, and $\texttt{tgt}(\mathtt{i}_2) = q_4$. In other words, we label the new indices with the source and target states of t_1 resp. t_2.
- ϕ consists of all conjuncts of the following forms:
  - **(i)** $z^{\mathtt{i}_1}_{\mathtt{s}} = z^{\mathtt{i}_1}_{\mathtt{t}}$ for all $z \in \mathbb{X}$, i.e., the values of the variables are not changed during t_1.
  - **(ii)** $z^{\mathtt{i}_2}_{\mathtt{s}} = z^{\mathtt{i}_2}_{\mathtt{t}}$ for all $z \in \mathbb{X} - \{y\}$, i.e., the values of the variables, except y, are not changed during t_2.
  - **(iii)** $y^{\mathtt{i}_2}_{\mathtt{t}} = x^{\mathtt{i}_1}_{\mathtt{s}}$.

  This condition corresponds to the fact that the value of a when pushed to the stack during t_1 is equal to the value of variable x. This value is identical to the value stored in y after the pop operation of transition t_2.

5.3 Operations

We define two operations for building new traces, namely shuffling and contraction.

Shuffling

Consider two traces $\tau_1 = \langle \mathbb{I}_1, \preceq_1, \texttt{src}_1, \texttt{tgt}_1, E_1, \phi_1\rangle$, and $\tau_2 = \langle \mathbb{I}_2, \preceq_2, \texttt{src}_2, \texttt{tgt}_2, E_2, \phi_2\rangle$, where $\mathbb{I}_1 \cap \mathbb{I}_2 = \varnothing$. We will build a new trace by shuffling the index sets of τ_1 and τ_2. We define $\tau_1 \otimes \tau_2$ to be the set of traces of the form $\langle \mathbb{I}, \preceq, \texttt{src}, \texttt{tgt}, E, \phi\rangle$ satisfying the following conditions:

- $\mathbb{I} = \mathbb{I}_1 \cup \mathbb{I}_2$, i.e., the new trace contains exactly all the segments that are in τ_1 and τ_2.
- $\preceq$ is a total ordering on $\mathbb{I}$ such that $\preceq_1 \subseteq \preceq$ and $\preceq_2 \subseteq \preceq$. We do not change the original orderings of the indices, but we do not constrain the places of the two sets of indices relative to each other.
- $\texttt{src}(\mathtt{i}) = \texttt{src}_1(\mathtt{i})$, and $\texttt{tgt}(\mathtt{i}) = \texttt{tgt}_1(\mathtt{i})$ for all $\mathtt{i} \in \mathbb{I}_1$. Furthermore, $\texttt{src}(\mathtt{i}) = \texttt{src}_2(\mathtt{i})$, and $\texttt{tgt}(\mathtt{i}) = \texttt{tgt}_2(\mathtt{i})$ for all $\mathtt{i} \in \mathbb{I}_2$. (We keep the state labelings of the old indices.)
- $E(\mathtt{i}_1, \mathtt{i}_2) = E_1(\mathtt{i}_1, \mathtt{i}_2)$ if $\mathtt{i}_1, \mathtt{i}_2 \in \mathbb{I}_1$, and $E(\mathtt{i}_1, \mathtt{i}_2) = E_2(\mathtt{i}_1, \mathtt{i}_2)$ if $\mathtt{i}_1, \mathtt{i}_2 \in \mathbb{I}_2$, i.e., all the edges in τ_1 and τ_2 are maintained in $\tau_1 \otimes \tau_2$. Also, $E(\mathtt{i}_1, \mathtt{i}_2) = \varnothing$, if $\mathtt{i}_1 \in \mathbb{I}_1$ and $\mathtt{i}_2 \in \mathbb{I}_2$ or if $\mathtt{i}_1 \in \mathbb{I}_2$ and $\mathtt{i}_2 \in \mathbb{I}_1$, i.e., we do not add any edges between the two sets of indices. Furthermore, we require that there are no $\mathtt{i}_1, \mathtt{i}_2 \in \mathbb{I}_1$ and $\mathtt{i}_3, \mathtt{i}_4 \in \mathbb{I}_2$ such that
  - **(i)** $\sigma \in E(\mathtt{i}_1, \mathtt{i}_2)$,
  - **(ii)** $\sigma \in E(\mathtt{i}_3, \mathtt{i}_4)$, and
  - **(iii)** either $\mathtt{i}_1 \prec \mathtt{i}_3 \prec \mathtt{i}_2 \prec \mathtt{i}_4$ or $\mathtt{i}_3 \prec \mathtt{i}_1 \prec \mathtt{i}_4 \prec \mathtt{i}_2$.

  This is to ensure that $\tau_1 \otimes \tau_2$ respects the stack semantics.
- $\phi = \phi_1 \wedge \phi_2$. Notice that the values of the variables indexed by elements from $\mathbb{I}_1$ are not related to the values of the variables indexed by elements from $\mathbb{I}_2$.

