Supplementary material to

" Android Multitasking Mechanism: Formal Semantics and Static Analysis of Apps"

A ANALYSIS OF THE INTENT FLAGS NOT CONSIDERED IN THIS PAPER

There are 21 intent flags whose names start with FLAG_ACTIVITY. In this paper, we only consider 7 intent flags. We explain why by having a closer look at the rest 14 flags.

- The following 7 flags are irrelevant to the evolution of the task stack,
  - FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS,
  - FLAG_ACTIVITY_FORWARD_RESULT,
  - FLAG_ACTIVITY_NO_ANIMATION,
  - FLAG_ACTIVITY_NO_USER_ACTION,
  - FLAG_ACTIVITY_MATCH_EXTERNAL,
  - FLAG_ACTIVITY_PREVIOUS_IS_TOP,
  - FLAG_ACTIVITY_RETAIN_IN_RECENTS.
- The following 2 flags are not normally set by app code,
  - FLAG_ACTIVITY_BROUGHT_TO_FRONT,
  - FLAG_ACTIVITY_LAUNCHED_FROM_HISTORY.
- The flag FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET was depreciated.
- The flag FLAG_ACTIVITY_LAUNCH_ADJACENT is used only in the split-screen multi-window mode.
- The other 3 flags, namely,
  - FLAG_ACTIVITY_RESET_TASK_IF_NEEDED,
  - FLAG_ACTIVITY_NO_HISTORY,
  - FLAG_ACTIVITY_NEW_DOCUMENT,

  are arguably less commonly used in Android apps, although they are related to the evolution of the task stack.

B FORMAL SEMANTICS OF ASM

Tasks and configurations. A *task* is encoded as a word $S = [A_1, \cdots, A_n] \in \mathsf{Act}^+$ which denotes the content of its back stack. For $S_1 \in \mathsf{Act}^*$ and $S_2 \in \mathsf{Act}^*$, $S_1 \cdot S_2$ denotes the concatenation of S_1 and S_2, and [] denotes the empty word in Act^*. A *task stack* is encoded as a non-empty sequence $\rho = ((S_1, A_1), \cdots, (S_n, A_n))$, where for each $i \in [n]$, S_i is a task and A_i is the real activity of S_i. For any activity A, we refer to an A-task as a task whose real activity is A. The tasks S_1 and S_n are called the top and the bottom task respectively. (Intuitively, S_1 is the foreground task.) ε is used to denote the empty task stack.

A task is called the *main task* of the task stack if it is the first task that was created when launching the app. Note that the current task stack may *not* contain the main task, since it may have been popped out from the task stack. This notion is introduced since, as we will see, the semantics of ASM is also dependent on whether the task stack contains the main task or not.

A *configuration* of $\mathcal{A}$ is a pair $= (\rho, \ell)$, where $\rho = ((S_1, A_1), \cdots, (S_n, A_n))$ with $S_i = [B_{i,1}, \cdots, B_{i,m_i}]$ for each $i \in [n]$ and $\ell \in [n] \cup \{0\}$. We require (ρ, ℓ) to satisfy that if $\ell \in [n]$, then $A_\ell = A_0$. Intuitively, ℓ is the index of the main task. If $\ell = 0$, then ρ contains no main task. Let $\mathsf{Conf}_{\mathcal{A}}$ denote the set of configurations of $\mathcal{A}$. The *initial* configuration of $\mathcal{A}$ is $(\varepsilon, 0)$. The *height* of ρ is defined as $\max_{i \in [m]} |S_i|$. By convention, the height of ε is defined as 0.

Auxiliary functions and predicates. To specify the transition relation precisely and concisely, we define the following functions and predicates. Let (ρ, ℓ) be a configuration with $\rho = ((S_1, A_1), \cdots, (S_n, A_n))$, and $S = [B_1, \cdots, B_m]$ be a task.

- $\mathsf{Top}(S) = A_1$, $\mathsf{Btm}(S) = A_m$.
- $\mathsf{TopTsk}(\rho) = S_1$, $\mathsf{TopAct}(\rho) = \mathsf{Top}(\mathsf{TopTsk}(\rho))$.
- $\mathsf{Push}((\rho, \ell), B) = ((([B] \cdot S_1), A_1), (S_2, A_2), \cdots, (S_n, A_n)), \ell)$.
- $\mathsf{MvAct2Top}((\rho, \ell), B) = ((([B] \cdot S_1' \cdot S_1''), (S_2, A_2), \cdots, (S_n, A_n)), \ell)$, if $S_1 = S_1' \cdot [B] \cdot S_1''$ with $S_1' \in (\mathsf{Act} \setminus \{B\})^*$.
- $\mathsf{ClrTop}((\rho, \ell), B) = (((S_1'', A_1), (S_2, A_2), \cdots, (S_n, A_n)), \ell)$ if $S_1 = S_1' \cdot S_1''$ with $S_1' \in (\mathsf{Act} \setminus \{B\})^* B$,
- $\mathsf{ClrTsk}((\rho, \ell)) = ((([], A_1), (S_2, A_2), \cdots, (S_n, A_n)), \ell)$,
- $\mathsf{MvTsk2Top}((\rho, \ell), S_i) = (((S_i, A_i), (S_1, A_1), \cdots, (S_{i-1}, A_{i-1}), (S_{i+1}, A_{i+1}), \cdots, (S_n, A_n)), \ell')$, where $\ell' = \begin{cases} 0 & \text{if } \ell = 0, \\ 1 & \text{if } \ell = i, \\ \ell & \text{if } i+1 \le \ell \le n, \\ \ell+1 & \text{if } 1 \le \ell \le i-1. \end{cases}$

  [Note that ℓ' is the simply the new position of the main task.]
- $\mathsf{NewTsk}((\rho, \ell), B) = ((([B], B), (S_1, A_1), \cdots, (S_n, A_n)), \ell')$, where $\ell' = 0$ if $\ell = 0$, and $\ell' = \ell + 1$ otherwise.
- $\mathsf{GetRealTsk}(\rho, B) = S_i$ such that $i \in [n]$ is the *minimum* index satisfying $A_i = B$ if such an index i exists; $\mathsf{GetRealTsk}(\rho, B) = *$ otherwise.

