A Type System for Android Applications

Mohamed A. El-Zawawy[1,2]

[1] College of Computer and Information Sciences,
Al Imam Mohammad Ibn Saud Islamic University (IMSIU),
Riyadh, Kingdom of Saudi Arabia
[2] Department of Mathematics, Faculty of Science, Cairo University,
Giza 12613, Egypt
maelzawawy@cu.edu.eg

Abstract. The most common form of computers today is the hand held form and specially smart phones and tablets. The platform for the majority of smart phones is the Android operating system. Therefore analyzing and verifying the Android applications become an issue of extreme importance. However classical techniques of static analysis are not applicable directly to Android applications. This is so because typically Android applications are developed using Java code that is translated into Java bytecode and executed using Dalvik virtual machine. Also developing new techniques for static analysis of Android applications is an involved problem due to many facts such as the concept of gestures used in Android interaction with users.

One of the main tools of analyzing and verifying programs is types systems [11]. This paper presents a new type system, Android$\mathcal{T}$, which is specific to Android applications. The proposed type system ensures the absence of errors like "method is undefined". Also the type system is applicable to check the soundness of static alanyses of Android applications.
Keywords: Android applications; Type systems; Android Activities; Android Applications analysis; Typed Programming Languages; *Android*$\mathcal{T}$; *Android*$\mathcal{P}$.

1 Introduction

The growing highest percentage of today's number of computers comes from hand held devices; tablets and smart phones. The most common operating systems for these computers is Android. For hand held devices, several properties of Android OS have led it to become the main platform [26]. The number of produced hand held computers in 2015 exceeded billion devices. This number is much more than the number of produced iOS-MacOS-Windows machines [2, 4, 33]. The nature of computing has greatly been affected, developed, and changed due to the great dominance of tablets and smart phones. The widespread also of these devices made their security, accuracy, reliability important issues for both developers and users. Software engineering is the branch of computer science that is most responsible for taking care of these issues by producing a variety of tools for verifying, analyzing, optimizing, and correcting Android applications.

$$
\begin{aligned}
& n \in N. \\
& l \in \mathcal{L} = \textit{The set of all memory locations}. \\
& v \in \mathcal{V} = \mathcal{L} \cup \textit{Integers}. \\
& \delta \in \Delta = \textit{The set of all labels of program points}. \\
& f \in \mathcal{F} = \textit{The set of all names of class fields}. \\
& m \in \textit{Me} = \textit{The set of all method names}. \\
& c \in \mathcal{C} = \textit{The set of all class names}. \\
& \textit{app} \in \mathcal{A} = \textit{The set of all application names}. \\
t \in \textit{types} ::= & \{\textit{int}, \textit{Boolean}, \textit{ref}\, t, \textit{double}\} \cup \mathcal{C} \cup \textit{array}\, t. \\
\textit{in} \in \textit{Instructions} ::= & \textit{nop} \mid \textit{const}\, i, v \mid \textit{move}\, i, j \mid \textit{goto}\, \delta \mid \textit{new-array}\, i, j, t \mid \\
& \textit{array-length}\, i, j \mid \textit{aget}\, i, j, k \mid \textit{aput}\, i, j, k \mid \textit{new-instance}\, i, c \mid \\
& \textit{iget}\, i, j, f \mid \textit{iput}\, i, j, f \mid \textit{invoke-direct}\, i_1, \ldots, i_n, \textit{method}\, m \mid \\
& \textit{neg-int}\, i, j \mid \textit{add}\, i, j, k \mid \textit{if-eq}\, i, j, k \mid \textit{if-eqz}\, i, j \mid \textit{move-result}\, i \mid \\
& \textit{return-void} \mid \textit{return}\, i. \\
\textit{mc} \in \textit{Macros} ::= & \textit{SetContentView xml} \mid \textit{findViewById}\, i, j \mid \\
& \textit{startActivityForResut}\, A \mid \textit{setResult}\, i \mid \textit{finish}. \\
\textit{com} \in \textit{Commands} ::= & \textit{in} \mid \textit{mc} \mid \textit{skip} \mid \textit{Label}\, \delta \mid \textit{com}_1; \textit{com}_2. \\
M \in \textit{Methods} ::= & (\textit{method}\, m, c, t, t^*, \textit{com}). \\
F \in \textit{Fields} ::= & (\textit{field}\, f, c, t). \\
C \in \textit{Classes} ::= & \textit{class}\, c \lhd \textit{AppCompatActivity} \\
& \{\textit{app}; (\textit{field}, \textit{root}, c, \textit{int}); (\textit{field}, \textit{result}, c, \textit{int}); \\
& (\textit{field}, \textit{finished}, c, \textit{Boolean}); F^*; M^*\}. \\
\textit{App} \in \textit{Applications} ::= & (\textit{Application app}, C^*).
\end{aligned}
$$

Fig. 1. Android$\mathcal{P}$: A Model for Android Applications.

