# A Note on the Complexity of Model-Checking Bounded Multi-Pushdown Systems*

Kshitij Bansal[1] and Stéphane Demri[1,2]

[1] New York University, USA
[2] LSV, CNRS, France

December 5, 2012

**Abstract.** In this note, we provide complexity characterizations of model checking multi-pushdown systems. Multi-pushdown systems model recursive concurrent programs in which any sequential process has a finite control. We consider three standard notions for boundedness: context boundedness, phase boundedness and stack ordering. The logical formalism is a linear-time temporal logic extending well-known logic `CaRet` but dedicated to multi-pushdown systems in which abstract operators (related to calls and returns) such as those for next-time and until are parameterized by stacks. We show that the problem is EXPTIME-complete for context-bounded runs and unary encoding of the number of context switches; we also prove that the problem is 2EXPTIME-complete for phase-bounded runs and unary encoding of the number of phase switches. In both cases, the value $k$ is given as an input (whence it is not a constant of the model-checking problem), which makes a substantial difference in the complexity. In certain cases, our results improve previous complexity results.

## 1 Introduction

*Multi-pushdown systems.* Verification problems for pushdown systems, systems with a finite automaton and an unbounded stack, have been extensively studied and decidability can be obtained as in the case for finite-state systems. Indeed, many problems of interest like computing $\mathrm{pre}^\star(X)$ (set of configurations reaching a regular set $X$), $\mathrm{post}^\star(X)$ (set of configurations accessible from a regular set $X$), reachability and LTL model checking have been shown to be decidable [11,16,22,28]. More precisely, existence of infinite runs with Büchi acceptance condition is decidable for pushdown systems. The proof is based on the fact that $\mathrm{pre}^\star(X)$ is computable and regular for any regular set $X$ of configuration ($\mathrm{post}^\star(X)$ is also regular and computable) [11]. These have also been implemented, for instance in the model-checker Moped [22]. It can be argued that they are natural models for modeling recursive programs. Two limitations

* Work partially supported by projects *ARCUS IdF/Inde* and *EU Seventh Framework Programme* under grant agreement No. PIOF-GA-2011-301166 (DATAVERIF).

though of the model are the inability to model programs with infinite domains (like integers) and modeling concurrency. Having an infinite automaton to handle the former limitation leads to undecidability [20]. An approach to tackle this has been to abstract infinite-state programs to Boolean programs using, for instance, predicate abstraction. The model is repeatedly refining, as needed, like in the SLAM tool [7], SATABS [12] etc. For concurrency, a natural way to extend this model would be to consider pushdown automata with multiple stacks, which has seen significant interest in the recent past [2,3,9,15]. This is the main object of study in this paper which we call *multi-pushdown systems*.

*The difficulty of model-checking multi-pushdown systems.* A pushdown system with even two stacks and with a singleton stack alphabet is sufficient to model a Turing machine (see e.g. this classical result [20]), hence making the problem of even testing reachability undecidable. This is not a unique situation and similar issues exists with other abstractions, like model-checking problems on counter systems; other models of multithreaded programs are also known to admit undecidable verification problems. That is why subclasses of runs have been introduced as well as problems related to the search for 'bounded runs' that may satisfy a desirable or undesirable property. For instance, reversal-bounded counter automata have effectively semilinear reachability sets [18], context-bounded model-checking (bound on the number of context switches) [21] (see also the more recent work dealing with complexity issues in [15]), and of course bounded model-checking (BMC) (bound on the distance of the reached positions) [8].

*Motivations.* In [21], NP-completeness of the reachability problem restricted to context-bounded runs was shown. This was also implemented in a tool called ZINC to verify safety properties and find bugs in actual programs in few context switches, showing that feasibility of the approach. Since, there has been significant work on considering weaker restrictions and other related problems. This paper focuses on the study of model-checking problems for multi-pushdown systems based on LTL-like dialects, naturally allowing to express liveness properties, when some bounds are fixed. Though decidability of these problems has been established in some recent works (as a consequence of considering more expressive logics like monadic second-order logic), we aim to provide optimal computational complexity analysis for LTL-like properties. In particular, we consider a LTL-like specification language based on `CaRet` [1], which strikes to us as fitting given the interest of the model in program verification.

*Content.* We consider the logic Multi-`CaRet`, an extension of `CaRet` (which itself is a generalization of LTL) to be able to reason about runs of concurrent recursive programs. Next, we study the model-checking problem of Multi-`CaRet` formulae over multi-pushdown systems restricted to $k$-bounded runs and give an EXPTIME upper bound, when $k$ is encoded in unary. Since this problem is a generalization of LTL model checking pushdown systems which is known to be EXPTIME-hard, this is an optimal result. Viewed as an extension of [11],

we consider both a more general model and a more general logic, while still preserving the complexity bounds. At a technical level, we focus on combining several approaches in order to achieve optimal complexity bounds. In particular, we combine the approach taken in `CaRet` model-checking of recursive state machines machines (equivalent to pushdown systems in expressiveness, but more explicitly model recursive programs) [1], ideas from reachability analysis of multi-pushdown systems [22] and the techniques introduced in work on pushdown systems in [11,22]. Also, we go further and consider less restrictive notions of boundedness that have been considered in the literature, and obtain optimal or near-optimal complexity bounds for these. To summarize the results in the paper,

- Multi-`CaRet` model-checking over multi-pushdown systems with $k$-**context bounded runs** [21] is EXPTIME-complete when $k$ is encoded in unary and it is in 2EXPTIME if the encoding is binary. The value $k$ is given as an input and not as a parameter of the problem, which makes a substantial difference when complexity analysis is provided.
- Multi-`CaRet` model-checking over multi-pushdown systems with $k$-**phase bounded runs** [24] is in 2EXPTIME when $k$ encoded in unary, and it is in 3-EXPTIME if the encoding is binary. Note that this problem can be encoded from developments in [9] but the EXPTIME upper bound from [9] applies when the number of phases is fixed. Otherwise, one gets an 3EXPTIME upper bound if $k$ is encoded in binary.
- Similarly, Multi-`CaRet` model-checking over **ordered multi-pushdown systems** [4] is in 2EXPTIME when $k$ encoded in unary, and it is in 3EXPTIME if the encoding is binary. This is the best we can hope for since by [3], reachability problem, existence of infinite runs with Büchi acceptance condition and LTL model-checking are decidable for ordered multi-pushdown systems, the latter problem being 2ETIME-complete. Global model-checking is also decidable [3, Theorem 12].

In [19], decidability results can be found for several classes of automata with auxiliary storage based on MSO property. This includes multi-pushdown systems with bounded context and ordered multi-pushdown systems. This might also include problems with temporal logics as stated in [19]. Another work that is worth looking at is the one on games on multi-stack systems [23] where parity games on bounded multi-stack systems are shown decidable thanks to a new proof for the decidability of the emptiness problem. Though, the complexity is non-elementary in the size of the formula, arrived at by using celebrated Courcelle's Theorem [13], which has parameterized complexity non-elementary, the parameter being the size of formula plus the tree-width. Non-elementary lower bounds can be also reached with branching-time logics, see e.g. [5].

*Comparison with two recent works.* In this paper, we generalize the automata-based approach for LTL to a linear-time temporal logic for multi-pushdown systems. A similar approach have been also followed in the recent works [6,26]. Let us explain below the main differences with the present work.

In [6], LTL model-checking on multi-pushdown systems when runs are $k$-scope-bounded is shown EXPTIME-complete. Scope-boundedness strictly extends context-boundedness and therefore Corollary 13(I) and [6, Theorem 7] are closely related even though Corollary 13(I) deals with Multi-`CaRet` and it takes into account only context-boundedness. Moreover, we are able to deal with regular constraints on stack contents while keeping the optimal complexity upper bound. By contrast, [26] introduces an extension of `CaRet` that is identical to the variant we consider in our paper. Again, [26] deals with scope-boundedness and Corollary 13(I) and [26, Theorem 6] are closely related even though Corollary 13(I) takes into account only context-boundedness, which leads to a slightly different result. The upper bound for OBMC from Corollary 14 is a relatively simple consequence of the way we can reduce model-checking to repeated reachability with generalized Büchi acceptance conditions. Similarly, [26, Theorem 7] provides an optimal complexity upper bound for ordered multiply nested words, whence Corollary 14 and [26, Theorem 7] are also related.

As a concluding remark, the work presented in this note is partly subsumed by the recent developments presented in [6,26]. Nevertheless, the upper bounds in Corollary 12(I) and in Corollary 13(I) are original results apart from the way we build the synchronized product in Section 4. Moreover, we believe that our developments shed some useful light on technical issues. For instance, we deal with context-boundedness, phase-boundedness and ordered multi-pushdown systems uniformly while providing in several cases optimal complexity upper bounds. Our complexity analysis for context-boundedness relies on [11,22] whereas for ordered multi-pushdown systems it relies on [3] (for instance, this contrasts with developments from [26, Section 5]). Finally, our construction allows us to add regularity constraints, that are known to go beyond first-order language, by a simple adaptation of the case for Multi-`CaRet`.

## 2 Preliminaries

We write $[N]$ to denote the set $\{1, 2, \ldots, N\}$. We also use a boldface as a shorthand for elements indexed by $[N]$, for e.g., $\boldsymbol{a} = \{a_i \mid i \in [N]\}$. For an alphabet $\Sigma$, $\Sigma^*$ represents the set of finite words over $\Sigma$, $\Sigma^+$ the set of finite non-empty words over $\Sigma$. For a finite word $w = a_1 \ldots a_k$ over $\Sigma$, we write $|w|$ to denote its *length* $k$. For $0 \leq i < |w|$, $w(i)$ represents the $(i+1)$-th letter of the word, here $a_{i+1}$. We use $\mathrm{card}(X)$ to denote the number of elements of a finite set $X$.

### 2.1 Multi-Pushdown Systems

In this section, we first define multi-pushdown systems and then present a simple reduction into multi-pushdown systems with global states in which the next active stack can be read. There exists a correspondence in terms of traces (which is what we need for the forthcoming model-checking problems).

Pushdown systems provide a natural execution model for programs with recursion. A generalization with multiple stacks allows us to model threads.

The formal model is described below, which we will call *multi-pushdown systems*.

**Definition 1.** *A* multi-pushdown system *is a tuple of the form*

$$P = (G, N, \Gamma, \Delta_1, \ldots, \Delta_N),$$

*for some* $N \geq 1$ *such that:*

- $G$ *is a non-empty finite set of* global states*,*
- $\Gamma$ *is the finite* stack alphabet *containing the distinguished letter* $\perp$*,*
- *for every* $s \in [N]$*,* $\Delta_s$ *is the* transition relation *acting on the* $s$*-th stack where* $\Delta_s$ *is a relation included in* $G \times \Gamma \times G \times \mathfrak{A}(\Gamma)$ *with* $\mathfrak{A}(\Gamma)$ *defined as*

$$\mathfrak{A}(\Gamma) \stackrel{def}{=} \bigcup_{a \in \Gamma} \{\mathsf{call}(a), \mathsf{return}(a), \mathsf{internal}(a)\}$$

  *Elements of the set* $\mathfrak{A}(\Gamma)$ *are to be thought of as* actions *modifying the stack with alphabet* $\Gamma$*.*

A *configuration* $c$ of the multi-pushdown system $P$ is the global state along with contents of the $N$ stacks, i.e. $c$ belongs to $G \times (\Gamma^*)^N$. For every $s \in [N]$, we write $\rightarrow_s$ to denote the *one-step relation* with respect to the $s$-th stack. Given two configurations $c = (g, w_1, \ldots, w_s a, \ldots w_N)$ and $c' = (g', w_1, \ldots, w'_s, \ldots, w_N)$, $c \rightarrow_s c' \stackrel{\text{def}}{\Leftrightarrow} (g, a, g', \mathfrak{a}(b)) \in \Delta_s$ where $\mathfrak{a}(b)$ reflects the change in the stack enforcing one of the conditions below:

- $w_s = w'_s$, $\mathfrak{a} = \mathsf{return}$ and $a = b$,
- $w'_s = w_s b$ and $\mathfrak{a} = \mathsf{internal}$,
- $w'_s = w_s ab$ and $\mathfrak{a} = \mathsf{call}$.