- $\mathsf{GetTsk}(\rho, B) = S_i$ such that $i \in [n]$ is the *minimum* index satisfying $\mathsf{Aft}(A_i) = \mathsf{Aft}(B)$, if such an index i exists; $\mathsf{GetTsk}(\rho, B) = *$ otherwise.
- $$\mathsf{RmAct}((\rho, \ell), i) = \begin{cases} (0, (((S_1, A_1), \cdots, (S_{i-1}, A_{i-1}), (S_{i+1}, A_{i+1}), \cdots, (S_n, A_n)), 0)) & \text{if } m = 1 \text{ and } \ell = 0 \text{ or } i, \\ (0, (((S_1, A_1), \cdots, (S_{i-1}, A_{i-1}), (S_{i+1}, A_{i+1}), \cdots, (S_n, A_n)), \ell)) & \text{if } m = 1 \text{ and } 1 \le \ell \le i-1, \\ (0, (((S_1, A_1), \cdots, (S_{i-1}, A_{i-1}), (S_{i+1}, A_{i+1}), \cdots, (S_n, A_n)), \ell - 1)) & \text{if } m = 1 \text{ and } i+1 \le \ell \le n, \\ (i, (((S_1, A_1), \cdots, (S_{i-1}, A_{i-1}), ([B_2, \cdots, B_m], A_i), (S_{i+1}, A_{i+1}), \cdots, (S_n, A_n)), \ell)) & \text{if } m > 1. \end{cases}$$

Transition relation. We define the relation $\xrightarrow{\mathcal{A}}$ which comprises the quadruples $((\rho, \ell), \tau, i, (\rho', \ell')) \in \mathsf{Conf}_{\mathcal{A}} \times \Delta \times \{0, 1, 2\} \times \mathsf{Conf}_{\mathcal{A}}$ to formalise the semantics of $\mathcal{A}$. For readability, we write $((\rho, \ell), \tau, i, (\rho', \ell')) \in \xrightarrow{\mathcal{A}}$ as $(\rho, \ell) \xrightarrow[\tau, i]{\mathcal{A}} (\rho', \ell')$. Intuitively, $i = 0, 1, 2$ corresponds to the cases that the top task of ρ is absent in ρ', remains to be the top task of ρ', or becomes the task immediately below the top task, of ρ' respectively. Let (ρ, ℓ) be the current configuration.

If (ρ, ℓ) is the initial configuration $(\varepsilon, 0)$, then for $\tau = \triangleright \to \mathsf{start}(A_0, \bigwedge_{F \in \mathcal{F}} \neg F)$, we have $(\rho, \ell) \xrightarrow[\tau, 0]{\mathcal{A}} ([A_0], 1)$. (Note that 0 is used because there is no top task in ρ.)

If (ρ, ℓ) is *not* the initial configuration, then we have that $\rho = ((S_1, A_1), \cdots, (S_n, A_n))$ for some $n \ge 1$. Let $A = \mathsf{TopAct}(\rho)$. In the following, we will present the semantics for the transition rules $\tau = A \xrightarrow{\mathsf{start}(\phi)} B$ first, the case $\tau = A \xrightarrow{\mathsf{finishStart}(\phi)} B$ is similar, we present later.

We distinguish two cases, i.e., $\phi \models \neg\mathsf{TOH}$ or $\phi \models \mathsf{TOH}$. We start with the first case.

The case $\phi \models \neg\mathsf{TOH}$

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

- if $\mathsf{GetRealTsk}(\rho, B) = S_j$ for some $j \in [n]$, then
  - if $j = 1$, then $i = 1$ and $(\rho', \ell') = (\rho, \ell)$,
  - if $j \neq 1$, then $i = 2$ and $(\rho', \ell') = \mathsf{MvTsk2Top}((\rho, \ell), S_j)$,
- if $\mathsf{GetRealTsk}(\rho, B) = *$, then $i = 2$ and $(\rho', \ell') = \mathsf{NewTsk}((\rho, \ell), B)$.

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

- if $\mathsf{GetRealTsk}(\rho, B) = S_j$ or $\mathsf{GetRealTsk}(\rho, B) = * \wedge \mathsf{GetTsk}(\rho, B) = S_j$, then $i = 1$ if $j = 1$, and $i = 2$ otherwise. Moreover,
  - if $\phi \models \neg\mathsf{CTK}$, then
    - if $B \notin S_j$, then $(\rho', \ell') = \mathsf{Push}(\mathsf{MvTsk2Top}((\rho, \ell), S_j), B)$,
    - if $B \in S_j$, then $(\rho', \ell') = \mathsf{Push}(\mathsf{ClrTop}(\mathsf{MvTsk2Top}((\rho, \ell), S_j), B), B)$
  - if $\phi \models \mathsf{CTK}$, then $(\rho', \ell') = \mathsf{Push}(\mathsf{ClrTsk}(\mathsf{MvTsk2Top}((\rho, \ell), S_j)), B)$,
- if $\mathsf{GetTsk}(\rho, B) = *$, then $i = 2$ and $(\rho', \ell') = \mathsf{NewTsk}((\rho, \ell), B)$,