More precisely, techniques and strategies of static analyses are the main tools to approach, take care, and treat these issues. However existing techniques of static and dynamic analyses do not fit well for studying Android applications [32, 29, 20, 22]. This is so as Android applications are developed mainly using Java which gets translated into Dalvik bytecode [33, 26].

There are many challengers concerning the analysis of Android applications. The capabilities of hardware are increasing (memory capacity and processor speed). This results in better ability to produce complex and advanced Android applications. This of course results in more difficulty in studying these applications. The concepts of event-driven and framework-based of Android platform contributes to the difficulty of studding Android applications. In Android applications, gestures (such as swipe and click) are main tools of the user to use the

applications [15, 5]. The Android applications are responsible for capturing user gestures and act accordingly. This includes recognizing accurate parameters like screen pixel and event type. The interaction between users and applications requires executing macros methods which are part of the Android framework. Taking care of these issues and details in a precise way makes it difficult to produce an efficient and accurate static analysis [12] for Android applications [33, 26].

In this paper, we present *Android*$\mathcal{T}$, a simple type system [30, 7, 18] for Android applications (considering activity composition). Our proposed type system *Android*$\mathcal{T}$ ensures that the Android applications that are typeable in the type systems do not cause method is undefined errors. The proposed type system treats macros that are executed during interaction with users via gestures. The problem of developing such type systems is involved due to the fact that macro instructions related to the life cycle of Android activities have to be considered by the type system. Also the fact that the composition of many activities in different states of execution is allowed in Android applications complicates the development of a type system for Android applications. A key idea in developing the type system is to let the typing environment composed of two parts; one part reports types of registers and memory locations and the other part links current activities with memory locations.

There are many applications to type type system developed in this paper. Besides that the type system guarantees the absence of specific errors, the type system can be used as a verification tool for Android applications. The type system can be used to ensure the convenience of any modifications done to Android applications by techniques of program analysis [25, 9, 31, 24]. Although the proposed type system is simple, it is powerful enough as it is designed on a rich model of Android programming, *Android*$\mathcal{P}$, that include main Dalvik bytecode operations. For example the array-operations are included in our programming model and considered in our type system. Another advantage of the proposed type system is that the used set of types is rich and specific to Android applications. For example the type "reference to view" ref_v is defined and used in the type system to capture types of registers hosting addresses of views composing Android activities.

Motivation The work in this paper is motivated by the need to a type system that

- is specific to Android applications and macro instructions related to the life cycle of Android activities,
- ensures the absence of errors like "method is undefined", and
- is applicable to check the soundness of static alanyses of Android applications.

Paper Outline

The rest of the paper is organized as follows. Section 2 presents the langauge model, *Android*$\mathcal{P}$, used to develop the type system. The detailed type system,

Android$\mathcal{T}$, is presented in section 3. Section 4 presents a review to the most related up-to-date work. The paper is concluded in Section 5.

2 Langauge Model

Figure 1 presents the syntax of our programming model, *Android*$\mathcal{P}$, for Android applications. The model has a rich set of types including integers, Boolean values, double numbers, classes, and arrays. The model uses a set of 19 Dalvik instructions. These inductions were selected carefully out of the full set which includes more than 200 instructions. The other main component in *Android*$\mathcal{P}$ is a set of macros instructions [14, 1, 27].

The selection of Dalvik instructions and macros instructions in *Android*$\mathcal{P}$ was made to cover main functionalities of Dalivk bytecode in a simple way. The macros included in the syntax are those used in almost all Android applications and that controls the application execution. Therefore the instructions are expected to be the focus of most analysis techniques of Android applications. This will help our semantics to serve these techniques by proving a mathematical background to the analysis techniques.

The set of commands in *Android*$\mathcal{P}$ is a sequence of instructions and macros. Every method has the form (*method* m, c, t, t^*, com) where

- m is the method name,
- c is the class owns the method,
- t is the return type of the method,
- t^* is the sequence of type for method arguments, and
- *textitcom* is the body of the method.

Each class filed in *Android*$\mathcal{P}$ has the syntax (*field* f, c, t) where c denotes the class having the field, f denotes the field name, and t denotes type of the field. Android classes in *Android*$\mathcal{P}$ are extensions of *AppCompatActivity* and hence have the following components:

- c denotes the class name,
- *app* denotes the name of the application having the class,
- A special field *root* of type integer to record the address of view file of the activity having the class,
- A special field *result* of type integer to keep the result of the executing an object (activity) of the class,
- A special field *finished* of type Boolean to inform if an object (activity) of the class is done,
- F^* denotes a sequence of more fields, and
- M^* denotes a sequence of method structures.

Finally an application in *Android*$\mathcal{P}$ has the form (*Application app*, C^*) which includes a name *app* and a sequence of classes C^*.