The letter $\perp$ from the stack alphabet plays a special role; indeed the initial content of each stack is precisely $\perp$. Moreover, $\perp$ cannot be pushed, popped or replaced by any other symbol. This is a standard way to constrain the transition relations and to check for 'emptiness' of the stack (i.e. equal to $\perp$). We write $\rightarrow_P$ to denote the relation $(\bigcup_{s \in [N]} \rightarrow_s)$. Note that given a configuration $c$, there may exist $c_1$, $c_2$ and $i_1 \neq i_2 \in [N]$ such that $c \rightarrow_{i_1} c_1$ and $c \rightarrow_{i_2} c_2$, which is the fundamental property to consider such models as adequate for modeling concurrency. An infinite *run* is an $\omega$-sequence of configurations $c_0, c_1, c_2, \ldots$ such that for every $i \geq 0$, we have $c_i \rightarrow_P c_{i+1}$. If $c_i \rightarrow_s c_{i+1}$, then we say that for that step, the $s$-th stack is *active*. Similar notions can be defined for finite runs. As usual, we write $c \stackrel{*}{\rightarrow} c'$ whenever there is a finite run from $c$ to $c'$. A standard problem on multi-pushdown systems is the state reachability problem defined below:

**input:** $(P, c, g)$ where $P$ is a multi-pushdown system, $c$ is a configuration of $P$ and $g$ a global state of $P$.
**question:** is there a finite run from $c$ to some configuration $c'$ such that the global state of $c'$ is $g$?

An *enhanced* multi-pushdown system is a multi-pushdown system of the form $P = (G \times [N], N, \Gamma, \Delta_1, \ldots, \Delta_N)$ such that for every $s \in [N]$, $\Delta_s \subseteq (G \times \{s\}) \times \Gamma \times (G \times [N]) \times \mathfrak{A}(\Gamma)$. In such multi-pushdown systems, the global state contains enough information to determine the next *active* stack. Observe that the way the one-step relation is defined, we do not necessarily need to carry this information as part of the finite control (see Lemma 2). We do that in order to enable us to assert about active stack in our logic (see Section 3), and for technical convenience.

**Lemma 2.** *Given a multi-pushdown system $P = (G, N, \Gamma, \boldsymbol{\Delta})$, one can construct in polynomial time an enhanced multi-pushdown system $P' = (G \times [N], N, \Gamma, \boldsymbol{\Delta'})$ such that*

**(I)** *For every infinite run of $P$ of the form*

$$c_0 \to_{s_0} c_1 \to_{s_1} \cdots c_t \to_{s_t} c_{t+1} \cdots$$

*there exists an infinite run $c'_0 \to_{s_0} c'_1 \to_{s_1} \cdots c'_t \to_{s_t} c'_{t+1} \cdots$ of $P'$ such that $(\star)$ for $t \geq 0$, if $c_t = (g_t, \{w_s^t\}_s)$, then $c'_t = ((g_t, s_t), \{w_s^t\}_s)$.*

**(II)** *Similarly, for every infinite run of $P'$ of the form*

$$c'_0 \to_{s_0} c'_1 \to_{s_1} \cdots c'_t \to_{s_t} c'_{t+1} \cdots$$

*there exists an infinite run $c_0 \to_{s_0} c_1 \to_{s_1} \cdots c_t \to_{s_t} c_{t+1} \cdots$ of $P$ such that $(\star)$.*

The proof is by an easy verification. In the sequel, without any loss of generality, we shall consider enhanced multi-pushdown systems only since all the properties that can be expressed in our logical languages are linear-time properties. For instance, there is a logarithmic-space reduction from the state reachability problem to its restriction to enhanced multi-pushdown systems.

### 2.2 Standard Restrictions on Multi-Pushdown Systems

State reachability problem is known to be undecidable by a simple reduction from the non-emptiness problem for intersection of context-free grammars. This has motivated works on the definition of restrictions on runs so that decidability can be regained (for state reachability problem but also for model-checking problems). We recall below three standard notions for boundedness; other notions can be found in [25,14]. Definitions are provided for infinite runs but they can be easily adapted to finite runs too.

In the notion of $k$-boundedness defined below, a phase is understood as a sub-run such that a single stack is active (see e.g. [21]).

**Definition 3.** *Let $\rho = c_0 \to_{s_0} c_1 \to_{s_1} \cdots c_t \to_{s_t} c_{t+1} \cdots$ be an infinite run and $k \geq 0$. We say that $\rho$ is $k$-*bounded *if there exist positions $i_1 \leq i_2 \leq \ldots \leq i_{k-1}$ such that $s_t = s_{t+1}$ for all $t \in \mathbb{N} \setminus \{i_1 \ldots i_{k-1}\}$.*

In the notion of $k$-phase-boundedness defined below, a phase is understood as a sub-run such that return actions are performed on a single stack, see e.g. [24].

**Definition 4.** *Let $\rho = c_0 \rightarrow_{s_0} c_1 \rightarrow_{s_1} \cdots c_t \rightarrow_{s_t} c_{t+1} \cdots$ be an infinite run and $k \geq 0$. We say that $\rho$ is $k$-*phase-bounded *if there is a partition $Y_1, \ldots, Y_\alpha$ of $\mathbb{N}$ with $\alpha \leq k$ such that for every $j \in [1, \alpha]$ there is $s \in [N]$ such that for every $i \in Y_j$, if a return action is performed from $c_i$ to $c_{i+1}$, then it is done on the $s$th stack.*

In the notion of order-boundedness defined below, the stacks are linearly ordered and a return action on a stack can only be performed if the smallest stacks are empty, see e.g. [4].

**Definition 5.** *Let $P$ be a multi-pushdown system and $\preceq = ([N], \leq)$ be a (finite) total ordering of the stacks. Let $\rho = c_0 \rightarrow_{s_0} c_1 \rightarrow_{s_1} \cdots c_t \rightarrow_{s_t} c_{t+1} \cdots$ be an infinite run. We say that $\rho$ is $\preceq$-*bounded *if for every $t \in \mathbb{N}$ that a return is performed on the $s$-th stack, all the stacks strictly smaller than $s$ with respect to $\preceq$ are empty.*

## 3 A Rich Specification Language for Multi-Pushdown Systems: Multi-CaRet

Below, we introduce Multi-CaRet, an extension of the logic CaRet proposed in [1], and dedicated to runs of multi-pushdown systems (instead of for runs of recursive state machines as done in [1]). For instance, the logical language can state that a stack is active. Moreover, the temporal operators are sometimes parameterized by a stack; for instance, the abstract next binary relation can be naturally extended to the case when several stacks are present. Note that the logic presented below can be easily seen as a fragment of monadic second-order logic and therefore the decidability results from [19,9] apply to the forthcoming model-checking problems. However, our definition makes a compromise between a language of linear-time temporal properties that extends the logic from [1] and the most expressive logic for which our model-checking problems are known to be decidable. Indeed, we aim at proposing optimal complexity characterizations. The logic presented below is identical the one presented in [26] except for the presence of regular constraints.

### 3.1 Definition

Models of Multi-CaRet are infinite runs of multi-pushdown systems. For each (enhanced) multi-pushdown system $P = (G \times [N], N, \Gamma, \Delta_1, \ldots, \Delta_N)$, we define the fragment Multi-CaRet($P$) of CaRet that uses syntactic resources from $P$ (namely $G$ and $[N]$). The full language Multi-CaRet is defined as the union of all the sub-languages Multi-CaRet($P$). Formulae of Multi-CaRet($P$) are defined according to the grammar below:

$$\begin{aligned}\phi := {} & g \mid s \mid \mathsf{call} \mid \mathsf{return} \mid \mathsf{internal} \\ & \mid \phi \vee \phi \mid \neg\phi \\ & \mid \mathsf{X}\phi \mid \phi \mathsf{U} \phi \mid \mathsf{X}^{\mathsf{a}}_s \phi \mid \phi \mathsf{U}^{\mathsf{a}}_s \phi \mid \mathsf{X}^{\mathsf{c}}_s \phi \mid \phi \mathsf{U}^{\mathsf{c}}_s \phi\end{aligned}$$

where $s \in [N]$, $g \in G$. Models of Multi-CaRet($P$) formulae are $\omega$-sequences in $\left(G \times [N] \times (\Gamma^*)^N\right)^\omega$, which can be obviously understood as infinite runs of $P$.

*Semantics.* Given an infinite run $\rho = c_0 c_1 \ldots c_t \ldots$ with $c_t = (g_t, s_t, w_1^t, \ldots, w_N^t)$ for every position $t \in \mathbb{N}$, the satisfaction relation $\rho, t \models \phi$ with $\phi$ in Multi-CaRet($P$) is defined inductively as follows (successor relations are defined just below):

$$
\begin{array}{ll}
\rho, t \models g & \text{iff } g_t = g \\
\rho, t \models s & \text{iff } s_t = s \\
\rho, t \models \mathfrak{a} & \text{iff } \left(\mathfrak{a}, \left|w_{s_t}^{t+1}\right| - \left|w_{s_t}^{t}\right|\right) \in \{(\mathsf{call}, 1), (\mathsf{internal}, 0), (\mathsf{return}, -1)\} \\
\rho, t \models \phi_1 \vee \phi_2 & \text{iff } \rho, t \models \phi_1 \text{ or } \rho, t \models \phi_2 \\
\rho, t \models \neg\phi & \text{iff } \rho, t \not\models \phi \\
\rho, t \models \mathsf{X}\phi & \text{iff } \rho, \mathrm{succ}_\rho(t) \models \phi \\
\rho, t \models \phi_1 \mathsf{U} \phi_2 & \text{iff there is a sequence of positions } i_0 = t, i_1 \ldots, i_k, \text{ s.t.} \\
 & \quad \text{for } j < k,\ i_{j+1} = \mathrm{succ}_\rho(i_j),\ \rho, i_j \models \phi_1 \text{ and } \rho, i_k \models \phi_2
\end{array}
$$

For $b \in \{\mathsf{a}, \mathsf{c}\}$ and $s \in [N]$:

$$
\begin{array}{ll}
\rho, t \models \mathsf{X}_s^b \phi & \text{iff } \mathrm{succ}_\rho^{b,s}(t) \text{ is defined and } \rho, \mathrm{succ}_\rho^{b,s}(t) \models \phi \\
\rho, t \models \phi_1 \mathsf{U}_s^a \phi_2 & \text{iff there exists a sequence of positions } t \leq i_0 < i_1 \\
 & \quad \cdots < i_k, \text{ where } i_0 \text{ smallest such with } s_{i_0} = s, \text{ for} \\
 & \quad j < k,\ i_{j+1} = \mathrm{succ}_\rho^{a,s}(i_j),\ \rho, i_j \models \phi_1 \text{ and } \rho, i_k \models \phi_2 \\
\rho, t \models \phi_1 \mathsf{U}_s^c \phi_2 & \text{iff there exists a sequence of positions } t \geq i_0 > i_1 \\
 & \quad \cdots > i_k, \text{ where } i_0 \text{ greatest such with } s_{i_0} = s, \text{ for} \\
 & \quad j < k,\ i_{j+1} = \mathrm{succ}_\rho^{c,s}(i_j),\ \rho, i_j \models \phi_1 \text{ and } \rho, i_k \models \phi_2
\end{array}
$$

The above definition for $\models$ distinguishes three successor relations: *global* successor relation, the *abstract* successor relation that jumps to the first future position after a return action at the same level, if any, and the *caller* successor relation that jumps to the latest past position before a call action at the same level, if any. Here are the formal definitions:

- $\mathrm{succ}_\rho(t) \stackrel{\text{def}}{=} t + 1$ for every $t \in \mathbb{N}$ (standard).
- $\mathrm{succ}_\rho^{\mathsf{a},s}(t)$ is defined below whenever $s$ is active at position $t$:
  1. If $\left|w_s^{t+1}\right| = |w_s^t| + 1$ (call), then $\mathrm{succ}_\rho^{\mathsf{a},s}(t)$ is the smallest $t' > t$ such that $s_{t'} = s$ and $\left|w_s^{t'}\right| = |w_s^t|$. If there is no such $t'$ then $\mathrm{succ}_\rho^{\mathsf{a},s}(t)$ is undefined.
  2. If $\left|w_s^{t+1}\right| = |w_s^t|$ (internal), then $\mathrm{succ}_\rho^{\mathsf{a},s}(t)$ is the smallest $t' > t$ such that $s_{t'} = s$ (first position when $s$th stack is active).
  3. If $\left|w_s^{t+1}\right| = |w_s^t| - 1$ (return), then $\mathrm{succ}_\rho^{\mathsf{a},s}(t)$ is undefined.
- $\mathrm{succ}_\rho^{\mathsf{c},s}(t)$ (caller of $s$-th stack): largest $t' < t$ such that $s_{t'} = s$ and $\left|w_s^{t'}\right| = |w_s^t| - 1$. If such a $t'$ does not exist, then $\mathrm{succ}_\rho^{\mathsf{c},s}(t)$ is undefined.