Case $\mathsf{Lmd}(B) = \mathsf{STD}$

- if $\mathsf{Lmd}(A) \neq \mathsf{SIT}$ and $\phi \models \neg\mathsf{NTK}$, then $i = 1$ and
  - if $\phi \models \mathsf{STP} \vee \mathsf{RTF} \vee \mathsf{CTP}$ and $B = \mathsf{TopAct}(\rho)$, then $(\rho', \ell') = (\rho, \ell)$,
  - if $\phi \models \neg\mathsf{STP} \wedge \neg\mathsf{RTF} \wedge \neg\mathsf{CTP}$, or $\phi \models \mathsf{STP} \wedge \neg\mathsf{RTF} \wedge \neg\mathsf{CTP}$ and $B \neq \mathsf{TopAct}(\rho)$, or $\phi \models \mathsf{RTF} \vee \mathsf{CTP}$ and $B \notin \mathsf{TopTsk}(\rho)$, then $(\rho', \ell') = \mathsf{Push}((\rho, \ell), B)$,
  - if $\phi \models \mathsf{RTF} \wedge \neg\mathsf{CTP}$ and $B \in \mathsf{TopTsk}(\rho)$, then $(\rho', \ell') = \mathsf{MvAct2Top}((\rho, \ell), B)$,
  - if $\phi \models \mathsf{CTP}$ and $B \in \mathsf{TopTsk}(\rho)$, then $(\rho', \ell') = \mathsf{Push}(\mathsf{ClrTop}((\rho, \ell), B), B)$,
- if $\phi \models \mathsf{NTK} \wedge \mathsf{MTK}$, or $\mathsf{Lmd}(A) = \mathsf{SIT}$ and $\phi \models \mathsf{MTK}$, then $i = 2$ and $(\rho', \ell') = \mathsf{NewTsk}((\rho, \ell), B)$,
- if $\phi \models \mathsf{NTK} \wedge \neg\mathsf{MTK}$, or $\mathsf{Lmd}(A) = \mathsf{SIT}$ and $\phi \models \neg\mathsf{MTK}$, then
  - if $\mathsf{GetRealTsk}(\rho, B) = S_j$, then $i = 1$ if $j = 1$, and $i = 2$ otherwise. Moreover,
    - if $\phi \models \neg\mathsf{CTP} \wedge \neg\mathsf{CTK}$, then
      - if $j \neq \ell$, or $\phi \models \mathsf{STP}$ and $Top(S_j) = B$, then $(\rho', \ell') = \mathsf{MvTsk2Top}((\rho, \ell), S_j)$,
      - otherwise, $(\rho', \ell') = \mathsf{Push}(\mathsf{MvTsk2Top}((\rho, \ell), S_j), B)$,
    - if $\phi \models \mathsf{CTP} \wedge \neg\mathsf{CTK}$, then
      - if $B \notin S_j$, then $(\rho', \ell') = \mathsf{Push}(\mathsf{MvTsk2Top}((\rho, \ell), S_j), B)$,
      - otherwise,$(\rho', \ell') = \mathsf{Push}(\mathsf{ClrTop}(\mathsf{MvTsk2Top}((\rho, \ell), S_j), B), B)$,
    - if $\phi \models \mathsf{CTK}$, then $(\rho', \ell') = \mathsf{Push}(\mathsf{ClrTsk}(\mathsf{MvTsk2Top}((\rho, \ell), S_j)), B)$,
  - if $\mathsf{GetRealTsk}(\rho, B) = *$ and $\mathsf{GetTsk}(\rho, B) = S_j$, then $i = 1$ if $j = 1$, and $i = 2$ otherwise. Moreover,
    - if $\phi \models \neg\mathsf{STP} \wedge \neg\mathsf{CTP} \wedge \neg\mathsf{CTK}$, or $\phi \models \mathsf{STP} \wedge \neg\mathsf{CTP} \wedge \neg\mathsf{CTK}$ and $\mathsf{TopAct}(S_j) \neq B$, then $(\rho', \ell') = \mathsf{Push}(\mathsf{MvTsk2Top}((\rho, \ell), S_j), B)$,
    - if $\phi \models \mathsf{STP} \wedge \neg\mathsf{CTP} \wedge \neg\mathsf{CTK}$ and $\mathsf{TopAct}(S_j) = B$, then $(\rho', \ell') = \mathsf{MvTsk2Top}((\rho, \ell), S_j)$,
    - if $\phi \models \mathsf{CTP} \wedge \neg\mathsf{CTK}$, then
      - if $B \notin S_j$, then $(\rho', \ell') = \mathsf{Push}(\mathsf{MvTsk2Top}((\rho, \ell), S_j), B)$,
      - otherwise, $(\rho', \ell') = \mathsf{Push}(\mathsf{ClrTop}(\mathsf{MvTsk2Top}((\rho, \ell), S_j), B), B)$,

  - $*$ if $\phi \models \mathsf{CTK}$, then $(\rho', \ell') = \mathsf{Push}(\mathsf{ClrTsk}(\mathsf{MvTsk2Top}((\rho, \ell), S_j)), B)$,
- if $\mathsf{GetTsk}(\rho, B) = *$, then $i = 2$ and $(\rho', \ell') = \mathsf{NewTsk}((\rho, \ell), B)$.