Our model can be realized as a combination model of the models used in [14, 1, 27] to overcome disadvantages in the related models. The idea of adapting

$$\frac{n \in integers}{\Pi, \Gamma : n \vdash int} \qquad \frac{l \in \mathcal{L} \wedge l \text{ points to value of type } \tau}{\Pi, \Gamma : l \vdash ref\ \tau} \tag{1}$$

$$\frac{\{i \mapsto \tau\} \subseteq \Gamma}{\Pi, \Gamma : i \vdash \tau} \qquad \frac{\{l \mapsto \tau\} \subseteq \Gamma}{\Pi, \Gamma : l \vdash \tau} \tag{2}$$

$$\frac{}{\Pi, \Gamma : nop \vdash OK} \tag{3}$$

$$\frac{\Pi, \Gamma : v \vdash \tau}{\Pi, \Gamma : const\ i, v \vdash \tau} \tag{4}$$

$$\frac{\Pi, \Gamma : j \vdash \tau}{\Pi, \Gamma : move\ i, j \vdash \tau} \tag{5}$$

$$\frac{}{\Pi, \Gamma : goto\ \delta \vdash OK} \tag{6}$$

$$\frac{\Pi, \Gamma : i \vdash ref\ array\ \tau \qquad \Pi, \Gamma : j \vdash int}{\Pi, \Gamma : new\text{-}array\ i, j, \tau \vdash ref\ array\ \tau} \tag{7}$$

$$\frac{\Pi, \Gamma : i \vdash int \qquad \Pi, \Gamma : j \vdash ref\ array\ t}{\Pi, \Gamma : array\text{-}length\ i, j \vdash int} \tag{8}$$

$$\frac{\Pi, \Gamma : i \vdash \tau \qquad \Pi, \Gamma : j \vdash ref\ array\ \tau \qquad \Pi, \Gamma : k \vdash int}{\Pi, \Gamma : aget\ i, j, k \vdash \tau} \tag{9}$$

$$\frac{\Pi, \Gamma : i \vdash \tau \qquad \Pi, \Gamma : j \vdash ref\ array\ \tau \qquad \Pi, \Gamma : k \vdash int}{\Pi, \Gamma : aput\ i, j, k \vdash \tau} \tag{10}$$

$$\frac{\begin{array}{l} class\text{-}size(c) = n \qquad \Pi, \Gamma : i \vdash ref\ c \\ \{\{l, l+1, \ldots, l+n-1\} \mapsto fields(c)\} \subseteq \Gamma \\ (c, l) \in \Pi \end{array}}{\Pi, \Gamma : new\text{-}instance\ i, c \vdash ref\ c} \tag{11}$$

$$\frac{\Pi, \Gamma : i \vdash \tau \qquad \Pi, \Gamma : j \vdash ref\ c \qquad type(c, f) = \tau}{\Pi, \Gamma : iget\ i, j, f \vdash \tau} \tag{12}$$

$$\frac{\Pi, \Gamma : i \vdash \tau \qquad \Pi, \Gamma : j \vdash ref\ c \qquad type(c, f) = \tau}{\Pi, \Gamma : iput\ i, j, f \vdash \tau} \tag{13}$$

Fig. 2. Inference Rules of the Type system Android$\mathcal{T}$ (Set 1).

the set of macro instructions and the three special fields of classes was followed from [27]. We believe the langauge model in [27] is very simple while that in [14, 1] is very complex and our model is moderate; in between the two models. We use the same model *Android*$\mathcal{P}$ to develop an operational semantics for Android application in [10].

3 Type systems

This section presents a new type system, *Android*$\mathcal{T}$, for *Android*$\mathcal{P}$. The type system *Android*$\mathcal{T}$ is meant to guarantee the soundness of types in Android applications; guaranteeing the absence of dynamic type-errors such as method-not-found and field-not-found. For *Android*$\mathcal{P}$, our proposed type system also supports typing macro instruction related to activity state methods like onCreate and onDestroy.

Definition 1 introduces the typing judgments and their components of *Android*$\mathcal{T}$.

Definition 1. – *The set of judgment types,* $\mathcal{T}$*, used in the typing judgments are the set of types defined in the langauge syntax Android*$\mathcal{P}$ *plus view-reference type (*ref_v*), the file-reference type (*ref_f*), the void type, and the method types:*

$$\tau \in \mathcal{T} = types \cup \{ref_v, ref_f, void, (\tau_1, \ldots, \tau_n) \rightarrow \tau\}.$$

- *A type environment,* Γ*, is a partial map from* $\mathcal{L} \cup \mathcal{R}$ *to types.*
- *A class environment,* Π*, is a stack of pairs of activity classes and memory locations.*
- *A type judgement for* $e \in \{commands, Methods, Fields, Classes\}$*, has one of the following forms:*

$$\Pi, \Gamma : e \vdash \tau \qquad\qquad \Pi, \Gamma : e \vdash OK.$$

Definition 2 introduces the sub-typing relation defined on the set $\mathcal{T}$.