Contraction

We define a *contraction operation* $\downarrow$ on indices that represents merging the corresponding segments. Consider $\mathtt{i}_1, \mathtt{i}_2 \in \mathbb{I}$ such that $\mathtt{i}_1 \lessdot \mathtt{i}_2$, i.e., $\mathtt{i}_2$ is the immediate successor of $\mathtt{i}_1$. Let $\mathtt{src}(\mathtt{i}_1) = q_1$, $\mathtt{tgt}(\mathtt{i}_1) = q_2$, $\mathtt{src}(\mathtt{i}_2) = q_2$, and $\mathtt{tgt}(\mathtt{i}_1) = q_3$, i.e., the target state of $\mathtt{i}_1$ is identical to the source state of $\mathtt{i}_2$. We will merge $\mathtt{i}_1$ and $\mathtt{i}_2$ to a new (single) index $\mathtt{j}$. We define $\tau \downarrow \langle \mathtt{i}_1, \mathtt{i}_2 \rangle := \langle \mathbb{I}', \preceq', \mathtt{src}', \mathtt{tgt}', E', \phi' \rangle$ as follows:

- $\mathbb{I}' = \mathbb{I} - \{\mathtt{i}_1, \mathtt{i}_2\} \cup \{\mathtt{j}\}$ where $\mathtt{j} \notin \mathbb{I}$, i.e., we replace the two merged indices by a new one.
- $\mathtt{k} \preceq' \mathtt{j}$ iff $\mathtt{k} \preceq \mathtt{i}_1$, and $\mathtt{j} \preceq' \mathtt{k}$ iff $\mathtt{i}_2 \preceq \mathtt{k}$ for all $\mathtt{k} \in \mathbb{I} - \{\mathtt{i}_1, \mathtt{i}_2\}$, i.e., in the new ordering, the new index $\mathtt{j}$ will take the places of the two (consecutive) indices $\mathtt{i}_1$ and $\mathtt{i}_2$. Furthermore, $\mathtt{k}_1 \preceq' \mathtt{k}_2$ iff $\mathtt{k}_1 \preceq \mathtt{k}_2$ for all $\mathtt{k}_1, \mathtt{k}_2 \in \mathbb{I} - \{\mathtt{i}_1, \mathtt{i}_2\}$, i.e., the relative orderings of the original indices are not changed.
- $\mathtt{src}'(\mathtt{j}) = q_1$, $\mathtt{tgt}'(\mathtt{j}) = q_3$, $\mathtt{src}'(\mathtt{k}) = \mathtt{src}(\mathtt{k})$ and $\mathtt{tgt}'(\mathtt{k}) = \mathtt{tgt}(\mathtt{k})$ for all $\mathtt{k} \in \mathbb{I} - \{\mathtt{i}_1, \mathtt{i}_2\}$. In other words, we keep the state labelings of the old indices, while we take the source and target states of $\mathtt{j}$ to be the source state of $\mathtt{i}_1$ and the target state of $\mathtt{i}_2$ respectively.
- $E'(\mathtt{j}, \mathtt{k}) = E(\mathtt{i}_1, \mathtt{k}) \cup E(\mathtt{i}_2, \mathtt{k})$ and $E'(\mathtt{k}, \mathtt{j}) = E(\mathtt{k}, \mathtt{i}_1) \cup E(\mathtt{k}, \mathtt{i}_2)$ for all $\mathtt{k} \in \mathbb{I} - \{\mathtt{i}_1, \mathtt{i}_2\}$, i.e., we merge the edges originating from/to the two merged indices. Also, $E'(\mathtt{k}_1, \mathtt{k}_2) = E(\mathtt{k}_1, \mathtt{k}_2)$ for all $\mathtt{k}_1, \mathtt{k}_2 \in \mathbb{I} - \{\mathtt{i}_1, \mathtt{i}_2\}$, i.e., the edges to/from the other indices are maintained. Notice that any edges between $\mathtt{i}_1$ and $\mathtt{i}_2$ are deleted.
- $\phi' = \exists \left(\mathbb{X}^{\mathtt{i}_1} \cup \mathbb{X}^{\mathtt{i}_2}\right). \phi''$, where ϕ'' is defined as $\phi \wedge \left(\bigwedge_{x \in \mathbb{X}} \left(x_{\mathtt{t}}^{\mathtt{i}_1} = x_{\mathtt{s}}^{\mathtt{i}_2}\right)\right) \wedge \left(\bigwedge_{x \in \mathbb{X}} \left(x_{\mathtt{s}}^{\mathtt{i}_1} = x_{\mathtt{s}}^{\mathtt{j}}\right)\right) \wedge \left(\bigwedge_{x \in \mathbb{X}} \left(x_{\mathtt{t}}^{\mathtt{i}_2} = x_{\mathtt{t}}^{\mathtt{j}}\right)\right)$. We require that ϕ' is consistent. Since we merge the two segments corresponding to $\mathtt{i}_1$ and $\mathtt{i}_2$, we require the values of the variables at the end of the first segment to be consistent with the values of the variables at the start of the second segment. If these conditions are not satisfied, then the resulting formula ϕ' will not be consistent. Furthermore, the values of variables at the start of the new segment are defined to be the values of the variables at the start of the segment corresponding to $\mathtt{i}_1$. Analogously, the values of variables at the end of the new segment are defined to be the values of the variables at the end of the segment corresponding to $\mathtt{i}_2$. By Lemma 1, we know that ϕ'' can be transformed to an equivalent formula in normal form, and hence by Corollary 3, we can compute ϕ' as a gap-order formula.

