Android Stack Machine[⋆]

Taolue Chen[1,6], Jinlong He[2,5], Fu Song[3],
Guozhen Wang[4], Zhilin Wu[2], Jun Yan[2,5]

[1] Birkbeck, University of London, UK
[2] State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences, China
[3] ShanghaiTech University, China
[4] Beijing University of Technology, China
[5] University of Chinese Academy of Sciences, China
[6] State Key Laboratory of Novel Software Technology, Nanjing University, China

Abstract. In this paper, we propose Android Stack Machine (ASM), a formal model to capture key mechanisms of Android multi-tasking such as activities, back stacks, launch modes, as well as task affinities. The model is based on pushdown systems with multiple stacks, and focuses on the evolution of the back stack of the Android system when interacting with activities carrying specific launch modes and task affinities. For formal analysis, we study the reachability problem of ASM. While the general problem is shown to be undecidable, we identify expressive fragments for which various verification techniques for pushdown systems or their extensions are harnessed to show decidability of the problem.

1 Introduction

Multi-tasking plays a central role in the Android platform. Its unique design, via activities and back stacks, greatly facilitates organizing user sessions through tasks, and provides rich features such as handy application switching, background app state maintenance, smooth task history navigation (using the "back" button), etc [16]. We refer the readers to Section 2 for an overview.

Android task management mechanism has substantially enhanced user experiences of the Android system and promoted personalized features in app design. However, the mechanism is also notoriously difficult to understand. As a witness, it constantly baffles app developers and has become a common topic of question-and-answer websites (for instance, [2]). Surprisingly, the Android multi-tasking mechanism, despite its importance, has not been thoroughly studied before, let along a formal treatment. This has impeded further developments of computer-aided (static) analysis and verification for Android apps, which are indispensable

[⋆] This work was partially supported by UK EPSRC grant (EP/P00430X/1), ARC grants (DP160101652, DP180100691), NSFC grants (61532019, 61761136011, 61662035, 61672505, 61472474, 61572478) and the National Key Basic Research (973) Program of China (2014CB340701), the INRIA-CAS joint research project "Verification, Interaction, and Proofs", and Key Research Program of Frontier Sciences, CAS, Grant No. QYZDJ-SSW-JSC036.

for vulnerability analysis (for example, detection of task hijacking [16]) and app performance enhancement (for example, estimation of energy consumption [8]).

This paper provides a formal model, i.e., *Android Stack Machine* (ASM), aiming to capture the key features of Android multi-tasking. ASM addresses the behavior of Android *back stacks*, a key component of the multi-tasking machinery, and their interplay with attributes of the activity. In this paper, for these attributes we consider four basic *launch modes*, i.e., standard (STD), singleTop (STP), singleTask (STK), singleInstance (SIT), and *task affinities*. (For simplicity more complicated activity attributes such as *allowTaskReparenting* will not be addressed in the present paper.) We believe that the semantics of ASM, specified as a transition system, captures faithfully the actual mechanism of Android systems. For each case of the semantics, we have created "diagnosis" apps with corresponding launch modes and task affinities, and carried out extensive experiments using these apps, ascertaining its conformance to the Android platform. (Details will be provided in Section 3.)

For Android, technically ASM can be viewed as the counterpart of pushdown systems with multiple stacks, which are the *de facto* model for (multi-threaded) concurrent programs. Being rigours, this model opens a door towards a formal account of Android's multi-tasking mechanism, which would greatly facilitate developers' understanding, freeing them from lengthy, ambiguous, elusive Android documentations. We remark that it is known that the evolution of Android back stacks could also be affected by the *intent flags* of the activities. ASM does not address intent flags explicitly. However, the effects of most intent flags (e.g., FLAG_ACTIVITY_NEW_TASK, FLAG_ACTIVITY_CLEAR_TOP) can be simulated by launch modes, so this is *not* a real limitation of ASM.

Based on ASM, we also make the first step towards a formal analysis of Android multi-tasking apps by investigating the *reachability problem* which is fundamental to all such analysis. ASM is akin to pushdown systems with multiple stacks, so it is perhaps not surprising that the problem is undecidable in general; in fact, we show undecidability for most interesting fragments even with just two launch modes. In the interest of seeking more expressive, practice-relevant decidable fragments, we identify a fragment **STK-dominating ASM** which assumes STK activities have different task affinities and which further restricts the use of SIT activities. This fragment covers a majority of open-source Android apps (e.g., from Github) we have found so far. One of our technical contributions is to give a decision procedure for the reachability problem of STK-dominating ASM, which combines a range of techniques from simulations by pushdown systems with transductions [19] to abstraction methods for multi-stacks. The work, apart from independent interests in the study of multi-stack pushdown systems, lays a solid foundation for further (static) analysis and verification of Android apps related to multi-tasking, enabling model checking of Android apps, security analysis (such as discovering task hijacking), or typical tasks in software engineering such as automatic debugging, model-based testing, etc.

We summarize the main contributions as follows: (1) We propose—to the best of our knowledge—the first comprehensive formal model, Android stack machine, for Android back stacks, which is also validated by extensive experiments. (2) We study the reachability problem for Android stack machine. Apart from strongest possible undecidablity results in the general case, we provide a decision procedure for a practically relevant fragment.

2 Android stack machine: An informal overview

In Android, an application, usually referred to as an *app*, is regarded as a collection of *activities*. An activity is a type of app components, an instance of which provides a graphical user interface on screen and serves the entry point for interacting with the user [1]. An app typically has many activities for different user interactions (e.g., dialling phone numbers, reading contact lists, etc). A distinguished activity is the *main* activity, which is started when the app is launched. A *task* is a collection of activities that users interact with when performing a certain job. The activities in a task are arranged in a stack in the order in which each activity is opened. For example, an email app might have one activity to show a list of latest messages. When the user selects a message, a new activity opens to view that message. This new activity is pushed to the stack. If the user presses the “Back” button, an activity is finished and is popped off the stack. [In practice, the onBackPressed() method can be overloaded and triggered when the “Back” button is clicked. Here we assume—as a model abstraction—that the onBackPressed() method is not overloaded.] Furthermore, multiple tasks may run concurrently in the Android platform and the *back stack* stores all the tasks as a stack as well. In other words, it has a nested structure being a stack of stacks (tasks). We remark that in android, activities from different apps can stay in the same task, and activities from the same app can enter different tasks.

Typically, the evolution of the back stack is dependent mainly on two attributes of activities: *launch modes* and *task affinities*. All the activities of an app, as well as their attributes, including the launch modes and task affinities, are defined in the *manifest file* of the app. The launch mode of an activity decides the corresponding operation of the back stack when the activity is launched. As mentioned in Section 1, there are four basic launch modes in Android: “standard”, “singleTop”, “singleTask” and “singleInstance”. The task affinity of an activity indicates to which task the activity prefers to belong. By default, all the activities from the same app have the same affinity (i.e., all activities in the same app prefer to be in the same task). However, one can modify the default affinity of the activity. Activities defined in different apps can share a task affinity, or activities defined in the same app can be assigned with different task affinities. Below we will use a simple app to demonstrate the evolution of the back stack.

Example 1. In Fig. 1, an app ActivitiesLaunchDemo[1] is illustrated. The app contains four activities of the launch modes STD, STP, STK and SIT, depicted by green, blue, yellow and red, respectively. We will use the colours to name the activities. The green, blue and red activities have the same task affinity, while

[1] Adapted from an open-source app `https://github.com/wauoen/LaunchModeDemo`

the yellow activity has a distinct one. The *main activity* of the app is the green activity. Each activity contains four buttons, i.e., the green, blue, yellow and red button. When a button is clicked, an instance of the activity with the colour starts. Moreover, the identifiers of all the tasks of the back stack, as well as their contents, are shown in the white zones of the window. We use the following execution trace to demonstrate how the back stack evolves according to the launch modes and the task affinities of the activities: The user clicks the buttons in the order of green, blue, blue, yellow, red, and green.

1. [*Launch the app*] When the app is launched, an instance of the main activity starts, and the back stack contains exactly one task, which contains exactly one green activity (see Fig. 1(a)). For convenience, this task is called the green task (with id: 23963).
2. [*Start an* STD *activity*] When the green button is clicked, since the launch mode of the green activity is STD, a new instance of the green activity starts and is pushed into the green task (see Fig. 1(b)).
3. [*Start an* STP *activity*] When the blue button is clicked, since the top activity of the green task is *not* the blue activity, a new instance of the blue activity is pushed into the green task (see Fig. 1(c)). On the other hand, if the blue button is clicked again, because the launch mode of the blue activity is STP and the top activity of the green task is already the blue one, a new instance of the blue activity will *not* be pushed into the green task and its content is kept unchanged.
4. [*Start an* STK *activity*] Suppose now that the yellow button is clicked, since the launch mode of the yellow activity is STK, and the task *affinity* of the yellow activity is different from that of the bottom activity of the green task, a new task is created and an instance of the yellow activity is pushed into the new task (called the yellow task, with id: 23964, see Fig. 1(d), where the leftmost task is the top task of the back stack).
5. [*Start an* SIT *activity*] Next, suppose that the red button is clicked, because the launch mode of the red activity is SIT, a new task is created and an instance of the red activity is pushed into the new task (called the red task, with id: 23965, see Fig. 1(e)). Moreover, at any future moment, the red activity is the only activity of the red task. Note that here a new task is created in spite of the affinity of the red activity.
6. [*Start an* STD *activity from an* SIT *activity*] Finally, suppose the green button is clicked again. Since the top task is the red task, which is supposed to contain only one activity (i.e., the red activity), the green task is then moved to the top of the back stack and a new instance of the green activity is pushed into the green task (see Fig. 1(f)).

3 Android stack machine

For $k \in \mathbb{N}$, let $[k] = \{1, \cdots, k\}$. For a function $f : X \to Y$, let $\mathsf{dom}(f)$ and $\mathsf{rng}(f)$ denote the domain (X) and range (Y) of f respectively.

Definition 1 (Android stack machine). *An* Android stack machine *(ASM) is a tuple* $\mathcal{A} = (Q, \mathsf{Sig}, q_0, \Delta)$*, where*

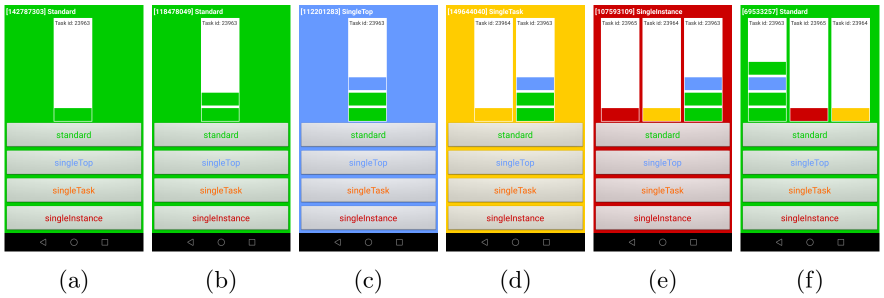

Fig. 1: ActivitiesLaunchDemo: The running example

– *Q is a finite set of control states, and $q_0 \in Q$ is the initial state,*
– $\mathsf{Sig} = (\mathsf{Act}, \mathsf{Lmd}, \mathsf{Aft}, A_0)$ *is the* activity signature, *where*
  - Act *is a finite set of activities,*
  - $\mathsf{Lmd} : \mathsf{Act} \to \{\mathsf{STD}, \mathsf{STP}, \mathsf{STK}, \mathsf{SIT}\}$ *is the launch-mode function,*
  - $\mathsf{Aft} : \mathsf{Act} \to [m]$ *is the task-affinity function, where $m = |\mathsf{Act}|$,*
  - $A_0 \in \mathsf{Act}$ *is the* main *activity,*
– *$\Delta \subseteq Q \times (\mathsf{Act} \cup \{\rhd\}) \times \mathsf{Inst} \times Q$ is the transition relation, where $\mathsf{Inst} = \{\square, \mathsf{back}\} \cup \{\mathsf{start}(A) \mid A \in \mathsf{Act}\}$, such that (1) for each transition $(q, A, \alpha, q') \in \Delta$, it holds that $q' \neq q_0$, and (2) for each transition $(q, \rhd, \alpha, q') \in \Delta$, it holds that $q = q_0$, $\alpha = \mathsf{start}(A_0)$, and $q' \neq q_0$.*

For convenience, we usually write a transition $(q, A, \alpha, q') \in \Delta$ as $q \xrightarrow{A,\alpha} q'$, and $(q, \rhd, \alpha, q') \in \Delta$ as $q \xrightarrow{\rhd,\alpha} q'$. Intuitively, $\rhd$ denotes an empty back stack, $\square$ denotes there is no change over the back stack, back denotes the pop action, and $\mathsf{start}(A)$ denotes the activity A being started. We assume that, if the back stack is empty, the Android stack system terminates (i.e., no further continuation is possible) unless it is in the initial state q_0, We use $\mathsf{Act}_\star$ to denote $\{B \in \mathsf{Act} \mid \mathsf{Lmd}(B) = \star\}$ for $\star \in \{\mathsf{STD}, \mathsf{STP}, \mathsf{STK}, \mathsf{SIT}\}$.

Semantics. Let $\mathcal{A} = (Q, \mathsf{Sig}, q_0, \Delta)$ be an ASM with $\mathsf{Sig} = (\mathsf{Act}, \mathsf{Lmd}, \mathsf{Aft}, A_0)$.

A *task* of $\mathcal{A}$ is encoded as a word $S = [A_1, \cdots, A_n] \in \mathsf{Act}^+$ which denotes the content of the stack, with A_1 (resp. A_n) as the top (resp. bottom) symbol, denoted by $\mathsf{top}(S)$ (resp. $\mathsf{btm}(S)$). **We also call the bottom activity of a non-empty task S as the *root* activity of the task.** (Intuitively, this is the *first* activity of the task.) For $\star \in \{\mathsf{STD}, \mathsf{STP}, \mathsf{STK}, \mathsf{SIT}\}$, a task S is called a $\star$*-task* if $\mathsf{Lmd}(\mathsf{btm}(S)) = \star$. We define the *affinity of a task* S, denoted by $\mathsf{Aft}(S)$, to be $\mathsf{Aft}(\mathsf{btm}(S))$. For $S_1 \in \mathsf{Act}^*$ and $S_2 \in \mathsf{Act}^*$, we use $S_1 \cdot S_2$ to denote the concatenation of S_1 and S_2, and ϵ is used to denote the empty word in Act^*.

As mentioned in Section 2, the (running) tasks on Android are organized as the *back stack*, which is the main modelling object of ASM. Typically we write a back stack ρ as *a sequence of non-empty tasks*, i.e., $\rho = (S_1, \cdots, S_n)$, where S_1 and S_n are called the top and the bottom task respectively. (Intuitively, S_1 is the currently active task.) ε is used to denote the empty back stack. For a non-empty back stack $\rho = (S_1, \cdots, S_n)$, we overload top by using $\mathsf{top}(\rho)$ to refer to the task S_1, and thus $\mathsf{top}^2(\rho)$ the top activity of S_1.

Definition 2 (Configurations). *A* configuration *of $\mathcal{A}$ is a pair (q, ρ) where $q \in Q$ and ρ is a back stack. Assume that $\rho = (S_1, \cdots, S_n)$ with $S_i = [A_{i,1}, \cdots, A_{i,m_i}]$ for each $i \in [n]$. We require ρ to satisfy the following constraints:*

1. *For each $A \in \mathsf{Act}_{\mathsf{STK}}$ or $A \in \mathsf{Act}_{\mathsf{SIT}}$, A occurs in at most one task. Moreover, if A occurs in a task, then A occurs at most once in that task.* [**At most one instance for each STK/SIT-activity**]
2. *For each $i \in [n]$ and $j \in [m_i - 1]$ such that $A_{i,j} \in \mathsf{Act}_{\mathsf{STP}}$, $A_{i,j} \neq A_{i,j+1}$.* [**Non-stuttering for STP-activities**]
3. *For each $i \in [n]$ and $j \in [m_i]$ such that $A_{i,j} \in \mathsf{Act}_{\mathsf{STK}}$, $\mathsf{Aft}(A_{i,j}) = \mathsf{Aft}(S_i)$.* [**Affinities of STK-activities agree to the host task**]
4. *For each $i \in [n]$ and $j \in [m_i]$ such that $A_{i,j} \in \mathsf{Act}_{\mathsf{SIT}}$, $m_i = 1$.* [**SIT-activities monopolize a task**]
5. *For $i \neq j \in [n]$ such that $\mathsf{btm}(S_i) \notin \mathsf{Act}_{\mathsf{SIT}}$ and $\mathsf{btm}(S_j) \notin \mathsf{Act}_{\mathsf{SIT}}$, $\mathsf{Aft}(S_i) \neq \mathsf{Aft}(S_j)$.* [**Affinities of tasks are mutually distinct, except for those rooted at SIT-activities**]

By Definition 2(5), each back stack ρ contains at most $|\mathsf{Act}_{\mathsf{SIT}}| + |\mathsf{rng}(\mathsf{Aft})|$ (more precisely, $|\mathsf{Act}_{\mathsf{SIT}}| + |\{\mathsf{Aft}(A) \mid A \in \mathsf{Act} \setminus \mathsf{Act}_{\mathsf{SIT}}\}|$) tasks. Moreover, by Definition 2(1-5), all the root activities in a configuration are pairwise distinct, which allows to refer to a task whose root activity is A as *the* A-task.

Let $\mathsf{Conf}_{\mathcal{A}}$ denote the set of configurations of $\mathcal{A}$. The *initial* configuration of $\mathcal{A}$ is (q_0, ε). To formalize the semantics of $\mathcal{A}$ concisely, we introduce the following shorthand stack operations and one auxiliary function. Here $\rho = (S_1, \cdots, S_n)$ is a non-empty back stack.

$$\mathsf{Noaction}(\rho) \equiv \rho \qquad \mathsf{Push}(\rho, B) \equiv (([B] \cdot S_1), S_2, \cdots, S_n)$$

$$\mathsf{NewTask}(B) \equiv ([B]) \qquad \mathsf{NewTask}(\rho, B) \equiv ([B], S_1, \cdots, S_n)$$

$$\mathsf{Pop}(\rho) \equiv \begin{cases} \varepsilon, & \text{if } n = 1 \text{ and } S_1 = [A]; \\ (S_2, \cdots, S_n), & \text{if } n > 1 \text{ and } S_1 = [A]; \\ (S_1', S_2, \cdots, S_n), & \text{if } S_1 = [A] \cdot S_1' \text{ with } S_1' \in \mathsf{Act}^+; \end{cases}$$

$$\mathsf{PopUntil}(\rho, B) \equiv (S_1'', S_2, \cdots, S_n), \text{ where}$$
$$S_1 = S_1' \cdot S_1'' \text{ with } S_1' \in (\mathsf{Act} \setminus \{B\})^* \text{ and } \mathsf{top}(S_1'') = B;$$

$$\mathsf{Move2Top}(\rho, i) \equiv (S_i, S_1, \cdots, S_{i-1}, S_{i+1}, \cdots, S_n)$$

$$\mathsf{GetNonSITTaskByAft}(\rho, k) \equiv \begin{cases} S_i, & \text{if } \mathsf{Aft}(S_i) = k \text{ and } \mathsf{Lmd}(\mathsf{btm}(S_i)) \neq \mathsf{SIT}; \\ \mathsf{Undef}, & \text{otherwise.} \end{cases}$$

Intuitively, $\mathsf{GetNonSITTaskByAft}(\rho, k)$ returns a non-SIT task whose affinity is k if it exists, otherwise returns Undef.

In the sequel, we define the transition relation $(q, \rho) \xrightarrow{\mathcal{A}} (q', \rho')$ on $\mathsf{Conf}_{\mathcal{A}}$ to formalize the semantics of $\mathcal{A}$. We start with the transitions out of the initial state q_0 and those with $\square$ or back action.

– For each transition $q_0 \xrightarrow{\triangleright, \mathsf{start}(A_0)} q$, $(q_0, \varepsilon) \xrightarrow{\mathcal{A}} (q, \mathsf{NewTask}(A_0))$.

- For each transition $q \xrightarrow{A,\Box} q'$ and $(q,\rho) \in \mathsf{Conf}_{\mathcal{A}}$ such that $\mathsf{top}^2(\rho) = A$, $(q,\rho) \xrightarrow{\mathcal{A}} (q', \mathsf{Noaction}(\rho))$.
- For each transition $q \xrightarrow{A,\mathsf{back}} q'$ and $(q,\rho) \in \mathsf{Conf}_{\mathcal{A}}$ such that $\mathsf{top}^2(\rho) = A$, $(q,\rho) \xrightarrow{\mathcal{A}} (q', \mathsf{Pop}(\rho))$.