Definition 2. *We let* $\Subset$ *denotes the least reflexive transitive closure of* $\sqsubseteq$ *on the set types where:*

1. $Boolean \sqsubseteq int \sqsubseteq double$.
2. $\forall \tau \in \mathcal{T}.\ \tau \sqsubseteq \tau$.
3. $\forall \tau, \tau' \in \mathcal{T}.\ \tau \sqsubseteq \tau' \Rightarrow arrary\ t \sqsubseteq arrary\ t'$.
4. $\forall \tau, \tau' \in types.\ \tau \sqsubseteq \tau' \Rightarrow ref\ \tau \sqsubseteq ref\ \tau'$.
5. $\forall \{\tau_1, \ldots, \tau_n, \tau, \tau'_1, \ldots, \tau'_n, \tau'\} \subseteq types.\ (\forall i.\ \tau_i \sqsubseteq \tau'_i) \wedge \tau \sqsubseteq \tau' \Rightarrow (\tau_1, \ldots, \tau_n \rightarrow \tau) \sqsubseteq (\tau'_1, \ldots, \tau'_n \rightarrow \tau')$.
6. $\forall c \in \mathcal{C}.\ c \sqsubseteq AppCompatActivity$.

Definition 2 guarantees the sub-typing relationship is defined conveniently over arrays types, references types, and methods types. The definition also makes it explicit that all activities (as defined classes) in Android applications are extensions for the class *AppCompatActivity*. The proof of Lemma 1 is straightforward. However it is important for the sub-typing relationship to be a partial order.

$$\frac{\Pi, \Gamma : com \ \vdash \tau}{\Pi, \Gamma : method\ m, c, t, t_1, \ldots, t_n, com \vdash \tau} \quad (14)$$

$$\frac{\Pi, \{i_1 \mapsto t_1, \ldots i_n \mapsto, t_n\} : (method\ m, c, t, t_1, \ldots, t_n, com) \vdash \tau \quad \forall j.\ \Pi, \Gamma : i_j \vdash \tau_i}{\Pi, \Gamma : invoke\text{-}direct\ i_1, \ldots, i_n, method\ m \vdash \tau} \quad (15)$$

$$\frac{\Pi, \Gamma : i \vdash int \qquad \Pi, \Gamma : j \vdash int}{\Pi, \Gamma : neg\text{-}int\ i, j \vdash int} \quad (16)$$

$$\frac{\Pi, \Gamma : i \vdash double \qquad \Pi, \Gamma : j \vdash double}{\Pi, \Gamma : neg\text{-}int\ i, j \vdash double} \quad (17)$$

$$\frac{\Pi, \Gamma : i \vdash int \qquad \Pi, \Gamma : j \vdash int \qquad \Pi, \Gamma : k \vdash int}{\Pi, \Gamma : add\ i, j, k \vdash int} \quad (18)$$

$$\frac{\Pi, \Gamma : i \vdash double \qquad \Pi, \Gamma : j \vdash int \qquad \Pi, \Gamma : k \vdash double \quad Or\ \Pi, \Gamma : i \vdash int \qquad \Pi, \Gamma : j \vdash double \qquad \Pi, \Gamma : k \vdash double}{\Pi, \Gamma : add\ i, j, k \vdash double} \quad (19)$$

$$\frac{\Pi, \Gamma : i \vdash \tau_1 \qquad \Pi, \Gamma : j \vdash \tau_2}{\Pi, \Gamma : if\text{-}eq\ i, j, k \vdash OK} \quad (20)$$

$$\frac{\Pi, \Gamma : i \vdash int}{\Pi, \Gamma : if\text{-}eqz\ i, j \vdash OK} \quad (21)$$

$$\frac{\Pi, \Gamma : i \vdash \tau \qquad \Pi, \Gamma : MVal \vdash \tau}{\Pi, \Gamma : move\text{-}result\ i \vdash \tau} \quad (22)$$

$$\frac{}{\Pi, \Gamma : return\text{-}void \vdash void} \quad (23)$$

$$\frac{\Pi, \Gamma : i \vdash \tau \qquad \Pi, \Gamma : MVal \vdash \tau}{\Pi, \Gamma : return\ i \vdash \tau} \quad (24)$$

Fig. 3. Inference Rules of the Type system Android$\mathcal{T}$ (Set 2).

Lemma 1. *The binary relation $\Subset$ is a partial order.*

Figure 2 presents the first set of inference rules of the proposed type system *Android*$\mathcal{T}$, for *Android*$\mathcal{P}$. This figure presents the typing rules for Dalvik bytecdoe inductions. Rule 7 presents the inference rule for the instruction *new-array* i, j, τ that establishes a new array of type τ and of length j and returns the address of the array to the register i. Therefore the rule requires i to be of type *ref* τ, and j to be of type *int*. In this case, the rule infers that the instruction is to type *ref* τ.

