The Improvement of GK-tail Algorithm of Software Behavior Modeling

Yiman Zhang, Xiaolin Zhao*, Yin Zhou, Yong Wang and Xinyu Hou
School of Software, Beijing Institute of Technology, 100081 Beijing, China

ABSTRACT

Software behavior modeling is very important to analyze software system security. The goal of software behavior analysis is to obtain the semantic description and understanding of the software behavior. Software behavior modeling and testing are the key steps to achieve the goal. Therefore, this paper studies the software behavior model algorithm, GK-tail algorithm. GK-tail algorithm uses interaction trace to describe the software behavior, which is concerned interaction between the software components. However, GK-tail algorithm is not accurate. Because GK-tail algorithm uses Daikon analysis tool to obtain the predicates, which contain some errors. In order to improve the accuracy of the GK-tail algorithm in software behavior modeling, this paper improves the traditional GK-tail algorithm based on the extended finite automaton. The improved GK-tail algorithm utilizes the combination of Daikon and static (ESC/Java) tools to filter out some errors of the generated predicates. In addition, this paper designs and implements a prototype system of the improved GK-tail algorithm. The prototype system proves that the improved GK-tail algorithm can filter out errors in the second step of GK-tail algorithm effectively.

INTRODUCTION

In recent years, with the repaid development of computer, communication and network technology, software system has become the social infrastructure [1]. But the software security has become increasingly serious and software threats have caused huge damage for the life of people and even hindered the progress of society. In order to improve the software security and protect the effectiveness of information resources, the behavior of the software must be monitored [2]. Software behavior modeling is a premise for software behavior monitoring. Therefore, this paper focuses on the software behavior modeling.

Software behavior modeling is divided into two categories: application software modeling and system-call modeling [3]. Application software modeling is to maintain the state information, obtain the context information of the user behavior, analyze and predict the user's behavior and then establish the corresponding behavior model. In application software modeling, Petri Net, Finite-State Automaton, IDEF3 and state diagram are the most commonly used methods [4,5,6]. The system-call modeling refers to the behavior model that characterizes the normal behavior of the software, which depends on the related information and methods of system calls. The system-call modeling is mainly used in the field of intrusion detection system. The common methods of system-call modeling are the Forrest's N-gram model, the improved Vargram model and the subsequent FSA model, the Vt-path model, the AbstractStack...
model, ExecutionGraph model, Dyck model, VPStatic model and HFA model[7,8,9,10]. According to the different ways of collecting software system call information, the behavior modeling based on system call can be divided into three ways: dynamic modeling, static modeling and mixed modeling [11]. TABLE I illustrates a comparison of the three modeling methods.

<table>
<thead>
<tr>
<th>The ways of collecting software system call information</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic modeling</td>
<td>Dynamic training</td>
<td>The model represents the real execution information of the software</td>
</tr>
<tr>
<td>Static modeling</td>
<td>Static analysis</td>
<td>The model can extract all possible execution paths of the software, &quot;zero false positives&quot;</td>
</tr>
<tr>
<td>Mixed modeling</td>
<td>Static analysis and dynamic analysis</td>
<td>The model combines with the advantages of static and dynamic analysis. It mainly utilizes the static analysis and uses the dynamic analysis for supplementary amendments.</td>
</tr>
</tbody>
</table>

After comparing the methods of software behavior modeling, we utilize the dynamic behavior modeling method, GK-tail algorithm to model the software behavior. In the field of dynamic behavior modeling, GK algorithm is very important for dealing with the software behavior traces in the software running process. While there is a limit, GK-tail algorithm is not accurate to process the software behavior traces. Therefore, this paper improves the GK-tail algorithm to improve the accuracy.

**DEFINITIONS IN SOFTWARE BEHAVIOR MODELING**

**Definition of Interaction Traces**

First, we will define interaction traces. A parameter trace is a triple \((x,p_x,V)\). \(x \in X\), \(p_x \in D_{Rx}\), \(v \in D_v\). \(X\) is a finite set of a method, \(D_{Rx}\) is the domain corresponding to possible input parameters \(R\), \(D_v\) is the domain corresponding to variables \(V\). In the definition, \(R\) and \(V\) may be an empty set. While an interaction trace is a sequence of a parameter trace, \(it = (x_1, p_{x_1}, v_1) \ldots (x_n, p_{x_n}, v_n)\), in which \(x_i \in X\), \(p_{x_i} \in D_{Rx}\), \(v_i \in D_v\).
Finite Automaton and Extended Finite Automaton

The key to build software behavior model is to automatically generate software interaction traces. Therefore, the automaton is very important in software behavior modeling. Finite Automaton is the basic models of finite computation, many formal specifications and verification methods [12]. They are often used in behavior modeling. The most prominent feature is that the FSA is well suited to describe the systems where the states are limited and the occurrence of events leads to state migration.