In the sequel, we write $\rho \models \phi$ whenever $\rho, 0 \models \phi$.

*Adding regularity constraints.* Regularity constraints are the most natural and simple constraints on stack contents and still such properties are not always expressible in first-order logic (or equivalently in plain LTL). Such constraints have a second-order flavour thanks to the close relationship between MSO and regular languages. We define Multi-$\texttt{CaRet}^{reg}$ as the extension of Multi-$\texttt{CaRet}$ in which regularity constraints on stack contents can be expressed. Logic Multi-$\texttt{CaRet}^{reg}$ is defined from Multi-$\texttt{CaRet}$ by adding atomic formulae of the form $\texttt{in}(s, \mathcal{A})$ where $s$ is a stack identifier and $\mathcal{A}$ is a finite-state automaton over the stack alphabet $\Gamma$. The satisfaction relation $\models$ is extended accordingly: $\rho, t \models \texttt{in}(s, \mathcal{A}) \stackrel{\text{def}}{\Leftrightarrow} w_s^t \in \mathrm{L}(\mathcal{A})$ where $\mathrm{L}(\mathcal{A})$ is the set of finite words accepted by $\mathcal{A}$. Note that regularity constraints can be expressed on each stack. Even though most of the developments in the paper are done with Multi-$\texttt{CaRet}$, we shall see that all our complexity upper bounds still hold true with Multi-$\texttt{CaRet}^{reg}$. This is despite the fact that these new constraints have a second-order flavour.

*Another set of temporal operators.* Temporal operators $\mathsf{X}_s^a$ and $\mathsf{U}_s^a$ in Multi-$\texttt{CaRet}$ not only are abstract operators that refer to future positions reached after returns but also they are parameterized by stacks. We made the choice to present these operators for their expressive power but also because they are quite handy in forthcoming technical developments. Below, we briefly present the alternative operators $\mathsf{X}_s$, $\mathsf{X}^a$ and $\mathsf{U}^a$ and we show how they are related to the operators from Multi-$\texttt{CaRet}$.

- $\rho, t \models \mathsf{X}_s\phi \stackrel{\text{def}}{\Leftrightarrow}$ there is $t' > t$ such that $s_{t'} = s$ and for the smallest $t'$, we have $\rho, t' \models \phi$. So, $\mathsf{X}_s\phi$ states that the next position when the stack $s$ is active (if any), $\phi$ holds true.
- $\rho, t \models \mathsf{X}^a\phi \stackrel{\text{def}}{\Leftrightarrow} \mathrm{succ}_\rho^{a,s_t}(t)$ is defined and $\rho, \mathrm{succ}_\rho^{a,s_t}(t) \models \phi$. So, $\mathsf{X}^a\phi$ states that next time the current stack performs a return action, $\phi$ holds true.
- $\rho, t \models \phi_1 \mathsf{U}^a \phi_2 \stackrel{\text{def}}{\Leftrightarrow}$ there exists a sequence of positions $t = i_0 < i_1 \cdots < i_k$ where for $j < k$, $i_{j+1} = \mathrm{succ}_\rho^{a,s_t}(i_j)$, $\rho, i_j \models \phi_1$ and $\rho, i_k \models \phi_2$.

Let us write $s$ to denote the formula (disjunction of atomic formulae) stating that the current active stack is $s$. Note that ($\equiv$ denotes logical equivalence):

$$\mathsf{X}_s\phi \equiv (\neg s \mathsf{U}(s \wedge \phi)) \quad \mathsf{X}^a\phi \equiv (\bigwedge_s (s \Rightarrow \mathsf{X}_s^a\phi)) \quad \phi_1 \mathsf{U}^a \phi_2 \equiv (\bigwedge_s (s \Rightarrow \phi_1 \mathsf{U}_s^a \phi_2))$$

Similarly, we have the following equivalences:

$$\mathsf{X}_s^a\phi \equiv (\neg s \Rightarrow \mathsf{X}_s\phi) \wedge (s \Rightarrow \mathsf{X}^a\phi) \quad \phi_1 \mathsf{U}_s^a \phi_2 \equiv (\neg s \Rightarrow \mathsf{X}_s(\phi_1 \mathsf{U}^a \phi_2)) \wedge (s \Rightarrow \phi_1 \mathsf{U}^a \phi_2)$$

Hence, it is worth noting that the choice we made about the set of primitive operators, does not strictly decrease the expressive power but we shall see that the operators from Multi-$\texttt{CaRet}$ happen to be extremely helpful in forthcoming technicalities.

### 3.2 Decision Problems

Let us introduce the model-checking problems considered in the paper. Model-checking problem for multi-pushdown systems (MC):

**input:** $(P, g_0, \phi)$ where $P$ is a multi-pushdown system $P$, $g_0$ gives an initial configuration $\left(g_0, (\bot)^N\right)$, $\phi$ is a formula in Multi-$\texttt{CaRet}(P)$.
**question:** Is there an infinite run $\rho$ from $\left(g, (\bot)^N\right)$ such that $\rho \models \phi$?

We know that the model-checking problem for multi-pushdown systems is undecidable whereas its restriction to a single stack is EXPTIME-complete [1]. Now, let us turn to bounded model-checking problems. Bounded model-checking problem for multi-pushdown systems (BMC):

**input:** $(P, g_0, \phi, k)$ where $P$ is a multi-pushdown system $P$, $g_0$ gives an initial configuration $\left(g_0, (\bot)^N\right)$, $\phi$ is a formula in Multi-$\texttt{CaRet}(P)$ and $k \in \mathbb{N}$ is a natural number thought of as a bound.
**question:** Is there an infinite $k$-bounded run $\rho$ from $\left(g, (\bot)^N\right)$ such that $\rho \models \phi$?

Note that $k \in \mathbb{N}$ is an input of the problem and not a parameter of BMC. This makes a significant difference for complexity since usually complexity can increase when passing from being a constant to being an input.

Phase-bounded model-checking problem (PBMC) is defined similarly by replacing in the above definition '$k$-bounded run' by '$k$-phase-bounded run'. Similarly, we can obtain a definition with order-boundedness. Order-bounded model-checking problem for multi-pushdown systems (OBMC):

**input:** $(P, g_0, \phi, \preceq)$ where $P$ is a multi-pushdown system $P$, $g_0$ gives an initial configuration $\left(g_0, (\bot)^N\right)$, $\phi$ is a formula in Multi-$\texttt{CaRet}(P)$ and $\preceq = ([N], \leq)$ is a total ordering of the stacks.
**question:** Is there an infinite $\preceq$-bounded run $\rho$ from $\left(g, (\bot)^N\right)$ such that $\rho \models \phi$?

We present below the problem of repeated reachability of multi-pushdown systems, denoted REP. In Section 4, we present how MC can be reduced to REP while obtaining optimal complexity upper bounds.

**input:** $(P, I_0, \mathcal{F})$ where $P$ is a multi-pushdown system, $I_0$ is a subset of global states of $P$ denoting the initial states, and $\mathcal{F}$ is a collection of Büchi acceptance sets,
**question:** Is there an infinite run $\rho$ from some $\left(g_0, (\bot)^N\right)$ with $g_0 \in I_0$ such that for each $F \in \mathcal{F}$ there exists a $g_f \in F$ that is repeated infinitely often?

We will refer to problem restricted to $k$-bounded runs by BREP. Obviously, the variants with other notions of boundedness can be defined too.

Finally, the simplified version of Multi-$\texttt{CaRet}$ consists of the restriction of Multi-$\texttt{CaRet}$ in which atomic formulae are of the form $(g, s)$ when enhanced multi-pushdown systems are involved. Logarithmic-space reductions exist between the full problems and their restrictions to the simplified languages.

**Lemma 6.** *For every problem $\mathcal{P}$ in { MC, BMC, PBMC, OBMC }, there is a logarithmic-space reduction to $\mathcal{P}$ restricted to formulae from the simplified language.*

The proof is by an easy verification and its very idea consists in adding to global states information about the next active stack and about the type of action. In the sequel, without any loss of generality, we restrict ourselves to the simplified languages.

**Theorem 7.** *[19] BMC, PBMC and OBMC are decidable.*

Decidability proof from [19] is very general and partly relies on Courcelle's Theorem. However, it provides non-elementary complexity upper bounds. As a main result of the paper, we shall show that BMC is EXPTIME-complete when $k$ is encoded in unary and in 2EXPTIME when $k$ is encoded in binary.

## 4 From Model-Checking to Repeated Reachability

Herein, we reduce the problem of model checking (MC) to the problem of repeated reachability (REP) while noting complexity features that are helpful later on (Theorem 10). This generalizes the reduction from LTL model-checking for finite-state systems into non-emptiness for generalized Büchi automata (see e.g. [27]), similarly to the approach followed in [26]; not only we have to tailor the reduction to Multi-`CaRet` and to multi-pushdown systems but also we aim at getting tight complexity bounds afterwards. The instance of the problem MC that we have is a multi-pushdown system $P$, a formula $\phi$ and initial state $(g_0, i_0)$. For the instance of REP we will reduce to, we will denote the multi-pushdown system by $\widehat{P}$, the set of acceptance sets by $\mathcal{F}$ and set of initial states by $I_0$.

### 4.1 Augmented Runs

Let $\rho$ be a run of the multi-pushdown system $P = (G \times [N], N, \Gamma, \boldsymbol{\Delta})$ with $\rho \in (G \times [N] \times (\Gamma^*)^N)^\omega$. The multi-pushdown system $\widehat{P}$ is built in such a way that its runs correspond exactly to runs from $P$ but augmented with pieces of information related to the satisfaction of subformulae (taken from the closure set $\mathrm{Cl}(\phi)$ elaborated on shortly), whether a stack is dead or not (using a tag from $\{\mathsf{alive}, \mathsf{dead}\}$) and whether the current call will ever be returned or not (using a tag from $\{\mathsf{noreturn}, \mathsf{willreturn}\}$). These additional tags will suffice to reduce the existence of a run satisfying $\phi$ to the existence of a run satisfying a generalized Büchi condition. First, we define from $\rho$ an "augmented run" $\gamma(\rho)$ which is an infinite sequence from $\left(\widehat{G} \times [N] \times (\widehat{\Gamma}^*)^N\right)^\omega$ where $\widehat{G} = G \times \mathcal{P}(\mathrm{Cl}(\phi))^N \times \{\mathsf{noreturn}, \mathsf{willreturn}\}^N \times \{\mathsf{alive}, \mathsf{dead}\}^N$ and $\widehat{\Gamma} = \Gamma \times \mathcal{P}(\mathrm{Cl}(\phi)) \times \{\mathsf{noreturn}, \mathsf{willreturn}\}$. By definition, an augmented run is simply an $\omega$-sequence but it remains to check that indeed, it will be also a run of the new system. We

will see that $\widehat{G} \times [N]$ is the set of global states of $\widehat{P}$ and $\widehat{\Gamma}$ is the stack alphabet of $\widehat{P}$.

Before defining $\gamma(\cdot)$ which maps runs to augmented runs, let us introduce the standard notion for *closure* but slightly tailored to our needs. Note that each global state is partially made of sets of formulas that can be viewed as obligations for the future. In order to consider only runs satisfying a formula $\phi$, it is sufficient to require that at the first position, obligations include the satisfaction of $\phi$. Obligations can be enforced by the transition relation but also by the satisfaction of Büchi acceptance conditions. It is our intention that the set of runs of $\widehat{P}$ satisfying such generalized Büchi acceptance condition correspond exactly to the set of augmented runs obtained from runs of $P$. Not only the new system simulates all the runs of the original system but it also keeps track of which subformulas holds true at each position. So, the projection of $\gamma(\rho)$ over $G \times [N]$ and $\Gamma$ (i.e, the operation of getting rid of the tags) correspond exactly to $\rho$.