Case $\mathsf{Lmd}(B) = \mathsf{STP}$

- if $\mathsf{Lmd}(A) \neq \mathsf{SIT}$ and $\phi \models \neg\mathsf{NTK}$, then $i = 1$ and
  - if $B = \mathsf{TopAct}(\rho)$, then $(\rho', \ell') = (\rho, \ell)$,
  - otherwise,
    - if $\phi \models \neg\mathsf{RTF} \wedge \neg\mathsf{CTP}$, or $\phi \models \mathsf{RTF} \vee \mathsf{CTP}$ and $B \notin \mathsf{TopTsk}(\rho)$, then $(\rho', \ell') = \mathsf{Push}((\rho, \ell), B)$,
    - if $\phi \models \mathsf{RTF} \wedge \neg\mathsf{CTP}$ and $B \in \mathsf{TopTsk}(\rho)$, then $(\rho', \ell') = \mathsf{MvAct2Top}((\rho, \ell), B)$,
    - if $\phi \models \mathsf{CTP}$ and $B \in \mathsf{TopTsk}(\rho)$, then $(\rho', \ell') = \mathsf{Push}(\mathsf{ClrTop}((\rho, \ell), B), B)$,
- if $\phi \models \mathsf{NTK} \wedge \mathsf{MTK}$, or $\mathsf{Lmd}(A) = \mathsf{SIT}$ and $\phi \models \mathsf{MTK}$, then $i = 2$ and $(\rho', \ell') = \mathsf{NewTsk}((\rho, \ell), B)$,
- if $\phi \models \mathsf{NTK} \wedge \neg\mathsf{MTK}$, or $\mathsf{Lmd}(A) = \mathsf{SIT}$ and $\phi \models \neg\mathsf{MTK}$, then
  - if $\mathsf{GetRealTsk}(\rho, B) = S_j$, then $i = 1$ if $j = 1$, and $i = 2$ otherwise. Moreover,
    - if $\phi \models \neg\mathsf{CTP} \wedge \neg\mathsf{CTK}$, then
      - if $j \neq \ell$ or $Top(S_j) = B$, then $(\rho', \ell') = \mathsf{MvTsk2Top}((\rho, \ell), S_j)$,
      - otherwise, $(\rho', \ell') = \mathsf{Push}(\mathsf{MvTsk2Top}((\rho, \ell), S_j), B)$,
    - if $\phi \models \mathsf{CTP} \wedge \neg\mathsf{CTK}$, then
      - if $B \notin S_j$, then $(\rho', \ell') = \mathsf{Push}(\mathsf{MvTsk2Top}((\rho, \ell), S_j), B)$,
      - otherwise, $(\rho', \ell') = \mathsf{Push}(\mathsf{ClrTop}(\mathsf{MvTsk2Top}((\rho, \ell), S_j), B), B)$,
    - if $\phi \models \mathsf{CTK}$, then $(\rho', \ell') = \mathsf{Push}(\mathsf{ClrTsk}(\mathsf{MvTsk2Top}((\rho, \ell), S_j)), B)$,
  - if $\mathsf{GetRealTsk}(\rho, B) = *$ and $\mathsf{GetTsk}(\rho, B) = S_j$, then $i = 1$ if $j = 1$, and $i = 2$ otherwise. Moreover,
    - if $\phi \models \neg\mathsf{CTP} \wedge \neg\mathsf{CTK}$ and $\mathsf{TopAct}(S_j) \neq B$, then $(\rho', \ell') = \mathsf{Push}(\mathsf{MvTsk2Top}((\rho, \ell), S_j), B)$,
    - if $\phi \models \neg\mathsf{CTP} \wedge \neg\mathsf{CTK}$ and $\mathsf{TopAct}(S_j) = B$, then $(\rho', \ell') = \mathsf{MvTsk2Top}((\rho, \ell), S_j)$,
    - if $\phi \models \mathsf{CTP} \wedge \neg\mathsf{CTK}$, then $(\rho', \ell') = \mathsf{Push}(\mathsf{ClrTop}(\mathsf{MvTsk2Top}((\rho, \ell), S_j), B), B)$,
    - if $\phi \models \mathsf{CTK}$, then $(\rho', \ell') = \mathsf{Push}(\mathsf{ClrTsk}(\mathsf{MvTsk2Top}((\rho, \ell), S_j)), B)$,
  - if $\mathsf{GetTsk}(\rho, B) = *$, then $i = 2$ and $(\rho', \ell') = \mathsf{NewTsk}((\rho, \ell), B)$.

The case $\phi \models \mathsf{TOH}$

We then consider the transition rules $\tau = A \xrightarrow{\mathsf{start}(\phi)} B$ with $\phi \models \mathsf{TOH}$. It turns out that we can largely reuse the semantic definitions of the case that $\phi \models \neg\mathsf{TOH}$. Namely, let $\tau' = A \xrightarrow{\mathsf{start}(\phi')} B$ where ϕ' is obtained from ϕ by replacing TOH with $\neg\mathsf{TOH}$. (The behavior of τ' is fully prescribed before, viz, $(\rho, \ell) \xrightarrow[\tau', i]{\mathcal{A}} (\rho', \ell')$ where $\rho' = ((S'_1, A'_1), \cdots, (S'_{n'}, A'_{n'}))$. Then we have that

- if $i = 1$, then $(\rho, \ell) \xrightarrow[\tau, i]{\mathcal{A}} (\rho', \ell')$,
- if $i = 2$, then $(\rho, \ell) \xrightarrow[\tau, 0]{\mathcal{A}} (((S'_1, A'_1)), \ell'')$, and $\ell'' = 1$ if $\ell' = 1$; $\ell'' = 0$ otherwise.

The case $\alpha = \mathsf{finishStart}$

Finally, we consider the semantics of the transition rules $\tau = A \to \mathsf{finishStart}(B, \phi)$. Intuitively, $A \to \mathsf{finishStart}(B, \phi)$ specifies that B is started with the intent flags ϕ followed by the termination of A, popped from the task stack. Suppose that (ρ, ℓ) is the current configuration with $\rho = ((S_1, A_1), \cdots, (S_n, A_n))$. Moreover, let $\tau' = A \to \mathsf{start}(B, \phi)$ and $(\rho, \ell) \xrightarrow[\tau', i]{\mathcal{A}} (\rho', \ell')$, with $\rho' = ((S'_1, A'_1), \cdots, (S'_{n'}, A'_{n'}))$.

- If $i = 0$ (this may happen when $\phi \models \mathsf{TOH}$), then $(\rho, \ell) \xrightarrow[\tau, 0]{\mathcal{A}} (\rho', \ell')$.
- If $i = 2$, then $(\rho, \ell) \xrightarrow[\tau, i']{\mathcal{A}} (\rho'', \ell'')$, where $(i', (\rho'', \ell'')) = \mathsf{RmAct}((\rho', \ell'), 2)$.
- If $i = 1$, then
  - if $|S'_1| = |S_1| + 1$ (in this case, the top activity of S'_1 is B, and the top second is A), then let S''_1 obtained from S'_1 by removing the second activity from the top, and ρ'' obtained from ρ' by replacing S'_1 with S''_1, then we have $(\rho, \ell) \xrightarrow[\tau, 1]{\mathcal{A}} (\rho'', \ell')$,
  - if $|S'_1| = |S_1|$, then $(\rho, \ell) \xrightarrow[\tau, i']{\mathcal{A}} (\rho'', \ell'')$, where if $\mathsf{Top}(S'_1) = A$, then $(i', (\rho'', \ell'')) = \mathsf{RmAct}((\rho', \ell'), 1)$, otherwise (in this case, $\phi \models \mathsf{RTF}$, $\mathsf{Top}(S'_1) = B$, and the top second activity of S'_1 is A), $i' = 1$, $\ell'' = \ell'$, and ρ'' is obtained from ρ' by removing from S'_1 the top second activity,
  - if $|S'_1| < |S_1|$, then $(\rho, \ell) \xrightarrow[\tau, 1]{\mathcal{A}} (\rho', \ell')$.