Finite Automaton, denoted by $M=\langle Q, \Sigma, \delta, q_0, F \rangle$. $Q = \{ q_0, q_1, \ldots q_n \}$ is the collection of limited state. At any given moment, the finite automaton only is in a certain state $q_i$. $\Sigma$ is the alphabet and $\delta: Q \times \Sigma \rightarrow Q$ is the state transfer function, in which $q_0$ is the initial state. $F$ is a subset of $Q$ and a set of final states. The finite automaton starts to accept input from $q_0$ and no longer receives input after reaching the final state $F$. The finite automaton is determined if $\delta(q, e)$ has up to one member, in which $q$ in $Q$ and $e$ in $\Sigma$ (see Figure 1).

![Figure 1. State transition of function call error detection model.](image)

EFA is built through $X, R, V$, denoted $(S, T, s_0, s_T)$. $S$ is a finite state set and $T$ is a finite conversion set in $S$. In $S$, $s_0 \in S$ is the initial state and $s_T \subset S$ is the final set. A conversion $t \in T$ is denoted $(s, x, P, s')$, in which $s', s' \in S$, $x \in X$ and $P: D_{Rx} \times D_V \rightarrow \{ \text{True}, \text{False} \}$. More specifically, $s$ and $s'$ is the source status and target status in the conversion, $x$ is the call method in the conversion and $P$ is the conversion predicate indicating that the conversion method receives the value of the input parameters and variables in the conversion.

Then, we will define EFA.

(1) Predicate is true. For $t = (s, x, P, s')$ and $r \subset D_{Rx} \times D_V$, at least there exists a conversion $t'= (s, x, P', s'')$ is true for $P'(x, r)$. So each method can be called in a state, at least one of these values that are the values of the parameters and variables associated with this method is convertible.

(2) Input is true. For $(s, x) \in S \times X$, at least one of the input $x$ and the state $s$ is convertible. So at least for some values of parameters and variables, all methods can be called in all states.
Conversion is determined. The predicate of the conversion with the same states is mutually exclusive for any input \( x \). So for a given method call and a parameters and a set of local variable values, there is at most one state of the conversion.

In this paper, EFA (Extended Finite Automaton) is used to build the software behavior model [13].

SOFTWARE BEHAVIOR MODELING METHOD AND TOOLS

GK-tail Algorithm

GK-tail is a dynamic behavior modeling algorithm for software behavior based on extended finite Automaton model [14]. The model is built fully automatically and the GK-tail algorithm depends on the software system execution information, which is obtained by the monitoring the software. Therefore, the GK-tail algorithm not only relies on the correct sample but also cannot identify the wrong sample, so this algorithm does not have the ability to self-learning.

GK-tail algorithm handles a large number of interaction trace sets and generates an extended finite automaton model. The behavior traces of software is the interaction and calls between software components, which refers to a series of function methods calls with parameters, associated variable information. Moreover, a software behavior trace is also a specific execution of the software system. The trace of method call describes the sequence of the method call, which mark the values of parameters and variables in the call. Figure 2 shows the sequence of the method call and Figure 3 demonstrates the corresponding interaction trace set. In Figure 4, the mark under the label indicates the value of the parameter. The process of GK-tail algorithm is as following.

1. Merge the software behavior trace of “input equivalence”.
2. Generate the predicates of the software system.
3. Establish an initial extended finite automaton.
4. Merge the equivalent state and obtain the final extended finite automaton.

![Figure 2. Sequence diagram of the method call.](image)
In first step of GK-tail algorithm. The interaction trace represents the set of executions. Several different interaction traces often trigger an identical behavior. Therefore, the different values of the parameters call the same method sequence. GK-tail algorithm utilizes this feature to simplify the input of “input equivalence” to the data set. Figure 4 indicates the data set form interaction trace in the Figure 3.