With this understanding of our intentions, we define obligations as a tuple of sets of formulas indexed by the stacks; each stack comes with a finite set of formulas. These formulas are obtained from the closure of $\phi$, defined as the set of subformulas of $\phi$ enriched with formulas for the until formulas. This is similar to what is usually done for LTL and is just a variant of Fischer-Ladner closure [17]. Given a formula $\phi$, its *closure*, denoted $\mathrm{Cl}(\phi)$, is the smallest set that contains $\phi$, the elements of $G \times [N]$, and satisfies the following properties ($b \in \{a, c\}$ and $s \in [N]$):

- If $\neg\phi' \in \mathrm{Cl}(\phi)$ or $\mathsf{X}\phi' \in \mathrm{Cl}(\phi)$ or $\mathsf{X}^b_s\phi' \in \mathrm{Cl}(\phi)$ then $\phi' \in \mathrm{Cl}(\phi)$.
- If $\phi' \vee \phi'' \in \mathrm{Cl}(\phi)$, then $\phi', \phi'' \in \mathrm{Cl}(\phi)$.
- If $\phi'\mathsf{U}\phi'' \in \mathrm{Cl}(\phi)$, then $\phi', \phi''$, and $\mathsf{X}(\phi'\mathsf{U}\phi'')$ are in $\mathrm{Cl}(\phi)$.
- If $\phi'\mathsf{U}^b_s\phi'' \in \mathrm{Cl}(\phi)$, then $\phi', \phi''$, and $\mathsf{X}^b_s(\phi'\mathsf{U}\phi'')$ are in $\mathrm{Cl}(\phi)$.
- If $\phi' \in \mathrm{Cl}(\phi)$ and $\phi'$ in not of the form $\neg\phi''$, then $\neg\phi' \in \mathrm{Cl}(\phi)$.

Note that the number of formulas in $\mathrm{Cl}(\phi)$ is linear in the size of $\phi$ and $P$. An *atom* of $\phi$, is a set $A \subseteq \mathrm{Cl}(\phi)$ that satisfies the following properties:

- For $\neg\phi' \in \mathrm{Cl}(\phi)$, $\phi' \in A$ iff $\neg\phi' \notin A$.
- For $\phi' \vee \phi'' \in \mathrm{Cl}(\phi)$, $\phi' \vee \phi'' \in A$ iff ($\phi' \in A$ or $\phi'' \in A$).
- For $\phi'\mathsf{U}\phi'' \in \mathrm{Cl}(\phi)$, $\phi'\mathsf{U}\phi'' \in A$ iff $\phi'' \in A$ or ($\phi' \in A$ and $\mathsf{X}(\phi'\mathsf{U}\phi'') \in A$).
- $A$ contains exactly one element from $G \times [N]$.

Let $\mathrm{Atoms}(\phi)$ denote the set of atoms of $\phi$, along with empty set (used as special atom, use will become clear later). Note that there are $2^{\mathcal{O}(|\phi|)}$ atoms of $\phi$.

We write $((g^t, s^t), \boldsymbol{w}^t)$ to denote the $t$-th configuration of $\rho$. We define the augmented run $\gamma(\rho)$ so that its $t$-th configuration is of the form $\left(\left(\widehat{g^t}, s^t\right), \widehat{\boldsymbol{w}^t}\right)$ with $\widehat{g^t} = (g^t, \boldsymbol{A}^t, \boldsymbol{r}^t, \boldsymbol{d}^t)$ and $\widehat{w^t_j} = (w^t_j, v^t_j, u^t_j)$ for every $j$ in $[N]$. We say that the stack $j$ is *active* at time $t$ if $s^t = j$.

Then, we define dead-alive tag to be dead if and only if the stack is not active at or after the corresponding position.

$$\forall t \geq 0, j \in [N] : (d^t_j = \mathsf{dead}) \quad \overset{\text{def}}{\Leftrightarrow} \quad (\forall t' \geq t,\ s^{t'} \neq j). \tag{1}$$

The idea of the closure as we discussed is to maintain the set of subformulas that hold true at each step. We will expect it to be the empty set if the stack is dead.

$$\forall t \geq 0, j \in [N] \text{ with } d_j^t = \mathsf{alive}, \psi \in \mathrm{Cl}(\phi):$$
$$\psi \in A_j^t \overset{\text{def}}{\Leftrightarrow} \rho, t' \models \psi \text{ where } t' \text{ is the least } t' \geq t \text{ with } s^{t'} = j. \tag{2}$$
$$\forall t \geq 0, j \in [N] \text{ with } d_j^t = \mathsf{dead}: A_j^t \overset{\text{def}}{=} \emptyset. \tag{3}$$

As for willreturn-noreturn tag, it reflects whether a call action has a "matching" return. This is similar to the $\{\infty, ret\}$ tags in [1]. This may be done by defining tag to be noreturn if stack will never become smaller than what it is now.

$$\forall t \geq 0, j \in [N] \text{ with } d_j^t = \mathsf{alive}:$$
$$(r_j^t = \mathsf{noreturn}) \overset{\text{def}}{\Leftrightarrow} (\forall t' \geq t, \left|w_j^{t'}\right| \geq \left|w_j^t\right|). \tag{4}$$

Finally, the formulas and willreturn-noreturn tag on the stack are defined to be what they were in the global state at the time when the corresponding letter was pushed on the stack.

$$\forall t \geq 0, j \in [N]: v_j^t \overset{\text{def}}{=} A_j^{t_1} A_j^{t_2} \dots A_j^{t_l} \text{ and } u_j^t \overset{\text{def}}{=} d_j^{t_1} d_j^{t_2} \dots d_j^{t_l},$$
$$\text{where for } k \text{ in } [l]: t_k \text{ is largest } t_k \leq t \text{ such that } \left|w_j^{t_k}\right| = k - 1. \tag{5}$$

We observe below properties about the way tags are placed in $\gamma(\rho)$. Later on, we shall establish that these conditions are sufficient to guarantee that any sequence satisfying these conditions correspond to a run of $P$ whose augmented run is exactly the sequence. Here are properties of $\gamma(\rho)$ that are easy to check using the definition of $\gamma(\cdot)$ and the satisfaction relation $\models$.

1. For all $t \geq 0$ and $j \in [N]$, $d_j^t = \mathsf{alive}$ if $j = s^t$.
2. For all $t \geq 0$ and $j \in [N] \setminus \{s^t\}$, $d_j^{t+1} = d_j^t$ , $r_j^{t+1} = r_j^t$ and $A_j^{t+1} = A_j^t$ (only tags related to the active stack may change).
3. For all $t \geq 0$ and $j \in [N]$, $\{\psi \in \mathrm{Cl}(\phi) : \rho, t \models \psi\}$ is an atom and therefore for all $t \geq 0$ and $j \in [N]$, if $d_j^t = \mathsf{alive}$ then $A_j^t$ is an atom.
4. For all $t \geq 0$ and $j \in [N]$, $(g^t, s^t) \in A_{s^t}^t$.
5. For all $t \geq 0$, if the $(t+1)$-th action on the stack is a call and $r_{s^t}^t = \mathsf{willreturn}$, then $r_{s^t}^{t+1} = \mathsf{willreturn}$ and willreturn is the top symbol of $w_{s^t}^{t+1}$. Moreover, $A_{s^t}^t$ is the top symbol of $v_{s^t}^{t+1}$.
6. For all $t \geq 0$, if the $(t+1)$-th action on the stack is a return then $r_{s^t}^t = \mathsf{willreturn}$ and $r_{s^t}^{t+1}$ is equal to the top symbol of $w_{s^t}^t$. Moreover, $\mathsf{X}_{s^t}^a A \subseteq A_{s^t}^{t+1}$ where $A$ is the top symbol of $v_{s^t}^t$. As a notational convenience, we denote the next formulas in a set $A$, using $\mathsf{X}A$. In other words, $\mathsf{X}A = \{\psi \mid \mathsf{X}\psi \in A\}$, $\mathsf{X}_1^{\mathsf{a}} A \overset{\text{def}}{=} \{\psi \mid \mathsf{X}_1^{\mathsf{a}} \psi \in A\}$ etc.
7. For all $t \geq 0$, $\mathsf{X}A_{s^t}^t \subseteq A_{s^t}^{t+1}$.

8. For all $t \geq 0$, if the $(t+1)$-th action on the stack is internal then $r_{s^t}^{t+1} = r_{s^t}^t$. Moreover, $\mathsf{X}_{s^t}^a A_{s^t}^t \subseteq A_{s^t}^{t+1}$ and $\mathsf{X}_{s^t}^c A_{s^t}^t = \mathsf{X}_{s^t}^c A_{s^t}^{t+1}$.
9. For all $t \geq 0$, the set $\mathsf{X}_{s^t}^a A_{s^t}^t$ is empty if one of the conditions below is met:
   (a) the $(t+1)$-th action on the stack is a call and $r_{s^t}^t = \mathsf{noreturn}$,
   (b) the $(t+1)$-th action on the stack is a return,
   (c) $d_{s^t}^{t+1} = \mathsf{dead}$.
10. For all $t \geq 0$, if the $(t+1)$-th action on the stack is a call, then for every $\mathsf{X}_{s^t}^c \psi \in \mathrm{Cl}(\phi)$, $\mathsf{X}_{s^t}^c \psi \in A_{s^{t+1}}^{t+1}$ iff $\psi \in A_{s^t}^t$.
11. For all $t \geq 0$ and $j \in [N]$, if $\psi \in \mathrm{Cl}(\phi)$ with $\psi = \phi_1 \mathsf{U}_i^{\mathsf{a}} \phi_2$, then:
   (a) if $i = j$, then $\psi \in A_j^t$ iff $\phi_1 \in A_j^t$ or ($\phi_2 \in A_j^t$ and $\mathsf{X}_j^{\mathsf{a}} \psi \in A_j^t$),
   (b) if $i \neq j$, then $\psi \in A_j^t$ iff $\psi \in A_i^t$.
12. For all $t \geq 0$ and $j \in [N]$, if $d_j^t = \mathsf{dead}$ then $A_j^t = \emptyset$.

Properties stated above are local since they involve at most two successive configurations. It is also possible to observe Büchi conditions that involve the infinite part of $\gamma(\rho)$.

1. As standard until formulas in LTL, for every $\phi_1 \mathsf{U} \phi_2 \in \mathrm{Cl}(\phi)$, infinitely often in $\gamma(\rho)$ there is a global state $\left(\left(\widehat{g^t}, s^t\right), \widehat{\boldsymbol{w}^t}\right)$ with $\widehat{g^t} = (g^t, \boldsymbol{A}^t, \boldsymbol{r}^t, \boldsymbol{d}^t)$ such that either $\phi_2 \in A_{s^t}^t$ or $\phi_1 \mathsf{U} \phi_2 \notin A_{s^t}^t$.
2. There is a similar property with abstract until: for every $\phi_1 \mathsf{U}_s^a \phi_2 \in \mathrm{Cl}(\phi)$, infinitely often in $\gamma(\rho)$ there is a state $\left(\left(\widehat{g^t}, s^t\right), \widehat{\boldsymbol{w}^t}\right)$ with $\widehat{g^t} = (g^t, \boldsymbol{A}^t, \boldsymbol{r}^t, \boldsymbol{d}^t)$ such that $s = s^t$, $r_s^t = \mathsf{noreturn}$, and $\phi_2 \in A_s^t$ or $\phi_1 \mathsf{U}_s^a \phi_2 \notin A_s^t$.
3. For every $s \in [N]$, infinitely often in $\gamma(\rho)$ there is a global state $\left(\left(\widehat{g^t}, s^t\right), \widehat{\boldsymbol{w}^t}\right)$ with $\widehat{g^t} = (g^t, \boldsymbol{A}^t, \boldsymbol{r}^t, \boldsymbol{d}^t)$ such that either $s^t = s$ or $d_s^t = \mathsf{dead}$. Moreover, note that $d_s^t = \mathsf{dead}$ implies $d_s^{t+1} = \mathsf{dead}$.
4. For every $s \in [N]$, infinitely often in $\gamma(\rho)$ there is a global state $\left(\left(\widehat{g^t}, s^t\right), \widehat{\boldsymbol{w}^t}\right)$ with $\widehat{g^t} = (g^t, \boldsymbol{A}^t, \boldsymbol{r}^t, \boldsymbol{d}^t)$ such that either $d_s^t = \mathsf{dead}$ or ($s^t = s$ and $d_s^t = \mathsf{noreturn}$).