The most interesting case is, however, the transitions of the form $q \xrightarrow{A,\mathsf{start}(B)} q'$. We shall make case distinctions based on the launch mode of B. For each transition $q \xrightarrow{A,\mathsf{start}(B)} q'$ and $(q,\rho) \in \mathsf{Conf}_{\mathcal{A}}$ such that $\mathsf{top}^2(\rho) = A$, $(q,\rho) \xrightarrow{\mathcal{A}} (q',\rho')$ if one of the following cases holds. Assume $\rho = (S_1, \cdots, S_n)$.

$\boxed{\text{CASE } \mathsf{Lmd}(B) = \mathsf{STD}}$

- $\mathsf{Lmd}(A) \neq \mathsf{SIT}$, then $\rho' = \mathsf{Push}(\rho, B)$;
- $\mathsf{Lmd}(A) = \mathsf{SIT}$ [2], then
  - if $\mathsf{GetNonSITTaskByAft}(\rho, \mathsf{Aft}(B)) = S_i$ [3], then $\rho' = \mathsf{Push}(\mathsf{Move2Top}(\rho, i), B)$,
  - if $\mathsf{GetNonSITTaskByAft}(\rho, \mathsf{Aft}(B)) = \mathsf{Undef}$, then $\rho' = \mathsf{NewTask}(\rho, B)$;

$\boxed{\text{CASE } \mathsf{Lmd}(B) = \mathsf{STP}}$

- $\mathsf{Lmd}(A) \neq \mathsf{SIT}$ and $A \neq B$, then $\rho' = \mathsf{Push}(\rho, B)$;
- $\mathsf{Lmd}(A) \neq \mathsf{SIT}$ and $A = B$, then $\rho' = \mathsf{Noaction}(\rho)$;
- $\mathsf{Lmd}(A) = \mathsf{SIT}$ [2],
  - if $\mathsf{GetNonSITTaskByAft}(\rho, \mathsf{Aft}(B)) = S_i$ [3], then
    - $*$ if $\mathsf{top}(S_i) \neq B$, $\rho' = \mathsf{Push}(\mathsf{Move2Top}(\rho, i), B)$,
    - $*$ if $\mathsf{top}(S_i) = B$, $\rho' = \mathsf{Move2Top}(\rho, i)$;
  - if $\mathsf{GetNonSITTaskByAft}(\rho, \mathsf{Aft}(B)) = \mathsf{Undef}$, then $\rho' = \mathsf{NewTask}(\rho, B)$;

$\boxed{\text{CASE } \mathsf{Lmd}(B) = \mathsf{SIT}}$

- $A = B$ [2], then $\rho' = \mathsf{Noaction}(\rho)$;
- $A \neq B$ and $S_i = [B]$ for some $i \in [n]$ [4], then $\rho' = \mathsf{Move2Top}(\rho, i)$;
- $A \neq B$ and $S_i \neq [B]$ for each $i \in [n]$, then $\rho' = \mathsf{NewTask}(\rho, B)$;

$\boxed{\text{CASE } \mathsf{Lmd}(B) = \mathsf{STK}}$

- $\mathsf{Lmd}(A) \neq \mathsf{SIT}$ and $\mathsf{Aft}(B) = \mathsf{Aft}(S_1)$, then
  - if B does *not* occur in S_1 [5], then $\rho' = \mathsf{Push}(\rho, B)$;
  - if B occurs in S_1 [6], then $\rho' = \mathsf{PopUntil}(\rho, B)$;
- $\mathsf{Lmd}(A) \neq \mathsf{SIT} \Longrightarrow \mathsf{Aft}(B) \neq \mathsf{Aft}(S_1)$, then
  - if $\mathsf{GetNonSITTaskByAft}(\rho, \mathsf{Aft}(B)) = S_i$ [7],

[2] By Definition 2(4), $S_1 = [A]$.
[3] If i exists, it must be unique by Definition 2(5). Moreover, $i > 1$, as $\mathsf{Lmd}(A) = \mathsf{SIT}$.
[4] If i exists, it must be unique by Definition 2(1). Moreover, $i > 1$, as $A \neq B$.
[5] B does *not* occur in ρ at all by Definition 2(3-5).
[6] Note that B occurs at most once in S_1 by Definition 2(1).
[7] If i exists, it must be unique by Definition 2(5). Moreover, $i > 1$, as $\mathsf{Lmd}(A) \neq \mathsf{SIT} \Longrightarrow \mathsf{Aft}(B) \neq \mathsf{Aft}(S_1)$.

    * if B does *not* occur in S_i [5], then $\rho' = \mathsf{Push}(\mathsf{Move2Top}(\rho, i), B)$;
    * if B occurs in S_i [8], then $\rho' = \mathsf{PopUntil}(\mathsf{Move2Top}(\rho, i), B)$,
- if $\mathsf{GetNonSITTaskByAft}(\rho, \mathsf{Aft}(B)) = \mathsf{Undef}$, then $\rho' = \mathsf{NewTask}(\rho, B)$;

This concludes the definition of the transition definition of $\xrightarrow{\mathcal{A}}$. As usual, we use $\overset{\mathcal{A}}{\Rightarrow}$ to denote the reflexive and transitive closure of $\xrightarrow{\mathcal{A}}$.

Example 2. The ASM for the ActivitiesLaunchDemo app in Example 1 is $\mathcal{A} = (Q, \mathsf{Sig}, q_0, \Delta)$, where $Q = \{q_0, q_1\}$, $\mathsf{Sig} = (\mathsf{Act}, \mathsf{Lmd}, \mathsf{Aft}, A_g)$ with

– $\mathsf{Act} = \{A_g, A_b, A_y, A_r\}$, corresponding to the green, blue, yellow and red activity respectively in the ActivitiesLaunchDemo app,
– $\mathsf{Lmd}(A_g) = \mathsf{STD}$, $\mathsf{Lmd}(A_b) = \mathsf{STP}$, $\mathsf{Lmd}(A_y) = \mathsf{STK}$, $\mathsf{Lmd}(A_r) = \mathsf{SIT}$,
– $\mathsf{Aft}(A_g) = \mathsf{Aft}(A_b) = \mathsf{Aft}(A_r) = 1$, $\mathsf{Aft}(A_y) = 2$,

and Δ comprises the transitions illustrated in Fig. 2. Below is a path in the graph $\xrightarrow{\mathcal{A}}$ corresponding to the sequence of user actions clicking the green, blue, blue, yellow, red, blue button (cf. Example 1),

$$
\begin{aligned}
&(q_0, \varepsilon) \xrightarrow{\triangleright, \mathsf{start}(A_g)} (q_1, ([A_g])) \xrightarrow{A_g, \mathsf{start}(A_b)} (q_1, ([A_b, A_g])) \xrightarrow{A_b, \mathsf{start}(A_b)} \\
&(q_1, ([A_b, A_g])) \xrightarrow{A_b, \mathsf{start}(A_y)} (q_1, ([A_y], [A_b, A_g])) \xrightarrow{A_y, \mathsf{start}(A_r)} \\
&(q_1, ([A_r], [A_y], [A_b, A_g])) \xrightarrow{A_r, \mathsf{start}(A_g)} (q_1, ([A_g, A_b, A_g], [A_r], [A_y])).
\end{aligned}
$$

Proposition 1 reassures that $\xrightarrow{\mathcal{A}}$ is indeed a relation on $\mathsf{Conf}_{\mathcal{A}}$ as per Definition 2.

Proposition 1. *Let $\mathcal{A}$ be an ASM. For each $(q, \rho) \in \mathsf{Conf}_{\mathcal{A}}$ and $(q, \rho) \xrightarrow{\mathcal{A}} (q', \rho')$, $(q', \rho') \in \mathsf{Conf}_{\mathcal{A}}$, namely, (q', ρ') satisfies the five constraints in Definition 2.*

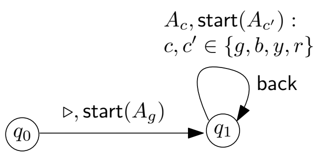
$A_c, \mathsf{start}(A_{c'})$:
$c, c' \in \{g, b, y, r\}$
back
$\triangleright, \mathsf{start}(A_g)$
q_0
q_1

Fig. 2: ASM corresponding to the ActivitiesLaunchDemo app

Remark 1. A single app can clearly be modeled by an ASM. However, ASM can also be used to model multiple apps which may share tasks/activities. (In this case, these multiple apps can be composed into a single app, where a new main activity is added.) This is especially useful when analysing, for instance, task hijacking [16]. We sometimes do not specify the main activity explicit for convenience. The translation from app source code to ASM is not trivial, but follows standard routines. In particular, in ASM, the symbols stored into the back stack are just names of activities. Android apps typically need to, similar to function calls of programs, store additional local state information. This can be dealt with by introducing an extend activity alphabet such that each symbol is of the form $A(\boldsymbol{b})$, where $A \in \mathsf{Act}$ and $\boldsymbol{b}$ represents local information. When we present examples, we also adopt this general syntax.

[8] Note that B occurs at most once in S_i by Definition 2(1).

Model validation. We validate the ASM model by designing "diagnosis" Android apps with extensive experiments. For each case in the semantics of ASM, we design an app which contains activities with the corresponding launch modes and task affinities. To simulate the transition rules of the ASM, each activity contains some buttons, which, when clicked, will launch other activities. For instance, in the case of $\mathsf{Lmd}(B) = \mathsf{STD}$, $\mathsf{Lmd}(A) = \mathsf{SIT}$, $\mathsf{GetNonSITTaskByAft}(\rho, \mathsf{Aft}(B)) = \mathsf{Undef}$, the app contains two activities A and B of launch modes SIT and STD respectively, where A is the main activity. When the app is launched, an instance of A is started. A contains a button, which, when clicked, starts an instance of B. We carry out the experiment by clicking the button, monitoring the content of the back stack, and checking whether the content of the back stack conforms to the definition of the semantics. Specifically, we check that there are exactly two tasks in the back stack, one task comprising a single instance of A and another task comprising a single instance of B, with the latter task on the top. Our experiments are done in a Redmi-4A mobile phone with Android version 6.0.1. The details of the experiments can be found at `https://sites.google.com/site/assconformancetesting/`.

4 Reachability of ASM

Towards formal (static) analysis and verification of Android apps, we study the fundamental *reachability* problem of ASM. Fix an ASM $\mathcal{A} = (Q, \mathsf{Sig}, q_0, \Delta)$ with $\mathsf{Sig} = (\mathsf{Act}, \mathsf{Lmd}, \mathsf{Aft}, A_0)$ and a *target state* $q \in Q$. There are usually two variants: the *state reachability problem* asks whether $(q_0, \varepsilon) \overset{\mathcal{A}}{\Rightarrow} (q, \rho)$ for *some* back stack ρ, and the *configuration reachability problem* asks whether $(q_0, \varepsilon) \overset{\mathcal{A}}{\Rightarrow} (q, \rho)$ when ρ is also given. We show they are interchangeable as far as decidability is concerned.

Proposition 2. *The configuration reachability problem and the state reachability problem of ASM are interreducible in exponential time.*

Proposition 2 allows to focus on the state reachability problem in the rest of this paper. Observe that, when the activities in an ASM are of the same launch mode, the problem degenerates to that of standard pushdown systems or even finite-state systems. These systems are well-understood, and we refer to [6] for explanations. To proceed, we deal with the cases where there are exactly two launch modes, for which we have $\binom{4}{2} = 6$ possibilities. The classification is given in Theorem 1–2. Clearly, they entail that the reachability for general ASM (with at least two launch modes) is undecidable. To show the undecidablity, we reduce from Minsky's two-counter machines [14], which, albeit standard, reveals the expressibility of ASM. We remark that the capability of *swapping the order* of two distinct non-SIT-tasks in the back stack—*without resetting* the content of any of them—is the main source of undecidability.

Theorem 1. *The reachability problem of ASM is undecidable, even when the ASM contains only (1)* STD *and* STK *activities, or (2)* STD *and* SIT *activities, or (3)* STK *and* STP *activities, or (4)* SIT *and* STP *activities.*

In contrast, we have some relatively straightforward positive results:

Theorem 2. *The state reachability problem of ASM is decidable in polynomial time when the ASM contains* STD *and* STP *activities only, and in polynomial space when the ASM contains* STK *and* SIT *activities only.*

As mentioned in Section 1, we aim to identify expressive fragments of ASM with decidable reachability problems. To this end, we introduce a fragment called **STK-dominating ASM**, which accommodates all four launch modes.

Definition 3 (STK-dominating ASM). *An ASM is said to be* STK-dominating *if the following two constraints are satisfied:*

(1) the task affinities of the STK *activities are mutually distinct,*
(2) for each transition $q \xrightarrow{A,\mathsf{start}(B)} q' \in \Delta$ *such that* $A \in \mathsf{Act}_{\mathsf{SIT}}$*, it holds that either* $B \in \mathsf{Act}_{\mathsf{SIT}} \cup \mathsf{Act}_{\mathsf{STK}}$*, or* $B \in \mathsf{Act}_{\mathsf{STD}} \cup \mathsf{Act}_{\mathsf{STP}}$ *and* $\mathsf{Aft}(B) = \mathsf{Aft}(A_0)$*.*

The following result explains the name "STK-dominating".

Proposition 3. *Let* $\mathcal{A} = (Q, \mathsf{Sig}, q_0, \Delta)$ *be an* STK*-dominating ASM with* $\mathsf{Sig} = (\mathsf{Act}, \mathsf{Lmd}, \mathsf{Aft}, A_0)$*. Then each configuration* (q, ρ) *that is reachable from the initial configuration* (q_0, ε) *in* $\mathcal{A}$ *satisfies the following constraints: (1) for each* STK *activity* $A \in \mathsf{Act}$ *with* $\mathsf{Aft}(A) \neq \mathsf{Aft}(A_0)$*,* A *can only occur at the bottom of some task in* ρ*, (2)* ρ *contains at most one* $\mathsf{STD}/\mathsf{STP}$*-task, which, when it exists, has the same affinity as* A_0*.*

It is not difficult to verify that the ASM given in Example 2 is STK-dominating.

Theorem 3. *The state reachability of* STK*-dominating ASM is in* $2\text{-}\mathsf{EXPTIME}$*.*

The proof of Theorem 3 is technically the most challenging part of this paper. We shall give a sketch in Section 5 with the full details in [6].

5 STK-dominating ASM

For simplicity, we assume that $\mathcal{A}$ **contains STD and STK activities only**[9]. To tackle the (state) reachability problem for STK-dominating ASM, we consider two cases, i.e., $\mathsf{Lmd}(A_0) = \mathsf{STK}$ and $\mathsf{Lmd}(A_0) \neq \mathsf{STK}$. The former case is simpler because, by Proposition 3, all tasks will be rooted at STK activities. For the latter, more general case, the back stack may contain, apart from several tasks rooted at STK activities, one single task rooted at A_0. Section 5.1 and Section 5.2 will handle these two cases respectively.

We will, however, first introduce some standard, but necessary, backgrounds on pushdown systems. We assume familiarity with standard *finite-state automata* (NFA) and *finite-state transducers* (FST). We emphasize that, in this paper,

[9] The more general case that $\mathcal{A}$ also contains STP and SIT activities is slightly more involved and requires more space to present, which can be found in [6].

FST refers to a special class of finite-state transducers, namely, *letter-to-letter* finite-state transducers where the input and output alphabets are the same.

Preliminaries of Pushdown systems. A *pushdown system* (PDS) is a tuple $\mathcal{P} = (Q, \Gamma, \Delta)$, where Q is a finite set of *control states*, Γ is a finite *stack alphabet*, and $\Delta \subseteq Q \times \Gamma \times \Gamma^* \times Q$ is a finite set of transition rules. The size of $\mathcal{P}$, denoted by $|\mathcal{P}|$, is defined as $|\Delta|$.

Let $\mathcal{P} = (Q, \Gamma, \Delta)$ be a PDS. A *configuration* of $\mathcal{P}$ is a pair $(q, w) \in Q \times \Gamma^*$, where w denotes the *content* of the stack (with the leftmost symbol being the top of the stack). Let $\mathsf{Conf}_{\mathcal{P}}$ denote the set of configurations of $\mathcal{P}$. We define a binary relation $\xrightarrow{\mathcal{P}}$ over $\mathsf{Conf}_{\mathcal{P}}$ as follows: $(q, w) \xrightarrow{\mathcal{P}} (q', w')$ iff $w = \gamma w_1$ and there exists $w'' \in \Gamma^*$ such that $(q, \gamma, w'', q') \in \Delta$ and $w' = w'' w_1$. We use $\overset{\mathcal{P}}{\Rightarrow}$ to denote the *reflexive and transitive closure* of $\xrightarrow{\mathcal{P}}$.

A configuration (q', w') is *reachable* from (q, w) if $(q, w) \overset{\mathcal{P}}{\Rightarrow} (q', w')$. For $C \subseteq \mathsf{Conf}_{\mathcal{P}}$, $\mathsf{pre}^*(C)$ (resp. $\mathsf{post}^*(C)$) denotes the set of *predecessor* (resp. *successor*) reachable configurations $\{(q', w') \mid \exists (q, w) \in C, (q', w') \overset{\mathcal{P}}{\Rightarrow} (q, w)\}$ (resp. $\{(q', w') \mid \exists (q, w) \in C, (q, w) \overset{\mathcal{P}}{\Rightarrow} (q', w')\}$). For $q \in Q$, we define $C_q = \{q\} \times \Gamma^*$ and write $\mathsf{pre}^*(q)$ and $\mathsf{post}^*(q)$ as shorthand of $\mathsf{pre}^*(C_q)$ and $\mathsf{post}^*(C_q)$ respectively.

As a standard machinery to solve reachability for PDS, a *$\mathcal{P}$-multi-automaton* ($\mathcal{P}$-MA) is an NFA $\mathcal{A} = (Q', \Gamma, \delta, I, F)$ such that $I \subseteq Q \subseteq Q'$ [4]. Evidently, multi-automata are a special class of NFA. Let $\mathcal{A} = (Q', \Gamma, \delta, I, F)$ be a $\mathcal{P}$-MA and $(q, w) \in \mathsf{Conf}_{\mathcal{P}}$, (q, w) is *accepted* by $\mathcal{A}$ if $q \in I$ and there is an accepting run $q_0 q_1 \cdots q_n$ of $\mathcal{A}$ on w with $q_0 = q$. Let $\mathsf{Conf}_{\mathcal{A}}$ denote the set of configurations accepted by $\mathcal{A}$. Moreover, let $\mathcal{L}(\mathcal{A})$ denote the set of words w such that $(q, w) \in \mathsf{Conf}_{\mathcal{A}}$ for some $q \in I$. For brevity, we usually write MA instead of $\mathcal{P}$-MA when $\mathcal{P}$ is clear from the context. Moreover, for an MA $\mathcal{A} = (Q', \Gamma, \delta, I, F)$ and $q' \in Q$, we use $\mathcal{A}(q')$ to denote the MA obtained from $\mathcal{A}$ by replacing I with $\{q'\}$. A set of configurations $C \subseteq \mathsf{Conf}_{\mathcal{P}}$ is *regular* if there is an MA $\mathcal{A}$ such that $\mathsf{Conf}_{\mathcal{A}} = C$.

Theorem 4 ([4]). *Given a PDS $\mathcal{P}$ and a set of configurations accepted by an MA $\mathcal{A}$, we can compute, in polynomial time in $|\mathcal{P}| + |\mathcal{A}|$, two MAs $\mathcal{A}_{\mathsf{pre}^*}$ and $\mathcal{A}_{\mathsf{post}^*}$ that recognise $\mathsf{pre}^*(\mathsf{Conf}_{\mathcal{A}})$ and $\mathsf{post}^*(\mathsf{Conf}_{\mathcal{A}})$ respectively.*

The connection between ASM and PDS is rather obvious. In a nutshell, ASM can be considered as a PDS with *multiple* stacks, which is well-known to be undecidable in general. Our overall strategy to attack the state reachability problem for the fragments of ASM is to simulate them (in particular, the multiple stacks) via—in some cases, decidable extensions of—PDS.

5.1 Case $\mathsf{Lmd}(A_0) = \mathsf{STK}$

Our approach to tackle this case is to simulate $\mathcal{A}$ by an *extension* of PDS, i.e., *pushdown systems with transductions* (TrPDS), proposed in [19]. In TrPDS, each transition is associated with an FST defining how the stack content is modified. Formally, a TrPDS is a tuple $\mathcal{P} = (Q, \Gamma, \mathscr{T}, \Delta)$, where Q and Γ are precisely the same as those of PDS, $\mathscr{T}$ is a finite set of FSTs over the alphabet Γ, and $\Delta \subseteq Q \times \Gamma \times \Gamma^* \times \mathscr{T} \times Q$ is a finite set of transition rules. Let $\mathcal{R}(\mathscr{T})$ denote the set of transductions defined by FSTs from $\mathscr{T}$ and $[\![\mathcal{R}(\mathscr{T})]\!]$ denote the *closure* of $\mathcal{R}(\mathscr{T})$ under composition and left-quotient. A TrPDS $\mathcal{P}$ is said to be *finite* if $[\![\mathcal{R}(\mathscr{T})]\!]$ is finite.