We define $\tau \downarrow$ to be the set of traces τ' such that: (1) there are indices $\mathtt{i}_1, \mathtt{i}_2 \in \mathbb{I}$ with $\mathtt{i}_1 \lessdot \mathtt{i}_2$, (2) $\mathtt{tgt}(\mathtt{i}_1) = \mathtt{src}(\mathtt{i}_2)$, and (3) $\tau' = \tau \downarrow \langle \mathtt{i}_1, \mathtt{i}_2 \rangle$.

5.4 Entailment

We define an entailment relation $\sqsubseteq$ on traces. Intuitively, a trace τ_1 is *weaker* than a trace τ_2, denoted $\tau_1 \sqsubseteq \tau_2$, if their graphs are isomorphic (equivalent up to the renaming of indices), but the gap-order formula of τ_1 is weaker than the one of τ_2. Later in the paper, when we compute reachability relations, we let τ_1 "subsume" τ_2 in the sense that if we encounter both τ_1 and τ_2 then we only include τ_1 (and discard τ_2), without suffering any loss of any precision in the analysis. Formally, consider traces $\tau_1 = \langle \mathbb{I}_1, \preceq_1, \mathtt{src}_1, \mathtt{tgt}_1, E_1, \phi_1 \rangle$ and $\tau_2 = \langle \mathbb{I}_2, \preceq_2, \mathtt{src}_2, \mathtt{tgt}_2, E_2, \phi_2 \rangle$. Let $h : \mathbb{I}_1 \to \mathbb{I}_2$ be a bijection. We write $\tau_1 \sqsubseteq_h \tau_2$ to denote that the following conditions are satisfied: (1) $\mathtt{i}_1 \preceq_1 \mathtt{i}_2$ iff $h(\mathtt{i}_1) \preceq_2 h(\mathtt{i}_2)$, (2) $\mathtt{src}_1(\mathtt{i}) = \mathtt{src}_2(h(\mathtt{i}))$ and $\mathtt{tgt}_1(\mathtt{i}) = \mathtt{tgt}_2(h(\mathtt{i}))$, (3) $E_1(\mathtt{i}_1, \mathtt{i}_2) = E_2(h(\mathtt{i}_1), h(\mathtt{i}_2))$, and (4) $\phi_1^h \sqsubseteq \phi_2$. We obtain ϕ_1^h from ϕ_1 by replacing each $x_{\mathtt{s}}^{\mathtt{i}}$ by $x_{\mathtt{s}}^{h(\mathtt{i})}$ and replacing each $x_{\mathtt{t}}^{\mathtt{i}}$ by $x_{\mathtt{t}}^{h(\mathtt{i})}$. We use $\tau_1 \sqsubseteq \tau_2$ to denote that $\tau_1 \sqsubseteq_h \tau_2$ for some h.

5.5 Split Width

We say that a trace has *split width* θ if it can be derived by starting from the atomic traces and repeatedly applying the shuffling and contraction operations without letting any of the intermediately generated traces have a degree larger than θ.

A *proof tree* $\mathcal{T}$ of split width θ is a binary tree whose nodes are labeled with traces such that the following conditions are satisfied:

- The leaves are labeled with atomic traces.
- If an internal node, labeled with τ, has two children then the children are labeled with traces τ_1 and τ_2 such that $\tau \in \tau_1 \otimes \tau_2$.
- If an internal node, labeled with τ', has a single child then that child is labeled with a trace τ such that $\tau \in \tau' \downarrow$.
- For each label τ of a node in $\mathcal{T}$, we have $\#\tau \leq \theta$.

A trace τ is of *split width* θ if θ is the smallest number such that τ is the root of a proof tree of split width θ. We use $\mathtt{SW}(\tau)$ to denote the split width of τ.