### 4.2 Synchronized Product

Let us define the multi-pushdown system $\widehat{P}$ as $(\widehat{G} \times [N], N, \widehat{\Gamma}, \widehat{\boldsymbol{\Delta}})$ with

- $\widehat{G} = G \times \mathrm{Atoms}(\phi)^N \times \{\mathsf{noreturn}, \mathsf{willreturn}\}^N \times \{\mathsf{alive}, \mathsf{dead}\}^N$,
- $\widehat{\Gamma} = \Gamma \times \mathrm{Atoms}(\phi) \times \{\mathsf{noreturn}, \mathsf{willreturn}\}$,
- each transition relation $\widehat{\Delta_s}$ is defined such that $(\widehat{g}, s, \widehat{a}, \widehat{g'}, s', \mathfrak{a}(a'))$ is in $\widehat{\Delta_s}$ $\overset{\text{def}}{\Leftrightarrow}$ the conditions from Figure 4.2. are satisfied.
  These conditions are actually the syntactic counterparts of the semantical properties stated a bit earlier. To refer to elements in the set we use $g$ to denote a state in original multi-pushdown system, $A_i$ for atoms, $r_i$ for return/no-return tags, $d_i$ for dead/alive tags, $a_S$ to denote stack letter from original multi-pushdown system, $a_A$ for stack atom, $a_r$ for return tag saved on the stack. We use unprimed version to denote state and letter on top of stack before the transition is taken and primed ones for after. Finally, $\mathfrak{a}$ denotes the action to perform on the stack, one of $\{\mathsf{call}, \mathsf{internal}, \mathsf{return}\}$.

1. $((g,s), a_S, (g',s'), \mathfrak{a}(a'_S)) \in \Delta_s$

2. $d_s = \mathsf{alive}$
3. $\forall j \neq s, d_j = d'_j$

4. If $\mathfrak{a} = \mathsf{call}$, then $r_s = \mathsf{willreturn} \Rightarrow r'_s = \mathsf{willreturn}$ and $a'_r = r_s$
5. If $\mathfrak{a} = \mathsf{internal}$, then $r'_s = r_s$ and $a'_r = a_r$
6. If $\mathfrak{a} = \mathsf{return}$, then $r_s = \mathsf{willreturn}$ and $r'_s = a_r$
7. $\forall j \neq s, r_j = r'_j$

8. $(g,s) \in A_s$
9. $\forall j \neq s, A_j = A'_j$
10. $\mathsf{X}A_s \subseteq A'_{s'}$ $(= A_{s'})$

11. If $\mathfrak{a} = \mathsf{call}$, then $a'_A = A_s$
12. If $\mathfrak{a} = \mathsf{internal}$, then $\mathsf{X}^{\mathsf{a}}_s A_s \subseteq A'_s$ and $a'_A = a_A$.
13. If $\mathfrak{a} = \mathsf{return}$, then $\mathsf{X}^{\mathsf{a}}_s a_A \subseteq A'_s$
14. Further, $\mathsf{X}^{\mathsf{a}}_s A_s = \emptyset$ if
    (a) $\mathfrak{a} = \mathsf{call}$ and $r_s = \mathsf{noreturn}$, or
    (b) $\mathfrak{a} = \mathsf{return}$, or
    (c) $d'_s = \mathsf{dead}$

15. If $\mathfrak{a} = \mathsf{call}$, $\mathsf{X}^{\mathsf{c}}_s A'_{s'} = (\mathsf{X}^{\mathsf{c}}_s \operatorname{Atoms}(\phi)) \cap A_s$
16. If $\mathfrak{a} = \mathsf{internal}$, then $\mathsf{X}^{\mathsf{c}}_s A'_{s'} = \mathsf{X}^{\mathsf{c}}_s A_s$.

17. Let $b \in \{\mathsf{a}, \mathsf{c}\}$. Let $\psi \in \mathrm{Cl}(\phi)$, $\psi = \phi_1 \mathsf{U}^b_s \phi_2$. Then, $\psi \in A_s$ iff either $\phi_2 \in A_s$ or ($\phi_1 \in A_s$ and $\mathsf{X}^{\mathsf{a}}_s \psi \in A_s$).
18. Let $b \in \{\mathsf{a}, \mathsf{c}\}$. Let $\psi \in \mathrm{Cl}(\phi)$, $\psi = \phi_1 \mathsf{U}^b_j \phi_2$ with $j \neq s$. Then $\psi \in A_s$ iff $\psi \in A'_s$.
19. $\forall j$: If $d_j = \mathsf{dead}$, then $A_j = \emptyset$ and $r_j = \mathsf{noreturn}$.

**Fig. 1.** Conditions for the transition relation $\widehat{\Delta_s}$. We recall that $\mathsf{X}A = \{\psi \mid \mathsf{X}\psi \in A\}$, $\mathsf{X}^{\mathsf{a}}_1 A = \{\psi \mid \mathsf{X}^{\mathsf{a}}_1 \psi \in A\}$.

The set $\mathcal{F}$ is defined by the following sets of accepting states:

(a) For each until formula $\psi = \phi_1 \mathsf{U} \phi_2 \in \mathrm{Cl}(\phi)$, we define

$$F^1_\psi \stackrel{\text{def}}{=} \{(\widehat{g}, s) \mid \phi_2 \in A_s \text{ or } \psi \notin A_s\}.$$

(b) For each abstract-until formula $\psi = \phi_1 \mathsf{U}^{\mathsf{a}}_s \phi_2 \in \mathrm{Cl}(\phi)$, we define

$$F^2_\psi \stackrel{\text{def}}{=} \{(\widehat{g}, s) \mid r_s = \mathsf{noreturn} \text{ and } (\phi_2 \in A_s \text{ or } \psi \notin A_s)\}.$$

(c) For each $j \in [N]$, we define

$$F^3_j \stackrel{\text{def}}{=} \{(\widehat{g}, s) \mid j = s\} \cup \{(\widehat{g}, s) \mid d_j = \mathsf{dead}\}.$$

(d) For each $j \in [N]$, we define

$$F^4_j \stackrel{\text{def}}{=} \{(\widehat{g}, s) \mid d_j = \mathsf{dead}\} \cup \{(\widehat{g}, s) \mid j = s, d_s = \mathsf{noreturn}\}.$$

Transition relations in $\widehat{P}$ and the acceptance conditions in $\mathcal{F}$ mimic syntactically the semantical properties satisfied by the augmented runs defined from run of $P$. That is why, the correctness lemma stated below follows from the observation about $\gamma(\cdot)$ earlier.

**Lemma 8.** *Let $\rho$ be a run of $P$. Then, $\gamma(\rho)$ is a run of $\widehat{P}$ such that for every $F \in \mathcal{F}$, there is a global state in $F$ that is repeated infinitely often.*

It remains to show that any accepting run corresponds to a run of the original system, we show that this run in fact the run obtained by "forgetting" the augmentations.

**Lemma 9.** *Let $\widehat{\rho}$ be a run of $\widehat{P}$ satisfying the acceptance condition $\mathcal{F}$. Then, $\widehat{\rho}$ is the augmented run corresponding to $\Pi(\widehat{\rho})$, which can be shown to be a run of $P$:*

$$\gamma(\Pi(\widehat{\rho})) = \widehat{\rho}$$

From Lemmas 8 and 9 the soundness and completeness of the reduction follows if we define the set of new initial states $I_0$ for the REP problem as states with initial state $(g_0, i_0)$ for the MC problem and $\phi$ present in the part tracking formulae that hold true:

$$I_0 = \{((g_0, \boldsymbol{A}, \boldsymbol{d}, \boldsymbol{r}), i_0) \mid \phi \in A_{i_0}\}$$

This gives an exponential-time reduction from MC to REP as well as with their bounded variants.

**Theorem 10.** *Let $P$ be a multi-pushdown system with initial configuration $(g, (\bot)^N)$ and $\phi$ be a Multi-*`CaRet` *formula. Let $\widehat{P}$ be the multi-pushdown system built from $P$, $g$ and $\phi$, $I_0$ be the associated set of initial states and $\mathcal{F}$ be the acceptance condition.*

- **(I)** *If $\rho_1$ is a run of $P$ from $(g, (\bot)^N)$ then $\rho_2 = \gamma(\rho_1)$ is a run of $\widehat{P}$ satisfying $\mathcal{F}$ and (A)-(C) hold true.*
- **(II)** *If $\rho_2$ is a run of $\widehat{P}$ from some initial configuration with global state in $I_0$ and satisfying $\mathcal{F}$, then $\Pi(\rho_2)$ is a run of $P$ and (A)-(C) hold true too.*

*Conditions (A)–(C) are defined as follows:*

- **(A)** *$\rho_1$ is $k$-bounded iff $\rho_2$ is $k$-bounded, for all $k \geq 0$;*
- **(B)** *$\rho_1$ is $k$-phase-bounded iff $\rho_2$ is $k$-phase-bounded, for all $k \geq 0$;*
- **(C)** *$\rho_1$ is $\preceq$-bounded iff $\rho_2$ is $\preceq$-bounded, for all total orderings of the stacks $\preceq = ([N], \leq)$.*

Note that at each position, $\rho_1$ and $\rho_2$ work on the same stack and perform the same type of action (call, return, internal move), possibly with slightly different letters. This is sufficient to guarantee the satisfaction of the conditions (A)–(C).

In the rest of this section we prove Lemma 9.

*Notation.* Let $\widehat{\rho}$ be a run of $\widehat{P}$ satisfying the acceptance condition $\mathcal{F}$. First of all, we observe that $\rho \stackrel{\text{def}}{=} \Pi(\widehat{\rho})$ is indeed a run of $P$ because of constraint 1 for the transition relation $\widehat{\Delta}$. Thus, we may conveniently use the same notation as earlier to denote configurations: $((g^t, s^t), \boldsymbol{w}^t)$ for $t$-th configuration of $\rho$, and $\left((\widehat{g^t}, s^t), \widehat{\boldsymbol{w}^t}\right)$ for $\widehat{\rho}$ with $\widehat{g^t} = (g^t, \boldsymbol{A}^t, \boldsymbol{d}^t, \boldsymbol{r}^t)$ and $\widehat{w_j^t} = (w_j^t, v_j^t, u_j^t)$ for every $j$ in $[N]$.

*Proof strategy.* By a case-by-case analysis, we will show that $\widehat{\rho}$ matches each augmentation of $\rho$ as would be obtained by (1)-(6). We will show this first for contents of the stack, then for the two kinds of tags. The order will be important, as for some proofs we would need the assumption that the other parts of the run match. Eventually, we will show the case for atoms. From the technical aspect, the case for correctness of parameterized-abstract-until we introduced is the most interesting, for which we will provide details. The other cases, we shall just point to the relevant ingredients in the construction, and leave the details to the reader.

***Case 1**: Stack content differs.* Let $t$ denote the first position where augmented stack content in $\widehat{\rho}$ differs from $\gamma(\rho)$ as defined in (5). Let $j$ denote a stack that differs. First, we observe that the length of the stacks must be the same as the actions for both runs are the same (this, of course, can be proven more formally – but in interest of readability, where clear, we use informal arguments as these). Further, since $t$ is the first position they differ, the difference has to be in the contents at the "top of the stack". According to the semantics of multi-pushdown systems, the change in the stack is possible only in the one stack active at time $t-1$. We do a case analysis on the action:

- If $\mathfrak{a}^{t-1} = \mathsf{call}$, then according to constraints 4 and 11 the character at the top of the stack will be the same as defined in (5), a contradiction.
- If $\mathfrak{a}^{t-1} = \mathsf{internal}$, the stack atom and stack return tag will not change from time $t-1$ because of constraints 5 and 12, contradicting that it differs from $\gamma(\rho)$ defined in (5).
- If $\mathfrak{a}^{t-1} = \mathsf{return}$, they could not possibly be different, since for both the top character is popped.

Thus, the augmented stack contents must be identical and if at all, the difference must be in the *augmented global state*. We already observed that the global state and active stack match, it remains to show that the tags and the atoms match.

***Case 2**: dead-alive tag differs.* Let $t$ denote the least such where the two tags may differ. Fix $j \in [N]$, a stack corresponding to which the tag differ at this position $t$.

***Case 2(a)**: $d_j^t = \mathsf{dead}$ but $\exists t' \geq t$ such that $s^{t'} = j$.* We rule this out by the following observation about $\widehat{\rho}$:

*Claim.* Let $j \in [N]$. Let $d_j^t = \mathsf{dead}$. Then, for all $t' \geq t$, $s^{t'} \neq j$ and $d_j^{t'} = \mathsf{dead}$.