The configurations of $\mathcal{P}$ are defined similarly as in PDS. We define a binary relation $\xrightarrow{\mathcal{P}}$ on $\mathsf{Conf}_{\mathcal{P}}$ as follows: $(q, w) \xrightarrow{\mathcal{P}} (q', w')$ if there are $\gamma \in \Gamma$, the words w_1, u, w_2, and $\mathcal{T} \in \mathscr{T}$ such that $w = \gamma w_1$, $(q, \gamma, u, \mathcal{T}, q') \in \Delta$, $w_1 \xrightarrow{\mathcal{T}} w_2$, and $w' = uw_2$. Let $\overset{\mathcal{P}}{\Rightarrow}$ denote the reflexive and transitive closure of $\xrightarrow{\mathcal{P}}$. Similarly to PDS, we can define $\mathsf{pre}^*(\cdot)$ and $\mathsf{post}^*(\cdot)$ respectively. Regular sets of configurations of TrPDS can be represented by MA, in line with PDS. More precisely, given a finite TrPDS $\mathcal{P} = (Q, \Gamma, \mathscr{T}, \Delta)$ and an MA $\mathcal{A}$ for $\mathcal{P}$, one can compute, in time polynomial in $|\mathcal{P}| + |[\![\mathcal{R}(\mathscr{T})]\!]| + |\mathcal{A}|$, two MAs $\mathcal{A}_{\mathsf{pre}^*}$ and $\mathcal{A}_{\mathsf{post}^*}$ that recognize the sets $\mathsf{pre}^*(\mathsf{Conf}_{\mathcal{A}})$ and $\mathsf{post}^*(\mathsf{Conf}_{\mathcal{A}})$ respectively [19,18,17].

To simulate $\mathcal{A}$ via a finite TrPDS $\mathcal{P}$, the back stack $\rho = (S_1, \cdots, S_n)$ of $\mathcal{A}$ is encoded by a word $S_1 \sharp \cdots \sharp S_n \sharp \bot$ (where $\sharp$ is a delimiter and $\bot$ is the bottom symbol of the stack), which is stored in the stack of $\mathcal{P}$. Recall that, in this case, each task S_i is rooted at an STK-activity which sits on the bottom of S_i. Suppose $\mathsf{top}(S_1) = A$. When a transition $(q, A, \mathsf{start}(B), q')$ with $B \in \mathsf{Act}_{\mathsf{STK}}$ is fired, according to the semantics of $\mathcal{A}$, the B-task of ρ, say S_i, is switched to the top of ρ and changed into $[B]$ (i.e., all the activities in the B-task, except B itself, are popped). To simulate this in $\mathcal{P}$, we replace every stack symbol in the place of S_i with a dummy symbol $\dagger$ and keep the other symbols unchanged. On the other hand, to simulate a back action of $\mathcal{A}$, $\mathcal{P}$ continues popping until the next non-dummy and non-delimiter symbol is seen.

Proposition 4. *Let $\mathcal{A} = (Q, \mathsf{Sig}, q_0, \Delta)$ be an STK-dominating ASM with $\mathsf{Sig} = (\mathsf{Act}, \mathsf{Lmd}, \mathsf{Aft}, A_0)$ and $\mathsf{Lmd}(A_0) = \mathsf{STK}$. Then a finite TrPDS $\mathcal{P} = (Q', \Gamma, \mathscr{T}, \Delta')$ with $Q \subseteq Q'$ can be constructed in time polynomial in $|\mathcal{A}|$ such that, for each $q \in Q$, q is reachable from (q_0, ε) in $\mathcal{A}$ iff q is reachable from $(q_0, \bot)$ in $\mathcal{P}$.*

For a state $q \in Q$, $\mathsf{pre}^*_{\mathcal{P}}(q)$ can be effectively computed as an MA $\mathcal{B}_q$, and the reachability of q in $\mathcal{A}$ is reduced to checking whether $(q_0, \bot) \in \mathsf{Conf}_{\mathcal{B}_q}$.

5.2 Case $\mathsf{Lmd}(A_0) \neq \mathsf{STK}$

We then turn to the more general case $\mathsf{Lmd}(A_0) \neq \mathsf{STK}$ which is significantly more involved. For exposition purpose, we consider an ASM $\mathcal{A}$ where **there are exactly two STK activities** A_1, A_2, and the task affinity of A_2 is the same as that of the main task A_0 (and thus the task affinity of A_1 is different from that of A_0). We also assume that all the activities in $\mathcal{A}$ are "standard" except A_1, A_2. Namely $\mathsf{Act} = \mathsf{Act}_{\mathsf{STD}} \cup \{A_1, A_2\}$ and $A_0 \in \mathsf{Act}_{\mathsf{STD}}$ in particular. Neither of these two assumptions is fundamental and their generalization is given in [6].

By Proposition 3, there are at most two tasks in the back stack of $\mathcal{A}$. The two tasks are either an A_0-task and an A_1-task, or an A_2-task and an A_1-task. An A_2-task can only surface when the original A_0-task is popped empty. If this happens, no A_0-task will be recreated again, and thus, according to the arguments in Section 5.1, we can simulate the ASM by TrPDS directly and we are done. The challenging case is that we have both an A_0-task and an A_1-task. To solve the state reachability problem, the main technical difficulty is that the order of the A_0-task and the A_1-task may be switched for arbitrarily many times before reaching the target state q. Readers may be wondering why they *cannot* simply simulate two-counter machines. The reason is that the two tasks are *asymmetric* in the sense that, each time when the A_1-task is switched from the bottom to the top (by starting the activity A_1), the content of the A_1-task is reset into $[A_1]$. But this is *not* the case for A_0-task: when the A_0-task is switched from the bottom to the top (by starting the activity A_2), if it does not contain A_2, then A_2 will be pushed into the A_0-task; otherwise all the activities above A_2 will be popped and A_2 becomes the top activity of the A_0-task. Our decision procedure below utilises the asymmetry of the two tasks.

Intuition of construction. The crux of reachability analysis is to construct a *finite abstraction* for the A_1-task and incorporate it into the control states of $\mathcal{A}$, so we can reduce the state reachability of $\mathcal{A}$ into that of a pushdown system $\mathcal{P}_{\mathcal{A}}$ (with a single stack). Observe that a run of $\mathcal{A}$ can be seen as a sequence of task switching. In particular, an $A_0; A_1; A_0$ *switching* denotes a path in $\xrightarrow{\mathcal{A}}$ where the A_0-task is on the top in the *first* and the *last* configuration, while the A_1-task is on the top in all the *intermediate* configurations. The main idea of the reduction is to simulate the $A_0; A_1; A_0$ switching by a "macro"-transition of $\mathcal{P}_{\mathcal{A}}$. Note that the A_0-task regains the top task in the last configuration either by starting the activity A_2 or by emptying the A_1-task. Suppose that, for an $A_0; A_1; A_0$ switching, in the first (resp. last) configuration, q (resp. q') is the control state and α (resp. β) is the finite abstraction of the A_1-task. Then for the "macro"-transition of $\mathcal{P}_{\mathcal{A}}$, the control state will be updated from (q, α) to (q', β), and the stack content of $\mathcal{P}_{\mathcal{A}}$ is updated accordingly:

– If the A_0-task regains the top task by starting A_2, then the stack content is updated as follows: if the stack does not contain A_2, then A_2 will be pushed into the stack; otherwise all the symbols above A_2 will be popped.
– On the other hand, if the A_0-task regains the top task by emptying the A_1-task, then the stack content is not changed.

Roughly speaking, the abstraction of the A_1-task must carry the information that, when A_0-task and A_1-task are the top resp. bottom task of the back stack and A_0-task is emptied, whether the target state q can be reached from the configuration at that time. As a result, we define the abstraction of the A_1-task whose content is encoded by a word $w \in \mathsf{Act}^*$, denoted by $\alpha(w)$, as the set of all states $q'' \in Q$ such that the target state q can be reached from $(q'', (w))$ in $\mathcal{A}$. [Note that during the process that q is reached from $(q'', (w))$ in $\mathcal{A}$, the A_0-task does not exist anymore, but a (new) A_2-task, may be formed.] Let $\mathsf{Abs}_{A_1} = 2^Q$.

To facilitate the construction of the PDS $\mathcal{P}_{\mathcal{A}}$, we also need to record how the abstraction "evolves". For each $(q', A, \alpha) \in Q \times (\mathsf{Act} \setminus \{A_1\}) \times \mathsf{Abs}_{A_1}$, we compute the set $\mathsf{Reach}(q', A, \alpha)$ consisting of pairs (q'', β) satisfying: there is an $A_0; A_1; A_0$

switching such that in the first configuration, A is the top symbol of the A_0-task, q' (resp. q'') is the control state of the first (resp. last) configuration, and α (resp. β) is the abstraction for the A_1-task in the first (resp. last) configuration.[10]

Computing $\mathsf{Reach}(q', A, \alpha)$. Let $(q', A, \alpha) \in Q \times (\mathsf{Act} \setminus \{A_1\}) \times \mathsf{Abs}_{A_1}$. We first simulate relevant parts of $\mathcal{A}$ as follows:

– Following Section 5.1, we construct a TrPDS $\mathcal{P}_{\boxed{A_0}} = (Q_{\boxed{A_0}}, \Gamma_{\boxed{A_0}}, \mathscr{T}_{\boxed{A_0}}, \Delta_{\boxed{A_0}})$ to simulate *the* A_1*-task and* A_2*-task* of $\mathcal{A}$ after the A_0-task is emptied, where $Q_{\boxed{A_0}} = Q \cup Q \times Q$ and $\Gamma_{\boxed{A_0}} = \mathsf{Act} \cup \{\sharp, \dagger, \bot\}$. Note that A_0 may still—as a "standard" activity—occur in $\mathcal{P}_{\boxed{A_0}}$ though the A_0-task disappears.
  In addition, we construct an MA $\mathcal{B}_q = (Q_q, \Gamma_{\boxed{A_0}}, \delta_q, I_q, F_q)$ to represent $\mathsf{pre}^*_{\mathcal{P}_{\boxed{A_0}}}(q)$, where $I_q \subseteq Q_{\boxed{A_0}}$. Then given a stack content $w \in \mathsf{Act}^*_{\mathsf{STD}} A_1$ of the A_1-task, the abstraction $\alpha(w)$ of w, is the set of $q'' \in I_q \cap Q$ such that $(q'', w\sharp\bot) \in \mathsf{Conf}_{\mathcal{B}_q}$.
– We construct a PDS $\mathcal{P}_{\boxed{A_0, A_2}} = (Q_{\boxed{A_0, A_2}}, \Gamma_{\boxed{A_0, A_2}}, \mathscr{T}_{\boxed{A_0, A_2}}, \Delta_{\boxed{A_0, A_2}})$ to simulate the A_1*-task* of $\mathcal{A}$, where $\Gamma_{\boxed{A_0, A_2}} = (\mathsf{Act} \setminus \{A_2\}) \cup \{\bot\}$. In addition, to compute $\mathsf{Reach}(q', A, \alpha)$ later, we construct an MA $\mathcal{M}_{(q',A,\alpha)} = (Q_{(q',A,\alpha)}, \Gamma_{\boxed{A_0, A_2}}, \delta_{(q',A,\alpha)}, I_{(q',A,\alpha)}, F_{(q',A,\alpha)})$ to represent

$$\mathsf{post}^*_{\mathcal{P}_{\boxed{A_0, A_2}}}(\{(q_1, A_1\bot) \mid (q', A, \mathsf{start}(A_1), q_1) \in \Delta\}).$$

Definition 4. $\mathsf{Reach}(q', A, \alpha)$ *comprises*

– *the pairs* $(q'', \beta) \in Q \times \mathsf{Abs}_{A_1}$ *satisfying that (1)* $(q', A, \mathsf{start}(A_1), q_1) \in \Delta$, *(2)* $(q_1, A_1\bot) \xRightarrow{\mathcal{P}_{\boxed{A_0, A_2}}} (q_2, Bw\bot)$, *(3)* $(q_2, B, \mathsf{start}(A_2), q'') \in \Delta$, *and (4)* β *is the abstraction of* Bw, *for some* $B \in \mathsf{Act} \setminus \{A_2\}$, $w \in (\mathsf{Act} \setminus \{A_2\})^*$ *and* $q_1, q_2 \in Q$,
– *the pairs* $(q'', \bot)$ *such that* $(q', A, \mathsf{start}(A_1), q_1) \in \Delta$ *and* $(q_1, A_1\bot) \xRightarrow{\mathcal{P}_{\boxed{A_0, A_2}}} (q'', \bot)$ *for some* $q_1 \in Q$.

Importantly, conditions in Definition 4 can be characterized algorithmically.

Lemma 1. *For* $(q', A, \alpha) \in Q \times (\mathsf{Act} \setminus \{A_1\}) \times \mathsf{Abs}_{A_1}$, $\mathsf{Reach}(q', A, \alpha)$ *is the union of*

– $\{(q'', \bot) \mid (q'', \bot) \in \mathsf{Conf}_{\mathcal{M}_{(q',A,\alpha)}}\}$ *and*
– *the set of pairs* $(q'', \beta) \in Q \times \mathsf{Abs}_{A_1}$ *such that there exist* $q_2 \in Q$ *and* $B \in \mathsf{Act} \setminus \{A_2\}$ *satisfying that* $(q_2, B, \mathsf{start}(A_2), q'')$, *and*

  $$(B(\mathsf{Act} \setminus \{A_2\})^*\sharp\bot) \cap (\mathsf{Act}^*_{\mathsf{STD}} A_1 \sharp\bot) \cap (\mathcal{L}(\mathcal{M}_{(q',A,\alpha)}(q_2))\langle\bot\rangle^{-1})\sharp\bot \cap \mathcal{L}_\beta \neq \emptyset,$$

  where $\mathcal{L}(\mathcal{M}_{(q',A,\alpha)}(q_2))\langle\bot\rangle^{-1}$ *is the set of words* w *such that* $w\bot$ *belongs to* $\mathcal{L}(\mathcal{M}_{(q',A,\alpha)}(q_2))$, *and* $\mathcal{L}_\beta = \bigcap\limits_{q''' \in \beta} \mathcal{L}(\mathcal{B}_q(q''')) \cap \bigcap\limits_{q''' \in Q \setminus \beta} \overline{\mathcal{L}(\mathcal{B}_q(q'''))}$, *with* $\overline{\mathcal{L}}$ *representing the complement language of* $\mathcal{L}$.

[10] As we can see later, $\mathsf{Reach}(q', A, \alpha)$ does not depend on α for the two-task special case considered here. We choose to keep α in view of readability.

Construction of $\mathcal{P}_{\mathcal{A}}$. We first construct a PDS $\mathcal{P}_{A_0} = (Q_{A_0}, \Gamma_{A_0}, \Delta_{A_0})$, to simulate the A_0-task of $\mathcal{A}$. Here $Q_{A_0} = (Q \times \{0,1\}) \cup (Q \times \{1\} \times \{\mathsf{pop}\})$, $\Gamma_{A_0} = \mathsf{Act}_{\mathsf{STD}} \cup \{A_2, \bot\}$, and Δ_{A_0} comprises the transitions. Here 1 (resp. 0) marks that the activity A_2 is in the stack (resp. is not in the stack) and the tag pop marks that the PDS is in the process of popping until A_2. The construction of $\mathcal{P}_{A_0}$ is relatively straightforward, the details of which can be found in [6].

We then define the PDS $\mathcal{P}_{\mathcal{A}} = (Q_{\mathcal{A}}, \Gamma_{A_0}, \Delta_{\mathcal{A}})$, where $Q_{\mathcal{A}} = (\mathsf{Abs}_{A_1} \times Q_{A_0}) \cup \{q\}$, and $\Delta_{\mathcal{A}}$ comprises the following transitions,

– for each $(p, \gamma, w, p') \in \Delta_{A_0}$ and $\alpha \in \mathsf{Abs}_{A_1}$, we have $((\alpha, p), \gamma, w, (\alpha, p')) \in \Delta_{\mathcal{A}}$ (here $p, p' \in Q_{A_0}$, that is, of the form (q', b) or (q', b, pop)), **[behaviour of the A_0-task]**
– for each $(q', A, \alpha) \in Q \times (\mathsf{Act} \setminus \{A_1\}) \times \mathsf{Abs}_{A_1}$ and $b \in \{0,1\}$ such that $\mathcal{M}_{(q',A,\alpha)}(q) \neq \emptyset$, we have $((\alpha, (q', b)), A, A, q) \in \Delta_{\mathcal{A}}$, **[switch to the A_1-task and reach q before switching back]**
– for each $(q', A, \alpha) \in Q \times (\mathsf{Act} \setminus \{A_1\}) \times \mathsf{Abs}_{A_1}$ and $(q'', \beta) \in \mathsf{Reach}(q', A, \alpha)$ such that $\beta \neq \bot$,
  - if $A \neq A_2$, then we have $((\alpha, (q', 0)), A, A_2 A, (\beta, (q'', 1))) \in \Delta_{\mathcal{A}}$ and $((\alpha, (q', 1)), A, \varepsilon, (\beta, (q'', 1, \mathsf{pop}))) \in \Delta_{\mathcal{A}}$,
  - if $A = A_2$, then we have $((\alpha, (q', 1)), A_2, A_2, (\beta, (q'', 1))) \in \Delta_{\mathcal{A}}$,

  [switch to the A_1-task and switch back to the A_0-task later by launching A_2]
– for each $(q', A, \alpha) \in Q \times (\mathsf{Act} \setminus \{A_1\}) \times \mathsf{Abs}_{A_1}$, $(q'', \bot) \in \mathsf{Reach}(q', A, \alpha)$ and $b \in \{0,1\}$, we have $((\alpha, (q', b)), A, A, (\emptyset, (q'', b))) \in \Delta_{\mathcal{A}}$,
  [switch to the A_1-task and switch back to the A_0-task later when the A_1-task becomes empty]
– for each $\alpha \in \mathsf{Abs}_{A_1}$, $b \in \{0,1\}$ and $A \in \mathsf{Act}_{\mathsf{STD}} \cup \{A_2\}$, $((\alpha, (q, b)), A, A, q) \in \Delta_{\mathcal{A}}$, **[$q$ is reached when the A_0-task is the top task]**
– for each $q' \in Q$ and $\alpha \in \mathsf{Abs}_{A_1}$ with $q' \in \alpha$, $((\alpha, (q', 0)), \bot, \bot, q) \in \Delta_{\mathcal{A}}$.
  [q is reached after the A_0-task becomes empty and the A_1-task becomes the top task]

Proposition 5. *Let $\mathcal{A}$ be an STK-dominating ASM where there are exactly two STK-activities A_1, A_2 and $\mathsf{Aft}(A_2) = \mathsf{Aft}(A_0)$. Then q is reachable from the initial configuration (q_0, ε) in $\mathcal{A}$ iff q is reachable from the initial configuration $((\emptyset, (q_0, 0)), \bot)$ in $\mathcal{P}_{\mathcal{A}}$.*

6 Related work

We first discuss *pushdown systems with multiple stacks* (MPDSs) which are the most relevant to ASM. (For space reasons we will skip results on general pushdown systems though.) A multitude of classes of MPDSs have been considered, mostly as a model for *concurrent* recursive programs. In general, an ASM can be encoded as an MPDS. However, this view is hardly profitable as general MPDSs are obviously Turing-complete, leaving the reachability problem undecidable.

To regain decidability at least for reachability, several subclasses of MPDSs were proposed in literature: (1) bounding the number of context-switches [15], or more generally, phases [10], scopes [11], or budgets [3]; (2) imposing a linear ordering on stacks and pop operations being reserved to the first non-empty stack [5]; (3) restricting control states (e.g., *weak* MPDSs [7]). However, our decidable subclasses of ASM admit none of the above bounded conditions. A unified and generalized criterion [12] based on MSO over graphs of bounded tree-width was proposed to show the decidability of the emptiness problem for several restricted classes of automata with auxiliary storage, including MPDSs, automata with queues, or a mix of them. Since ASMs work in a way fairly different from multi-stack models in the literature, it is unclear—literally for us—to obtain the decidability by using bounded tree-width approach. Moreover, [12] only provides decidability proofs, but without complexity upper bounds. Our decision procedure is based on symbolic approaches for pushdown systems, which provides complexity upper bounds and which is amenable to implementation.

Higher-order pushdown systems represent another type of generalization of pushdown systems through higher-order stacks, i.e., a nested "stack of stack" structure [13], with decidable reachability problems [9]. Despite apparent resemblance, the back stack of ASM can *not* be simulated by an order-2 pushdown system. The reason is that the order between tasks in a back stack may be dynamically changed, which is not supported by order-2 pushdown systems.