Example 1 *Rule 25 provides an example for applying Rule 7.*

$$\frac{\begin{array}{l}\{(A,l_A)::(B,l_B)::(C,l_C)\},\{i\mapsto \mathit{ref\ int}, j\mapsto \mathit{int}, l\mapsto \mathit{ref\ c}, l_C\mapsto \mathit{double}, l_C+1\mapsto \mathit{double}\}\\ : i \vdash \mathit{ref\ int}\\ \{(A,l_A)::(B,l_B)::(C,l_C)\},\{i\mapsto \mathit{ref\ int}, j\mapsto \mathit{int}, l\mapsto \mathit{ref\ c}, l_C\mapsto \mathit{double}, l_C+1\mapsto \mathit{double}\}\\ : j \vdash \mathit{int}\end{array}}{\begin{array}{l}\{(A,l_A)::(B,l_B)::(C,l_C)\},\{i\mapsto \mathit{ref\ int}, j\mapsto \mathit{int}, l\mapsto \mathit{ref\ c}, l_C\mapsto \mathit{double}, l_C+1\mapsto \mathit{double}\}:\\ \mathit{new\text{-}array}\ i, j, \mathit{int} \vdash \mathit{ref\ int}\end{array}} \quad (25)$$

Rule 11 formulates the typing rule for the instruction *new-instance* i, c which establishes a new object of the class c and returns the address of the object to the register i. The rule first calls the function *class-size* that calculates the size, n, of the class in memory unites (bytes). The rule then assumes that the address of the object is l from the environment $(c, l) \in \Pi$. The rules also requires that according to the environment Γ the memory locations to have the same types as the class fields; $\{\{l, l+1, \ldots, l+n-1\} \mapsto \mathit{fields(c)}\} \subseteq \Gamma$.

Example 2 *Inference rule 26 presents an example of the application of Rule 11. We assume that the class c is of size* 2*. The fields of the calls are of type double.*

$$\frac{\begin{array}{l}\mathit{class\text{-}size}(c) = 2\\ \{(A,l_A)::(B,l_B)::(C,l_C)\},\{i\mapsto \mathit{ref\ int}, j\mapsto \mathit{int}, l\mapsto \mathit{ref\ c}, l_C\mapsto \mathit{double}, l_C+1\mapsto \mathit{double}\}\\ : k \vdash \mathit{ref\ c}\\ \{\{l_C\mapsto \mathit{double}, l_C+1\mapsto \mathit{double}\}\subseteq \Gamma\\ (C,l_C)\in\{(A,l_A)::(B,l_B)::(C,l_C)\}\end{array}}{\begin{array}{l}\{(A,l_A)::(B,l_B)::(C,l_C)\},\{i\mapsto \mathit{ref\ int}, j\mapsto \mathit{int}, l\mapsto \mathit{ref\ c}, l_C\mapsto \mathit{double}, l_C+1\mapsto \mathit{double}\}:\\ \mathit{new\text{-}instance}\ k, c \vdash \mathit{ref\ c}\end{array}} \quad (26)$$

Figure 3 presents the second set of inference rules of the proposed type system $\mathit{Android}\mathcal{T}$, for $\mathit{Android}\mathcal{P}$. Rule 20 presents the typing rule for the instruction *if-eq* i, j, k which checks if the registers i and j are equal. If this is the case, then the control goes to location k. The rule requires that i and j are just typeable. If so then the instruction is *OK*.

Example 3 *Inference rule 34 presents an example of the application of Rule 20.*

$$\frac{\begin{array}{l}\{(A,l_A)::(B,l_B)::(C,l_C)\},\{i\mapsto \mathit{ref\ int}, j\mapsto \mathit{int}, l\mapsto \mathit{ref\ c}, l_C\mapsto \mathit{double}, l_C+1\mapsto \mathit{double}\}\\ : i \vdash \mathit{ref\ int}\\ \{(A,l_A)::(B,l_B)::(C,l_C)\},\{i\mapsto \mathit{ref\ int}, j\mapsto \mathit{int}, l\mapsto \mathit{ref\ c}, l_C\mapsto \mathit{double}, l_C+1\mapsto \mathit{double}\}\\ : j \vdash \mathit{int}\end{array}}{\begin{array}{l}\{(A,l_A)::(B,l_B)::(C,l_C)\},\{i\mapsto \mathit{ref\ int}, j\mapsto \mathit{int}, l\mapsto \mathit{ref\ c}, l_C\mapsto \mathit{double}, l_C+1\mapsto \mathit{double}\}:\\ \mathit{if\text{-}eq}\ i, j, k \vdash \mathit{OK}\end{array}} \quad (34)$$

Figure 4 presents the third set of inference rules of the proposed type system $\mathit{Android}\mathcal{T}$, for $\mathit{Android}\mathcal{P}$. Rule 30 introduces the typing rule for the macro instruction *setResulti* that sets the content of the register i into the field *Result*

$$\frac{\Pi, \Gamma : l \vdash ref\ c \qquad filed\text{-}order(c, root) = m \qquad \Pi, \Gamma : l + m \vdash ref_f \qquad \Pi = (c, l) :: \Pi'}{(c, l) :: \Pi, \Gamma : SetContentView\ xml \vdash ref_f} \tag{27}$$