*Proof.* If $d_j^t = \mathsf{dead}$, then from $\widehat{\rho}$ being a run of $\widehat{P}$ constraint 2 forces that $s^t \neq j$. Consequently, from constraint 3 we conclude that $d_j^{t+1} = d_j^t = \mathsf{dead}$. By induction on time the claim follows.

***Case 2(b)***: $d_j^t = \mathsf{alive}$ *but* $\forall t' \geq t\ s^t \neq j$. We make another observation which follows directly from constraint 3 that states a tag may not change unless the corresponding stack is active.

*Claim.* Let $j \in [N]$. Let $d_j^t = \mathsf{alive}$ and further assume $\forall t' \geq t\ s^{t'} \neq j$. Then, $d_j^{t'} = \mathsf{alive}$ for all $t' \geq t$.

As a consequence, for the tags to mismatch it must be the case that for all $t' \geq t$, $d_j^{t'} = \mathsf{alive}$ and $s^{t'} \neq j$. Though, such a run would not satisfy the Büchi condition in (c) leading to a contradiction.

***Case 3***: willreturn-noreturn *tag differs.* Let $t$ denote the least such position.

*Claim.* Let $j$ be in $[N]$. Let $r_j^t = \mathsf{noreturn}$, then for all $t' > t$ where $\left|w_j^{t'}\right| = \left|w_j^t\right|$, $r_j^{t'} = \mathsf{noreturn}$.

*Proof.* We will prove the statement for the smallest $t'$ with $t' > t$, and the claim will follow by induction on time. We consider the case where $j = s^t$, as otherwise $t' = t + 1$ and the statement follows from condition 7. If $\mathfrak{a}^t = \mathsf{internal}$, again $t' = t + 1$ and the statement follows from condition 5. $\mathfrak{a}^t = \mathsf{return}$ is not possible because of condition 6. Hence, the interesting case is when $j = s^t$ and $\mathfrak{a}^t = \mathsf{call}$. Then if $t'$ is as defined above, we can conclude from semantics of the multi-pushdown systems that $s^{t'-1} = j$ with $\mathfrak{a}^{t'-1} = \mathsf{return}$. Further, the character "popped" from the stack at time $t' - 1$ is the same as the one that was "pushed" at time $t$ – which was noreturn (condition 4). This in turn shows, that $r_j^{t'} = \mathsf{noreturn}$ (condition 6).

The following claim which rules out the tag being incorrectly marked noreturn follows from the previous claim, along with condition 6 that disallows the action to be return when tag is noreturn.

*Claim.* Let $j$ in $[N]$. If $r_j^t = \mathsf{noreturn}$, then $\forall t' > t$, $\left|w_j^{t'}\right| \geq \left|w_j^t\right|$.

Next, we consider the case when the tag is marked willreturn and show there is indeed a position in the future when the stack returns.

*Claim.* Let $j \in [N]$ and $r_j^t = \mathsf{willreturn}$. Then, $\exists t' > t$ such that $\left|w_j^{t'}\right| < \left|w_j^t\right|$.

*Proof.* It is easy to show that while the height of the stack is more than that at time $t$ the tag will be willreturn (using constraints 4-6). In case this stack never becomes dead, the Büchi condition (d) will not be satisfied. In the case the stack eventually becomes dead, condition 19 in the transition relation which requires tag to be noreturn when stack goes dead leads to contradiction.

***Case 4**: Atom differs.* Let $\psi$ denote the smallest formula on which the atoms differ. Let $t$ be the smallest position where the difference is for $\psi$. That is, for some $j \in [N]$, $A_j^t$ does not match definition obtained for atoms for $\gamma(\rho)$ in (2) with respect to presence (or absence) of $\psi$. $j = s^{t-1}$, for otherwise it is easy to conclude using constraint 9 (which states only the atom for an active stack can change) that one may find a smaller $t$ where the runs differ with respect to $\psi$. Let $t^* \geq t$ be the smallest such that $s^{t^*} = j$. If such a $t^*$ does not exist then the stack would in fact be dead, and constraint 19 will imply that the atom is the empty set – matching definition of atom for $\gamma(\rho)$, a contradiction. Thus, we have $\psi$ present (respectively, absent) in atom $A_j^{t^*}$ but $\rho, t^* \not\models \psi$ (respectively, $\rho, t^* \models \psi$). To conclude the previous statement, we made use of constraint 9 and definition of atom for $\gamma(\rho)$ in (2).

The rest of the analysis depends of the type of formula $\psi$ is. We consider the case when $\psi$ is an abstract-until formula in detail. The other cases for the atom are much simpler we point to parts of transition relation constraints ($\widehat{\boldsymbol{\Delta}}$ in Figure 4.2) and Büchi acceptance conditions ($\mathcal{F}$) required to reach a contradiction. Correctness of atomic propositions is ensured by constraint 8, that of propositional parts of the formulas by definition of atom, that of propagating next formulas by constraint 10, and for abstract-next by constraints 11-13, until formulas by atom and Büchi acceptance sets (a), and finally caller formulas (which are easier to handle since they are only passed down the stack) by constraints 15-18. We also do not elaborate on the case of negation of abstract-until is incorrectly present (i.e. *release*) which can be established with the help of transition constraints 11-14 and 17-18.

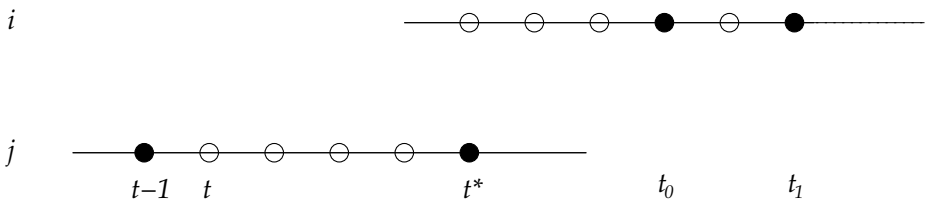


**Fig. 2.** Visual representation of $t$-s in proof for abstract-until. Solid circles correspond to the particular stack being active.

For the rest of the proof we focus on the subcase where $\psi = \phi_1 \mathsf{U}_i^{\mathsf{a}} \phi_2$ such that $\psi \in A_j^{t^*}$ but $\rho, t^* \not\models \psi$. Let $t_0 \geq t^*$ be smallest such with $s^{t_0} = i$. We can conclude that $\psi \in A_i^{t_0}$ (trivially if $i = j$, and using constraint 18 and 9 otherwise), and $\rho, t_0 \not\models \psi$ (semantics of operator $\mathsf{U}_i^{\mathsf{a}}$). Define $t_1$, $t_2$, ... as sequence of positions with $t_{k+1} = \mathrm{succ}_\rho^{\mathsf{a},i}(t_k)$. From constraint 17 about abstract-until formula for active stack in atom corresponding to active stack, if $\psi \in A_i^{t_0}$, then either $\phi_2 \in A_i^{t_0}$ or $\phi_1 \in A_i^{t_0}$ and $\mathsf{X}_i^{\mathsf{a}} \psi \in A_i^{t_0}$. Using the fact that $\psi$ is the smallest "incorrect" formula we rule out the former possibility (it will imply $\rho, t_0 \models \phi_2$ –

a contradiction to $\rho, t_0 \not\models \psi$). Thus, $\rho, t_0 \models \phi_1$ and $\mathsf{X}^{\mathsf{a}}_i \psi \in A_i^{t_0}$. From constraints 11-13 it follows that $\psi \in A_i^{t_1}$. Inductively, one may show that $\psi$ is in $A_i^{t_0}$, $A_i^{t_1}$, $A_i^{t_2}$ etc., and hence also $\rho, t_0 \models \phi_1$, $\rho, t_1 \models \phi_1$ etc.

We have established if an abstract-until formula is in an atom, the first part of the until formula is true for successive-abstract positions for the particular stack it is asserted on. The rest of the argument would have proceeded analogous to the standard until formula, by having a Büchi acceptance condition to ensure that eventually the second part of the formula holds. As such, the argument does not work since the sequence of successive positions is not guaranteed to be infinite. We need to ensure that we have satisfied the until formula by the last position through the transition relation. To be able to do that, we need to be able to identify these last positions locally. This is where the tags will be helpful – they help us exactly determine the cases when the sequence of successive positions where abstract until formula is to be asserted is finite and the case when it is infinite and we require a Büchi acceptance set. We now present a more formal treatment, which will illustrate some subtleties (or technicalities, as one may prefer) of the argument.

First of all, we recall that from previous arguments we already know the tags are correct (that is, they follow the semantics as in (1), (4)). The sequence would be finite if $\mathrm{succ}_\rho^{\mathsf{a},i}(t_k)$ does not exist for some $k$. We do a case analysis on the action performed on the stack. If the action $\mathfrak{a}^{t_k}$ is call, we consider two cases when the abstract-successor may not exist. First case is when the call does not return at all, that is $\forall t' > t_k$, $\left|w_i^{t'}\right| > \left|w_i^{t_k}\right|$. In other words, $\forall t' \geq t_k + 1$, $\left|w_i^{t'}\right| \geq \left|w^{t_k+1}\right|$. In such a case $r_i^{t_k+1}$ will be noreturn and hence because of constraint 14a there will be no abstract-next obligations. Second case is when the call does return. Even in such a case the successor may not exist if stack is dead at this point. Formally, if $t' > t$ denotes the least position such that $w_i^{t'} = w_i^{t_k}$, successor will not exist if $\forall t'' \geq t'$, $s^{t''} \neq i$. In other words, $d_i^{t'} =$ dead. Since this is the least such position, we can conclude that $s^{t'-1} = i$ with $\mathfrak{a}^{t'-1} =$ return with the character being popped the one that was pushed at time $t_k$. Since a dead stack will not have any obligations (constraint 19), we may conclude from constraint 13 that the atom pushed at time $t_k$ has no abstract-next formula. To conclude, $A_i^{t_k}$ cannot have any abstract-next obligations if $\mathfrak{a}^{t_k} =$ call and $\mathrm{succ}_\rho^{\mathsf{a},i}(t_k)$ is undefined. If the action $\mathfrak{a}^{t_k} =$ return, constraint 14b ensures the same. Finally, for the case $\mathfrak{a}^{t_k} =$ internal, abstract-successor will not exist only if stack becomes dead, which is handled by constraint 14c.

On the contrary, let us consider the case the sequence of abstract-successors is infinite. It must be the case that at each of these positions the tags are noreturn (follows from semantics of abstract-successor and definition of tags). We also observe that for each until formula there might be at most one such infinite sequence. Together these observations would imply the run does not satisfy the Büchi acceptance conditions (b).

We have exhausted all possible cases and may safely claim that, indeed, $\widehat{\rho} = \gamma(\rho)$.

## 5 Complexity Analysis with Bounded Runs

### 5.1 Bounded Repeated Global State Reachability Problem

In this section, we evaluate the computational complexity of the problem BREP as well as its variant restricted to a single accepting global state, written below $\text{BREP}_{\text{single}}$. First, note that there is a logarithmic-space reduction from BREP to $\text{BREP}_{\text{single}}$ by copying the multi-pushdown system as many times as the cardinality of $\mathcal{F}$ (as done to reduce non-emptiness problem for generalized Büchi automata to non-emptiness problem for standard Büchi automata). This allows us to conclude about the complexity upper bound for BMC itself but it is worth noting that the multi-pushdown system obtained by synchronization has an exponential number of global states and therefore a refined complexity analysis is required to get optimal upper bounds.

In order to analyze the complexity for $\text{BREP}_{\text{single}}$, we take advantage of two proof techniques that have been introduced earlier and for which we provide a complexity analysis that will suit our final goal. Namely, existence of an infinite $k$-bounded run such that a final global state $(g_f, i_f)$ is repeated infinitely often is checked:

(1) by first guessing a sequence of intermediate global states witnessing context switches of length at most $k+1$,
(2) by computing the (regular) set of reachable configurations following that sequence and then,
(3) by verifying whether there is a reachable configuration leading to an infinite run such that $(g_f, i_f)$ is repeated infinitely often and no context switch occurs.