On a different line, there are some models which have addressed, for instance, GUI activities of Android apps. *Window transition graphs* were proposed for representing the possible GUI activity (window) sequences and their associated events and callbacks, which can capture how the events and callbacks modify the back stack [21]. However, the key mechanisms of back stacks (launch modes and task affinities) were not covered in this model. Moreover, the reachability problem for this model was not investigated. A similar model, labeled transition graph with stack and widget (LATTE [20]) considered the effects of launch modes on the back stacks, but not task affinities. LATTE is essentially a finite-state abstraction of the back stack. However, to faithfully capture the launch modes and task affinities, one needs an infinite-state system, as we have studied here.

7 Conclusion

In this paper, we have introduced Android stack machine to formalize the back stack system of the Android platform. We have also investigated the decidability of the reachability problem of ASM. While the reachability problem of ASM is undecidable in general, we have identified a fragment, i.e., STK-dominating ASM, which is expressive and admits decision procedures for reachability.

The implementation of the decision procedures is in progress. We also plan to consider other features of Android back stack systems, e.g., the "allowTaskReparenting" attribute of activities. A long-term program is to develop an efficient and scalable formal analysis and verification framework for Android apps, towards which the work reported in this paper is the first cornerstone.

References

1. Android documentation. `https://developer.android.com/guide/components/activities/tasks-and-back-stack.html`.
2. Stackoverflow entry: Android singletask or singleinstance launch mode? `https://stackoverflow.com/questions/3219726/`.
3. Parosh Aziz Abdulla, Mohamed Faouzi Atig, Othmane Rezine, and Jari Stenman. Multi-pushdown systems with budgets. In *Formal Methods in Computer-Aided Design (FMCAD)*, pages 24–33, 2012.
4. Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proceedings of the 8th International Conference on Concurrency Theory (CONCUR)*, pages 135–150, 1997.
5. Luca Breveglieri, Alessandra Cherubini, Claudio Citrini, and Stefano Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996.
6. Taolue Chen, Jinlong He, Fu Song, Guozhen Wang, Zhilin Wu, and Jun Yan. Android stack machine (full version), 2018. `http://www.dcs.bbk.ac.uk/~taolue/pub-papers/ASM-full.pdf`.
7. Wojciech Czerwinski, Piotr Hofman, and Slawomir Lasota. Reachability problem for weak multi-pushdown automata. In *Proceedings of the 23rd International Conference on (CONCUR)*, pages 53–68, 2012.
8. Shuai Hao, Ding Li, William G. J. Halfond, and Ramesh Govindan. Estimating mobile application energy consumption using program analysis. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, pages 92–101, 2013.
9. Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-order pushdown trees are easy. In *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures (FOSSACS)*, pages 205–222, 2002.
10. Salvatore La Torre, Parthasarathy Madhusudan, and Gennaro Parlato. A robust class of context-sensitive languages. In *Proceedings of the 22nd IEEE Symposium on Logic in Computer Science (LICS)*, pages 161–170, 2007.
11. Salvatore La Torre and Margherita Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In *Proceedings of the 22nd International Conference on Concurrency Theory (CONCUR)*, pages 203–218, 2011.
12. P. Madhusudan and Gennaro Parlato. The tree width of auxiliary storage. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 283–294, 2011.
13. A. N. Maslov. Multilevel stack automata. *Problems of Information Transmission*, 15:1170–1174, 1976.
14. M. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall Int., 1967.
15. Shaz Qadeer and Jakob Rehof. Context-bounded model checking of concurrent software. In *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 93–107, 2005.
16. Chuangang Ren, Yulong Zhang, Hui Xue, Tao Wei, and Peng Liu. Towards discovering and understanding task hijacking in android. In *Proceedings of the 24th USENIX Security Symposium (USENIX Security)*, pages 945–959, 2015.
17. Fu Song. Analyzing pushdown systems with stack manipulation. *Information and Computation*, 259(1):41–71, 2018.

18. Fu Song, Weikai Miao, Geguang Pu, and Min Zhang. On reachability analysis of pushdown systems with transductions: Application to boolean programs with call-by-reference. In *Proceedings of the 26th International Conference on Concurrency Theory (CONCUR)*, pages 383–397, 2015.
19. Yuya Uezato and Yasuhiko Minamide. Pushdown systems with stack manipulation. In *Proceedings of the 11th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 412–426, 2013.
20. Jiwei Yan, Tianyong Wu, Jun Yan, and Jian Zhang. Widget-sensitive and back-stack-aware GUI exploration for testing android apps. In *Proceedings of the 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 42–53, 2017.
21. Shengqian Yang, Hailong Zhang, Haowei Wu, Yan Wang, Dacong Yan, and Atanas Rountev. Static window transition graphs for android. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 658–668, 2015.

A Preliminaries

For $k \in \mathbb{N}$, let $[k]$ denote $\{1, \cdots, k\}$. For a function $f : X \to Y$, let $\mathsf{dom}(f)$ and $\mathsf{rng}(f)$ denote the domain (X) and range (Y) of f respectively.

A.1 Finite-state automata

A *nondeterministic finite-state automaton* (NFA) is a tuple $\mathcal{A} = (Q, \varGamma, \delta, I, F)$, where Q is a finite set of *states*, $\varGamma$ is a finite *alphabet*, $\delta \subseteq Q \times \varGamma \times Q$ is a *transition* relation, $I \subseteq Q$ is a set of *initial* states, $F \subseteq Q$ is a set of *accepting* states. The *size* of $\mathcal{A}$, denoted by $|\mathcal{A}|$, is defined as the number of transitions in δ. For $q \in Q$, let $\mathcal{A}(q)$ denote the NFA obtained from $\mathcal{A}$ by replacing the set of initial states with $\{q\}$.

A (finite) *word* w is a finite sequence of symbols from $\varGamma$. We use $\varGamma^*$ to denote the set of words, and $\varepsilon \in \varGamma^*$ to denote the empty word. For a word w, let $|w|$ denote the length of w. Assume an NFA $\mathcal{A} = (Q, \varGamma, \delta, I, F)$ and word $w = \gamma_1 \cdots \gamma_n$. A *run* of $\mathcal{A}$ on w is a sequence of states $q_0 q_1 \cdots q_n$, such that $q_0 \in I$ and $(q_{i-1}, \gamma_i, q_i) \in \delta$ for each $i \in [n]$. The *run* $q_0 q_1 \cdots q_n$ is *accepting* if $q_n \in F$. A word w is *accepted* by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on w. In particular, ε is accepted by $\mathcal{A}$ iff $I \cap F \neq \emptyset$. Let $\mathcal{L}(\mathcal{A})$ denote the set of words accepted by $\mathcal{A}$. For an NFA $\mathcal{A}$ and $\gamma \in \varGamma$, we use $\mathcal{L}(\mathcal{A})\langle\gamma\rangle^{-1}$ to denote $\{w \in \varGamma^* \mid w\gamma \in \mathcal{L}(\mathcal{A})\}$.

A.2 Finite-state transducers

We also consider finite-state *transducers*, but focus on a special class of them, i.e., letter-to-letter finite-state transducers where the input and output alphabets are the same. A *letter-to-letter finite-state transducer* (FST for short) is a tuple $\mathcal{T} = (Q, \varGamma, \delta, I, F)$, where $Q, \varSigma, I, F$ are the same as those in NFA, while the transition relation δ is defined as $\delta \subseteq Q \times \varGamma \times \varGamma \times Q$.

Intuitively, a transition (q, γ, γ', q') means that when $\mathcal{T}$ is in the state q and reads the symbol γ, it may output a symbol γ' and move to the state q'. For readability, we usually write a transition $(q, \gamma, \gamma', q') \in \delta$ as $q \xrightarrow{\gamma,\gamma'} q'$. Similar to NFA, the *size* of $\mathcal{T}$, denoted by $|\mathcal{T}|$, is defined as the number of transitions in δ.

Let $\mathcal{T} = (Q, \varGamma, \delta, I, F)$ be an FST and $w = \gamma_1 \cdots \gamma_n$ be a word. Then a *run* of $\mathcal{T}$ on w is a sequence $q_0 \gamma'_1 q_1 \cdots \gamma'_n q_n$ such that $q_0 \in I$ and $q_{i-1} \xrightarrow{\gamma_i,\gamma'_i} q_i \in \delta$ for each $i \in [n]$. A run $q_0 \gamma'_1 q_1 \cdots \gamma'_n q_n$ is *accepting* if $q_n \in F$. If $q_0 \gamma'_1 q_1 \cdots \gamma'_n q_n$ is an accepting run of $\mathcal{T}$ on w, then $w' = \gamma'_1 \cdots \gamma'_n$ is said to be an *output* of $\mathcal{T}$ on w. In particular, ε is an output of $\mathcal{T}$ on ε iff $I \cap F \neq \emptyset$. We use $\mathcal{R}(\mathcal{T})$ to denote the set of pairs (w, w') such that w' is an output of $\mathcal{T}$ on w. For convenience, we also write $(w, w') \in \mathcal{R}(\mathcal{T})$ as $w \xrightarrow{\mathcal{T}} w'$.

We use $\mathcal{T}_{id}$ to denote the identity transducer, that is, the FST $(Q, \varGamma, \delta, I, F)$, where $Q = I = F = \{q_0\}$ and $\delta = \{(q_0, \gamma, \gamma, q_0) \mid \gamma \in \varGamma\}$. A relation $\tau \subseteq \varGamma^* \times \varGamma^*$ is called a *transduction* if $\tau = \mathcal{R}(\mathcal{T})$ for some FST $\mathcal{T}$. In particular, we use $\tau_\emptyset$ to denote the empty transduction (that is, the transduction relation is $\emptyset$), and τ_{id} to denote the identity transduction (that is, the relation $\{(w, w) \mid w \in \varGamma^*\}$).

Let τ, τ' be two transductions. The *composition* of τ and τ', denoted by $\tau \circ \tau'$, is defined as $\{(w, w') \mid \exists w''. \ (w, w'') \in \tau, (w'', w') \in \tau'\}$. For each transduction τ and $\mathcal{L} \subseteq \Gamma^*$, we use $\tau|_{\mathcal{L}}$ to denote the restriction of τ to the domain $\mathcal{L}$, that is, $\tau|_{\mathcal{L}} = \{(w, w') \in \tau \mid w \in \mathcal{L}\}$.

Let w, w' be two words of the same length. We define the *left-quotient* of τ w.r.t. (w, w'), denoted by $\langle w, w' \rangle^{-1}\tau$, inductively as follows: $\langle \varepsilon, \varepsilon \rangle^{-1}\tau = \tau$, $\langle \gamma, \gamma' \rangle^{-1}\tau = \{(w, w') \mid (\gamma w, \gamma' w') \in \tau\}$, and $\langle \gamma w, \gamma' w' \rangle^{-1}\tau = \langle w, w' \rangle^{-1}(\langle \gamma, \gamma' \rangle^{-1}\tau)$. Moreover, we define the *left-extension* of τ w.r.t. (w, w'), denoted by $\langle w, w' \rangle^{+1}\tau$, as follows: $\langle w, w' \rangle^{+1}\tau = \{(ww_1, w'w'_1) \mid (w_1, w'_1) \in \tau\}$.

Definition 5 (Closure of transductions). *Let $\mathscr{T}$ be a set of transductions. Then the* closure *of $\mathscr{T}$ under composition and left-quotient, denoted by $[\![\mathscr{T}]\!]$, is inductively defined as follows,*

- *$\mathscr{T} \subseteq [\![\mathscr{T}]\!]$, $\tau_\emptyset \in [\![\mathscr{T}]\!]$, $\tau_{id} \in [\![\mathscr{T}]\!]$,*
- *if $\tau_1, \tau_2 \in [\![\mathscr{T}]\!]$, then $\tau_1 \circ \tau_2 \in [\![\mathscr{T}]\!]$,*
- *if $\tau \in [\![\mathscr{T}]\!]$ and $\gamma, \gamma' \in \Gamma$, then $\langle \gamma, \gamma' \rangle^{-1}\tau \in [\![\mathscr{T}]\!]$.*

Recall that for a set of *transducers* $\mathscr{T}$, for brevity, we use $\mathcal{R}(\mathscr{T})$ to denote $\{\mathcal{R}(\mathcal{T}) \mid \mathcal{T} \in \mathscr{T}\}$.

A.3 Pushdown systems with transductions

A *pushdown system with transductions* (TrPDS) is a tuple $\mathcal{P} = (Q, \Gamma, \mathscr{T}, \Delta)$, where Q and Γ are exactly the same as those of pushdown systems, $\mathscr{T}$ is a finite set of FST over the alphabet Γ, and $\Delta \subseteq Q \times \Gamma \times \Gamma^* \times \mathscr{T} \times Q$ is a finite set of transition rules. A TrPDS $\mathcal{P}$ is said to be *finite* if $[\![\mathcal{R}(\mathscr{T})]\!]$ is finite.

Let $\mathcal{P} = (Q, \Gamma, \mathscr{T}, \Delta)$ be a TrPDS. The configurations of $\mathcal{P}$ are defined similarly as in PDS. Let $\mathsf{Conf}_{\mathcal{P}}$ denote the set of configurations of $\mathcal{P}$. We define a binary relation $\xrightarrow{\mathcal{P}}$ on $\mathsf{Conf}_{\mathcal{P}}$ as follows: $(q, w) \xrightarrow{\mathcal{P}} (q', w')$ if there are $\gamma \in \Gamma$, the words w_1, u, w_2, and $\mathcal{T} \in \mathscr{T}$ such that $w = \gamma w_1$, $(q, \gamma, u, \mathcal{T}, q') \in \Delta$, $w_1 \xrightarrow{\mathcal{T}} w_2$, and $w' = uw_2$.

Let $\overset{\mathcal{P}}{\Rightarrow}$ denote the reflexive and transitive closure of $\xrightarrow{\mathcal{P}}$. Similarly to PDS, we can define $\mathsf{pre}^*(\cdot)$ and $\mathsf{post}^*(\cdot)$ respectively. Moreover, in parallel to multi-automata, we use *NFA with transductions* to represent regular sets of configurations of TrPDS [19].

Given a finite TrPDS $\mathcal{P} = (Q, \Gamma, \mathscr{T}, \Delta)$, an *NFA with transductions* (TrNFA for short) for $\mathcal{P}$ is a tuple $\mathcal{A} = (Q', \Gamma, \mathscr{T}, \delta, I, F)$, where Q' is a finite set of states with $Q \subseteq Q'$, $\delta \subseteq Q' \times \Gamma \times [\![\mathcal{R}(\mathscr{T})]\!] \times Q'$ is a finite set of transition rules, $I \subseteq Q$ is a set of initial states, and $F \subseteq Q'$ is a set of final states. For readability, we usually write a transition $(q, \gamma, \tau, q') \in \delta$ as $q \xrightarrow{\gamma|\tau} q'$. Intuitively, a transition $q \xrightarrow{\gamma|\tau} q'$ means that the current symbol is γ and the transduction τ is applied to the rest of the input (i.e., the suffix after γ).

Let $\mathcal{A} = (Q', \Gamma, \mathscr{T}, \delta, I, F)$ be a TrNFA for $\mathcal{P}$. A configuration (q, ε) is *accepted* by $\mathcal{A}$ if $q \in I \cap F$. On the other hand, for a configuration (q, w) such that

$w = \gamma_1 \cdots \gamma_n$ is a nonempty word, (q, w) is *accepted* by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on w, that is, a sequence of transitions $q_0 \xrightarrow{\gamma'_1 | \tau_1} q_1 \xrightarrow{\gamma'_2 | \tau_2} \cdots \xrightarrow{\gamma'_{n-1} | \tau_{n-1}} q_{n-1} \xrightarrow{\gamma'_n | \tau_n} q_n$ such that $q = q_0 \in I$, $q_n \in F$, $\gamma'_1 = \gamma_1$, and $(\epsilon, \epsilon) \in \tau'_n$, where $\tau'_1 = \tau_1$ and for each $2 \leq i \leq n$, $\tau'_i = (\langle \gamma_i, \gamma'_i \rangle^{-1} \tau'_{i-1}) \circ \tau_i$. Note that $(\epsilon, \epsilon) \in \tau'_n$ implies that, for each $2 \leq i \leq n$, $\tau'_i \neq \tau_\emptyset$, and thus $\langle \gamma_i, \gamma'_i \rangle^{-1} \tau'_{i-1} \neq \tau_\emptyset$, that is, $\tau'_{i-1}(\gamma_i u) = \gamma'_i u'$ for words u, u'. Intuitively, for each $2 \leq i \leq n$, the transduction τ'_{i-1} summarizes the effect of the transductions $\tau_1, \cdots, \tau_{i-1}$ on the positions $\geq i$, and γ'_i is produced from γ_i by applying τ'_{i-1}. We use $\mathsf{Conf}_{\mathcal{A}}$ to denote the set of configurations accepted by $\mathcal{A}$.

Theorem 5 ([19,18,17]). *Let $\mathcal{P} = (Q, \Gamma, \mathscr{T}, \Delta)$ be a finite TrPDS. Given a set of configurations represented by a TrNFA $\mathcal{A}$ for $\mathcal{P}$, we can compute, in time polynomial in $|\mathcal{P}| + |[\![\mathcal{R}(\mathscr{T})]\!]| + |\mathcal{A}|$, two TrNFA $\mathcal{A}_{\mathsf{pre}^*}$ and $\mathcal{A}_{\mathsf{post}^*}$ that recognise the sets $\mathsf{pre}^*(\mathsf{Conf}_{\mathcal{A}})$ and $\mathsf{post}^*(\mathsf{Conf}_{\mathcal{A}})$ respectively. Moreover, for each TrNFA $\mathcal{A}$, an equivalent MA can be computed in time polynomial in $|\mathcal{A}| + |[\![\mathcal{R}(\mathscr{T})]\!]|$.*

B Details of Section 4

We sketch how to tackle the state reachability problem when there is a single launch mode for the activities in an ASM.

- STD: this is simply a PDS;
- SIT: the back stack contains tasks, each of which contains only one activity. Clearly we treat the back stack as a permutation of activities, and reduce to finite-state systems;
- STP: this can be treated as a PDS;
- STK: activities with the same affinities are grouped together, and each activity can appear at most once. This can be encoded by a finite-state system.

Remark 2. For those cases reduced to PDS, the reachability can be decided in polynomial time. For those reduced to finite-state systems, there is in general an exponential blowup in size, and one can obtain PSPACE-upper bound.

B.1 Proof of Proposition 2

The reduction from the state reachability problem to the configuration reachability is easy. The idea is that once the target state q is reached, we can enter a special state $q_\perp$, where a transition with the back action is continuously applied until the back stack becomes empty. Evidently, the reduction is in polynomial time.

Next, we show the reduction from the configuration reachability to the state reachability. The reduction uses ASM with parameters, an extension of ASM where parameters are attached to activities.

An *ASM with parameters* is a tuple $\mathcal{A} = (Q, \mathsf{Sig}, q_0, \Delta)$ such that $\mathsf{Sig} = (\mathsf{Act}, \mathsf{Lmd}, \mathsf{Aft}, \mathsf{Art}, A_0(\boldsymbol{b_0}))$, where

- $Q, q_0, \mathsf{Act}, \mathsf{Lmd}, \mathsf{Aft}$ are defined as for ASM (without parameters);
- $\mathsf{Art} : \mathsf{Act} \to \mathbb{N}$ is an arity function which denotes the numbers of parameters of activities;
- $A_0(\boldsymbol{b_0})$ is the main activity A_0, with $\boldsymbol{b_0} \in \{0,1\}^{\mathsf{Art}(A_0)}$ as the initial values of the parameters;
- $\Delta \subseteq Q \times (\mathsf{EAct} \cup \{\triangleright\}) \times \mathsf{Inst} \times Q$ is the transition relation, where $\mathsf{EAct} = \{A(\boldsymbol{b}) \mid A \in \mathsf{Act}, \boldsymbol{b} \in \{0,1\}^{\mathsf{Art}(A)}\}$, and $\mathsf{Inst} = \{\square, \mathsf{back}\} \cup \{\mathsf{start}(A(\boldsymbol{b})) \mid A(\boldsymbol{b}) \in \mathsf{EAct}\}$, such that: (1) for each transition $(q, A(\boldsymbol{b}), \alpha, q') \in \Delta$, it holds that $q' \neq q_0$, and (2) for each transition $(q, \triangleright, \alpha, q') \in \Delta$, it satisfies that $q = q_0$, $\alpha = \mathsf{start}(A_0(\boldsymbol{b_0}))$, and $q' \neq q_0$.