B.1 Android 6.0

As mentioned before, the task allocation mechanism of Android 6.0 is different, which has nothing to do with the real activities of tasks but only uses the affinities of tasks. More precisely, the semantics for Android 6.0 can be adapted from that for Android 8.0 as follows,

- for the case $\mathsf{Lmd}(B) = \mathsf{STK}$, replace the constraint $\mathsf{GetRealTsk}(\rho, B) = S_j$ or $\mathsf{GetRealTsk}(\rho, B) = *$ and $\mathsf{GetTsk}(\rho, B) = S_j$ with $\mathsf{GetTsk}(\rho, B) = S_j$,
- for the case $\mathsf{Lmd}(B) = \mathsf{STD}$, remove the item corresponding to the constraint $\mathsf{GetRealTsk}(\rho, B) = S_j$, moreover, replace the constraint $\mathsf{GetRealTsk}(\rho, B) = *$ and $\mathsf{GetTsk}(\rho, B) = S_j$ with $\mathsf{GetTsk}(\rho, B) = S_j$.

Note that some similar adaptations should be done for the other cases, although they are omitted here. Moreover, the aforementioned strange behaviour of RTF for Android 7.0 does not exist for Android 6.0.

C VALIDATION OF THE FORMAL SEMANTICS

We do experiments to to validate the formal semantics of ASM against the actual behaviour of the multitasking mechanism in Android 6.0, 7.0, and 8.0.

We design a diagnosis app with 17 activities for the experiments. The names, launch modes and task affinities of these activities can be found in Table 8, where "" means the empty-string task affinity.

We observe that in each case of the definition of the semantics, if a flag does not take effect in that case, then these flags are omitted in the constraints corresponding to the case. However, in the validation experiments, we still need to consider all the possible values of these flags in order to be exhaustive. Therefore, the validation experiments are done in the following way: For each case of the semantics, a desired configuration of the diagnosis app is generated. Moreover, for each possible value of the intent flags satisfying the constraint corresponding to the case, a corresponding transition is selected to execute, and the resulting configuration is recorded.

The experimental results can be found by browsing the web page all.htm after unziping the semantics-validation.zip file. ZL: describe the information in the table :LZ . The cases where Android 6.0, 7.0, and 8.0 have different semantics are highlighted by colors.

Table 8: Attributes of activities of the diagnosis app

Name	LaunchMode	TaskAffinity	Name	LaunchMode	TaskAffinity
A	std	0	C	std	0
B1	std	0	B2	std	1
B3	std	""	B4	stp	0
B5	stp	1	B6	stp	""
B7	stk	0	B8	stk	1
B9	stk	""	B10	sit	0
B11	sit	1	B12	sit	""
D1	std	0	D2	std	1
D3	std	""			

D CONFIGURATION REACHABILITY ANALYSIS

In nuXmv, an FSM $\mathcal{M}$ is specified by $(\vec{x}, \vec{s}, I, O, T)$, where $\vec{x}$ (resp. $\vec{s}$) is a tuple of boolean variables to represent the inputs (resp. states), $I(\vec{s})$ (resp. $O(\vec{s})$) is a boolean formula involving the variables $\vec{s}$ to specify the set of initial states (resp. final states), $T(\vec{s}, \vec{x}, \vec{s'})$, called the transition relation, is a boolean formula over $\vec{s}$, $\vec{x}$, and $\vec{s'}$ to describe how inputs lead from one state to possibly many different states. (Here $\vec{s'}$ is the tuple of primed boolean variables corresponding to $\vec{s}$.) For instance, let $\mathcal{M} = (\vec{x}, \vec{s}, I, O, T)$ be the FSM, where $\vec{s} = (s_1)$, $\vec{x} = (x_1)$, $I(\vec{s}) = \neg s_1$, $O(\vec{s}) = s_1$, $T(\vec{s}, \vec{x}, \vec{s'}) = (\neg s_1 \wedge x_1 \wedge s_1') \vee (s_1 \wedge \neg x_1 \wedge \neg s_1')$, then $\mathcal{M}$ accepts the language $(10)^*1$.

Let $\Sigma = \mathsf{Act} \cup \{\bot\}$. Then each activity can be encoded by $\ell = \lceil \log_2 |\Sigma| \rceil$ boolean variables. For each $A \in \mathsf{Act}$, we use $enc(A)$ to denote the binary encoding of A. W.l.o.g., we assume that $\bot$ is encoded by 0^ℓ. As such, each configuration of $\mathcal{A}$ is encoded by $m(\hbar + 1)\ell + m\ell$ boolean variables. We will use $\vec{t} = \vec{t}_1\vec{r}_1 \cdots \vec{t}_m\vec{r}_m$, where each $\vec{t}_i$ (resp. $\vec{r}_i$) with $i \in [m]$ is a tuple of $\hbar\ell$ (resp. ℓ) boolean variables to denote the boolean variables encoding the task set, and $\vec{o} = \vec{o}_1 \cdots \vec{o}_m$, where each $\vec{o}_j$ ($j \in [m]$) is a tuple of ℓ boolean variables to denote the boolean variables encoding the linear order of the real activities of tasks.

To simulate the transition rules of $\mathcal{A}$ in $\mathcal{M}_\mathcal{A}$, we also need to encode them by boolean variables. Each transition rule $\tau = A \xrightarrow{\alpha(\phi)} B$ takes $\ell + 1 + 7 + \ell = 2\ell + 8$ boolean variables, where A, B are encoded by 2ℓ variables, $\alpha \in \{\mathsf{start}, \mathsf{finishStart}\}$ requires one variable, and the seven intent flags from $\mathcal{F}$ take up the rest 7 variables with each flag encoded by one variable. We will use $\vec{x} = \vec{y}\vec{a}\vec{z}$ to denote these boolean variables, where $\vec{y}$ and $\vec{z}$ are tuples of ℓ boolean variables and $\vec{a}$ is a tuple of 8 boolean variables.