$$\frac{\Pi, \Gamma : l \vdash ref\ c \qquad filed\text{-}order(c, root) = m \qquad \Pi, \Gamma : l + m \vdash ref_f \qquad \Pi = (c, l) :: \Pi' \qquad \Pi, \Gamma : i \vdash ref_v \qquad \Pi, \Gamma : j \vdash int}{\Pi, \Gamma : findViewById\ i, j \vdash ref_v} \tag{28}$$

$$\frac{\Pi, \Gamma : new\text{-}instance\ i, A \vdash ref\ A}{\Pi, \Gamma : startActivityForResut\ A \vdash ref\ A} \tag{29}$$

$$\frac{\Pi, \Gamma : l \vdash ref\ c \qquad filed\text{-}order(c, result) = m \qquad \Pi, \Gamma : l + m \vdash \tau \qquad \Pi, \Gamma : i \vdash \tau \qquad \Pi = (c, l) :: \Pi'}{\Pi, \Gamma : setResult\ i \vdash \tau} \tag{30}$$

$$\frac{class\text{-}size(c) = n \qquad \Gamma' = \Gamma \setminus \{l, l+1, \ldots l+n\} \qquad \Pi, \Gamma : l \vdash ref\ c \qquad filed\text{-}order(c, finish) = m \qquad \Pi, \Gamma : l + m \vdash Boolean \qquad \Pi = (c, l) :: \Pi'}{\Pi', \Gamma' : finish \vdash OK} \tag{31}$$

$$\frac{\begin{array}{l}\Pi, \Gamma : com_1 \vdash \tau_1 \qquad \Pi, \Gamma : com_2 \vdash \tau_2 \\ Or \qquad \Pi, \Gamma : com_1 \vdash OK \qquad \Pi, \Gamma : com_2 \vdash \tau_2 \\ \Pi, \Gamma : com_1 \vdash \tau_1 \qquad \Pi, \Gamma : com_2 \vdash OK \\ \Pi, \Gamma : com_1 \vdash OK \qquad \Pi, \Gamma : com_2 \vdash OK\end{array}}{\Pi, \Gamma : com_1; com_2 \vdash OK} \tag{32}$$

$$\frac{}{\Pi, \Gamma : Label\ \delta \vdash OK} \tag{33}$$

Fig. 4. Inference Rules of the Type system Android$\mathcal{T}$ (Set 3).

of the currently active object known from the first element of the environment $\Pi; \Pi = (c, l) :: \Pi'$. The rule calls the method *field-order* to determine the order of the field, m, *Result* in the class c; $filed\text{-}order(c, result) = m$. The rule also requires the register i and the location $l + m$ to have equal type τ. If this is the case, then the rule concludes that the type of the instruction is τ.

Figure 5 presents the fourth set of inference rules of the proposed type system *Android*$\mathcal{T}$, for *Android*$\mathcal{P}$. Rule 37 says that assigning a type to a command in an environment, $\Pi', \Gamma' : com \vdash \tau$, guarantees assigning the same type to the command in a bigger environment.

$$\frac{\Pi, \Gamma : com \ \vdash \tau}{\Pi, \Gamma : com \ \vdash OK} \tag{35}$$

$$\frac{\Pi, \Gamma : com \ \vdash \tau' \qquad \tau' \Subset \tau}{\Pi, \Gamma : com \ \vdash \tau} \tag{36}$$

$$\frac{\Pi' \subseteq \Pi \qquad \Gamma' \subseteq \Gamma \qquad \Pi', \Gamma' : com \ \vdash \tau}{\Pi, \Gamma : com \ \vdash \tau} \tag{37}$$

$$\frac{c \in \mathcal{C}}{\Pi, \Gamma : c \vdash AppCompatActivity} \tag{38}$$

$$\frac{c \in \mathcal{C} \qquad \Pi, \Gamma : i \vdash ref\ c}{\Pi, \Gamma : i \vdash ref\ AppCompatActivity} \tag{39}$$

$$\frac{c \in \mathcal{C} \qquad \Pi, \Gamma : l \vdash ref\ c}{\Pi, \Gamma : l \vdash ref\ AppCompatActivity} \tag{40}$$

Fig. 5. Inference Rules of the Type system Android$\mathcal{T}$ (Set 4).

Example 4 *Inference rule 41 presents an example of the application of Rule 32 on results of Example 1 and Example 3.*

$$\frac{\begin{array}{l}\{(A, l_A) :: (B, l_B) :: (C, l_C)\}, \{i \mapsto ref\ int, j \mapsto int, l \mapsto ref\ c, l_C \mapsto double, l_C + 1 \mapsto double\} : \\ new\text{-}array\ i, j, int \vdash ref\ int \\ \{(A, l_A) :: (B, l_B) :: (C, l_C)\}, \{i \mapsto ref\ int, j \mapsto int, l \mapsto ref\ c, l_C \mapsto double, l_C + 1 \mapsto double\} : \\ if\text{-}eq\ i, j, k \vdash OK\end{array}}{\begin{array}{l}\{(A, l_A) :: (B, l_B) :: (C, l_C)\}, \{i \mapsto ref\ int, j \mapsto int, l \mapsto ref\ c, l_C \mapsto double, l_C + 1 \mapsto double\} : \\ new\text{-}array\ i, j, int; if\text{-}eq\ i, j, k \vdash OK\end{array}} \tag{41}$$

It is not hard (using structure induction) to prove the following fact about the type system *Android*$\mathcal{T}$.