The principle behind (2) is best explained in [21] but we provide a complexity analysis using the computation of $\text{post}^\star(X)$ along the lines of [22]. Sets $\text{post}^\star(X)$ need to be computed at most $k$ times, which might cause an exponential blow-up (for instance if at each step the number of states were multiplied by a constant). Actually, computing $\text{post}^\star$ adds an additive factor at each step, which is essential for our complexity analysis. Let us define the problem $\text{BREP}_{\text{single}}$ (bounded repeated global state reachability problem for multi-pushdown systems):

**input:** a multi-pushdown system $P$, an initial configuration $\left((g,i),(\bot)^N\right)$, a global state $(g_f, i_f)$ and a bound $k \in \mathbb{N}$.
**question:** Is there an infinite $k$-bounded run $\rho$ from $\left((g,i),(\bot)^N\right)$ such that $(g_f, i_f)$ is repeated infinitely often?

**Proposition 11.** *$\text{BREP}_{\text{single}}$ can be solved in time $\mathcal{O}(|P|^{k+1} \times p(k,|P|))$ for some polynomial $p(\cdot,\cdot)$.*

The proof of Proposition 11 is at the heart of our complexity analysis and it relies on constructions from [11,22]. Moreover, we shall take advantage of it when the input system is precisely $\widehat{P}$.

*Proof.* We use the following results on pushdown systems. Let $P = (G, \Gamma, \Delta)$ be a pushdown system and $\mathcal{A}$ be a $P$-automaton encoding a regular set of configurations. We recall that a *$P$-automaton* $\mathcal{A} = (Q, \Gamma, \delta, G, F)$ is a finite-state automaton over the alphabet $\Gamma$ such that $G \subseteq Q$ [22]. A configuration $(g, w)$ is *recognized* by $\mathcal{A}$ (written $(g, w) \in \mathrm{L}(\mathcal{A})$) $\overset{\text{def}}{\Leftrightarrow}$ $g \rightarrow_w g'$ in $\mathcal{A}$ for some accepting state $g' \in F$. Hence, the set $G$ in $(Q, \Gamma, \delta, G, F)$ can be viewed as a set of initial states and acceptance is relative to the initial state $g$ that allows to satisfy $g \rightarrow_w g'$. Note that a $P$-automaton is nothing else than a means to represent a regular set of configurations.

Let us list a few essential properties.

(i) Let $\mathrm{post}^\star(\mathcal{A}) \overset{\text{def}}{=} \{(g, w) : \exists\, (g', w') \in \mathrm{L}(\mathcal{A}) \text{ s.t. } (g', w') \rightarrow_* (g, w) \text{ in } P\}$. The set $\mathrm{post}^\star(\mathcal{A})$ can be represented by a $P$-automaton $\mathcal{A}'$ such that the number of states for $\mathcal{A}'$ is bounded by the number of states for $\mathcal{A}$ plus $\mathrm{card}(G) \times \mathrm{card}(\Gamma)$ and the time required to compute $\mathcal{A}'$ is quadratic in its number of states and in $\mathrm{card}(\Gamma)$. Clearly, $\mathcal{A}'$ can be computed in polynomial time in $|\mathcal{A}| + |P|$, see e.g. [22] but we shall also take advantage of the fact that the number of states is at most augmented by a constant factor [22, Section 3.3.3].

(ii) Checking whether there is an infinite run $\rho$ starting from a configuration in $\mathrm{L}(\mathcal{A})$ and such that the global state $(g_f, i_f)$ is repeated infinitely often, can be done in polynomial time in $|\mathcal{A}| + |P|$, see e.g. [11].

Let $P = (G \times [N], N, \Gamma, \Delta_1, \ldots, \Delta_N)$ be an enhanced multi-pushdown system, $(g, i)$ and $(g_f, i_f)$ be global states and $k \in \mathbb{N}$. For every $(g_1, i_1) \cdots (g_l, i_l) \in (G \times [N])^*$ such that $l \leq k + 1$, $(g, i) = (g_1, i_1)$ and $i_f = i_l$, we check whether there is an infinite run $\rho$ from $((g, i), (\bot)^N)$ such that

1. $(g_f, i_f)$ is repeated infinitely often,
2. the sequence of active stacks in $\rho$ is exactly $i_1 \cdots i_l$ and $(g_1, i_1) \cdots (g_l, i_l)$ witnesses which are the intermediate global states that there is a context switch.

We show that the existence of such a run can be done in polynomial time, which entails an EXPTIME upper bound since there is an exponential amount of sequences of the form $(g_1, i_1) \cdots (g_l, i_l)$ with $l \leq k + 1$.

The algorithm has two steps. First, we build an automaton $\mathcal{A}$ encoding a (regular) set of configurations from $(G \times [N], \Gamma, \Delta_{i_l})$ corresponding to the configurations of $P$ restricted to the $i_l$-th stack that can be reached from $\big((g_1, i_1), (\bot)^N\big)$ via the sequence $(g_1, i_1) \cdots (g_l, i_l)$. This is precisely the approach followed in [21] by picking a sequence of context switches and doing a $\mathrm{post}^\star(\cdot)$ in that order to get the set of all reachable configurations. We shall see that $\mathcal{A}$ can be indeed computed in polynomial time in $|P|$. Then, we use the polynomial time algorithm from (ii) above to check whether there is an infinite run $\rho$ for the pushdown system $(G \times \{i_l\}, \Gamma, \Delta_{i_l})$ that starts from a configuration in $\mathrm{L}(\mathcal{A})$ and such that the global state $(g_f, i_f)$ is repeated infinitely often. It remains to check that $\mathcal{A}$ can be computed in polynomial time in $|P|$.

Let us introduce some notation. We write $P_j$ to denote the pushdown system $(G \times [N], \Gamma, \Delta_j)$. Note that $P$ and $P_j$ have identical sets of global states: $P_j$ corresponds to $P$ restricted to the transition relation $\Delta_j$. Given a $P$-automaton or a $P_j$-automaton $\mathcal{A} = (Q, \Gamma, \delta, G \times [N], F)$, we write $\mathrm{FSA}(\mathcal{A}, (g, i))$ to denote a finite-state automaton such that for every $w \in \Gamma^*$, we have $w \in \mathrm{L}(\mathrm{FSA}(\mathcal{A}, (g, i)))$ iff $((g, i), w) \in \mathrm{L}(\mathcal{A})$. Note that $\mathrm{FSA}(\mathcal{A}, (g, i))$ can be easily obtained from $\mathcal{A}$ by replacing $(g, i)$ by a new state $q_0$ that is also the initial state. Similarly, given a finite-state automaton $\mathcal{A}$ over the alphabet $\Gamma$, we write $\mathrm{PA}(\mathcal{A}, (g, i))$ to denote a $P$-automaton such that

1. for every $(g', i') \neq (g, i)$, for every $w \in \Gamma^*$, $((g', i'), w) \notin \mathrm{L}(\mathrm{PA}(\mathcal{A}, (g, i)))$,
2. for every $w \in \Gamma^*$, $((g, i), w) \in \mathrm{L}(\mathrm{PA}(\mathcal{A}, (g, i)))$ iff $w \in \mathrm{L}(\mathcal{A})$.

Without any loss of generality, we can assume that $G \times [N]$ is disjoint from the set of states of $\mathcal{A}$ (otherwise, we rename the states). $\mathrm{PA}(\mathcal{A}, (g, i))$ can be obtained from $\mathcal{A}$ by adding all the states from $G \times [N]$ and by replacing in the transition relation, the initial state of $\mathcal{A}$ by $(g, i)$. Observe that the operations $\mathrm{post}^\star(\cdot)$, $\mathrm{FSA}(\cdot)$ and $\mathrm{PA}(\cdot)$ define automata where the increase in the number of states is bounded by $\mathrm{card}(G) \times N$. Hence, performing such operations at most $3 \times (k + 1)$ times, irrespective of the ordering of these operations, will increase the number of states by at most $\mathrm{card}(G) \times N \times 3 \times (k + 1)$.

Now, let us define the (simple) algorithm that consists in computing a $P_{i_l}$-automaton that represents the set of configurations with global state $(g_l, i_l)$ reachable from the initial configuration and following the sequence of intermediate global states (that witness context switches too).

Let $\alpha := 1$ and define the finite-state automata $\mathcal{A}_1, \ldots, \mathcal{A}_N$ over the alphabet $\Gamma$ such that $\mathrm{L}(\mathcal{A}_1) = \cdots = \mathrm{L}(\mathcal{A}_N) = \{\bot\}$. While $\alpha \leq l$, perform the following steps:

1. Compute a $P_{i_\alpha}$-automaton $\mathcal{B}$ that represents all the configurations reachable from a configuration of the form $((g_\alpha, i_\alpha), w)$ with $w \in \mathrm{L}(\mathcal{A}_{i_\alpha})$ using only the stack $i_\alpha$.
   That is, $\mathcal{B} := \mathrm{post}^\star(\mathrm{PA}(\mathcal{A}_{i_\alpha}, (g_\alpha, i_\alpha)))$ (computed from $P_{i_\alpha}$);
2. If $\alpha < l$, then update $\mathcal{A}_{i_\alpha}$ so that it represents the set of contents for the stack $i_\alpha$ from the configurations represented by $\mathcal{B}$. Otherwise return the $P$-automaton $\mathrm{PA}(\mathcal{A}_{i_l}, (g_l, i_l))$.
   That is, if $\alpha < l$ then $\mathcal{A}_{i_\alpha} := \mathrm{FSA}(\mathcal{B}, (g_{\alpha+1}, i_{\alpha+1}))$ else return $\mathrm{PA}(\mathcal{A}_{i_l}, (g_l, i_l))$.
3. $\alpha := \alpha + 1$.

Let the $P_{i_l}$-automaton returned by the above algorithm be denoted by $\mathcal{A}$. Then, as explained earlier, we check whether there is an infinite run $\rho$ for $(G \times \{i_l\}, \Gamma, \Delta_{i_l})$ that starts from a configuration in $\mathrm{L}(\mathcal{A})$ and such that $(g_f, i_f)$ is repeated infinitely often.

Note that $\mathcal{A}$ is obtained from automata with a few states after applying the operations $\mathrm{post}^\star(\cdot)$, $\mathrm{FSA}(\cdot)$ and $\mathrm{PA}(\cdot)$ at most $3 \times (k + 1)$ times. Hence, the size of $\mathcal{A}$ is in $\mathcal{O}([3 \times (k + 1) \times \mathrm{card}(G) \times N]^2 \times \mathrm{card}(\Gamma))$. Detecting whether there is an infinite run in which $(g_f, i_f)$ is repeated infinitely often, will be polynomial

in $3 \times (k+1) \times \mathrm{card}(G) \times N \times \mathrm{card}(\Gamma)$. As a consequence, the complete decision procedure requires time in $\mathcal{O}(|P|^{k+1} \times p(k, |P|))$ for some polynomial $p(\cdot, \cdot)$.

**Corollary 12.** *(I) BMC with $k$ encoded with a unary representation is* EXPTIME*-complete. (II) BMC with $k$ encoded with a binary representation is in* 2EXPTIME*.*

In Corollary 12(I), EXPTIME-hardness is inherited from the case with a single stack [11]. We have seen that there is an infinite $k$-bounded run $\rho$ from $(g, (\bot)^N)$ such that $\rho \models \phi$ iff there exists a $k$-bounded run $\widehat{\rho}$ from $(\widehat{g}, (\bot)^N)$ for some $\widehat{g} \in I_0$ from some multi-pushdown system $\widehat{P}$ such that for each $F \in \mathcal{F}$ there exists a $g_f \in F$ that is repeated infinitely often. Since $\widehat{P}$, $I_0$ and $\mathcal{F}$ are of exponential size, the second proposition can be reduced to an exponential number of instances of $\mathrm{BREP}_{\mathrm{single}}$ in which the multi-pushdown system is of exponential-size only. Using the parameterized complexity upper bound $\mathcal{O}(|P|^{k+1} \times p(k, |P|))$, we can conclude that BMC with $k$ encoded with an unary representation can be solved in exponential time. Corollary 12(II) is then a consequence of the above argument.

It is worth also noting that [9, Theorem 15] would lead to an EXPTIME upper bound for BMC if $k$ is not part of the input, see the EXPTIME upper bound for the problem NESTED-TRACE-SAT($\mathcal{L}^-, k$) introduced in [9]; in our case $k$ *is* indeed part of the input and in that case, the developments in [9] will lead to a 2EXPTIME bound by using the method used for NESTED-TRACE-SAT($\mathcal{L}^-, k$) even if $k$ is encoded in unary. Indeed, somewhere in the proof, the path expression $succ_{\leq k}$ is exponential in the value $k$. Hence, Corollary 12(I) is the best we can hope for when $k$ is part of the input of the model-checking problem.