We extend the notion of split width to plain runs as follows. Consider a plain run $\pi = \mathbf{c}_0 t_1 \mathbf{c}_1 \cdots \mathbf{c}_{n-1} t_n \mathbf{c}_n$ where $\mathbf{c}_i = \langle q_i, \alpha_i, \beta_i \rangle$. We define $\widehat{\pi}$ to be the set of traces of degree one, of the form $\langle \{\mathtt{i}\}, \preceq, \mathtt{src}, \mathtt{tgt}, E, \phi \rangle$ where $\preceq = \{\langle \mathtt{i}, \mathtt{i} \rangle\}$, $\mathtt{src}(\mathtt{i}) = q_0$, $\mathtt{tgt}(\mathtt{i}) = q_n$, $E = \varnothing$, and $\mathtt{Val} \vDash \phi$ with $\mathtt{Val}(x^{\mathtt{i}}_{\mathtt{s}}) = \alpha_0(x)$ and $\mathtt{Val}(x^{\mathtt{i}}_{\mathtt{t}}) = \alpha_n(x)$ for all $x \in \mathbb{X}$. We define $\mathtt{SW}(\pi)$ to be the smallest $k \in \mathbb{N}$ such that there is a trace $\tau \in \widehat{\pi}$ with $\mathtt{SW}(\tau) = k$. For states $q, q' \in Q$ and $\theta \in \mathbb{N}$, we define $\mathbb{R}^{\leq\theta}(q, q') := \left\{ \langle \alpha, \alpha' \rangle \mid \exists \pi. (\mathtt{SW}(\pi) \leq \theta) \wedge \left(\langle q, \alpha, \beta_\epsilon \rangle \xrightarrow{\pi} \langle q', \alpha', \beta_\epsilon \rangle \right) \right\}$.

For a DMPDA $\mathcal{A}$, we define the split width $\mathtt{SW}(\mathcal{A})$ to be the largest k such that there is a plain run π in $\mathcal{A}$ with $\mathtt{SW}(\pi) = k$. For a class $\mathcal{C}$ of DMPDA, we define $\mathtt{SW}(\mathcal{C})$ to be the largest k such that there is an $\mathcal{A} \in \mathcal{C}$ with $\mathtt{SW}(\mathcal{A}) = k$. We say that $\mathcal{C}$ has *bounded split width* if $\mathtt{SW}(\mathcal{C}) = k$ for some $k \in \mathbb{N}$.

6 Decidability

In this section, we present the main result of the paper:

▸ **Theorem 4.** *The reachability problem is decidable for any class of* DMPDA *with bounded split width.*

To prove Theorem 4 we first describe an algorithm computing reachability relations, and then prove its correctness in Lemma 5 (termination), Lemma 6 (soundness), and Lemma 7 (completeness).

6.1 Algorithm

The algorithm inputs a DMPDA $\mathcal{A} = \langle Q, \Delta \rangle$ together with an upper limit θ on the degrees of the traces to be considered during the analysis. The algorithm maintains a set `W` of traces that have been detected but not analyzed, and a set `V` of traces that have been both detected and analyzed. Initially, the sets `W` and `V` are empty (Line 1–2). We add all the atomic traces induced by internal transitions (Line 3), and by matching push/pop transitions (Line 5) to the set `W`. After the initialization phase, the algorithm performs a number of iterations using the repeat-loop of Line 8. In each iteration, we first select and remove an element τ from `W` (Lines 9–10). We check that τ satisfies two conditions (Line 11), namely: that (i) the degree of τ is within the allowed limit, and that (ii) τ is not subsumed by any trace already in the set `V`. If the two conditions are satisfied, we use τ to generate new traces to analyze. These new traces are added to the set `W`. First, we take the shuffle of τ with each member of the set

Algorithm 1: Computing the Reachability Relation.

Input: $\mathcal{A} = \langle Q, \Delta \rangle$: DMPDA,
θ: maximal index size
Output: characterization of the reachability relation

1 $\mathtt{V} \leftarrow \varnothing$;
2 $\mathtt{W} \leftarrow \varnothing$;
3 **for** *each* $t \in \Delta^{\mathtt{intern}}$ **do**
4 $\quad \mathtt{W} \leftarrow \mathtt{W} \cup \{\mathtt{MkTrace}(t)\}$
5 **for** *each* $t_1 \in \Delta^{\mathtt{push}}_{\sigma_1,a_1}$ *and* $t_2 \in \Delta^{\mathtt{pop}}_{\sigma_2,a_2}$ **do**
6 $\quad$ **if** $\sigma_1 = \sigma_2$ *and* $a_1 = a_2$ **then**
7 $\quad\quad \mathtt{W} \leftarrow \mathtt{W} \cup \{\mathtt{MkTrace}(t_1, t_2)\}$
8 **repeat**
9 $\quad$ select some $\tau \in \mathtt{W}$;
10 $\quad \mathtt{W} \leftarrow \mathtt{W} - \{\tau\}$;
11 $\quad$ **if** $(\#\tau \leq \theta) \wedge (\nexists\, \tau' \in \mathtt{V}.\ \tau' \sqsubseteq \tau)$ **then**
12 $\quad\quad$ **for** *each* $\tau' \in \mathtt{V}$ **do**
13 $\quad\quad\quad \mathtt{W} \leftarrow \mathtt{W} \cup (\tau \otimes \tau')$
14 $\quad\quad \mathtt{W} \leftarrow \mathtt{W} \cup \tau\downarrow$;
15 $\quad\quad \mathtt{V} \leftarrow \{\tau' \in \mathtt{V} \mid \tau \not\sqsubseteq \tau'\} \cup \{\tau\}$;
16 **until** $\mathtt{W} = \varnothing$;
17 **for** *each* $q, q' \in Q$ **do**
18 $\quad \mathcal{R}(q, q') \leftarrow \varnothing$
19 **for** *each* $\tau \in \mathtt{V}$ **do**
20 $\quad$ Let $\tau = \langle \mathbb{I}, \preceq, \mathtt{src}, \mathtt{tgt}, E, \phi \rangle$;
21 $\quad$ **if** $\#\tau = 1$ **then**
22 $\quad\quad$ let $\mathbb{I} = \{\mathtt{i}\}$, $\mathtt{src}(\mathtt{i}) = q$, $\mathtt{tgt}(\mathtt{i}) = q'$;
23 $\quad\quad \mathcal{R}(q, q') \leftarrow \mathcal{R}(q, q') \cup \{\phi\}$
24 **return** $\mathcal{R}$