Theorem 1. *Every well-structured command of Android*$\mathcal{P}$ *is typeable in Android*$\mathcal{T}$.

4 Literature Review

The focus of a big program-analysis audience has been attracted by the common use of Android mobile devices. Methods such as dynamic monitoring, information-flow analysis, modifying the Android system have been applied to reveal, study, fix essential problems and subtle structure flaws of Android applications [6].

Type systems have been used as one of the main tools to study programs in general and Android programs in particular. There are a variety of analyses and studies that can be carried out for Android applications using type systems [26,

13, 23]. Security of Android programs is a famous example of an issue that can be analyzed using type systems [19, 6, 28, 17, 16].

In [19] for Android applications, a security type system was presented to achieve the analysis of static data-flow. This analysis has the direct application of revealing privacy leaks. The idea of the analysis is to check the code of Android applications against privacy protocol using the type system [19].

In [6], concepts of type systems were utilized to analyze Android applications. The Android API for inter-component communication was reasoned for using a theoretical calculus. This paper also presented a type system to stop privilege. The idea in this paper is that well-typed components of applications are guaranteed to be attacks-protected. This work was implemented to produce a type checker, Lintent [6].

A technique, Dare, for covering Android applications into Java classes was presented in [26]. The conversion included the introduction of a new intermediate form for Applications. The concept of typing inference was also utilized in this paper in the form of solving strong constraints. A few number of inference rules was used to convert the full set of DVM opcodes (more than 200) [3]. This paper provides a way to treat unverifiable Dalvik bytecode [26].

In [28], it is claimed that application-centric rather than resource-centric models of permissions for resources (like GPS, camera, and Internet connection) in Android applications would better serve both users and developers. For example a permission for the use of camera has to be application justified (for paper scanning for example) and another permission for the Internet access can be granted but for downloading from a specific server for example. It was also claimed in [28] that Android is already equipped with required mechanisms to adapt a precise and efficient structures of policies that are application-centric .

Proved sound against the concept of non-interference, a type system for DEX bytecode, an operational semantics for Dalvik VM were introduced in [13]. The aim of this work was to verify characteristics of non-interference for DEX bytecode. Moreover this paper introduced an abstract conversion (that is non-interference preserving) of Java bytecode into DEX bytecode.

In [17] it is claimed that although the already paid high cost in applying the process of application approval, some mobile application stores let malware sneaks to our mobile devices. It is obvious that application stores must adapt strong mechanisms guarantee the genuineness (not malicious) of their applications. In [17], a verification technique for achieving this task is presented.

Targeting Android applications, a type system for taint analysis was introduced in [16]. The type system, DFlow, is classified as data-flow and context-sensitive. This type system was associated with an analysis, DroidInfer, for type inference. The direct application of the type system and its analysis is to reveal privacy leaks. Also in [16] methods for treating Android characteristics such as inter-component communication, macros, and many entry program points were presented. DFlow and its analysis also are quipped with methods to report errors using CFL-reachability.

In [21], it was noted that methods that control access to private information such as location and contacts in mobile applications are not strong enough. In this paper, declassification policies, that control the reveal of private information by the interaction of users with constrains of mobile applications, were presented. Relying on appropriate event sequences in the application execution, the policies were independently formalized from the application implementation details.

In [8] a new programming langauge was introduced to facilitate expressing Android local policies and Android Inter-Process Communication logic. This can be realized as a try to effectively and efficiently formalize the application and verification of policies. The syntax of the proposed langauge is equipped with a scope operator used to force the application of local policies to specific parts of the code. Such techniques is also applicable to web services and object-oriented programming [8].

5 Conclusion

This paper presented a type system, *Android*$\mathcal{T}$, for Android applications. The type system was developed using a rich model for Android programming, *Android*$\mathcal{P}$. The model includes a completed set of Dalvik instructions and a set of important android macros instructions. The type system *Android*$\mathcal{T}$ can be used to prove that an Android application if free of type errors such as "method is undefined". The type system can also be used to prove the adequacy and correctness of new techniques of program analysis for Android applications. The set of types used in the type system is rich enough to host types such as reference to view which is the main component of activity which is the main component of Android applications.