The transition relation $T(\vec{t} \cdot \vec{o}, \vec{x}, \vec{t'} \cdot \vec{o'})$ of $\mathcal{M}_\mathcal{A}$ is constructed from $\mathcal{A}$ as: $T(\vec{t} \cdot \vec{o}, \vec{x}, \vec{t'} \cdot \vec{o'}) = \bigvee_{\tau \in \Delta} T_\tau(\vec{t} \cdot \vec{o}, \vec{x}, \vec{t'} \cdot \vec{o'})$, where T_τ is a boolean formula simulating the transition rule τ. We now show how T_τ is formulated for $\tau = A \xrightarrow{\alpha(\phi)} B$.

In general, T_τ is a disjunction of the subformulae constructed for different situations specified in the semantics of ASM. We will use the following instance to illustrate the construction of T_τ: $\mathsf{Lmd}(B) = \mathsf{STD}$, $\phi \models \mathsf{NTK} \wedge \neg\mathsf{MTK}$, $\mathsf{GetRealTsk}(\rho, B) = *$, and $\mathsf{GetTsk}(\rho, B) = S_j$, where ρ is the current configuration.

At first, it is easy to see that the following boolean formulae over $\vec{x}$ can be constructed to describe the launch modes and intent flags,

- for each $L \in \{\mathsf{STD}, \mathsf{STP}, \mathsf{STK}, \mathsf{SIT}\}$, a boolean formula $\mathsf{Lmd}_{1,L}(\vec{x})$ (resp. $\mathsf{Lmd}_{2,L}(\vec{x})$) to specify that $\mathsf{Lmd}(A) = L$ (resp. $\mathsf{Lmd}(B) = L$) is satisfied.
- for each $F \in \mathcal{F}$, a boolean formula $\mathsf{Flg}_F(\vec{x})$ to specify that $\phi \models F$ holds.

In addition, for $B \in \mathsf{Act} \setminus \mathsf{Act}_{\mathsf{SIT}}$, we define a boolean formula $\mathsf{Aft}_B(\vec{z}')$, where $\vec{z}' = (z'_1, \cdots, z'_\ell)$, as follows,

$$\mathsf{Aft}_B(\vec{z}') = \bigvee_{\mathsf{Lmd}(C)\neq\mathsf{SIT}, \mathsf{Aft}(C)=\mathsf{Aft}(B)} \vec{z}' = enc(C).$$

It follows that T_τ can be written as

$$\mathsf{Lmd}_{2,\mathsf{STD}}(\vec{x}) \wedge \mathsf{Flg}_{\mathsf{NTK}}(\vec{x}) \wedge \neg\mathsf{Flg}_{\mathsf{MTK}}(\vec{x}) \wedge \bigwedge_{i\in[m]} \vec{o}_i \neq enc(B) \wedge \bigvee_{j\in[m]} (\mathsf{Aft}_{\mathsf{Aft}(B)}(\vec{o}_j) \wedge \bigwedge_{1\le j'<j} \neg\mathsf{Aft}_{\mathsf{Aft}(B)}(\vec{o}_{j'}) \wedge \mathsf{Upd}_j),$$

where

- the formula $\bigwedge_{i\in[m]} \vec{o}_i \neq enc(B)$ specifies the condition $\mathsf{GetRealTsk}(\rho, B) = *$,
- the formula $\mathsf{Aft}_{\mathsf{Aft}(B)}(\vec{o}_j) \wedge \bigwedge_{1\le j'<j} \neg\mathsf{Aft}_{\mathsf{Aft}(B)}(\vec{o}_{j'})$ specifies the condition $\mathsf{GetTsk}(\rho, B) = S_j$, and
- the formula Upd_j updates the values of the boolean variables according to τ, where

$$\mathsf{Upd}_j ::= \vec{o}'_1 = \vec{o}_j \wedge \bigwedge_{2\le j'\le j} \vec{o}'_{j'} = \vec{o}_{j'-1} \wedge \bigwedge_{j<j'\le m} \vec{o}'_j = \vec{o}_j \wedge \bigvee_{i\in[m]} (\vec{r}_i = \vec{o}_j \wedge \mathsf{tskUpd}_i(\vec{t}, \vec{t}')).$$

The subformula tskUpd_i in Upd_j updates the values of $\vec{t}$ which encodes the task set and is defined as follows, where each disjunct corresponds to different cases specified in the semantics of ASM.

$$\begin{aligned}
\mathsf{tskUpd}_i(\vec{t}, \vec{t}') ::= {} & \Big(\mathsf{Flg}_{\mathsf{STP}}(\vec{x}) \wedge \neg\mathsf{Flg}_{\mathsf{CTP}}(\vec{x}) \wedge \neg\mathsf{Flg}_{\mathsf{CTK}}(\vec{x}) \wedge \bigvee_{1<j'\le\hbar} (\mathsf{topAct}_{\neg B,j'}(\vec{t}_i) \wedge \mathsf{Push}_{B,i,j'}(\vec{t}, \vec{t}')) \Big) \vee \\
& \Big(\mathsf{Flg}_{\mathsf{STP}}(\vec{x}) \wedge \neg\mathsf{Flg}_{\mathsf{CTP}}(\vec{x}) \wedge \neg\mathsf{Flg}_{\mathsf{CTK}}(\vec{x}) \wedge \bigvee_{j'\in[\hbar]} (\mathsf{topAct}_{B,j'}(\vec{t}_i) \wedge \vec{t}' = \vec{t}) \Big) \vee \\
& \Big(\mathsf{Flg}_{\mathsf{CTP}}(\vec{x}) \wedge \neg\mathsf{Flg}_{\mathsf{CTK}}(\vec{x}) \wedge \bigwedge_{j'\in[\hbar]} \vec{t}_{i,j'} \neq enc(B) \wedge \bigvee_{1<j'\le k} \mathsf{Push}_{B,i,j'}(\vec{t}, \vec{t}') \Big) \vee \\
& \Big(\mathsf{Flg}_{\mathsf{CTP}}(\vec{x}) \wedge \neg\mathsf{Flg}_{\mathsf{CTK}}(\vec{x}) \wedge \bigvee_{j'\in[\hbar]} (\vec{t}_{i,j'} = enc(B) \wedge \mathsf{clrTop}_{B,i,j'}(\vec{t}, \vec{t}')) \Big) \vee \\
& \Big(\neg\mathsf{Flg}_{\mathsf{STP}}(\vec{x}) \wedge \neg\mathsf{Flg}_{\mathsf{CTP}}(\vec{x}) \wedge \neg\mathsf{Flg}_{\mathsf{CTK}}(\vec{x}) \wedge \bigvee_{1<j'\le\hbar} \mathsf{Push}_{B,i,j'}(\vec{t}, \vec{t}') \Big) \vee \\
& \Big(\mathsf{Flg}_{\mathsf{CTK}}(\vec{x}) \wedge \mathsf{clrTsk}_{B,i}(\vec{t}, \vec{t}') \Big),
\end{aligned}$$