The semantics of ASM with parameters are defined similarly to those of ASM. A task of ASM with parameters is now $S = [A_1(\boldsymbol{b_1}), \cdots, A_n(\boldsymbol{b_n})] \in \mathsf{EAct}^+$, with A_1 as the top activity, denoted by $\mathsf{top}(S)$, and A_n as the bottom activity, denoted by $\mathsf{btm}(S)$. Back stack ρ is still *a sequence of non-empty tasks*, i.e., $\rho = (S_1, \cdots, S_n)$. A configurations is defined as a pair of a control state and a back stack satisfying the conditions in Definition 2 (with some minor adaptations).

For clarity, we would like to describe below the semantics of the transitions $q \xrightarrow{A(\boldsymbol{b}), \mathsf{start}(B(\boldsymbol{b'}))} q'$ with $\mathsf{Lmd}(B) = \mathsf{STK}$. Let $\rho = (S_1, \ldots, S_n)$. We first adapt the notations used for defining the semantics of ASM.

$$\mathsf{Noaction}(\rho) \equiv \rho \qquad \mathsf{Push}(\rho, B(\boldsymbol{b})) \equiv (([B(\boldsymbol{b})] \cdot S_1), S_2, \cdots, S_n)$$

$$\mathsf{NewTask}(B(\boldsymbol{b})) \equiv ([B(\boldsymbol{b})]) \qquad \mathsf{NewTask}(\rho, B(\boldsymbol{b})) \equiv ([B(\boldsymbol{b})], S_1, \cdots, S_n)$$

$$\mathsf{Pop}(\rho) \equiv \begin{cases} \varepsilon, & \text{if } n = 1 \text{ and } S_1 = [A(\boldsymbol{b})]; \\ (S_2, \cdots, S_n), & \text{if } n > 1 \text{ and } S_1 = [A(\boldsymbol{b})]; \\ (S_1', S_2, \cdots, S_n), & \text{if } S_1 = [A(\boldsymbol{b})] \cdot S_1' \text{ with } S_1' \in \mathsf{EAct}^+; \end{cases}$$

$$\mathsf{PopUntil}(\rho, B) \equiv (S_1'', S_2, \cdots, S_n), \text{ where}$$

$$S_1 = S_1' \cdot S_1'' \text{ with } S_1' \in (\mathsf{EAct} \setminus \{B(\boldsymbol{b}) \mid \boldsymbol{b} \in \{0,1\}^{\mathsf{Art}(B)}\})^* \text{ and } \mathsf{top}(S_1'') = B;$$

$$\mathsf{Move2Top}(\rho, i) \equiv (S_i, S_1, \cdots, S_{i-1}, S_{i+1}, \cdots, S_n)$$

$$\mathsf{GetNonSITTaskByAft}(\rho, k) \equiv \begin{cases} S_i, & \text{if } \mathsf{Aft}(S_i) = k \text{ and } \mathsf{Lmd}(\mathsf{btm}(S_i)) \neq \mathsf{SIT}; \\ \mathsf{Undef}, & \text{otherwise.} \end{cases}$$

Then $(q, \rho) \xrightarrow{\mathcal{A}} (q', \rho')$ if one of the following holds:

- $\mathsf{Lmd}(A) \neq \mathsf{SIT}$ and $\mathsf{Aft}(B) = \mathsf{Aft}(S_1)$, then
  - if B does *not* occur in S_1, then $\rho' = \mathsf{Push}(\rho, B(\boldsymbol{b}))$;
  - if B occurs in S_1, then $\rho' = \mathsf{PopUntil}(\rho, B)$;
- $\mathsf{Lmd}(A) \neq \mathsf{SIT}$ and $\mathsf{Aft}(B) \neq \mathsf{Aft}(S_1)$, then
  - if $\mathsf{GetNonSITTaskByAft}(\rho, \mathsf{Aft}(B)) = S_i$,
    - ∗ if B does *not* occur in S_i, then $\rho' = \mathsf{Push}(\mathsf{Move2Top}(\rho, i), B(\boldsymbol{b}))$;
    - ∗ if B occurs in S_i, then $\rho' = \mathsf{PopUntil}(\mathsf{Move2Top}(\rho, i), B)$,
  - if $\mathsf{GetNonSITTaskByAft}(\rho, \mathsf{Aft}(B)) = \mathsf{Undef}$, then $\rho' = \mathsf{NewTask}(\rho, B(\boldsymbol{b}))$;
- $\mathsf{Lmd}(A) = \mathsf{SIT}$, then
  - if $\mathsf{GetNonSITTaskByAft}(\rho, \mathsf{Aft}(B)) = S_i$, then
    - ∗ if B does *not* occur in S_i, then $\rho' = \mathsf{Push}(\mathsf{Move2Top}(\rho, i), B(\boldsymbol{b}))$;

    * if B occurs in S_i, then $\rho' = \mathsf{PopUntil}(\mathsf{Move2Top}(\rho, i), B)$;
- if $\mathsf{GetNonSITTaskByAft}(\rho, \mathsf{Aft}(B)) = \mathsf{Undef}$, then $\rho' = \mathsf{NewTask}(\rho, B(\boldsymbol{b}))$.

From the aforementioned description of the semantics, we know that for $B \in \mathsf{Act}_{\mathsf{STK}}$, if an instance of the B-activity, say $B(\boldsymbol{b'})$, is already in the back stack, then executing an action $\mathsf{start}(B(\boldsymbol{b}))$ will not modify the parameters of the instance of B in the back stack and the parameters $\boldsymbol{b}$ are lost, in other words, $B(\boldsymbol{b'})$ will be still stored in the back stack after executing the action.

Since in many apps, activities do have parameters, which typically take Boolean values, ASM with parameters is a natural extension of ASM to model such Apps. **All the results for ASM obtained in this paper can be easily extended to ASM with parameters.**

We are ready to present the reduction. Suppose $\mathcal{A} = (Q, \mathsf{Sig}, q_0, \Delta)$ is an ASM with $\mathsf{Sig} = (\mathsf{Act}, \mathsf{Lmd}, \mathsf{Aft}, A_0)$ and $(q, \rho) \in \mathsf{Conf}_{\mathcal{A}}$. Let $\rho = (S_1, \cdots, S_n)$, where $S_i = [A_{i,1}, \cdots, A_{i,m_i}]$ for each $i \in [n]$.

We use $\mathsf{Aft}_{\cancel{\mathsf{sit}}}$ to denote the set of affinities of non-SIT activities, i.e., $\{\mathsf{Aft}(A) \mid A \in \mathsf{Act} \setminus \mathsf{Act}_{\mathsf{sit}}\}$. Intuitively, in the reduction, we encode ρ as a word

$$\mathsf{enc}(\rho) = A_{1,1}(0), \cdots, A_{1,m_1-1}(0), A_{1,m_1}(1), \cdots, A_{n,1}(0), \cdots, A_{n,m_n-1}(0), A_{n,m_n}(1),$$

where $A_{1,m_1}(1)$ denotes the fact that A_{1,m_1} is the root of the top task, similarly for $A_{2,m_2}(1)$, and so on.

Let $\mathsf{EAct} = \{A(b) \mid A \in \mathsf{Act}, b \in \{0,1\}\}$ and Sfx_ρ denote the set of suffixes of $\mathsf{enc}(\rho)$. Moreover, we define a relation $\alpha \xrightarrow{A(b)} \alpha'$ as follows: For $\alpha, \alpha' \in \mathsf{Sfx}_\rho$ and $A(b) \in \mathsf{EAct}$, we have $\alpha \xrightarrow{A(b)} \alpha'$ if $\alpha = A(b) \cdot \alpha'$.

The ASM $\mathcal{B} = (Q', \mathsf{Sig}', q'_0, \Delta')$, such that

- $\mathsf{Sig}' = (\mathsf{Act}, \mathsf{Lmd}, \mathsf{Aft}, \mathsf{Art}, A_0(1))$ such that $\mathsf{Art}(A) = 1$ for each $A \in \mathsf{Act}$,
- $Q' = Q \times 2^{\mathsf{Act}_{\mathsf{sit}} \cup \mathsf{Aft}_{\cancel{\mathsf{sit}}}} \cup \{q_\alpha \mid \alpha \in \mathsf{Sfx}_\rho\}$, where each state (q', I) denotes the fact that the current state of $\mathcal{A}$ is q' and I is the union of the set of SIT activities in the back stack and the set of affinities of non-SIT tasks,
- $q'_0 = (q_0, \emptyset)$,
- Δ' is defined below.

We construct Δ' out of the transitions from Δ by mimicking their semantics.

- For each transition $q_0 \xrightarrow{\triangleright, \mathsf{start}(A_0)} q' \in \Delta$,
    - if $\mathsf{Lmd}(A_0) = \mathsf{SIT}$, then, $(q_0, \emptyset) \xrightarrow{\triangleright, \mathsf{start}(A_0(1))} (q', \{A_0\}) \in \Delta'$,
    - if $\mathsf{Lmd}(A_0) \neq \mathsf{SIT}$, then, $(q_0, \emptyset) \xrightarrow{\triangleright, \mathsf{start}(A_0(1))} (q', \{\mathsf{Aft}(A_0)\}) \in \Delta'$.
- For each transition $q' \xrightarrow{A, \square} q'' \in \Delta$, $I \subseteq \mathsf{Act}_{\mathsf{sit}} \cup \mathsf{Aft}_{\cancel{\mathsf{sit}}}$ and $b \in \{0,1\}$,

$$(q', I) \xrightarrow{A(b), \square} (q'', I) \in \Delta'.$$

- For each transition $q' \xrightarrow{A, \mathsf{back}} q'' \in \Delta$ and $I \subseteq \mathsf{Act}_{\mathsf{sit}} \cup \mathsf{Aft}_{\cancel{\mathsf{sit}}}$,
    - if $\mathsf{Lmd}(A) = \mathsf{SIT}$ and $A \in I$, then $(q', I) \xrightarrow{A(1), \mathsf{back}} (q'', I \setminus \{A\}) \in \Delta'$,

- if $\mathsf{Lmd}(A) \neq \mathsf{SIT}$, then $(q', I) \xrightarrow{A(0),\mathsf{back}} (q'', I) \in \Delta'$. Moreover, if $\mathsf{Aft}(A) \in I$, $(q', I) \xrightarrow{A(1),\mathsf{back}} (q'', I \setminus \{\mathsf{Aft}(A)\}) \in \Delta'$.

Next, we consider the transitions of the form $q' \xrightarrow{A,\mathsf{start}(B)} q'' \in \Delta$. For each transition $q' \xrightarrow{A,\mathsf{start}(B)} q'' \in \Delta$, $I, I' \subseteq \mathsf{Act}_{\mathbf{sit}} \cup \mathsf{Aft}_{\cancel{\mathbf{sit}}}$ and $b, b' \in \{0, 1\}$, we have $(q', I), \xrightarrow{A(b),\mathsf{start}(B(b'))} (q'', I') \in \Delta'$ if one of the following constraints holds:

CASE $\mathsf{Lmd}(B) = \mathsf{STD}$ or CASE $\mathsf{Lmd}(B) = \mathsf{STP}$

- $\mathsf{Lmd}(A) = \mathsf{SIT}$, $A \in I$ and $\mathsf{Aft}(B) \notin I$, then $I' = I \cup \{\mathsf{Aft}(B)\}$ and $b' = 1$;
- otherwise, $I = I'$ and $b' = 0$;

CASE $\mathsf{Lmd}(B) = \mathsf{SIT}$

- $I' = I \cup \{B\}$ and $b' = 1$;

CASE $\mathsf{Lmd}(B) = \mathsf{STK}$

- $\mathsf{Aft}(B) \notin I$, then $I' = I \cup \{\mathsf{Aft}(B)\}$ and $b' = 1$;
- otherwise, $I = I'$ and $b' = 0$.

Note that according to the semantics of $\mathsf{start}(B(\boldsymbol{b}))$, if $\mathsf{Aft}(B) \in I$, then an instance of B is already in the back stack, then it is safe to set $b' = 0$, since b' will not be stored into the back stack anyway.

Finally, we add the following transition rules in Δ' to verify that the configuration (q, ρ) is reached in $\mathcal{A}$.

- For every $I \subseteq \mathsf{Act}_{\mathbf{sit}} \cup \mathsf{Aft}_{\cancel{\mathbf{sit}}}$, $\alpha, \alpha' \in \mathsf{Sfx}_\rho$, and $A(b) \in \mathsf{EAct}$ such that $\alpha \xrightarrow{A(b)} \alpha'$, we have
  - if $\alpha = \mathsf{enc}(\rho)$, then $(q, I) \xrightarrow{A(b),\mathsf{back}} (q_{\alpha'}, I')$,
  - otherwise, $(q_\alpha, I) \xrightarrow{A(b),\mathsf{back}} (q_{\alpha'}, I')$,

  where
$$I' = \begin{cases} I, & \text{if } b = 0, \\ I \setminus \{A\}, & \text{if } b = 1 \wedge \mathsf{Lmd}(A) = \mathsf{SIT}, \\ I \setminus \{\mathsf{Aft}(A)\}, & \text{if } b = 1 \wedge \mathsf{Lmd}(A) \neq \mathsf{SIT}. \end{cases}$$

From the construction, we know that $(q_0, \varepsilon) \overset{\mathcal{A}}{\Rightarrow} (q, \rho)$ iff the state $(q_\varepsilon, \emptyset)$ is reachable from the configuration $((q_0, \emptyset), \varepsilon)$ in $\mathcal{B}$.

Since the size of Sfx_ρ is linear in the length of ρ and the set of states $(q', I) \in Q \times 2^{\mathsf{Act}_{\mathbf{sit}} \cup \mathsf{Aft}_{\cancel{\mathbf{sit}}}}$ has a size at most exponential in $|\mathsf{Act}_{\mathsf{sit}}| + |\mathsf{Aft}_{\cancel{\mathbf{sit}}}|$, we know that size of $\mathcal{B}$ is at most exponential in the size of $\mathcal{A}$ and (q, ρ). We conclude that the reduction takes exponential time in the worst case.

B.2 Proof of Theorem 1

We prove the theorem by reducing from the state reachability problem of two counter machines.

A *two-counter machine* $\mathcal{M}$ is a triple (Q, q_0, δ), where Q is a set of states, $q_0 \in Q$ is the initial state, and $\delta \subseteq Q \times \{\mathsf{ifz}_i, \mathsf{inc}_i, \mathsf{dec}_i \mid i = 1, 2\} \times Q$ is a set of transitions. The *state reachability problem* is to ask whether a given state q is reachable from q_0, with the zero initial values for the two counters.

The intuition of the reduction is to construct an ASM with two tasks that record the values of the two counters, and use STK (resp. SIT) activities to switch between the two tasks. We shall focus on the first two claims. Other claims can be shown in a very similar way, and are omitted.

Proof of the first claim. We construct an ASM $\mathcal{B} = (Q', \mathsf{Sig}, q'_0, \Delta)$ to simulate the two-counter machine $\mathcal{M}$, where

- $Q' = Q \cup \delta \cup \{q'_0, q'_1\}$, where q'_0, q'_1 are two fresh states not in $Q \cup \delta$,
- $\mathsf{Sig} = (\mathsf{Act}, \mathsf{Lmd}, \mathsf{Aft}, B_1)$, such that $\mathsf{Act} = \{A_1, A_2, B_1, B_2, C_1, C_2\}$, and $\mathsf{Lmd}, \mathsf{Aft}$ are defined as follows,
  - $\mathsf{Lmd}(A_1) = \mathsf{Lmd}(A_2) = \mathsf{STD}$, $\mathsf{Aft}(A_1) = 1$, $\mathsf{Aft}(A_2) = 2$ (intuitively, A_1, A_2 are put in two different tasks to simulate the values of the two counters),
  - B_1, B_2 are the root activities of the two tasks, $\mathsf{Lmd}(B_1) = \mathsf{Lmd}(B_2) = \mathsf{STK}$, $\mathsf{Aft}(B_1) = \mathsf{Aft}(A_1)$, $\mathsf{Aft}(B_2) = \mathsf{Aft}(A_2)$ (we will call the two tasks as B_1-task and B_2-task respectively),
  - $\mathsf{Lmd}(C_1) = \mathsf{Lmd}(C_2) = \mathsf{STK}$, $\mathsf{Aft}(C_1) = \mathsf{Aft}(A_1)$, $\mathsf{Aft}(C_2) = \mathsf{Aft}(A_2)$ (intuitively, C_1 and C_2 are used to switch between the two tasks when incrementing/decrementing the two counters),
- Δ comprises the following transitions,
  - $q'_0 \xrightarrow{\triangleright, \mathsf{start}(B_1)} q'_1$, $q'_1 \xrightarrow{B_1, \mathsf{start}(B_2)} q_0$,
  - for each transition $(q', \mathsf{ifz}_i, q'') \in \delta$,
    * $q' \xrightarrow{B_i, \square} q''$,
    * $q' \xrightarrow{B_{3-i}, \mathsf{start}(C_i)} (q', \mathsf{ifz}_i, q'')$, $q' \xrightarrow{A_{3-i}, \mathsf{start}(C_i)} (q', \mathsf{ifz}_i, q'')$, $(q', \mathsf{ifz}_i, q'') \xrightarrow{C_i, \mathsf{back}} (q', \mathsf{ifz}_i, q'')$, $(q', \mathsf{ifz}_i, q'') \xrightarrow{B_i, \square} q''$,
  - for each transition $(q', \mathsf{inc}_i, q'') \in \delta$,
    * $q' \xrightarrow{B_i, \mathsf{start}(A_i)} q''$, $q' \xrightarrow{A_i, \mathsf{start}(A_i)} q''$,
    * $q' \xrightarrow{B_{3-i}, \mathsf{start}(C_i)} (q', \mathsf{inc}_i, q'')$, $q' \xrightarrow{A_{3-i}, \mathsf{start}(C_i)} (q', \mathsf{inc}_i, q'')$, $(q', \mathsf{inc}_i, q'') \xrightarrow{C_i, \mathsf{back}} (q', \mathsf{inc}_i, q'')$, $(q', \mathsf{inc}_i, q'') \xrightarrow{B_i, \mathsf{start}(A_i)} q''$, $(q', \mathsf{inc}_i, q'') \xrightarrow{A_i, \mathsf{start}(A_i)} q''$,
  - for each transition $(q', \mathsf{dec}_i, q'') \in \delta$,
    * $q' \xrightarrow{A_i, \mathsf{back}} q''$,
    * $q' \xrightarrow{B_{3-i}, \mathsf{start}(C_i)} (q', \mathsf{dec}_i, q'')$, $q' \xrightarrow{A_{3-i}, \mathsf{start}(C_i)} (q', \mathsf{dec}_i, q'')$, $(q', \mathsf{dec}_i, q'') \xrightarrow{C_i, \mathsf{back}} (q', \mathsf{dec}_i, q'')$, $(q', \mathsf{dec}_i, q'') \xrightarrow{A_i, \mathsf{back}} q''$.

From the construction, the state q is reachable from the initial configuration $(q_0, (0, 0))$ in $\mathcal{M}$ iff q is reachable from (q'_0, ε) in $\mathcal{B}$.