$\mathtt{V}$ (Line 12). Then, we add all possible contractions of τ (Line 14). Finally, at Line 15, we add τ to $\mathtt{V}$, and at the same time remove all elements of $\mathtt{V}$ that are subsumed by τ. Notice that this means that all the traces in the set $\mathtt{V}$ will be pairwise incomparable wrt. $\sqsubseteq$. The iteration of the main loop is repeated until the set $\mathtt{W}$ becomes empty.

After the termination of the loop, we build the reachability relations successively by going through all traces that have been added to $\mathtt{V}$ (Lines 17–23). For each pair of states q and q', we maintain a set $\mathcal{R}(q, q')$ of gap-order formulas. Each time a trace τ of degree one is encountered in $\mathtt{V}$, we add the gap-order formula of τ to the set $\mathcal{R}(q, q')$ corresponding the source state q and target state q' of the (only) index of τ. At the end of the algorithm, the reachability relation between any pair of states is characterized by the union of the denotations of all the gap-order formulas in the corresponding set.

6.2 Correctness

We show correctness of the algorithm in several steps. We start by showing that the algorithm always terminates. For a set A, a pre-order $\preceq$ on A is said to be a *Well Quasi-Ordering*

(WQO) if, for each infinite sequence $a_0a_1a_2\cdots$ of elements from A, there are $i < j$ such that $a_i \preceq a_j$. For a set T of traces, we define its degree $\#T := \max_{\tau\in T}(\#\tau)$. Notice that $\#T$ need not exist in general. Gap-order formulas over $\mathbb{X}$ is WQO under the $\sqsubseteq$ relation [1, 4]. Hence, for any $k \in \mathbb{N}$ and any set of traces T with $\#T \leq k$, the entailment relation $\sqsubseteq$ is a WQO on T. This gives the following lemma.

▸ **Lemma 5.** *The algorithm is guaranteed to terminate.*

Next, we show soundness and completeness of the algorithm. To that end, we will consider the set $\mathcal{R}(q, q')$ of gap-order formulas for each pair of states q and q', returned by the algorithm. We define a denotation function for these formulas, and relate them to the reachability relation $\mathbb{R}(q, q')$. We show that each member in the denotation corresponds to a run (Lemma 6) implying the soundness of the algorithm. Conversely, we show that each run with split width θ belongs to the denotation (Lemma 7) implying the completeness of the algorithm.

A formula ϕ is said be *transitional* over $\mathbb{X}$ if ϕ is a gap-order formula over $\mathbb{X}^{\mathbb{I}}$ where $|\mathbb{I}| = 1$. Notice that, if $\mathbb{I} = \{\mathtt{i}\}$, then each variable in ϕ is either of the form $x^{\mathtt{i}}_{\mathtt{s}}$ or of the form $x^{\mathtt{i}}_{\mathtt{t}}$ where $x \in \mathbb{X}$. We define $\|\phi\|$ to be the set of pairs $\langle\alpha, \alpha'\rangle$ such that $\alpha : \mathbb{X} \mapsto \mathbb{N}$, $\alpha' : \mathbb{X} \to \mathbb{N}$, and there is a $\mathtt{Val} : \mathbb{X}^{\mathbb{I}} \to \mathbb{N}$ with $\mathtt{Val} \vDash \phi$, $\alpha(x) = \mathtt{Val}(x^{\mathtt{i}}_{\mathtt{s}})$ and $\alpha'(x) = \mathtt{Val}(x^{\mathtt{i}}_{\mathtt{t}})$ for all $x \in \mathbb{X}$. For a set of Φ of transitional gap-order formulas, we define $\|\Phi\| := \cup_{\phi\in\Phi}\|\phi\|$. Notice that all members of $\mathcal{R}(q, q')$ are transitional gap-order formulas. The following two lemmas then show the soundness and completeness of the algorithm.