where

$$\begin{aligned}
\mathsf{Push}_{B,i,j'}(\vec{t}, \vec{t}') ::= {} & \vec{t}_{i,j'} \neq 0^\ell \wedge \bigwedge_{1\le j''<j'} \vec{t}_{i,j''} = 0^\ell \wedge \vec{t}'_{i,j'-1} = enc(B) \wedge \bigwedge_{j''\in[\hbar]\setminus\{j'-1\}} \vec{t}'_{i,j''} = \vec{t}_{i,j''} \wedge \bigwedge_{i'\in[m]\setminus\{i\}} \vec{t}'_{i'} = \vec{t}_{i'} \wedge \bigwedge_{i'\in[m]} \vec{r}'_{i'} = \vec{r}_{i'}, \\
\mathsf{topAct}_{\neg B,j'}(\vec{t}_i) ::= {} & \vec{t}_{i,j'} \neq 0^\ell \wedge \bigwedge_{1\le j''<j'} \vec{t}_{i,j''} = 0^\ell \wedge \vec{t}_{i,j'} \neq enc(B), \\
\mathsf{topAct}_{B,j'}(\vec{t}_i) ::= {} & \vec{t}_{i,j'} = enc(B) \wedge \bigwedge_{1\le j''<j'} \vec{t}_{i,j''} = 0^\ell, \\
\mathsf{clrTop}_{B,i,j'}(\vec{t}, \vec{t}')) ::= {} & \bigwedge_{1\le j''<j'} \vec{t}'_{i,j''} = 0^\ell \wedge \bigwedge_{j'\le j''\le\hbar} \vec{t}'_{i,j''} = \vec{t}_{i,j''} \wedge \bigwedge_{i'\in[m]\setminus\{i\}} \vec{t}'_{i'} = \vec{t}_{i'} \wedge \bigwedge_{i'\in[m]} \vec{r}'_{i'} = \vec{r}_{i'}, \\
\mathsf{clrTsk}_{B,i'}(\vec{t}, \vec{t}')) ::= {} & \vec{t}'_{i',\hbar} = enc(B) \wedge \bigwedge_{1\le j'<\hbar} \vec{t}'_{i',j'} = 0^\ell \wedge \bigwedge_{i'\in[m]\setminus\{i\}} \vec{t}'_{i'} = \vec{t}_{i'} \wedge \bigwedge_{i'\in[m]} \vec{r}'_{i'} = \vec{r}_{i'}.
\end{aligned}$$

Other instances of τ can be defined analogously and are implemented.

E STACK BOUNDEDNESS ANALYSIS

Case $k = 1$. Let $A \in \mathsf{Act}_{\mathsf{real}}$. In this case, we check whether the height of some A-task is unbounded, but involving another task. The main difficulty of this case, in a contrast to $k = 0$, is that, when the system evolves, the A-task may give away as the top task, i.e., the other task may become the top task, but then gives back to the A-task later on. Even worse, the content of the A-task may have be changed during the round of switch over, and there may be several rounds during which the height of the A-task grows. To accommodate such a complex analysis, we will utilise a concept of *virtual transitions* to summarise the changes of the content of the A-task for each round of top task switching.

For an activity $A \in \mathsf{Act}_{\mathsf{real}}$, let $G_A = (V, E)$ be an edge-labeled graph, where the edge labels are of the form $\alpha(\phi)$ with $\alpha \in \{\mathsf{start}, \mathsf{finishStart}\}$ and $\phi \in \mathscr{B}(\mathscr{F})$. Intuitively, G_A is used to capture the set of transitions that can be applied to update the content of an A-task. Note that if $\mathsf{Lmd}(A) = \mathsf{SIT}$, then G is the graph that contains a single node A without edges.

A *task-switching transition from G to B* is a transition $A' \xrightarrow{\alpha'(\phi')} B' \in \Delta$ such that $A' \in V$, moreover, either $\mathsf{Lmd}(B) = \mathsf{SIT}$ and $B' = B$, or $\mathsf{Lmd}(B) \neq \mathsf{SIT}$ and $\mathsf{Aft}(B') = \mathsf{Aft}(B)$.

Suppose additionally $\mathsf{Lmd}(A) \neq \mathsf{SIT}$ holds. For the graph $G_A = (V, E)$, $A' \xrightarrow{\alpha'(\phi')} A''$ is a *virtual transition of G w.r.t. B* if there is a task-switching transition $A' \xrightarrow{\alpha(\phi)} B'$ from G to B and a task-switching transition $B'' \xrightarrow{\alpha'(\phi')} A''$ from $\mathsf{Reach}(\Delta, B')$ to A.

We define the *completion of* $\mathsf{Reach}(\Delta, A)$ *by the virtual transitions w.r.t. B*, denoted by $\mathsf{Comp}_B(\mathsf{Reach}(\Delta, A))$, as the graph computed by the following three-step algorithm.

(1) Let $G_0 := \mathsf{Reach}(\Delta, A)$ and $i := 0$.
(2) Iterate the following procedure until $G_{i+1} = G_i$: Let G_{i+1} be the graph obtained from G_i by adding all the virtual transitions of G_i w.r.t. B. Let $i := i + 1$.
(3) Let $\mathsf{Comp}_B(\mathsf{Reach}(\Delta, A)) := G_i$.

Our algorithm checks for each $A, B \in \mathsf{Act}_{\mathsf{real}}$ such that A, B represent different tasks and $\mathsf{Lmd}(A) \neq \mathsf{SIT}$, whether there exists a witness cycle in $\mathsf{Comp}_B(\mathsf{Reach}(\Delta, A))$.