Proof of the second claim. We construct an ASM $\mathcal{C} = (Q'', \mathsf{Sig}', q_0', \Delta')$ to simulate $\mathcal{M}$. The construction of $\mathcal{C}$ is similar to $\mathcal{B}$, except that we now use a SIT activity C to switch between two tasks in lieu of two STK activities. In addition, the root activities of the two tasks have the "standard" launch mode now. More specifically,

- $Q'' = Q \cup \delta \cup (\delta \times \{0,1\}) \cup \{q_0', q_1', q_2'\}$, where q_0', q_1', q_2' are three fresh states,
- $\mathsf{Sig}' = (\mathsf{Act}', \mathsf{Lmd}', \mathsf{Aft}', B_1)$, where
  - $\mathsf{Act}' = \{A_1, A_2, B_1, B_2, C\}$,
  - $\mathsf{Lmd}'(A_1) = \mathsf{Lmd}'(A_2) = \mathsf{Lmd}'(B_1) = \mathsf{Lmd}'(B_2) = \mathsf{STD}$, $\mathsf{Lmd}'(C) = \mathsf{SIT}$,
  - $\mathsf{Aft}'(A_1) = \mathsf{Aft}'(B_1) = 1$, $\mathsf{Aft}'(A_2) = \mathsf{Aft}'(B_2) = 2$, and $\mathsf{Aft}'(C) = 3$,
- Δ' comprises the following transitions,
  - $q_0' \xrightarrow{\triangleright, \mathsf{start}(B_1)} q_1'$, $q_1' \xrightarrow{B_1, \mathsf{start}(C)} q_2'$, $q_2' \xrightarrow{C, \mathsf{start}(B_2)} q_0$,
  - for each transition $(q', \mathsf{ifz}_i, q'') \in \delta$,
    - $q' \xrightarrow{B_i, \Box} q''$,
    - $q' \xrightarrow{B_{3-i}, \mathsf{start}(C)} ((q', \mathsf{ifz}_i, q''), 0)$, $q' \xrightarrow{A_{3-i}, \mathsf{start}(C)} ((q', \mathsf{ifz}_i, q''), 0)$, $((q', \mathsf{ifz}_i, q''), 0) \xrightarrow{C, \mathsf{start}(A_i)} ((q', \mathsf{ifz}_i, q''), 0)$, $((q', \mathsf{ifz}_i, q''), 0) \xrightarrow{A_i, \mathsf{back}} ((q', \mathsf{ifz}_i, q''), 1), ((q', \mathsf{ifz}_i, q''), 1) \xrightarrow{B_i, \Box} q''$,
  - for each transition $(q', \mathsf{inc}_i, q'') \in \delta$,
    - $q' \xrightarrow{B_i, \mathsf{start}(A_i)} q''$, $q' \xrightarrow{A_i, \mathsf{start}(A_i)} q''$,
    - $q' \xrightarrow{B_{3-i}, \mathsf{start}(C)} (q', \mathsf{inc}_i, q'')$, $q' \xrightarrow{A_{3-i}, \mathsf{start}(C)} (q', \mathsf{inc}_i, q'')$, $(q', \mathsf{inc}_i, q'') \xrightarrow{C, \mathsf{start}(A_i)} q''$,
  - for each transition $(q', \mathsf{dec}_i, q'') \in \delta$,
    - $q' \xrightarrow{A_i, \mathsf{back}} q''$,
    - $q' \xrightarrow{B_{3-i}, \mathsf{start}(C)} ((q', \mathsf{dec}_i, q''), 0)$, $q' \xrightarrow{A_{3-i}, \mathsf{start}(C)} ((q', \mathsf{dec}_i, q''), 0)$, $((q', \mathsf{dec}_i, q''), 0) \xrightarrow{C, \mathsf{start}(A_i)} ((q', \mathsf{dec}_i, q''), 0)$, $((q', \mathsf{dec}_i, q''), 0) \xrightarrow{A_i, \mathsf{back}} ((q', \mathsf{dec}_i, q''), 1), ((q', \mathsf{dec}_i, q''), 1) \xrightarrow{A_i, \mathsf{back}} q''$.

From the construction, the state q is reachable from the initial configuration $(q_0, (0,0))$ in $\mathcal{M}$ iff q is reachable from (q_0', ε) in $\mathcal{C}$.

C Details of Section 5.1

For the detailed construction of the TrPDS $\mathcal{P}$, we start with two transducers manipulating the back stack encoded by $S_1\sharp \cdots \sharp S_n \sharp$. Let $B \in \mathsf{Act}_{\mathsf{STK}}$.

The first transducer $\mathcal{T}_{B,1}$ checks that B is the root activity of the top task, and replaces every symbol of the top task with †. The second transducer $\mathcal{T}_{B,2}$ checks that B is *not* the root activity of the top task, and replaces each symbol of the task containing B—if it exists—with †. Note that for an input word, if B is not the root activity of the top task, then $\mathcal{T}_{B,1}$ fails and does not produce any output. Likewise, if B is the root activity of the top task, then $\mathcal{T}_{B,2}$ fails.

Let $\Gamma = \mathsf{Act} \cup \{\bot, \sharp, \dagger\}$. In the TrPDS $\mathcal{P}$, we shall encode each back stack of $\mathcal{A}$ as a word in the regular language

$$\mathcal{L}_{conf} = \left((\mathsf{Act}^*_{\mathsf{STD}}\mathsf{Act}_{\mathsf{STK}} \cup \dagger^+)\sharp\right)^* \bot.$$

For a word $w \in \mathcal{L}_{conf}$, w can be split into subwords from $(\mathsf{Act}^*_{\mathsf{STD}}\mathsf{Act}_{\mathsf{STK}} \cup \dagger^+)\sharp$, that is, $w = w_1\sharp w_2\sharp \cdots \sharp w_k\sharp\bot$, where $w_i \in \mathsf{Act}^*_{\mathsf{STD}}\mathsf{Act}_{\mathsf{STK}} \cup \dagger^+$ for each $i \in [k]$. We refer to $w_1, \cdots, w_k$ as the *blocks* of w. A block w_i of w is *non-trivial* if $w_i \in \mathsf{Act}^*_{\mathsf{STD}}\mathsf{Act}_{\mathsf{STK}}$. For $B \in \mathsf{Act}_{\mathsf{STK}}$, a non-trivial block w_i of w is said to be a B-block if $w_i \in \mathsf{Act}^*_{\mathsf{STD}}B$.

Let $\mathcal{B}_{conf} = (Q_{conf}, Q_{conf}, \Gamma, \delta_{conf}, I_{conf}, F_{conf})$ be the NFA recognizing $\mathcal{L}_{conf}$. The size of $\mathcal{B}_{conf}$ is polynomial in $|\mathsf{Act}_{\mathsf{STK}}|$. W.l.o.g., in $\mathcal{B}_{conf}$, we assume that there are *no* incoming transitions for each $p \in I_{conf}$.

Let $B \in \mathsf{Act}_{\mathsf{STK}}$. We define $\mathcal{T}_{B,1} = (Q_{B,1}, \Gamma, \delta_{B,1}, I_{B,1}, F_{B,1})$, where

- $Q_{B,1} = Q_{conf} \times \{q_0^{(B,1)}, q_1^{(B,1)}, q_2^{(B,1)}\}$,
- $\delta_{B,1}$ comprises
  - the transitions $(p, q_0^{(B,1)}) \xrightarrow{C,\dagger} (p', q_0^{(B,1)})$ for each $C \in \mathsf{Act}_{\mathsf{STD}}$ and $(p, C, p') \in \delta_{conf}$,
  - $(p, q_0^{(B,1)}) \xrightarrow{B,\dagger} (p', q_1^{(B,1)})$ for each $(p, B, p') \in \delta_{conf}$,
  - $(p, q_1^{(B,1)}) \xrightarrow{\gamma,\gamma} (p', q_1^{(B,1)})$ for each $\gamma \in \Gamma \setminus \{B\}$ and $(p, \gamma, p') \in \delta_{conf}$,
  - $(p, q_1^{(B,1)}) \xrightarrow{\sharp,\sharp} (p', q_2^{(B,1)})$ for each $(p, \sharp, p') \in \delta_{conf}$,
  - $(p, q_2^{(B,1)}) \xrightarrow{\gamma,\dagger} (p', q_2^{(B,1)})$ for each $\gamma \in \mathsf{Act}_{\mathsf{STD}}$ and $(p, \gamma, p') \in \delta_{conf}$,
  - $(p, q_2^{(B,1)}) \xrightarrow{B,\dagger} (p', q_1^{(B,1)})$ for each $(p, B, p') \in \delta_{conf}$,
- $I_{B,1} = I_{conf} \times \{q_0^{(B,1)}\}$,
- $F_{B,1} = F_{conf} \times \{q_1^{(B,1)}\}$.

In addition, we define $\mathcal{T}_{B,2} = (Q_{B,2}, \Gamma, \delta_{B,2}, I_{B,2}, F_{B,2})$, where

- $Q_{B,2} = Q_{conf} \times \{q_0^{(B,2)}, q_1^{(B,2)}\}$,
- $\delta_{B,2}$ comprises the following transitions
  - $(p, q_0^{(B,2)}) \xrightarrow{\gamma,\gamma} (p', q_0^{(B,2)})$ for each $\gamma \in \Gamma \setminus \{B\}$ and $(p, \gamma, p') \in \delta_{conf}$,
  - $(p, q_0^{(B,2)}) \xrightarrow{\sharp,\sharp} (p', q_1^{(B,2)})$ for each $(p, \sharp, p') \in \delta_{conf}$,
  - $(p, q_1^{(B,2)}) \xrightarrow{\gamma,\dagger} (p', q_1^{(B,2)})$ for each $\gamma \in \mathsf{Act}_{\mathsf{STD}}$ and $(p, \gamma, p') \in \delta_{conf}$,
  - $(p, q_1^{(B,2)}) \xrightarrow{B,\dagger} (p', q_0^{(B,2)})$ for each $(p, B, p') \in \delta_{conf}$,
- $I_{(B,2)} = I_{conf} \times \{q_0^{(B,2)}\}$,
- $F_{(B,2)} = F_{conf} \times \{q_0^{(B,2)}\}$.

Note that $F_{conf} \times \{q_0^{(B,2)}\} = F_{(B,2)}$ implies that $\mathcal{T}_{B,2}$ simply outputs the input string if B does not occur in the input string.

Let $\mathscr{T} = \{\mathcal{T}_{B,1}, \mathcal{T}_{B,2} \mid B \in \mathsf{Act}_{\mathsf{STK}}\} \cup \{\mathcal{T}_{id}\}$. According to the definition, we know that the domain of $\mathcal{T}_{B,1} \in \mathscr{T}$ is $\mathcal{L}_{conf} \cap (\mathsf{Act}^*_{\mathsf{STD}}B\sharp\Gamma^*)$, while that of $\mathcal{T}_{B,2} \in \mathscr{T}$ is $\mathcal{L}_{conf} \cup (\sharp \cdot \mathcal{L}_{conf})$. Note that we construct $\mathcal{T}_{B,1}$ and $\mathcal{T}_{B,2}$ in a way that, for each w in the domain of $\mathcal{T}_{B,1}$ (resp. $\mathcal{T}_{B,2}$), B may occur *multiple times* in w. This would facilitate the construction of $[\![\mathcal{R}(\mathscr{T})]\!]$ from $\mathscr{T}$ later.

Based on these transducers, $\mathcal{P}$ is constructed as $(Q', \Gamma, \mathscr{T}, \Delta')$, where $Q' = Q \cup (Q \times Q)$, and Δ' comprises the following transitions,

- for each $(q_0, \triangleright, \mathsf{start}(A_0), q) \in \Delta$, we have $(q_0, \perp, A_0\sharp\perp, \mathcal{T}_{id}, q) \in \Delta'$,
- for each $(q, A, \mathsf{start}(B), q') \in \Delta$ with $B \in \mathsf{Act}_{\mathsf{STD}}$, we have $(q, A, BA, \mathcal{T}_{id}, q') \in \Delta'$,
- for each $(q, A, \mathsf{start}(A), q') \in \Delta$ with $A \in \mathsf{Act}_{\mathsf{STK}}$, we have $(q, A, A, \mathcal{T}_{id}, q') \in \Delta'$,
- for each $(q, A, \mathsf{start}(B), q') \in \Delta$ with $B \in \mathsf{Act}_{\mathsf{STK}}$ and $A \neq B$, we have $(q, A, B\sharp\dagger, \mathcal{T}_{B,1}, q') \in \Delta'$ (corresponding to the situation that the top task is a B-task), and $(q, A, B\sharp A, \mathcal{T}_{B,2}, q') \in \Delta'$ (corresponding to the situation that the top task is not a B-task),
- for each $(q, A, \mathsf{back}, q') \in \Delta$, we have the transitions $(q, A, \varepsilon, \mathcal{T}_{id}, (q, q')) \in \Delta'$, $((q, q'), \gamma, \gamma, \mathcal{T}_{id}, q') \in \Delta'$ for each $\gamma \in \mathsf{Act} \cup \{\perp\}$, $((q, q'), \sharp, \varepsilon, \mathcal{T}_{id}, (q, q')) \in \Delta'$, and $((q, q'), \dagger, \varepsilon, \mathcal{T}_{id}, (q, q')) \in \Delta'$,
- for each $(q, A, \square, q') \in \Delta$, we have $(q, A, A, \mathcal{T}_{id}, q') \in \Delta'$.

In order to show that $\mathcal{P}$ is indeed a finite TrPDS, it remains to show that $[\![\mathcal{R}(\mathscr{T})]\!]$ is finite.

Lemma 2. *$[\![\mathcal{R}(\mathscr{T})]\!]$ is finite, moreover, the size of $[\![\mathcal{R}(\mathscr{T})]\!]$ is exponential in $|\mathsf{Act}_{\mathsf{STK}}|$.*

Proof. We will prove the lemma by utilising some special properties enjoyed by the transductions from $\mathcal{R}(\mathscr{T})$.

Let $B \in \mathsf{Act}_{\mathsf{STK}}$. Recall that $\mathcal{T}_{B,1}$ checks that the first block is a B-block and replaces every symbol in all the B-blocks with $\dagger$ and $\mathcal{T}_{B,2}$ replaces every symbol in all the B-blocks—if they exist—with $\dagger$.

For convenience, for $B \in \mathsf{Act}_{\mathsf{STK}}$, we use $\mathcal{R}(\mathcal{T}_B)$ to denote $\mathcal{R}(\mathcal{T}_{B,1}) \cup \mathcal{R}(\mathcal{T}_{B,2})$.

By the definition of $\mathcal{T}_{B,1}$ and $\mathcal{T}_{B,2}$, we know that $\mathcal{R}(\mathscr{T})$ satisfies the following properties:

- For each $B \in \mathsf{Act}_{\mathsf{STK}}$, $\mathcal{R}(\mathcal{T}_{B,2})$ is an idempotent, that is, $\mathcal{R}(\mathcal{T}_{B,2}) \circ \mathcal{R}(\mathcal{T}_{B,2}) = \mathcal{R}(\mathcal{T}_{B,2})$.
- For $B, B' \in \mathsf{Act}_{\mathsf{STK}}$ (B and B' are not necessarily distinct), $\mathcal{R}(\mathcal{T}_{B,1}) \circ \mathcal{R}(\mathcal{T}_{B',1}) = \tau_\emptyset$.
- $\mathcal{R}(\mathscr{T})$ is associative and communicative, namely, for all $\tau_1, \tau_2, \tau_3 \in \mathcal{R}(\mathscr{T})$, $(\tau_1 \circ \tau_2) \circ \tau_3 = \tau_1 \circ (\tau_2 \circ \tau_3)$, and $\tau_1 \circ \tau_2 = \tau_2 \circ \tau_1$.
- The left-quotients of $\mathcal{R}(\mathscr{T})$ satisfy that
  - for each $B \in \mathsf{Act}_{\mathsf{STK}}$, $A \in \mathsf{Act}_{\mathsf{STD}}$, and $B' \in \mathsf{Act}_{\mathsf{STK}} \setminus \{B\}$, we have
    * $\langle A, \dagger\rangle^{-1}\mathcal{R}(\mathcal{T}_{B,1}) = \mathcal{R}(\mathcal{T}_{B,1})$, and
    $$\langle B, \dagger\rangle^{-1}\mathcal{R}(\mathcal{T}_{B,1}) = \langle \sharp, \sharp\rangle^{+1}\mathcal{R}(\mathcal{T}_B),$$
    * $\langle A, A\rangle^{-1}\mathcal{R}(\mathcal{T}_{B,2}) = \mathcal{R}(\mathcal{T}_{B,2})$, $\langle \sharp, \sharp\rangle^{-1}\mathcal{R}(\mathcal{T}_{B,2}) = \mathcal{R}(\mathcal{T}_B)$,
    $$\langle B', B'\rangle^{-1}\mathcal{R}(\mathcal{T}_{B,2}) = \langle \sharp, \sharp\rangle^{+1}\mathcal{R}(\mathcal{T}_B),$$
    $$\langle \dagger, \dagger\rangle^{-1}\mathcal{R}(\mathcal{T}_{B,2}) = \mathcal{R}(\mathcal{T}_{B,2}) \cup \langle \sharp, \sharp\rangle^{+1}\mathcal{R}(\mathcal{T}_B);$$

- for each $B \in \mathsf{Act}_{\mathsf{STK}}$, except the aforementioned cases, $\langle \gamma_1, \gamma_2 \rangle^{-1} \mathcal{R}(\mathcal{T}_{B,1}) = \tau_\emptyset$ and $\langle \gamma_1, \gamma_2 \rangle^{-1} \mathcal{R}(\mathcal{T}_{B,2}) = \tau_\emptyset$.

As a result, $[\![\mathcal{R}(\mathscr{T})]\!]$ can be computed from $\mathscr{T}$ by repeatedly applying composition and left quotient until stablized. It turns out that the computation stabilizes after applying composition, left quotient, composition, and left quotient.

Let $B_1, \cdots, B_k$ be an enumeration of the activities in $\mathsf{Act}_{\mathsf{STK}}$.

Step I. At first, we compute the closure of $\mathcal{R}(\mathscr{T})$ under *compositions*, denoted by $\mathsf{Comp}(\mathcal{R}(\mathscr{T}))$, as the union of

$$\mathscr{T}_1' = \left\{ \mathcal{R}(\mathcal{T}_{B_{i_1},2}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r},2}) \mid r \geq 1, i_1 < \cdots < i_r \right\},$$

$$\mathscr{T}_2' = \left\{ \mathcal{R}(\mathcal{T}_{B_{i_0},1}) \circ \mathcal{R}(\mathcal{T}_{B_{i_1},2}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r},2}) \,\middle|\, \begin{array}{l} r \geq 1, i_1 < \cdots < i_r, \\ \forall j \in [r], i_j \neq i_0 \end{array} \right\},$$

and $\mathscr{T}_3' = \{ \mathcal{R}(\mathcal{T}_{B_i,1}) \mid i \in [k] \} \cup \{\tau_\emptyset\}$.

Step II. We compute the closure of $\mathsf{Comp}(\mathcal{R}(\mathscr{T}))$ under left-quotients, denoted by $\mathsf{Quot}(\mathsf{Comp}(\mathcal{R}(\mathscr{T})))$.

Let $i_1, \cdots, i_r \in [k]$ such that $i_1 < \cdots < i_r$. From the properties about left-quotients, we know that

– for $i_0 \in [k]$ such that $i_0 \neq i_1, \cdots, i_r$ and $A \in \mathsf{Act}_{\mathsf{STD}}$, we have

$$\begin{array}{l} \langle A, \dagger \rangle^{-1} (\mathcal{R}(\mathcal{T}_{B_{i_0},1}) \circ \mathcal{R}(\mathcal{T}_{B_{i_1},2}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r},2})) \\ = \mathcal{R}(\mathcal{T}_{B_{i_0},1}) \circ \mathcal{R}(\mathcal{T}_{B_{i_1},2}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r},2}), \\ \langle B_{i_0}, \dagger \rangle^{-1} (\mathcal{R}(\mathcal{T}_{B_{i_0},1}) \circ \mathcal{R}(\mathcal{T}_{B_{i_1},2}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r},2})) \\ = \langle \sharp, \sharp \rangle^{+1} (\mathcal{R}(\mathcal{T}_{B_{i_0}}) \circ \mathcal{R}(\mathcal{T}_{B_{i_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r}})), \end{array}$$

– for $A \in \mathsf{Act}_{\mathsf{STD}}$, we have

$$\langle A, A \rangle^{-1} (\mathcal{R}(\mathcal{T}_{B_{i_1},2}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r},2})) = \mathcal{R}(\mathcal{T}_{B_{i_1},2}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r},2}),$$

– $\langle \sharp, \sharp \rangle^{-1} (\mathcal{R}(\mathcal{T}_{B_{i_1},2}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r},2})) = \mathcal{R}(\mathcal{T}_{B_{i_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r}})$,

– for $i_0 \in [k]$ such that $i_0 \neq i_1, \cdots, i_r$,

$$\langle B_{i_0}, B_{i_0} \rangle^{-1} (\mathcal{R}(\mathcal{T}_{B_{i_1},2}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r},2})) = \langle \sharp, \sharp \rangle^{+1} (\mathcal{R}(\mathcal{T}_{B_{i_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r}})),$$

– and

$$\begin{array}{l} \langle \dagger, \dagger \rangle^{-1} (\mathcal{R}(\mathcal{T}_{B_{i_1},2}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r},2})) = \\ (\mathcal{R}(\mathcal{T}_{B_{i_1},2}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r},2})) \cup \langle \sharp, \sharp \rangle^{+1} (\mathcal{R}(\mathcal{T}_{B_{i_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r}})). \end{array}$$

Note that $\mathcal{R}(\mathcal{T}_{B_{i_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r}})$ is the union of $r + 2$ transductions from $\mathsf{Comp}(\mathcal{R}(\mathscr{T}))$, namely, $\tau_\emptyset$, $\mathcal{R}(\mathcal{T}_{B_{i_1},2}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r},2})$, and

$$\mathcal{R}(\mathcal{T}_{B_{i_j},1}) \circ \mathcal{R}(\mathcal{T}_{B_{i_1},2}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_{j-1}},2}) \circ \mathcal{R}(\mathcal{T}_{B_{i_{j+1}},2}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r},2})$$

for $j \in [r]$.

We then consider the left-quotients of $\mathcal{R}(\mathcal{T}_{B_{i_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r}})$ for $i_1, \cdots, i_r \in [k]$ such that $i_1 < \cdots < i_r$.