▸ **Lemma 6.** $\forall q, q' \in Q.\ \|\mathcal{R}(q, q')\| \subseteq \mathbb{R}(q, q')$

▸ **Lemma 7.** $\forall q, q' \in Q.\ \|\mathcal{R}(q, q')\| \supseteq \mathbb{R}^{\leq\theta}(q, q')$

7 Applications

7.1 Pushdown automata

A data push-down automata is a DMPDA with a single stack. All plain runs of a data-push-down automaton have split-width bounded by 3 [21]. Thus, it follows from Theorem 4 that

▸ **Corollary 8.** *The reachability problem is decidable for data push-down automata.*

7.2 Multi-push-down systems

As already mentioned in the introduction, the control state reachability is undecidable even in a finite data setting for multi-pushdown systems. Several under-approximation classes (cf. Introduction) have been proposed in the literature for regaining decidability in the finite data case. We recall their definitions below.

- **Bounded context-switch** [30] A context is a sequence of operations in which at most one stack is touched. A run is k-context bounded if it is the concatenation of at most k contexts.
- **Bounded phase** [25] A phase is a sequence of operations in which at most one stack can be popped, though there are no restrictions on pushes. An run is k-phase bounded if it is the concatenation of at most k phases. This subsumes the k- context bounded runs.
- **Ordered stacks** [10, 9]. In an ordered multi-pushdown system, the stacks are ordered linearly by priority. Further a stack may be popped only if all the stacks with higher

priority are empty. An ordered-stack run respects the priority ordering in its stack operations.

- **Bounded scope** [26] A run of a multi-pushdown system is k-scope bounded if the number of context switches between a push and the corresponding pop is always bounded by k. This definition is slightly general than the original round-based definition of [26]. This subsumes the k- context bounded runs,but is orthogonal to k-phase bounded runs.

Runs falling into any of the above class have bounded split-width.

▸ Fact 1 ([21]).
- Split-width of k-context bounded runs is at most $k+2$.
- Split-width of k-phase bounded runs is at most 2^k.
- Split-width of k-scope bounded runs is at most $k+2$.
- Split-width of ordered-stack runs is at most $2^{|\Sigma|}$.

Let U be an under-approximation class above. A U-run of a DMPDA $\mathcal{A}$ is a run that is in U. For the under-approximation U, let $\mathcal{C}_U$ be the class of DMPDA such that all runs of any DMPDA $\mathcal{A} \in \mathcal{C}_U$ are U-runs. By Fact 1, $\mathcal{C}_U$ has bounded split-width. Hence by Theorem 4, we get

▸ **Corollary 9.** *The reachability problem is decidable for {bounded context-switching, bounded phase, ordered stacks, bounded scope} - DMPDA.*

Let U be an under-approximation class above. The U-reachability problem asks, given a DMPDA $\mathcal{A}$ and two plain configurations $\mathbf{c}_1$ and $\mathbf{c}_2$, whether it is possible to reach c_2 from c_1 by a U-run.

If the input DMPDA $\mathcal{A}$ belongs to $\mathcal{C}_U$, then U-reachability problem is the same as the reachability problem, which is decidable by Corollary 9. However, an arbitrary $\mathcal{A}$ may have runs outside of U in general, but still having bounded split-width. Thus running our algorithm naively on any input $\mathcal{A}$ could say "yes" to a pair of configurations $\mathbf{c}_1$ and $\mathbf{c}_2$ even when they are not U-reachable.

In order to decide the U-reachability problem, given the class U and DMPDA $\mathcal{A}$, we will construct a new DMPDA $\mathcal{A}' \in \mathcal{C}_U$. The DMPDA $\mathcal{A}'$ in effect will enforce the semantic restriction of U-runs syntactically into the automaton $\mathcal{A}$. Thus runs of $\mathcal{A}'$ are precisely U-runs of $\mathcal{A}$. We will thus reduce the U-reachability problem in $\mathcal{A}$ to the reachability problem in $\mathcal{A}' \in \mathcal{C}_U$ which is decidable by Corollary 9.

More formally, we reduce the U-reachability problem to reachability problem in $\mathcal{C}_U$. The reduction depends on the under-approximation U. On input DMPDA $\mathcal{A}$, plain configurations $\mathbf{c}_1$ and $\mathbf{c}_2$, we will construct a DMPDA $\mathcal{A}'$ belonging to $\mathcal{C}_U$. Then we will compute a finite set of pairs of plain configurations $\mathbf{c}'_1$ and $\mathbf{c}'_2$ from $\mathbf{c}_1$ and $\mathbf{c}_2$. We check whether $\mathbf{c}'_2$ is reachable from $\mathbf{c}'_1$ in $\mathcal{A}'$ for at least one pair of computed plain configurations $\mathbf{c}'_1$ and $\mathbf{c}'_2$. If this is the case, we conclude that $\mathbf{c}_2$ is U-reachable from $\mathbf{c}_1$ in $\mathcal{A}$. Otherwise, $\mathbf{c}_2$ is not U-reachable from $\mathbf{c}_1$ in $\mathcal{A}$.

Due to lack of space, we will now describe the reduction in detail for only the case of bounded-phase.

Reduction for k-phase bounded.