If affirmative, then the algorithm returns "stack unbounded". Otherwise, if Δ is a directed acyclic graph, then the algorithm returns "stack bounded". Otherwise, the algorithm returns "unknown".

Case $k = 2$. For $A \in \mathsf{Act}_{\mathsf{real}}$ and $\mathbb{A} = \{A_1, \cdots, A_k\} \subseteq \mathsf{Act}_{\mathsf{real}}$, we say that $A, A_1, \cdots, A_k$ represent different tasks if they are mutually distinct, in addition, for each pair of them, say A', A'', one of the following constraints holds: 1) $\mathsf{Lmd}(A') = \mathsf{Lmd}(A'') = \mathsf{SIT}$, 2) $\mathsf{Lmd}(A') = \mathsf{SIT}$ and $\mathsf{Lmd}(A'') \neq \mathsf{SIT}$, 3) $\mathsf{Lmd}(A') \neq \mathsf{SIT}$ and $\mathsf{Lmd}(A'') = \mathsf{SIT}$, 4) $\mathsf{Lmd}(A') \neq \mathsf{SIT}$, $\mathsf{Lmd}(A'') \neq \mathsf{SIT}$, and $\mathsf{Aft}(A') \neq \mathsf{Aft}(A'')$.

Let $A \in \mathsf{Act}_{\mathsf{real}}$ and $\mathbb{A} = \{A_1, \cdots, A_k\} \subseteq \mathsf{Act}_{\mathsf{real}}$ such that $A, A_1, \cdots, A_k$ represent different tasks in the rest of this section.

Let $G = (V, E)$ be an edge-labeled graph, where the edge labels are of the form $\alpha(\phi)$ with $\alpha \in \{\mathsf{start}, \mathsf{finishStart}\}$ and $\phi \in \mathscr{B}(\mathscr{F})$, with the intention of capturing the set of transitions that can be applied to update the content of an A-task. Note that if $\mathsf{Lmd}(A) = \mathsf{SIT}$, then G is the graph that contains a single node A and no edges. A *task-switching transition from G to $\mathbb{A}$* is a transition $A' \xrightarrow{\alpha'(\phi')} B' \in \Delta$ such that $A' \in V$, moreover, either $\mathsf{Lmd}(B') = \mathsf{SIT}$ and $B' \in \mathbb{A}$, or $\mathsf{Lmd}(B') \neq \mathsf{SIT}$ and $\mathsf{Aft}(B') \in \mathsf{Aft}(\mathbb{A} \setminus \mathsf{Act}_{\mathsf{SIT}})$.

For $B' \in \mathsf{Act}$ such that either $\mathsf{Lmd}(B') = \mathsf{SIT}$ and $B' \in \mathbb{A}$, or $\mathsf{Lmd}(B') \neq \mathsf{SIT}$ and $\mathsf{Aft}(B') \in \mathsf{Aft}(\mathbb{A} \setminus \mathsf{Act}_{\mathsf{SIT}})$, we use $\mathsf{Reach}_{\mathbb{A}}(\Delta, B')$ to denote the least set of transitions $\Theta \subseteq \Delta$ satisfying the following constraints:

- $\mathsf{Reach}(\Delta, B') \subseteq \Theta$,
- for each transition $C \xrightarrow{\alpha(\phi)} D \in \Delta$ satisfying the following two conditions, we have $C \xrightarrow{\alpha(\phi)} D \in \Theta$: 1) C is either B' or the target node of some transition in Θ, 2) either $\mathsf{Lmd}(D) = \mathsf{SIT}$ and $D \in \mathbb{A}$, or $\mathsf{Lmd}(D) \neq \mathsf{SIT}$ and $\mathsf{Aft}(D) \in \mathsf{Aft}(\mathbb{A} \setminus \mathsf{Act}_{\mathsf{SIT}})$.

Intuitively, $\mathsf{Reach}_{\mathbb{A}}(\Delta, B')$ is the set of transitions that are reachable from B' that can be used to update the contents of the tasks whose real activities are from $\mathbb{A}$.

Suppose additionally $\mathsf{Lmd}(A) \neq \mathsf{SIT}$ holds. For a graph $G = (V, E)$ capturing the set of transitions that can be applied to update the content of an A-task, a *virtual transition of G w.r.t. $\mathbb{A}$* is some $A' \xrightarrow{\alpha'(\phi')} A''$ such that there is a task-switching transition $A' \xrightarrow{\alpha(\phi)} B'$ from G to $\mathbb{A}$ and a task-switching transition $B'' \xrightarrow{\alpha'(\phi')} A''$ from $\mathsf{Reach}_{\mathbb{A}}(\Delta, B')$ to A. We define the *completion of* $\mathsf{Reach}(\Delta, A)$ *by the virtual transitions w.r.t. $\mathbb{A}$*, denoted by $\mathsf{Comp}_{\mathbb{A}}(\mathsf{Reach}(\Delta, A))$, as the graph computed by the following three-step algorithm.

(1) Let $G_0 := \mathsf{Reach}(\Delta, A)$ and $i := 0$.
(2) Iterate the following procedure until $G_{i+1} = G_i$: Let G_{i+1} be the graph obtained from G_i by adding all the virtual transitions of G_i w.r.t. $\mathbb{A}$. Let $i := i + 1$.
(3) Let $\mathsf{Comp}_{\mathbb{A}}(\mathsf{Reach}(\Delta, A)) := G_i$.

We are ready to present the procedure to solve the stack boundedness problem for the case $k \geq 2$. The procedure checks for each $A \in \mathsf{Act}_{\mathsf{real}}$ and $\mathbb{A} = \{A_1, \cdots, A_k\} \subseteq \mathsf{Act}_{\mathsf{real}}$ such that $A, A_1, \cdots, A_k$ represent different tasks and $\mathsf{Lmd}(A) \neq \mathsf{SIT}$, whether there exists an su-witnessing cycle in the graph $\mathsf{Comp}_{\mathbb{A}}(\mathsf{Reach}(\Delta, A))$. If the answer is yes for any such a pair A and $\mathbb{A}$, then the procedure reports "stack unbounded". Otherwise, if Δ is a directed acyclic graph, then the procedure reports "stack bounded". Otherwise, the procedure reports "unknown".