In second step of GK-tail algorithm. The predicates are obtained from the data set generated in first step. A predicate summarizes the conditions from the accepted method calls. In GK-tail algorithm, Daikon analysis tool is used to generate the predicates. Figure 5 explains the predicates generation process.

In third step of GK-tail algorithm. An initial extended finite automaton is generated by adding a public initial state. Figure 6 indicates the initial EFA.

In fourth step of GK-tail algorithm. The initial EFA is streamlined to the final EFA by merging the equivalent state.
In GK-tail algorithm, a dynamic analysis tool Daikon is used to generate the predicates in the second step. In second step, it is not accurate to use Daikon analysis tool to obtain the predicates because there are many errors in the generated predicates. Therefore, this paper improves the analysis tool in this step to obtain more accurate predicate. Comparing the dynamic and static analysis tools, this paper will improve the analysis tools in the GK-tail algorithm.

**Software Behavior Modeling Tools**

ESC/Java, the “Extended Static Checker for Java” is a programming tool that attempts to find common run-time errors in Java programs at compile time [15]. ESC/Java is a collective name referring to a range of techniques for statically checking the correctness of various program constraints. ESC/java not only checks out the assertions in the source code but also checks the common errors in the Java code, such as bounds of an array, null pointer reference. Besides, ESC/java can check errors that occur in concurrent programs during synchronization. In ESC/Java, there are two technologies: verification condition generation and automatic theorem proving.

Daikon analysis tool is a computer program that detects likely invariants of programs [16]. An invariant is a condition that always holds true at certain points in the program. It is mainly used for debugging programs in late development, or checking modifications to existing code. Daikon can automatically insert the description of the program into the appropriate location of the program. There are three steps to detect the invariants of programs. Daikon records the value of the interest variables and the invariants can be deduced from the records.

Dynamic techniques may find invariants in a program by running a program. But the invariants of the program found may not be correct. Meanwhile, static technology can verify that certain properties are always correct. But it is very difficult to choose a validation target for a static validator and add comments. Therefore, it is good to combine the two tools to find and check the invariants in the software. Daikon analysis tool is used to get the invariants of programs firstly. Then, ESC/java tool is used to check the invariants generated by Daikon.
IMPROVING GK-TAIL ALGORITHM

In this paper, we combine ESC/java with Daikon analysis tool to improve the accuracy of GK-tail algorithm. In improved GK-tail algorithm, ESC/java and Daikon analysis tool are used to get the predicates of software behavior, which can filter out some errors and improve the correctness of GK-tail.

We improve the second step of the GK-tail algorithm by combining the Daikon with ESC/java analysis. This improved GK-tail algorithm can filter out error predicates and obtain more accurate results. The improved GK-tail algorithm is divided into four steps based on the definition of interaction traces and EFA. (1) Monitoring software Aspectwerkz is used to get the monitoring traces. Besides we merge the input-equivalent traces and record it. (2) Using the combination of Daikon and ESC/Java two analysis tools to generate the predicate of the relevant traces. (3) Initial EFA is generated. (4) Merge the equivalent states and optimize the initial EFA to get the final EFA.

In the first step. We use Aspectwerkz tool to monitor the target software and merge the traces of “input-equivalent”. GK-tail simplifies the results into the data set.

In the second step. The combination of Daikon and ESC/java analysis tool is used to generated the predicates of the traces. The difference of original GK-tail and improved GK-tail is the use of analysis tools. We define the combination function of Daikon and ESC/java daikonEsc:

\[ \varphi(D_{Rx}) \cup \varphi(D_{V}) \rightarrow P. \]

\( D_{Rx} \) is an evaluation set for any input \( x \in X \). \( D_{V} \) is an evaluation set of variables in the context. \( P : D_{Rx} \times D_{V} \rightarrow \{\text{True, False}\} \) is predict based on the parameters and / or context variables. When \( \text{daikonEsc(Data}_x, \text{Var}) = P, \forall p_x \in \text{Data}_x, d_v \in V_{ar}, P(p_x, d_v) = \text{True} \).

In the improved GK-tail algorithm, the combination of Daikon and ESC/java filters out the error information. Therefore, the predicates generated in this step are more effective, which improve the accuracy of software behavior model.