Given a DMPDA $\mathcal{A} = \langle Q, \Delta \rangle$ and a bound k on the number phases, we construct a new one $\mathcal{A}' = \langle Q', \Delta' \rangle$ where $Q' = Q \times \{1 \dots k\} \times \Sigma \cup Q \times \{0\}$. The state remembers, in addition to the state of $\mathcal{A}$, how many phases have been used so far, and the stack that is being popped from in the current phase. Thus a state of the form (q, i, σ) means that currently $\mathcal{A}$ would

have been in state q, and in the ith phase, which is allowed to pop only from stack σ. The state $(q,0)$ is used to start off a computation, where no stack has been popped yet. The transitions Δ' lifts Δ to smoothly extend to Q' while maintaining the intended semantics. For instance, if $\langle q_1, op, q_2\rangle \in \Delta$, then $\langle (q_1,i,\sigma), op, (q_2,i,\sigma)\rangle \in \Delta'$ and $\langle (q_1,0), op, (q_2,0)\rangle \in \Delta'$ if op is an operation of the forms (i) to (vii) (cf. Section 3). If $\langle q_1, op, q_2\rangle \in \Delta$ and if op is of the form (viii), i.e, $pop\,(x)\,(\sigma)\,(a)$, then we have i) $\langle (q_1,i,\sigma), op, (q_2,i,\sigma)\rangle \in \Delta'$, ii) $\langle (q_1,i,\sigma'), op, (q_2,i+1,\sigma)\rangle \in \Delta'$ if $\sigma' \neq \sigma$ and $i < k$, and, iii) $\langle (q_1,0), op, (q_2,1,\sigma)\rangle \in \Delta'$.

The Dmpda $\mathcal{A}'$ exhibits all and only executions of $\mathcal{A}$ in which the number of phases is bounded by k. Given the pair of plain configurations $\mathbf{c}_1 = \langle q_1, \alpha_1, \beta_\epsilon\rangle$ and $\mathbf{c}_2 = \langle q_2, \alpha_2, \beta_\epsilon\rangle$ of $\mathcal{A}$, we obtain the set of pairs of the form $(\langle q_1', \alpha_1, \beta_\epsilon\rangle, \langle q_2', \alpha_2, \beta_\epsilon\rangle)$ where $q_1' = (q_1, 0)$ and q_2' is either $(q_2, 0)$ or of the form $q_2' = (q_2, i, \sigma)$ for some i and σ.

If $\mathbf{c}_2' = \langle q_2', \alpha_2, \beta_\epsilon\rangle$ is reachable from $\mathbf{c}_1' = \langle q_1', \alpha_1, \beta_\epsilon\rangle$ in $\mathcal{A}'$ for one such computed pair, then indeed, $\mathbf{c}_2$ is k-phase reachable from $\mathbf{c}_1$ in $\mathcal{A}$. Conversely, if $\mathbf{c}_2$ is k-phase reachable from $\mathbf{c}_1$ in $\mathcal{A}$, then there exists $\mathbf{c}_2'$ and $\mathbf{c}_1'$ of the form described above such that $\mathbf{c}_2'$ is reachable from $\mathbf{c}_1'$ in $\mathcal{A}'$. This concludes our reduction.

▸ **Corollary 10.** *The k-phase reachability problem is decidable for any* Dmpda.

In a similar manner, we can have reductions for each of the under-approximations described above. Thus, we get

▸ **Corollary 11.** *{Bounded context-switching, bounded phase, ordered stacks, bounded scope}-reachability problem is decidable for any* Dmpda.

8 Conclusions

We have studied the reachability problem for multi-pushdown systems with gap-order constraints. We provide an algorithm for solving the reachability problem. The algorithm is sound and complete for the classes of automata that have a bounded split-width.

For future work, we plan to consider lifting the framework to a more general setting of auxiliary storages which include queues and multi-sets. Furthermore, it would be interesting to consider the case of distributed processes.

References

1. P. A. Abdulla, M. F. Atig, G. Delzanno, and A. Podelski. Push-down automata with gap-order constraints. In *FSEN*, volume 8161 of *LNCS*, pages 199–216. Springer, 2013.
2. P. A. Abdulla, M. F. Atig, O. Rezine, and J.Stenman. Budget-bounded model-checking pushdown systems. *Formal Methods in System Design*, 45(2):273–301, 2014.
3. P. A. Abdulla, M. F. Atig, and J. Stenman. The minimal cost reachability problem in priced timed pushdown systems. In *LATA*, volume 7183 of *LNCS*, 2012.
4. P. A. Abdulla and G. Delzanno. On the coverability problem for constrained multiset rewriting. In *Proc. AVIS'06, The fifth Int. Workshop on on Automated Verification of Infinite-State Systems*, 2006.
5. C. Aiswarya, P Gastin, and K. Narayan Kumar. Controllers for the verification of communicating multi-pushdown systems. In *CONCUR*, volume 8704 of *LNCS*, pages 297–311, 2014.
6. C. Aiswarya, P. Gastin, and K. Narayan Kumar. Verifying communicating multi-pushdown systems via split-width. In *ATVA'14*, volume 8837 of *LNCS*. Springer, 2014. To appear.
7. S. Akshay, P. Gastin, V. Juge, and S. N. Krishna. personal communication.