Let $i_1, \cdots, i_r \in [k]$ such that $i_1 < \cdots < i_r$. Then

– for $A \in \mathsf{Act}_{\mathsf{STD}}$ and $j \in [r]$,

$$\langle A, \dagger\rangle^{-1}(\mathcal{R}(\mathcal{T}_{B_{i_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r}})) = \\ \bigcup_{j' \in [r]} \mathcal{R}(\mathcal{T}_{B_{i_{j'}},1}) \circ \mathcal{R}(\mathcal{T}_{B_{i_1},2}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_{j'-1}},2}) \circ \mathcal{R}(\mathcal{T}_{B_{i_{j'+1}},2}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r},2}),$$

$$\langle B_{i_j}, \dagger\rangle^{-1}(\mathcal{R}(\mathcal{T}_{B_{i_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r}})) = \langle \sharp, \sharp\rangle^{+1}(\mathcal{R}(\mathcal{T}_{B_{i_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r}})),$$

– for $A \in \mathsf{Act}_{\mathsf{STD}}$,

$$\langle A, A\rangle^{-1}(\mathcal{R}(\mathcal{T}_{B_{i_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r}})) = \mathcal{R}(\mathcal{T}_{B_{i_1,2}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r,2}}),$$

– $\langle \sharp, \sharp\rangle^{-1}(\mathcal{R}(\mathcal{T}_{B_{i_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r}})) = \mathcal{R}(\mathcal{T}_{B_{i_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r}})$,
– for $i_0 \in [k]$ such that $i_0 \neq i_1, \cdots, i_r$,

$$\langle B_{i_0}, B_{i_0}\rangle^{-1}(\mathcal{R}(\mathcal{T}_{B_{i_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r}})) = \langle \sharp, \sharp\rangle^{+1}(\mathcal{R}(\mathcal{T}_{B_{i_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r}})),$$

– $\langle \dagger, \dagger\rangle^{-1}(\mathcal{R}(\mathcal{T}_{B_{i_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r}})) = (\mathcal{R}(\mathcal{T}_{B_{i_1,2}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r,2}})) \cup \langle \sharp, \sharp\rangle^{+1}(\mathcal{R}(\mathcal{T}_{B_{i_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r}}))$.

Let us consider the left-quotients of

$$\bigcup_{j' \in [r]} \mathcal{R}(\mathcal{T}_{B_{i_{j'}},1}) \circ \mathcal{R}(\mathcal{T}_{B_{i_1},2}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_{j'-1}},2}) \circ \mathcal{R}(\mathcal{T}_{B_{i_{j'+1}},2}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r},2})$$

for $i_1, \cdots, i_r \in [k]$ such that $i_1 < \cdots < i_r$. For convenience, we use $\tau^{\cup}_{i_1,\cdots,i_r}$ to denote this transduction. Then

– for $A \in \mathsf{Act}_{\mathsf{STD}}$, $\langle A, \dagger\rangle^{-1}\tau^{\cup}_{i_1,\cdots,i_r} = \tau^{\cup}_{i_1,\cdots,i_r}$,
– for $j \in [r]$, $\langle B_{i_j}, \dagger\rangle^{-1}\tau^{\cup}_{i_1,\cdots,i_r} = \langle \sharp, \sharp\rangle^{+1}(\mathcal{R}(\mathcal{T}_{B_{i_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r}}))$.

Therefore, $\mathsf{Quot}(\mathsf{Comp}(\mathcal{R}(\mathscr{T})))$ is

$$\mathsf{Comp}(\mathcal{R}(\mathscr{T})) \cup \{\mathcal{R}(\mathcal{T}_{B_{i_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r}}) \mid i_1 < \cdots < i_r\} \cup \\ \{\langle \sharp, \sharp\rangle^{+1}(\mathcal{R}(\mathcal{T}_{B_{i_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r}})) \mid i_1 < \cdots < i_r\} \cup \\ \{(\mathcal{R}(\mathcal{T}_{B_{i_1},2}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r},2})) \cup \langle \sharp, \sharp\rangle^{+1}(\mathcal{R}(\mathcal{T}_{B_{i_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r}})) \mid i_1 < \cdots < i_r\} \cup \\ \{\tau^{\cup}_{i_1,\cdots,i_r} \mid i_1 < \cdots < i_r\}.$$

Step III. We compute the closure of $\mathsf{Quot}(\mathsf{Comp}(\mathcal{R}(\mathscr{T})))$ under composition, denoted by $\mathsf{Comp}(\mathsf{Quot}(\mathsf{Comp}(\mathcal{R}(\mathscr{T}))))$.

It is easy to observe that for $B \in \mathsf{Act}_{\mathsf{STK}}, B' \in \mathsf{Act}_{\mathsf{STK}}$ such that $B \neq B'$, we have

– $\mathcal{R}(\mathcal{T}_{B,1}) \circ \mathcal{R}(\mathcal{T}_{B'}) = \mathcal{R}(\mathcal{T}_{B'}) \circ \mathcal{R}(\mathcal{T}_{B,1}) = \mathcal{R}(\mathcal{T}_{B,1}) \circ \mathcal{R}(\mathcal{T}_{B',2})$,
– $\mathcal{R}(\mathcal{T}_{B}) \circ \mathcal{R}(\mathcal{T}_{B}) = \mathcal{R}(\mathcal{T}_{B,2}) \circ \mathcal{R}(\mathcal{T}_{B}) = \mathcal{R}(\mathcal{T}_{B}) \circ \mathcal{R}(\mathcal{T}_{B,2}) = \mathcal{R}(\mathcal{T}_{B,2})$,
– $\mathcal{R}(\mathcal{T}_{B,2}) \circ \mathcal{R}(\mathcal{T}_{B'}) = \mathcal{R}(\mathcal{T}_{B'}) \circ \mathcal{R}(\mathcal{T}_{B,2})$.

Therefore, $\mathsf{Comp}(\mathsf{Quot}(\mathsf{Comp}(\mathcal{R}(\mathscr{T}))))$ is

$$\begin{aligned}
&\mathsf{Quot}(\mathsf{Comp}(\mathcal{R}(\mathscr{T}))) \cup \\
&\left\{ \begin{array}{l} \mathcal{R}(\mathcal{T}_{B_{i_1},2}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i_r},2}) \circ \\ \mathcal{R}(\mathcal{T}_{B_{i'_1}}) \circ \cdots \circ \mathcal{R}(\mathcal{T}_{B_{i'_s}}) \end{array} \middle| \begin{array}{l} i_1 < \cdots < i_r, i'_1 < \cdots < i'_s, \\ \{i_1, \cdots, i_r\} \cap \{i'_1, \cdots, i'_s\} = \emptyset \end{array} \right\} \cup \\
&\left\{ \mathcal{R}(\mathcal{T}_{B_{i'_1},2}) \circ \cdots \mathcal{R}(\mathcal{T}_{B_{i'_s},2}) \circ \tau^{\cup}_{i_1, \cdots, i_r} \middle| \begin{array}{l} i_1 < \cdots < i_r, i'_1 < \cdots < i'_s, \\ \{i_1, \cdots, i_r\} \cap \{i'_1, \cdots, i'_s\} = \emptyset \end{array} \right\}.
\end{aligned}$$

Step IV. Finally, we observe that the closure of $\mathsf{Comp}(\mathsf{Quot}(\mathsf{Comp}(\mathcal{R}(\mathscr{T}))))$ under quotient is equal to $\mathsf{Comp}(\mathsf{Quot}(\mathsf{Comp}(\mathcal{R}(\mathscr{T}))))$. Therefore, we conclude that

$$[\![\mathcal{R}(\mathscr{T})]\!] = \mathsf{Comp}(\mathsf{Quot}(\mathsf{Comp}(\mathcal{R}(\mathscr{T})))) \cup \{\tau_{id}\}.$$

It is easy to see that the size of $[\![\mathcal{R}(\mathscr{T})]\!]$ is exponential in $|\mathsf{Act}_{\mathsf{STK}}|$. $\square$

D Details of Section 5.2

D.1 Two-stack case

The construction of $\mathcal{P}_{\mathcal{A}}$ is divided into two steps. We first construct a PDS $\mathcal{P}_{A_0}$ to simulate the A_0-task of $\mathcal{A}$. Then we incorporate the aforementioned "macro" transitions into $\mathcal{P}_{A_0}$ to get $\mathcal{P}_{\mathcal{A}}$, by utilising

$$(\mathsf{Reach}(q', A, \alpha))_{(q',A,\alpha) \in Q \times (\mathsf{Act} \setminus \{A_1\}) \times \mathsf{Abs}_{A_1}}.$$

The PDS $\mathcal{P}_{A_0} = (Q_{A_0}, \varGamma_{A_0}, \varDelta_{A_0})$, where $Q_{A_0} = (Q \times \{0,1\}) \cup (Q \times \{1\} \times \{\mathsf{pop}\})$, $\varGamma_{A_0} = \mathsf{Act}_{\mathsf{STD}} \cup \{A_2, \bot\}$, and $\varDelta_{A_0}$ comprises the following transitions,

- for each transition $(q_0, \triangleright, \mathsf{start}(A_0), q') \in \varDelta$, we have $((q_0, 0), \bot, A_0\bot, (q', 0)) \in \varDelta_{A_0}$, [initialization]
- for each $b \in \{0,1\}$ and $(q', A, \mathsf{start}(B), q'') \in \varDelta$ such that $B \in \mathsf{Act}_{\mathsf{STD}}$, we have $((q', b), A, BA, (q'', b)) \in \varDelta_{A_0}$, [push a standard activity]
- for each transition $(q', A, \mathsf{start}(A_2), q'') \in \varDelta$ such that $A \in \mathsf{Act}_{\mathsf{STD}}$, we have both $((q', 0), A, A_2A, (q'', 1)) \in \varDelta_0$ and $((q', 1), A, \varepsilon, (q'', 1, \mathsf{pop})) \in \varDelta_{A_0}$, [push A_2 or start popping]
- for each $(q', A_2, \mathsf{start}(A_2), q'') \in \varDelta$, we have $((q', 1), A_2, A_2, (q'', 1)) \in \varDelta_{A_0}$, [the stack unchanged if A_2 starts itself]
- for each $q' \in Q$ and $A \in \mathsf{Act}_{\mathsf{STD}}$, $((q', 1, \mathsf{pop}), A, \varepsilon, (q', 1, \mathsf{pop})) \in \varDelta_{A_0}$, moreover, for each $q' \in Q$, $((q', 1, \mathsf{pop}), A_2, A_2, (q', 1)) \in \varDelta_{A_0}$, [pop until A_2]
- for each $b = 0, 1$ and $(q', A, \mathsf{back}, q'') \in \varDelta$ such that $A \in \mathsf{Act}_{\mathsf{STD}}$, we have $((q', b), A, \varepsilon, (q'', b)) \in \varDelta_{A_0}$, [pop a standard activity]
- for each $(q', A_2, \mathsf{back}, q'') \in \varDelta$, we have $((q', 1), A_2, \varepsilon, (q'', 0)) \in \varDelta_{A_0}$, [pop A_2]
- for each $b = 0, 1$ and each $(q, A, \square, q') \in \varDelta$, we have $((q, b), A, A, (q', b)) \in \varDelta'$. [no action]

D.2 General case

Finally, we show how the decision procedure can be generalised to the more general case that there are more than two "singleTask" activities. Let us assume that $A_1 \in \mathsf{Act}_{\mathsf{STK}}$ satisfies that $\mathsf{Aft}(A_1) = \mathsf{Aft}(A_0)$.

For the general case, the skeleton of the decision procedure for the reachability problem is similar to that of the two-task case. The additional technical intricacy lies in that the abstraction of the non-A_0 tasks is more involved and the construction of $\mathsf{Reach}(q', A, \alpha)$ is considerably more complex.

Similarly to the two-task case, to simulate the non-A_0 tasks of the ASM $\mathcal{A}$, we construct a TrPDS $\mathcal{P}_{\boxed{A_0}} = (Q_{\boxed{A_0}}, \Gamma_{\boxed{A_0}}, \mathscr{T}_{\boxed{A_0}}, \Delta_{\boxed{A_0}})$, where $\Gamma_{\boxed{A_0}} = \mathsf{Act} \cup \{\dagger, \sharp, \bot\}$. In addition, from Theorem 5, an MA $\mathcal{M}_q = (Q_q, \Gamma_{\boxed{A_0}}, \mathscr{T}_{\boxed{A_0}}, \delta_q, I_q, F_q)$ can be constructed to represent $\mathsf{pre}^*_{\mathcal{P}_{\boxed{A_0}}}(q)$.

Our goal is to define a finite abstraction for the non-A_0-tasks of $\mathcal{A}$, under the assumption that the A_0-task is nonempty in the back stack. When the A_0-task is nonempty, the non-A_0-tasks of $\mathcal{A}$ are encoded as a word $w \in (\mathsf{Act}^*_{\mathsf{STD}}(\mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\})\sharp \cup \dagger^+\sharp)^+\bot$, which describes the content of the stack of $\mathcal{P}_{\boxed{A_0}}$ where the A_1-task has not been started yet.

Similarly to the two-task case, we define a finite abstraction of the *non-A_0-tasks* of $\mathcal{A}$, incorporate it into the control states, and reduce the state reachability problem of $\mathcal{A}$ to that of a PDS $\mathcal{P}_{\mathcal{A}}$. The main idea of the reduction is to simulate each $A_0; \boxed{A_0}; A_0$ switching by a "macro"-transition of $\mathcal{P}_{\mathcal{A}}$, where an $A_0; \boxed{A_0}; A_0$ switching is a path in $\xrightarrow{\mathcal{A}}$ such that in both the first and last configuration of the path, the A_0-task is the top task, and in all the intermediate configurations, the A_0-task is *not* the top task. Suppose that, for an $A_0; \boxed{A_0}; A_0$ switching, in the first (resp. last) configuration, q' (resp. q'') is the control state and α (resp. β) is the finite abstraction of the non-A_0 tasks. Then for the "macro"-transition of $\mathcal{P}_{\mathcal{A}}$, the control state will be updated from (q', α) to (q'', β), and the stack content of $\mathcal{P}_{\mathcal{A}}$ is updated accordingly, viz.,

– if in the $A_0; \boxed{A_0}; A_0$ switching, the A_0-task becomes the top task again by starting the activity A_1 (in this case, the switching is called an *active* switching), then A_1 will be pushed into the stack of $\mathcal{P}_{\mathcal{A}}$ if the stack does not contain A_1, and all the symbols above A_1 will be popped otherwise,
– if in the $A_0; \boxed{A_0}; A_0$ switching, the A_0-task becomes the top task again by popping empty all the tasks on top of the A_0-task (in this case, the switching is called a *passive* switching), then the stack of $\mathcal{P}_{\mathcal{A}}$ stays unchanged.

Similarly to the two-task case, to construct $\mathcal{P}_{\mathcal{A}}$, we need compute $\mathsf{Reach}(q', A, \alpha)$, which is the union of

– the set of triples (q'', β, A_1) such that (q'', β) is reachable from (q', α) by an active $A_0; \boxed{A_0}; A_0$ switching, in which A is the top activity of the A_0-task in the first configuration,
– the set of triples $(q'', \beta, \bot)$ such that (q'', β) is reachable from (q', α) by a passive $A_0; \boxed{A_0}; A_0$ switching, in which A is the top activity of the A_0-task in the first configuration.

Since the non-A_0 tasks of $\mathcal{A}$ are encoded as the stack contents of $\mathcal{P}_{\boxed{A_0}}$, the finite abstraction can be actually defined for the words $w \in (\mathsf{Act}^*_{\mathsf{STD}}(\mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\})\sharp \cup \dagger^+\sharp)^+\bot$. Note that such a word w encodes the contents of the non-A_0-tasks of $\mathcal{A}$ for the situation that the A_0-task is nonempty in the back stack and the A_1-task has not been started yet.

To compute $\mathsf{Reach}(q', A, \alpha)$, we need an additional notation $\mathsf{Remv}(\alpha, B)$ for an abstraction α and $B \in \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$, which intuitively specifies how to obtain the new abstraction from the abstraction α when a B activity is started. Let us exemplify $\mathsf{Remv}(\alpha, B)$ for the situation that $w \in (\mathsf{Act}^*_{\mathsf{STD}}(\mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\})\sharp \cup \dagger^+\sharp)^+\bot$ is the current stack content of $\mathcal{P}_{\boxed{A_0}}$, the abstraction of w is α, and $w = w_1(w_2B)\sharp w_3\bot$ such that $w_1, w_3 \in (\mathsf{Act}^*_{\mathsf{STD}}(\mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\})\sharp \cup \dagger^+\sharp)^*$, $B \in \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$, and $w_2 \in \mathsf{Act}^*_{\mathsf{STD}}$. If the activity B is started in $\mathcal{A}$, then accordingly, the stack content of $\mathcal{P}_{\boxed{A_0}}$ is changed by rewriting the subword w_2B, which encodes the content of the B-task of $\mathcal{A}$, into $\dagger^{|w_2|+1}$, and adding $B\sharp$ to the front of the word. Thus, $\mathsf{Remv}(\alpha, B)$ is the abstraction of $w_1\dagger^{|w_2|+1}\sharp w_3\bot$, the new stack content of $\mathcal{P}_{\boxed{A_0}}$ (with the prefix $B\sharp$ ignored for technical reasons).

Abstraction. As a result, to facilitate the computation of $\mathsf{Remv}(\alpha, B)$, for a word $w \in (\mathsf{Act}^*_{\mathsf{STD}}(\mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\})\sharp \cup \dagger^+\sharp)^+\bot$, we define the abstraction of w, denoted by $\alpha(w)$, by splitting w into the segments corresponding to the nontrivial blocks or the maximal segments containing only trivial blocks. Specifically, suppose that w is split into $w_1, \cdots, w_k$, (therefore $w = w_1 \cdots w_k\bot$), such that for each $i \in [k]$, either $w_i \in \mathsf{Act}^+\sharp$, or $w_i \in (\dagger^+\sharp)^+$, moreover, for each $i \in [k-1]$, we have $w_i \in \mathsf{Act}^+\sharp$ or $w_{i+1} \in \mathsf{Act}^+\sharp$. Note that k is linear in the number of tasks in $\mathcal{A}$, more precisely, $k \leq 2 \cdot |\mathsf{rng}(\mathsf{Aft})| + 1$. If $w_i \in \mathsf{Act}^+\sharp$, we say w_i is a *nontrivial* segment of w, otherwise, it is a *trivial* segment of w. Then $\alpha(w)$ is defined as $\alpha = [\alpha_1, \cdots, \alpha_k]$ such that for each $i \in [k]$,

– if w_i is a trivial segment of w, then $\alpha_i = (\dagger, \alpha_{i,1})$, where $\alpha_{i,1}$ is the set of pairs $(q', q'') \in Q_q \times Q_q$ such that $q' \xrightarrow{w_i} q''$ in $\mathcal{M}_q$,
– otherwise, we have $\alpha_i = (A_i, \alpha_{i,1}, \alpha_{i,2})$, where $w_i \in \mathsf{Act}^*_{\mathsf{STD}}A_i\sharp$, $\alpha_{i,1}$ is the set of pairs $(q', q'') \in Q_q \times Q_q$ such that $q' \xrightarrow{w_i} q''$ in $\mathcal{M}_q$, and $\alpha_{i,2}$ is the set of pairs $(q', q'') \in Q_q \times Q_q$ such that $q' \xrightarrow{\dagger^{|w_i|-1}\sharp} q''$ in $\mathcal{M}_q$.

Let $\mathsf{Abs}_{\boxed{A_0, A_1}}$ denote the set of abstractions of words from $(\mathsf{Act}^*_{\mathsf{STD}}(\mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\})\sharp \cup \dagger^+\sharp)^+\bot$. By convention, we assume that $\mathsf{Abs}_{\boxed{A_0, A_1}}$ contains a special element $\bot$ to denote the special situation that there are no non-A_0 tasks in the back stack. For $\alpha = [\alpha_1, \cdots, \alpha_k] \in \mathsf{Abs}_{\boxed{A_0, A_1}}$, let $\mathcal{L}_\alpha$ denote the set of words $w \in (\mathsf{Act}^*_{\mathsf{STD}}(\mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\})\sharp \cup \dagger^+\sharp)^+\bot$ such that $\alpha(w) = \alpha$. Note that $\mathcal{L}_\alpha$ is a regular language, that is, we can construct an NFA $\mathcal{B}_\alpha$ to accept the set of words $w \in (\mathsf{Act}^*_{\mathsf{STD}}(\mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\})\sharp \cup \dagger^+\sharp)^+\bot$ such that the segments of w satisfy the aforementioned state-reachability constraints of $\mathcal{M}_q$ in the definition of $\alpha(w)$. Let $\alpha = [\alpha_1, \cdots, \alpha_k]$. For each $i \in [k]$, let $\mathcal{B}_{\alpha_i}$ be the product of the automata corresponding to the elements of α_i. For instance, if $\alpha_i = (A_i, \alpha_{i,1}, \alpha_{i,2})$ with $A_i \in \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$, then $\mathcal{B}_{\alpha_i}$ is the product of $\mathcal{M}_q(q', q'')$ for $(q', q'') \in \alpha_{i,1}$ and $\mathcal{M}'_q(q', q'')$ for $(q', q'') \in \alpha_{i,2}$, where $\mathcal{M}'_q$ is obtained from $\mathcal{M}_q$ by removing all the

Act-transitions and replacing each transition $(q_1, \dagger, q_2)$ with multiple transitions (q_1, A, q_2) for each $A \in \mathsf{Act} \setminus \{A_1\}$. Moreover, let $\mathcal{B}_\alpha$ be the NFA obtained by composing sequentially the NFA $\mathcal{B}_{\alpha_i}$ for $i \in [k]$. Therefore, the size of $\mathcal{B}_\alpha$ is at most exponential in that of α.