In the third step, the initial EFA is generated from the interaction traces that are annotated by the predicates [17]. For the interaction traces set annotated by predicates \( \{seq_1, ... , seq_m\}, seq_i = (x^i_1, p^i_1) ... (x^i_m, p^i_m) \). For \( \forall i = 1, ... , m \), the initial EFA \( (S, T, s_0, s_F) \), \( S = \{s_0, s_1, ... , s^1_n, s^2_1, ... , s^2_m, ... , s^m_1, ... , s^m_m\} \). For each \( (x^i_j, p^i_j) \), if \( j > 1 \), there exists a conversion \( t = (s^i_{j-1}, x^i_j, p^i_j, s^i_j) \in T \). In initial EFA, \( s_F = \{s^i_m | i = 1, ... , m\} \) and \( s_0 \) is the initial state.

In the last step. The final EFA is obtained. When emerging, if the states have the same path, the states are equivalent.

The improved GK-tail algorithm uses the combination analysis tool to generate the more accurate predicates and improve the accuracy of the software behavior model.
EXPERIMENT AND RESULTS
Experiment Design of Software Running State Monitoring System

In order to implement the improved GK-tail algorithm, we design a prototype system of software running state monitoring system. This system monitors the function of adding users in the students’ management system. Besides, software running state monitoring system also provides the state diagrams that are used to examine the accuracy of improved GK-tail algorithm in the following study.

Software running state monitoring system includes the four functions: monitoring the software running traces, checking the predicates for the running traces, merging the equivalent state and drawing the state diagram of software running. Figure 7 shows the process of software running state monitoring system. In this experiment, AspectWerkz is used to monitor the software, Daikon and ESC/Java tools are used to generate the predicates and SWT/Jface with Canvas are used to draw the state diagram.

In monitoring function part, the input is a series of specific actions of the user to the target monitoring system, and the output is a "software traces" record file. In generating predicates part, Daikon is used to analyze the invariant for such as .dtrace input file and JML file is got. Then, input the JML of invariant to ESC/java tool. ESC/java checks the JML file and system source and throws the conflicting program invariants. Finally, the predicates with high accuracy are output to the such as .data files. In merging the equivalent state part, EFA will be generated from such as .dtrace file and .data file. File .efsm stores the EFA. In the last part, SWT(JFace) and Canvas draw the software running state diagram based the .efsm file.
Experiment Results of Software Running Status Monitoring System

First, we write the monitoring class aspect and define the output file monitor.dtrace. In this experiment, we write 50 files and the 50 monitor.dtrace will be generated indicating the software traces.

Then we use Daikon tool to get the invariables from the .dtrace file and ESC/java to filter out the errors of invariables. Figure 8 is the invariables file. Then the EFA will be generated. Figure 9 shows the diagram of initial EFA and the Figure 10 shows the diagram of final EFA. In the last, the state diagram is got (see Figure 11).
From the Figure 9 and Figure 10, we can find that in the second step of experiment, some errors are filtered out by the tool ESC/java. Figure 9 shows 16 software running states and Figure 10 shows 8 software running states. So, the ESC/java tool filters out 8 software running states. Compared with the traditional GK-tail algorithm, we can find that the improved GK-tail algorithm is more accurate to build software behavior model. Moreover, the improved GK-tail algorithm can exclusive the errors of the generated predicates in the second step. Therefore, the improved GK-tail algorithm is more accurate than the traditional GK-tail algorithm.

CONCLUSIONS

This paper studies the algorithms of software behavior model. It is found that GK-tail algorithm is proper to analyze the software behavior. While the traditional GK-tail algorithm is not so accurate. Therefore, this paper improves the traditional GK-tail algorithm based on the analyzing the GK-tail and some analysis tools.
GK-tail algorithm not only considers the method call but also considers the call parameter information and context variable information. While the generated invariants in the traditional GK-tail algorithm contain some errors, which cause the results is not so accurate. Therefore, in order to improve the accuracy of GK-tail algorithm, this paper improves the traditional GK-tail algorithm using the combination of ESC/java and Daikon. A prototype system of software running state monitoring system is designed and implemented based on the improved GK-tail algorithm. This system also outputs the system state diagram. In the future, we will use the state diagram generated in the software running state monitoring system to examine the accuracy of the improved GK-tail algorithm.

ACKNOWLEDGEMENT

This work was supported by National Key R&D Program of China (Grant No. 2016YFB0800700).

REFERENCES