8 S. Akshay, P. Gastin, and S. N. Krishna. Analyzing timed systems using tree automata. In *CONCUR'16*, volume 59 of *LIPICS*, pages 27:1–27:14. Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CONCUR.2016.27.

9 M. F. Atig. Global Model Checking of Ordered Multi-Pushdown Systems. In *FSTTCS 2010*, volume 8, pages 216–227, 2010.

10 M. F. Atig, B. Bollig, and P. Habermehl. Emptiness of multi-pushdown automata is 2ETIME-complete. In *Proceedings of DLT'08*, volume 5257 of *LNCS*, pages 121–133. Springer, 2008.

11 M. F. Atig, K. Narayan Kumar, and P. Saivasan. Adjacent ordered multi-pushdown systems. *Int. J. Found. Comput. Sci.*, 25(8):1083–1096, 2014.

12 M. F. Atig, K. Narayan Kumar, and P. Saivasan. Acceleration in multi-pushdown systems. In *TACAS 2016*, volume 9636 of *LNCS*, pages 698–714. Springer, 2016.

13 Mohamed Faouzi Atig. Model-checking of ordered multi-pushdown automata. *Logical Methods in Computer Science*, 8(3), 2012.

14 B. Bollig, A. Cyriac, P. Gastin, and K. Narayan Kumar. Model checking languages of data words. In *FoSSaCS'12*, volume 7213 of *LNCS*, pages 391–405. Springer, March 2012.

15 A. Bouajjani, R. Echahed, and R. Robbana. On the automatic verification of systems with continuous variables and unbounded discrete data structures. In *Hybrid Systems II*, volume 999 of *LNCS*, pages 64–85. Springer, 1994.

16 L. Breveglieri, A. Cherubini, C. Citrini, and S. Crespi Reghizzi. Multi-push-down languages and grammars. *International Journal of Foundations of Computer Science*, 7(3):253–292, 1996.

17 X. Cai and M. Ogawa. Well-structured pushdown systems. In *CONCUR 2013*, volume 8052 of *LNCS*, pages 121–136. Springer, 2013.

18 K. Čerāns. Deciding properties of integral relational automata. In Abiteboul and Shamir, editors, *ICALP 94*, volume 820 of *LNCS*, pages 35–46. Springer Verlag, 1994.

19 Lorenzo Clemente and Slawomir Lasota. Reachability analysis of first-order definable pushdown systems. In *CSL 2015,*, volume 41 of *LIPIcs*, pages 244–259. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

20 B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, pages 313–400. World Scientific, 1997.

21 A. Cyriac, P. Gastin, and K. Narayan Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR'12*, volume 7454 of *LNCS*, pages 547–561. Springer, 2012.

22 F. S. de Boer, M. M. Bonsangue, and J. Rot. It is pointless to point in bounded heaps. *Sci. Comput. Program.*, 112:102–118, 2015.

23 M. Fortin and P. Gastin. Verification of parameterized communicating automata via split-width. In *FoSSaCS'16*, volume 9634 of *LNCS*, pages 197–213. Springer, 2016. doi:10.1007/978-3-662-49630-5_12.

24 A. Heußner, J. Leroux, A. Muscholl, and G. Sutre. Reachability analysis of communicating pushdown systems. *Logical Methods in Computer Science*, 8(3), 2012. doi:10.2168/LMCS-8(3:23)2012.

25 S. La Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS'07*, pages 161–170. IEEE Computer Society Press, 2007.

26 S. La Torre and M. Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In *CONCUR 2011*, volume 6901 of *LNCS*, pages 203–218. Springer, 2011.

27 S. La Torre and G. Parlato. Scope-bounded multistack pushdown systems: Fixed-point, sequentialization, and tree-width. In *FSTTCS 2012*, volume 18 of *LIPIcs*, pages 173–184. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

28 A. Lal and T.W. Reps. Reducing concurrent analysis under a context bound to sequential analysis. In *CAV*, volume 5123 of *LNCS*, pages 37–51. Springer, 2008.

29 P. Madhusudan and G. Parlato. The tree width of auxiliary storage. In Thomas Ball and Mooly Sagiv, editors, *POPL*, pages 283–294. ACM, 2011.

30 S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In N. Halbwachs and L.D. Zuck, editors, *TACAS 2005*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.

31 G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM Trans. Program. Lang. Syst.*, 22(2):416–430, 2000.

32 P. Revesz. A closed form evaluation for datalog queries with integer (gap)-order constraints. *Theoretical Computer Science*, 116(1):117–149, 1993.

33 A. Seth. Global reachability in bounded phase multi-stack pushdown systems. In *CAV'10*, LNCS, 2010.

34 S. La Torre, P. Madhusudan, and G. Parlato. Reducing context-bounded concurrent reachability to sequential reachability. In *CAV*, volume 5643 of *LNCS*, pages 477–492. Springer, 2009.