Adding regularity constraints about stack contents preserves the complexity upper bound. We write BMC$^{reg}$ to denote the extension of BMC in which Multi-`CaRet` is replaced by Multi-`CaRet`$^{reg}$ (see Section 3.1).

**Corollary 13.** *(I) BMC$^{reg}$ with $k$ encoded with an unary representation is* EXPTIME*-complete. (II) BMC$^{reg}$ with $k$ encoded with a binary representation is in* 2EXPTIME*.*

Let us explain how the construction of $\widehat{P}$ can be updated so that BMC$^{reg}$ can be solved almost as BMC. Obviously, we have to take care of regularity constraints and to do so, we enrich the global states with pieces of information about the regularity constraints satisfied by the current stack content. Typically, such pieces of information shall be finite-state automata enriched with a set of states and an update on a stack triggers an update on the set of states. Moreover, we take advantage of the stack mechanism to recover previous values of such pieces of information.

Let us provide a bit more detail. Suppose that the formula $\phi$ contains the following regularity constraints $\mathtt{in}(s_1, \mathcal{A}_1)$, ..., $\mathtt{in}(s_n, \mathcal{A}_n)$. We extend the notion of augmented run so that global states are enriched with triples $(s_1, \mathcal{A}_1, X_1)$, ..., $(s_n, \mathcal{A}_1, X_n)$ where each $X_i$ is a set of states from $\mathcal{A}_i$. By definition, $X_i$ is the set of states from $\mathcal{A}_i$ that can be reached from some initial state of $\mathcal{A}_i$ with the current content of the stack $s_i$. Hence, $X_i$ is uniquely defined but since $\mathcal{A}_i$

is not necessarily deterministic, $X_i$ may not be a singleton. In the definition of $\widehat{P}$, $(s_1, \mathcal{A}_1, X_1), \ldots, (s_n, \mathcal{A}_1, X_n)$ are updated according to the updates on stacks but we have to be a little bit careful. Before explaining the very reason, first note that if $\texttt{in}(s_i, \mathcal{A}_i)$ belongs to an atom of some global state of $\widehat{P}$, we impose that $X_i$ contains an accepting state of $\mathcal{A}_i$, which amounts to check that the current content of the stack $s_i$ is indeed a pattern from $\mathrm{L}(\mathcal{A}_i)$. Let us explain now how to update the values $(s_1, \mathcal{A}_1, X_1), \ldots, (s_n, \mathcal{A}_1, X_n)$. When a call action is performed on a stack $s$ with letter $a$, each value $(s_i, \mathcal{A}_i, X_i)$ with $s_i = s$ is replaced by $(s_i, \mathcal{A}_i, Y_i)$ where $Y_i$ is the set of states that can be reached from some state of $X_i$ by reading $a$ (as what is done in the power set construction for finite-state automata). Moreover, we extend the stack alphabet of $\widehat{P}$ so that each letter of the stack alphabet is also enriched with values $(s_1, \mathcal{A}_1, X_1), \ldots, (s_n, \mathcal{A}_1, X_n)$. When a call action is performed in $P$, we perform also a call in $\widehat{P}$, but with a letter enriched with the values $(s_1, \mathcal{A}_1, X_1), \ldots, (s_n, \mathcal{A}_1, X_n)$ on the stack. Now, when a return is performed on a stack $s$, the current values $(s_1, \mathcal{A}_1, X_1), \ldots, (s_n, \mathcal{A}_1, X_n)$ on the top of the stack are used to restore those values in the global state of $\widehat{P}$. Similarly, when an internal action is performed on a stack $s$, the current values $(s_1, \mathcal{A}_1, X_1), \ldots, (s_n, \mathcal{A}_1, X_n)$ on the top of the stack are also used to get the new values in the global state (but this time these values are not popped from the stack). By observing that values of the form $(s_1, \mathcal{A}_1, X_1), \ldots, (s_n, \mathcal{A}_1, X_n)$ are of linear size in the size of $\phi$, all the complexity analysis we have performed for BMC can be easily adapted to BMC$^{reg}$; actually, the arguments are identical except that the construction of $\widehat{P}$ is a bit more complex, as described above.

### 5.2 Complexity Results for Other Boundedness Notions

In this section, we focus on the complexity analysis for OBMC and PBMC based not only on previous developments but also on the complexity of repeated reachability problems when runs are either $k$-phase-bounded or from ordered multi-pushdown systems. Let OREP$_{\text{single}}$ be the variant of BREP$_{\text{single}}$ with ordered multi-pushdown systems:

**input:** an ordered multi-pushdown system $P$, a configuration $\left((g, i), (\bot)^N\right)$, a global state $(g_f, i_f)$.

**question:** Is there an infinite run $\rho$ from $\left((g, i), (\bot)^N\right)$ such that $(g_f, i_f)$ is repeated infinitely often?

According to [3, Theorem 11], OREP$_{\text{single}}$ restricted to ordered multi-pushdown systems with $k$ stacks can be checked in time $\mathcal{O}(|P|^{2^{d\ k}})$ where $d$ is a constant. Our synchronized product $\widehat{P}$ is exponential in the size of formulas (see Section 4), whence order-bounded model-checking problem OBMC can be solved in 2ExpTime too ($k$ is linear in the size of our initial $P$). Note that Condition (C) from Theorem 10 needs to be used here.

**Corollary 14.** *OBMC is in* 2ExpTime.

Corollary 14 is close to optimal since non-emptiness problem for ordered multi-pushdown systems is 2ETIME-complete. In addition, the same complexity upper bounds apply even when regularity constraints on stack contents are added.

Now, let us conclude this section by considering $k$-bounded-phase runs. Again, let us define the problem $\text{PBREP}_{\text{single}}$:

**input:** a multi-pushdown system $P$, an initial configuration $\left((g,i),(\bot)^N\right)$, a global state $(g_f,i_f)$ and a bound $k \in \mathbb{N}$.
**question:** Is there an infinite $k$-phase-bounded run $\rho$ from $\left((g,i),(\bot)^N\right)$ such that $(g_f,i_f)$ is repeated infinitely often?

In [4, Section 5], it is shown that non-emptiness for $k$-phase multi-pushdown systems can be reduced to non-emptiness for ordered multi-pushdown systems with $2k$ stacks. By inspecting the proof, we can conclude:

1. a similar reduction can be performed for reducing the repeated reachability of a global state,
2. non-emptiness of $k$-phase $P$ with $N$ stacks is reduced to non-emptiness of one of $N^k$ instances of $P'$ with $2k$ stacks and each $P'$ is polynomial-size in $k + |P|$.

Therefore, $\text{PBREP}_{\text{single}}$ is in 2EXPTIME too. Indeed, there is an exponential number of instances and checking non-emptiness for one of them can be done in double exponential time. By combining the different complexity measures above, checking an instance of $\text{PBREP}_{\text{single}}$ with $\widehat{P}$ requires time in

$$\mathcal{O}(N^k \times \left|\widehat{P}\right|^{2^{d\ 2k}})$$

which is clearly double-exponential in the size of $P$. Consequently, bounded model-checking with bounded-phase multi-pushdown systems is in 2EXPTIME too if the number of phases is encoded in unary (and in 3EXPTIME otherwise).

**Corollary 15.** *(I) PBMC where $k$ is encoded in unary is in* 2EXPTIME. *(II) PBMC where $k$ is encoded in binary is in* 3EXPTIME.

Again, the same complexity upper bounds apply when regularity constraints are added. Note that an alternative proof of Corollary 15(I) can be found in the recent paper [10, Theorem 5.2] where fragments of MSO are taken into account.

## 6 Conclusion

In this note, we have shown that model-checking over multi-pushdown systems with $k$-bounded runs is EXPTIME-complete when $k$ is an input bound encoded in unary, otherwise the problem is in 2EXPTIME with a binary encoding. The logical specification language is a version of `CaRet` in which abstract temporal operators are related to calls and returns and parameterized by the stacks,

and regularity constraints on stack contents are present too. A 2EXPTIME upper bound is also established with ordered multi-pushdown systems or with $k$-phase bounded runs and these are optimal upper bounds with a unary encoding of $k$. Our complexity analysis rests on the reduction from model-checking to repeated reachability and on complexity analysis for pushdown systems. The characterization of the complexity when $k$ is encoded in binary is still open and we conjecture that an exponential blow-up may occur. More generally, our work can be pursued in several directions including refinements of the complexity analysis but also increase of the expressive power of the logics while keeping the same worst-case complexity upper bounds.

## References


1. R. Alur, K. Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *TACAS'04*, volume 2988 of *LNCS*, pages 467–481. Springer, 2004.
2. M. Atig. From multi to single stack automata. In *CONCUR'10*, volume 6269 of *LNCS*, pages 117–131. Springer, 2010.
3. M. Atig. Global model checking of ordered multi-pushdown systems. In *FST&TCS'10*, pages 216–227. LIPICS, 2010.
4. M. Atig, B. Bollig, and P. Habermehl. Emptiness of multi-pushdown automata is 2ETIME-complete. In *DLT'08*, volume 5257 of *LNCS*, pages 121–133. Springer, 2008.
5. M. Atig, A. Bouajjani, K. Kumar, and P. Saivashan. Model checking branching-time properties of multi-pushdown systems is hard. Technical Report arXiv:1205.6928, arXiv:cs.LO, May 2012.
6. M. Atig, A. Bouajjani, K. N. Kumar, and P. Saivasan. Linear-time model-cheking for multithreaded programs under scope-bounding. In *ATVA'12*, volume 7561 of *LNCS*, pages 152–166. Springer, 2012.
7. T. Ball and S. Rajamani. The SLAM Toolkit. In *CAV'01*, volume 2102 of *LNCS*, pages 260–264. Springer, 2001.
8. A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.
9. B. Bollig, A. Cyriac, P. Gastin, and M. Zeitoun. Temporal logics for concurrent recursive programs: Satisfiability and model checking. In *MFCS'11*, volume 6907 of *LNCS*, pages 132–144. Springer, 2011.
10. B. Bollig, D. Kuske, and R. Mennicke. The complexity of model-checking multi-stack systems. 2012. Submitted.
11. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: application to model-checking. In *CONCUR'97*, volume 1243 of *LNCS*, pages 135–150. Springer, 1997.
12. E. Clarke, D. Kroening, N. Sharygina, and K. Yorav. SATABS: SAT-based predicate abstraction for ANSI-C. In *TACAS'05*, volume 3440 of *LNCS*, pages 570–574. Springer, 2005.
13. B. Courcelle. The monadic second order theory of graphs I: Recognisable sets of finnite graphs. *I&C*, 85:12–75, 1990.
14. A. Cyriac, P. Gastin, and K. N. Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR'12*, volume 7454 of *LNCS*, pages 547–561. Springer, 2012.
15. J. Esparza and P. Ganty. Complexity of pattern-based verification for multithreaded programs. In *POPL'11*, pages 499–510. ACM, 2011.

16. A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In *INFINITY'97*, volume 9 of *ENTCS*, 1997.
17. M. Fischer and R. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.
18. O. Ibarra. Reversal-bounded multicounter machines and their decision problems. *JACM*, 25(1):116–133, 1978.
19. P. Madhusudan and G. Parlato. The tree width of auxiliary storage. In *POPL'11*, pages 283–294. ACM, 2011.
20. M. Minsky. *Computation, Finite and Infinite Machines*. Prentice Hall, 1967.
21. S. Qaader and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS'05*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.
22. S. Schwoon. *Model-checking pushdown systems*. PhD thesis, TUM, 2002.
23. A. Seth. Games on multi-stack pushdown systems. In *LFCS'09*, volume 5407 of *LNCS*, pages 395–408. Springer, 2009.
24. S. L. Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS'07*, pages 161–170. IEEE, 2077.
25. S. L. Torre and M. Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In *CONCUR'11*, volume 6901 of *LNCS*, pages 203–218. Springer, 2011.
26. S. L. Torre and M. Napoli. A temporal logic for multi-threaded programs. In *TCS 2012*, volume 7604 of *LNCS*, pages 225–239. Springer, 2012.
27. M. Vardi and P. Wolper. Reasoning about infinite computations. *I&C*, 115:1–37, 1994.
28. I. Walukiewicz. Pushdown processes: games and model-checking. *I&C*, 164(2):234–263, 2001.