References

1. Dalvik bytecode. https://source.android.com/devices/tech/dalvik/dalvik-bytecode.html. Accessed: 2016-02-1.
2. Dalvik docs mirror. http://www.statista.com/topics/840/smartphones/. Accessed: February 2016.
3. Dalvik docs mirror. http://www.milk.com/kodebase/dalvik-docs-mirror/. Accessed: February 2016.
4. Gartner, inc. worldwide traditional pc, tablet, ultramobile and mobile phone shipments. www.gartner.com/newsroom/id/2692318.
5. Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De Carmine, and Atif M Memon. Using gui ripping for automated testing of android applications. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 258–261. ACM, 2012.
6. Michele Bugliesi, Stefano Calzavara, and Alvise Spanò. Lintent: towards security type-checking of android applications. In *Formal Techniques for Distributed Systems*, pages 289–304. Springer, 2013.
7. Luca Cardelli. Type systems. *ACM Computing Surveys*, 28(1):263–264, 1996.

8. Gabriele Costa. Securing android with local policies. In *Programming Languages with Applications to Biology and Security*, pages 202–218. Springer, 2015.
9. Patrick Cousot. Semantic foundations of program analysis. In *Prentice Hall*, 1981.
10. Mohamed A El-Zawawy. An operational semantics for android applications. In *Computational Science and Its Applications–ICCSA 2016*.
11. Mohamed A El-Zawawy. Heap slicing using type systems. In *Computational Science and Its Applications–ICCSA 2012*, pages 592–606. Springer, 2012.
12. Mohamed A El-Zawawy. Recognition of logically related regions based heap abstraction. *Journal of the Egyptian Mathematical Society*, 20(2):64–71, 2012.
13. Hendra Gunadi. Formal certification of non-interferent android bytecode (dex bytecode). In *Engineering of Complex Computer Systems (ICECCS), 2015 20th International Conference on*, pages 202–205. IEEE, 2015.
14. H.S.Karlsen and E.R.Wognsen. Static analysis of dalvik bytecode and reflection in android. Master's thesis, Aalborg University, June 2012.
15. Cuixiong Hu and Iulian Neamtiu. Automating gui testing for android applications. In *Proceedings of the 6th International Workshop on Automation of Software Test*, pages 77–83. ACM, 2011.
16. Wei Huang, Yao Dong, Ana Milanova, and Julian Dolby. Scalable and precise taint analysis for android. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, pages 106–117. ACM, 2015.
17. René Just, Michael D Ernst, and Suzanne Millstein. Collaborative verification of information flow for a high-assurance app store. In *Software Engineering & Management*, page 77, 2015.
18. Aravind Machiry, Rohan Tahiliani, and Mayur Naik. Dynodroid: An input generation system for android apps. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 224–234. ACM, 2013.
19. Christopher Mann and Artem Starostin. A framework for static detection of privacy leaks in android applications. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 1457–1462. ACM, 2012.
20. Zigurd Mednieks, Laird Dornin, G Blake Meike, and Masumi Nakamura. *Programming Android*. " O'Reilly Media, Inc.", 2012.
21. Kristopher Micinski, Jonathan Fetter-Degges, Jinseong Jeon, Jeffrey S Foster, and Michael R Clarkson. Checking interaction-based declassification policies for android using symbolic execution. In *Computer Security–ESORICS 2015*, pages 520–538. Springer, 2015.
22. Greg Milette and Adam Stroud. *Professional Android sensor programming*. John Wiley & Sons, 2012.
23. Martin Mohr, Jürgen Graf, and Martin Hecker. Jodroid: Adding android support to a static information flow control tool. In *Software Engineering (Workshops)*, pages 140–145, 2015.
24. Kathryn E Newcomer, Harry P Hatry, and Joseph S Wholey. *Handbook of practical program evaluation*. John Wiley & Sons, 2015.
25. Flemming Nielson, Hanne R Nielson, and Chris Hankin. *Principles of program analysis*. Springer, 2015.
26. Damien Octeau, Somesh Jha, and Patrick McDaniel. Retargeting android applications to java bytecode. In *Proceedings of the ACM SIGSOFT 20th international symposium on the foundations of software engineering*, page 6. ACM, 2012.
27. Étienne Payet and Fausto Spoto. An operational semantics for android activities. In Wei-Ngan Chin and Jurriaan Hage, editors, *Proceedings of the ACM SIGPLAN 2014 workshop on Partial evaluation and program manipulation, PEPM 2014, January 20-21, 2014, San Diego, California, USA*, pages 121–132. ACM, 2014.

28. Nikhilesh Reddy, Jinseong Jeon, J Vaughan, Todd Millstein, and J Foster. Application-centric security policies on unmodified android. *UCLA Computer Science Department, Tech. Rep*, 110017, 2011.
29. Rick Rogers, John Lombardo, Zigurd Mednieks, and Blake Meike. *Android application development: Programming with the Google SDK*. O'Reilly Media, Inc., 2009.
30. Michael I Schwartzbach and Jens Palsberg. *Object-oriented type systems*. Wiley., 1994.
31. Ben Wegbreit. Mechanical program analysis. *Communications of the ACM*, 18(9):528–539, 1975.
32. Rubin Xu, Hassen Saïdi, and Ross Anderson. Aurasium: Practical policy enforcement for android applications. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 539–552, 2012.
33. Shengqian Yang. *Static analyses of GUI behavior in Android applications*. PhD thesis, The Ohio State University, 2015.