Computation of $\mathsf{Reach}(q', A, \alpha)$. As mentioned before, in order to define $\mathsf{Reach}(q', A, \alpha)$, we need compute $\mathsf{Remv}(\alpha, B)$ for $\alpha \in \mathsf{Abs}_{\boxed{A_0, A_1}}$ and $B \in \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$.

Suppose $\alpha = [\alpha_1, \cdots, \alpha_k] \in \mathsf{Abs}_{\boxed{A_0, A_1}}$ and $B \in \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$. Then $\mathsf{Remv}(\alpha, B)$ is defined as follows: If there does not exist $i \in [k]$ such that $\alpha_i = (B, \alpha_{i,1}, \alpha_{i,2})$, then $\mathsf{Remv}(\alpha, B) = \alpha$. Otherwise, let us assume $\alpha_i = (B, \alpha_{i,1}, \alpha_{i,2})$ for some $i \in [k]$. Then $\mathsf{Remv}(\alpha, B)$ is defined as follows:

- If $i = 1$ and $\alpha_2 = (C, \alpha_{2,1}, \alpha_{2,2})$ for some $C \in \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$, then
$$\mathsf{Remv}(\alpha, B) = [(\dagger, \alpha_{1,2}), \alpha_2, \cdots, \alpha_k].$$
- If $i = 1$ and $\alpha_2 = (\dagger, \alpha_{2,1})$, then
$$\mathsf{Remv}(\alpha, B) = [(\dagger, \alpha_{1,2} \cdot \alpha_{2,1}), \alpha_3, \cdots, \alpha_k],$$
where $\alpha_{1,2} \cdot \alpha_{2,1}$ is the composition of the two relations $\alpha_{1,2}$ and $\alpha_{2,1}$.
- If $i = k$ and $\alpha_{k-1} = (C, \alpha_{k-1,1}, \alpha_{k-1,2})$ for some $C \in \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$, then
$$\mathsf{Remv}(\alpha, B) = [\alpha_2, \cdots, \alpha_{k-1}, (\dagger, \alpha_{k,2})].$$
- If $i = k$ and $\alpha_{k-1} = (\dagger, \alpha_{k-1,1})$, then
$$\mathsf{Remv}(\alpha, B) = [\alpha_2, \cdots, \alpha_{k-2}, (\dagger, \alpha_{k-1,1} \cdot \alpha_{k,2})].$$
- If $1 < i < k$, $\alpha_{i-1} = (\dagger, \alpha_{i-1,1})$ and $\alpha_{i+1} = (\dagger, \alpha_{i+1,1})$, then
$$\mathsf{Remv}(\alpha, B) = [\alpha_1, \cdots, \alpha_{i-2}, (\dagger, \alpha_{i-1,1} \cdot \alpha_{i,2} \cdot \alpha_{i+1,1}), \alpha_{i+2}, \cdots, \alpha_k].$$
- If $1 < i < k$, $\alpha_{i-1} = (C, \alpha_{i-1,1}, \alpha_{i-1,2})$ for some $C \in \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$ and $\alpha_{i+1} = (\dagger, \alpha_{i+1,1})$, then
$$\mathsf{Remv}(\alpha, B) = \alpha_1, \cdots, \alpha_{i-2}, \alpha_{i-1}, (\dagger, \alpha_{i,2} \cdot \alpha_{i+1,1}), \alpha_{i+2}, \cdots, \alpha_k],$$
- If $1 < i < k$, $\alpha_{i-1} = (\dagger, \alpha_{i-1,1})$, and $\alpha_{i+1} = (C, \alpha_{i+1,1}, \alpha_{i+1,2})$ for some $C \in \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$, then
$$\mathsf{Remv}(\alpha, B) = [\alpha_1, \cdots, \alpha_{i-2}, (\dagger, \alpha_{i-1,1} \cdot \alpha_{i,2}), \alpha_{i+1}, \cdots, \alpha_k],$$
- If $1 < i < k$, $\alpha_{i-1} = (C, \alpha_{i-1,1}, \alpha_{i-1,2})$ for some $C \in \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$ and $\alpha_{i+1} = (C', \alpha_{i+1,1}, \alpha_{i+1,2})$ for some $C' \in \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$, then
$$\mathsf{Remv}(\alpha, B) = [\alpha_1, \cdots, \alpha_{i-2}, \alpha_{i-1}, (\dagger, \alpha_{i,2}), \alpha_{i+1}, \cdots, \alpha_k].$$

For $\Lambda = \{B_1, \ldots, B_r\}$, we define $\mathsf{Remv}(\Lambda, \alpha) = \mathsf{Remv}(B_1, (\ldots, \mathsf{Remv}(B_r, \alpha)))$. (The order of activities in Λ is irrelevant here.)

In order to compute $\mathsf{Reach}(q', A, \alpha)$, we construct a $\mathcal{P}_{\boxed{A_0, A_1}}$ to simulate the non-A_0 tasks of $\mathcal{A}$, with an additional assumption that the A_1-task is not started. More precisely, $\mathcal{P}_{\boxed{A_0, A_1}} = (Q_{\boxed{A_0, A_1}}, \Gamma_{\boxed{A_0, A_1}}, \mathscr{T}_{\boxed{A_0, A_1}}, \Delta_{\boxed{A_0, A_1}})$, where

- $Q_{\boxed{A_0, A_1}} = (Q \cup (Q \times Q) \cup \{q'_0\}) \times 2^{\mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}}$ (where q'_0 is a fresh state),
- $\Gamma_{\boxed{A_0, A_1}} = (\mathsf{Act} \setminus \{A_1\}) \cup \{\dagger, \sharp, \bot\}$,
- $\mathscr{T}_{\boxed{A_0, A_1}} = \{\mathcal{T}_{id}\} \cup \{\mathcal{T}_{B,1}, \mathcal{T}_{B,2} \mid B \in \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}\}$,
- and $\Delta_{\boxed{A_0, A_1}}$ comprises the following transitions,
  - for each $(q', A, \mathsf{start}(B), q'') \in \Delta$ such that $A \in \mathsf{Act}_{\mathsf{STD}} \cup \{A_1\}$ and $B \in \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$, we have $((q'_0, \emptyset), \bot, B\sharp\bot, \mathcal{T}_{id}, (q'', \{B\})) \in \Delta_{\boxed{A_0, A_1}}$,
  - for each $(q', A, \mathsf{start}(B), q'') \in \Delta$ and $\Lambda \subseteq \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$ with $B \in \mathsf{Act}_{\mathsf{STD}}$, we have $(((q', \Lambda), A, BA, \mathcal{T}_{id}, (q'', \Lambda)) \in \Delta_{\boxed{A_0, A_1}}$,
  - for each $(q', A, \mathsf{start}(A), q'') \in \Delta$ and $\Lambda \subseteq \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$ with $A \in \mathsf{Act}_{\mathsf{STK}}$, we have $((q', \Lambda), A, A, \mathcal{T}_{id}, (q'', \Lambda)) \in \Delta_{\boxed{A_0, A_1}}$,
  - for each $(q', A, \mathsf{start}(B), q'') \in \Delta$ with $B \in \mathsf{Act}_{\mathsf{STK}}$ and $A \neq B$,
    * for each $\Lambda \subseteq \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$ such that $B \in \Lambda$, we have
$$((q', \Lambda), A, B\sharp\dagger, \mathcal{T}_{B,1}, (q'', \Lambda)) \in \Delta_{\boxed{A_0, A_1}}$$
(corresponding to the situation that the top task is a B-task),
    * for each $\Lambda \subseteq \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$, we have
$$((q', \Lambda), A, B\sharp A, \mathcal{T}_{B,2}, (q'', \Lambda \cup \{B\})) \in \Delta_{\boxed{A_0, A_1}}$$
(corresponding to the situation that the top task is not a B-task),
  - for each $(q', A, \mathsf{back}, q'') \in \Delta$ and $\Lambda \subseteq \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$, we have the transitions $((q', \Lambda), A, \varepsilon, \mathcal{T}_{id}, (q', q'', \Lambda)) \in \Delta_{\boxed{A_0, A_1}}$, $((q', q'', \Lambda), \gamma, \gamma, \mathcal{T}_{id}, (q'', \Lambda)) \in \Delta_{\boxed{A_0, A_1}}$ for each $\gamma \in (\mathsf{Act} \setminus \{A_1\}) \cup \{\bot\}$, $((q', q'', \Lambda), \sharp, \varepsilon, \mathcal{T}_{id}, (q', q'', \Lambda)) \in \Delta_{\boxed{A_0, A_1}}$, and $((q', q'', \Lambda), \dagger, \varepsilon, \mathcal{T}_{id}, (q', q'', \Lambda)) \in \Delta_{\boxed{A_0, A_1}}$,
  - for each $(q', A, \square, q'') \in \Delta$ and $\Lambda \subseteq \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$, we have
$$((q', \Lambda), A, A, \mathcal{T}_{id}, (q'', \Lambda)) \in \Delta_{\boxed{A_0, A_1}}.$$

We are ready to define $\mathsf{Reach}(q', A, \alpha)$. For each $(q', A, \alpha) \in Q \times (\mathsf{Act}_{\mathsf{STD}} \cup \{A_1\}) \times \mathsf{Abs}_{\boxed{A_0, A_1}}$, $\mathsf{Reach}(q', A, \alpha)$ comprises

- the triples (q'', β, A_1) such that there exist $B \in \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$, $q_1, q_2 \in Q$, $\Lambda \subseteq \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$, $C \in \mathsf{Act} \setminus \{A_1\}$, $w_1 \in C(\Gamma_{\boxed{A_0, A_1}})^*$, and $w_2 \in \mathcal{L}_{\mathsf{Remv}(\Lambda, \alpha)}$ satisfying that $(q', A, \mathsf{start}(B), q_1) \in \Delta$, $((q_1, \{B\}), B\sharp\bot) \xRightarrow{\mathcal{P}_{\boxed{A_0, A_1}}} ((q_2, \Lambda), w_1)$, $(q_2, C, \mathsf{start}(A_1), q'') \in \Delta$, and $(w_1 \bot^{-1}) w_2 \in \mathcal{L}_\beta$,
- or the triples $(q'', \mathsf{Remv}(\Lambda, \alpha), \bot)$ such that there exist $q_1 \in Q$, $B \in \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$, and $\Lambda \subseteq \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$ satisfying that $(q', A, \mathsf{start}(B), q_1) \in \Delta$ and $((q_1, \{B\}), B\sharp\bot) \xRightarrow{\mathcal{P}_{\boxed{A_0, A_1}}} ((q'', \Lambda), \bot)$.

Via Theorem 5, from $\mathcal{P}_{\boxed{A_0, A_1}}$, we construct a TrNFA

$$\mathcal{B}_{(q',A)} = (Q_{(q',A)}, \Gamma_{\boxed{A_0, A_1}}, \mathscr{T}_{\boxed{A_0, A_1}}, \delta_{(q',A)}, I_{(q',A)}, F_{(q',A)})$$

to represent $\mathsf{post}^*_{\mathcal{P}'_{\boxed{A_0, A_1}}}(\mathsf{Conf}_{(q',A)})$, where

$$\mathsf{Conf}_{(q',A)} = \bigcup_{B \in \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}, (q',A,\mathsf{start}(B),q_1) \in \Delta} \{(q_1, \{B\})\} \times \{B\sharp\bot\}.$$

Note that an equivalent MA $\mathcal{M}_{(q',A)}$ can be constructed from $\mathcal{B}_{(q',A)}$. The size of $\mathcal{M}_{(q',A)}$ is polynomial in that of $\mathcal{B}_{(q',A)}$, thus polynomial in $|\mathcal{P}_{\boxed{A_0, A_1}}| + |[\![\mathcal{R}(\mathscr{T}_{\boxed{A_0, A_1}})]\!]|$.

The sets $\mathsf{Reach}(q', A, \alpha)$ are characterised algorithmically by the following lemma.

Lemma 3. *For each* $(q', A, \alpha) \in Q \times (\mathsf{Act}_{\mathsf{STD}} \cup \{A_1\}) \times \mathsf{Abs}_{\boxed{A_0, A_1}}$, $\mathsf{Reach}(q', A, \alpha)$ *is the union of*

– *the set of triples* (q'', β, A_1) *such that there exist* $q_2 \in Q$, $B \in \mathsf{Act} \setminus \{A_1\}$, *and* $\Lambda \subseteq \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$ *satisfying that* $(q_2, B, \mathsf{start}(A_1), q'') \in \Delta$, *and*

$$\mathcal{L}_\beta \cap (([B(\Gamma_{\boxed{A_0, A_1}})^* \cap \mathcal{L}(\mathcal{M}_{(q',A)}((q_2, \Lambda)))]\bot^{-1}) \cdot \mathcal{L}_{\mathsf{Remv}(\Lambda,\alpha)}) \neq \emptyset.$$

– *the set of triples* $(q'', \mathsf{Remv}(\Lambda, \alpha), \bot)$ *such that* $\bot \in \mathcal{L}(\mathcal{M}_{(q',A)}((q'', \Lambda)))$ *for some nonempty* $\Lambda \subseteq \mathsf{Act}_{\mathsf{STK}}$ *(the "nonempty" constraint is due to the fact that in a switching at least one* STK*-activity is started).*

Finally, the construction of $\mathcal{P}_{\mathcal{A}}$ is the same as the two-task case, by utilising $(\mathsf{Reach}(q', A, \alpha))_{(q',A,\alpha) \in Q \times (\mathsf{Act}_{\mathsf{STD}} \cup \{A_1\}) \times \mathsf{Abs}_{\boxed{A_0, A_1}}}$.

Construction of $\mathcal{P}_{\mathcal{A}}$. We first construct a PDS $\mathcal{P}_{A_0} = (Q_{A_0}, \Gamma_{A_0}, \Delta_{A_0})$, to simulate the A_0-task of $\mathcal{A}$. Here $Q_{A_0} = (Q \times \{0,1\}) \cup (Q \times \{1\} \times \{\mathsf{pop}\})$, $\Gamma_{A_0} = \mathsf{Act}_{\mathsf{STD}} \cup \{A_1, \bot\}$, and Δ_{A_0} comprises the transitions. The construction of $\mathcal{P}_{A_0}$ is the same as the two-task case, except that A_2 is replaced by A_1.

We then define the PDS $\mathcal{P}_{\mathcal{A}} = (Q_{\mathcal{A}}, \Gamma_{A_0}, \Delta_{\mathcal{A}})$, where $Q_{\mathcal{A}} = (\mathsf{Abs}_{\boxed{A_0, A_1}} \times Q_{A_0}) \cup \{q\}$, and $\Delta_{\mathcal{A}}$ comprises the following transitions,

– for each $(p, \gamma, w, p') \in \Delta_{A_0}$ and $\alpha \in \mathsf{Abs}_{\boxed{A_0, A_1}}$, we have $((\alpha, p), \gamma, w, (\alpha, p')) \in \Delta_{\mathcal{A}}$ (here $p, p' \in Q_{A_0}$, that is, of the form (q', b) or (q', b, pop)), [**behaviour of the** A_0**-task**]
– for each $(q', A, \alpha) \in Q \times (\mathsf{Act}_{\mathsf{STD}} \cup \{A_1\}) \times \mathsf{Abs}_{\boxed{A_0, A_1}}$ and $b \in \{0, 1\}$ such that $\mathcal{L}(\mathcal{M}_{(q',A)}((q, \Lambda))) \neq \emptyset$ for some $\Lambda \subseteq \mathsf{Act}_{\mathsf{STK}} \setminus \{A_1\}$, we have $((\alpha, (q', b)), A, A, q) \in \Delta_{\mathcal{A}}$,
[**switch to the non-**A_0 **tasks and reach** q **before switching back**]
– for each $(q', A, \alpha) \in Q \times (\mathsf{Act}_{\mathsf{STD}} \cup \{A_1\}) \times \mathsf{Abs}_{\boxed{A_0, A_1}}$ and $(q'', \beta, A_1) \in \mathsf{Reach}(q', A, \alpha)$ such that $\beta \neq \bot$,
  • if $A \neq A_1$, then we have $((\alpha, (q', 0)), A, A_1 A, (\beta, (q'', 1))) \in \Delta_{\mathcal{A}}$ and $((\alpha, (q', 1)), A, \varepsilon, (\beta, (q'', 1, \mathsf{pop}))) \in \Delta_{\mathcal{A}}$,

  - if $A = A_1$, then we have $((\alpha, (q', 1)), A_1, A_1, (\beta, (q'', 1))) \in \Delta_{\mathcal{A}}$,

    [switch to the non-A_0 tasks and switch back to the A_0-task later by launching A_1]

– for each $(q', A, \alpha) \in Q \times (\mathsf{Act}_{\mathsf{STD}} \cup \{A_1\}) \times \mathsf{Abs}_{\boxed{A_0,\, A_1}}$, $(q'', \beta, \bot) \in \mathsf{Reach}(q', A, \alpha)$ and $b \in \{0, 1\}$, we have $((\alpha, (q', b)), A, A, (\beta, (q'', b))) \in \Delta_{\mathcal{A}}$,

  [switch to the non-A_0 tasks and switch back to the A_0-task later when the non-A_0 tasks become empty]

– for each $\alpha \in \mathsf{Abs}_{\boxed{A_0,\, A_1}}$, $b \in \{0, 1\}$, and $A \in \mathsf{Act}_{\mathsf{STD}} \cup \{A_1\}$, $((\alpha, (q, b)), A, A, q) \in \Delta_{\mathcal{A}}$,

  [q is reached when the A_0-task is the top task]

– for each $q' \in Q$ and $\alpha \in \mathsf{Abs}_{\boxed{A_0,\, A_1}}$ with $\mathcal{L}(\mathcal{B}_\alpha) \cap \mathcal{L}(\mathcal{M}_q(q')) \neq \emptyset$, we have $((\alpha, (q', 0)), \bot, \bot, q) \in \Delta_{\mathcal{A}}$.

  [q is reached after the A_0-task becomes empty and the some non-A_0 task becomes the top task]

Complexity analysis. We apply the complexity of the aforementioned construction and computation as follows.

– The size of $\mathsf{Abs}_{\boxed{A_0,\, A_1}}$ is exponential in $|\mathcal{M}_q|$ and $|\mathsf{Act}_{\mathsf{STK}}|$.
– For each (q', A, α), the computation of $\mathsf{Reach}(q', A, \alpha)$ takes time exponential in $|\mathsf{Act}_{\mathsf{STK}}|$ and $|\mathcal{M}_q|$. Therefore, the computation of all these $\mathsf{Reach}(q', A, \alpha)$ takes time exponential in $|\mathsf{Act}_{\mathsf{STK}}|$ and $|\mathcal{M}_q|$.
– Since $|\mathcal{M}_q|$ is polynomial in $|\mathcal{P}_{\boxed{A_0}}| + |[\![\mathcal{R}(\mathscr{T}_{\boxed{A_0}})]\!]|$, it holds that $|\mathcal{M}_q|$ is polynomial in $|\mathcal{A}|$ and exponential in $|\mathsf{Act}_{\mathsf{STK}}|$.
– Since the size of $\mathcal{P}_{\mathcal{A}}$ is polynomial in that of $\mathcal{P}_{A_0}$ and the collection of $\mathsf{Reach}(q', A, \alpha)$'s, we deduce that the construction of the PDS $\mathcal{P}_{\mathcal{A}}$ takes time doubly exponential in $|\mathsf{Act}_{\mathsf{STK}}|$ and exponential in $|\mathcal{A}|$.
– From Theorem 4, we know that the state reachability problem of PDS can be solved in polynomial time. Therefore, the state reachability of $\mathcal{A}$ can be solved in doubly exponential time in $|\mathcal{A}|$.

We conclude that the state reachability problem of STK-dominating ASM is in 2-EXPTIME.