Structured Algorithm for Software Behavior Model based on Finite State Automaton

Quanbao Chen, Xiaolin Zhao, Hang Yin, Jinyan Zhang and Meijing Wu
School of Software, Beijing Institute of Technology, 100081 Beijing, China

ABSTRACT

At present, with the rapid development of software, software security is very more and more serious. In order to solve the software security problem, it is necessary to monitor the software behavior. Therefore, this paper studies the differences between the running states of the software, and finds out and quantifies the differences, which further monitors the software behavior in software security. This paper analyzes the topology structure algorithm of software behavior model and proposes FSM Diff algorithm based on finite state automaton. FSM Diff algorithm is more effective to select the key state nodes. Besides, experiment shows that FSM Diff algorithm is effective to select the key state nodes in the software running state diagram.

INTRODUCTION

Software Security Research Status

Nowadays, information technology becomes more and more important. The security threats such as the new malicious code becomes more and more serious. In information system security area, software defects and software vulnerabilities are one of the biggest dangers. Software security has become an urgent problem to be solved.

If a software is attacked, its behavior is different from the standard software. Therefore the software behavior can be used to detect whether it has been attacked. The software behavior modeling is necessary to solve the software security problem. The software behavior model is applied to detect the software behavior. Because the standard software behavior model is used as the reference system in the detection. The accuracy of the standard model and the tested model directly determines the performance of the detection system. Therefore, good model algorithm is very important for software security.

Software Behavior Model

Software behavior model is divided into the static modeling and the dynamic modeling. Static modeling directly builds software behavior model based on analyzing the program source code or binary code and extracting all possible actions of system call. In theory, the static modeling method extracts all possible execution of system calls, but in practice, it is often limited by factors. For example, the source code is not convenient to get and the software behavior is greatly influenced by the run-time environment.

Dynamic model records the all behavior information of the system call. Although the dynamic modeling cannot capture all the execution paths of the program, it is better
than static modeling in practice. In dynamic modeling, Gopalakrishna\(^3\) present PDA (push-down automaton) model and Liu et al.\(^4\) proposed HPDA model. HPDA model extends the PDA model, which combines the advantages of static analysis with the advantages of dynamic learning. So, HPDA model is more accurate than dynamic and static modeling.

In this paper, we first model the state nodes that can express the software behavior in the process of running the software, and obtain the software running state diagram. In the second step, we build the finite state automaton model and utilize the structured algorithm to compare the software behavior state diagram. Finally, we identify and quantify the differences between the software running state diagrams and further monitor software behavior [5].

**RELATED THEORIES**

**Finite State Automaton**

Petri net, IDEF3 and finite state automaton are common methods to emulator the software behavior in software model. TABLE I shows the differences of three methods.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Petri nets</td>
<td>It is suitable for describing asynchronous concurrent systems [6-7. Rigorous math theory. Facilitate verification and analysis.</td>
<td>Resource competition is not prominent. It is not suitable for the single-user application software.</td>
</tr>
<tr>
<td>FSA</td>
<td>The relationship between the state is clear. It is suitable for object-oriented software and has low complexity 8.</td>
<td>It can’t express concurrency. It is too difficult to describe the complex system.</td>
</tr>
<tr>
<td>IDEF3</td>
<td>It is suitable for complex enterprise system modeling. 9</td>
<td>It is too complex.</td>
</tr>
</tbody>
</table>

Petri nets, IDEF3, process algebra [10] and π calculus [11] and regular expressions [12] are used to describe the software behavior in addition to the finite state automaton. The most software is object-oriented in software behavior model. Therefore, it is found it that finite state automaton is more suitable for software behavior modeling. Therefore, this paper utilizes finite state automaton to obtain software running state diagrams.

**Definition of Labeled Transition System**

In this paper, software behavior is modeled by the Labeled Transition System (LTS), which is a traditional accepted state transition model based on abstract state machine. Labeled Transition System was first proposed by Keller and Lien. Now we define Labeled Transition System 13.

The behavior model of software \( E \) can be defined as a five-tuple: \( E = (\mathcal{Q}_E, \sum E, \Delta_E, q_{E0}, Q_{EF}) \). In \( E \), \( \mathcal{Q}_E \) is a finite state set, \( \sum E \) is the set of actions,
∑EI is the set of input actions, ∑EH is the set of internal actions, ∑EP is the set of output actions and ΔE = Q_E × ∑E × Q_E is the set of state transitions. q_{E0} ∈ Q_E is the Initial state. Q_{EF} belongs to Q_E is the set of end states and a software may have many end statuses.

Finite state automaton can be interpreted as LTS. LTS includes data transition and state transition. The software behavior based on the states is usually assumed to be complete. While in practice, the model is often not complete. So, in this paper we assume the software behavior model is Partial Labeled Transition System (PLTS). [14]

Now, we define PLTS. We denote PLTS as 

\[ E = (Q_E, ∑E, Δ_E, q_{E0}, Q_{EF}, ψ_E) \]

E is a six-tuple, which is similar to LTS. In order to distinguish unknown behavior from invalid behavior, ψ is introduced. Invalid behavior is described as

\[ \psi_E : Q_E × ∑E → Q_E, \psi_E \cap Δ_E = \emptyset. \]

Now, we define (P)LTS. For a sequence \( α \in ∑^* \), if there is a path from the initial state \( q_0 \) to the state \( q \), then the sequence \( α \) is the language of FSM, which is present \( q_0 \xrightarrow{α} q, q ∈ Q \).

The software operating-state diagram is defined as a labeled transition system. The LTS is equivalent to a finite state automaton. All software running-state diagrams in this paper are the directed graph of the software behavior model based on the finite state automaton.

STRUCTURE COMPARISON ALGORITHM OF SOFTWARE BEHAVIOR MODEL

In order to make the comparison of the software running state diagram more effective, this paper proposes the FSM Diff algorithm based on the local similarity and global similarity. FSM Diff algorithm is used to select the key nodes. In addition, Precision and Recall parameters are used to quantify the state diagram in the structure. In the key node selection, FSM Diff algorithm can select the key state nodes with high similarity.

Calculate State Similarity

The core of the structure comparison algorithm is to quantify the similarity of state pairs in different software running state diagrams. Firstly, we get the relevant jumps of the state pairs and calculate the local similarity degree. Then, we according to the rules merge the local similarity degree and obtain the global similarity degree.

LOCAL SIMILARITY DEGREE

In the state diagram, the local similarity \( L_{AE} \) of state A and state E is calculated by that overlapping hop count in the surrounding is divided by the total hop count.

We assume that the finite state automaton of the state diagram is determined. When the hop condition is unique, the state node of the finite state automaton points
to the unique destination state node through the hop condition. The equation 1 shows
the Local similarity degree. If both states do not have a hop, the \( S_{EA} = 0 \).

\[
L_{AE} = \frac{|\sum A \cap \sum E|}{|\sum A \cup \sum E|}
\]

(1)

In Figure 1, the number of combinations of the jump condition q is 2*2 and the
number of combinations of the jump condition p is 1*1. While the number of
non-matching jumps c and d is 1+1. Therefore, the local similarity degree is as
following.

\[
L_{AE} = \frac{(2*2+1*1)}{(2*2+1*1)+(1+1)} = 5/7
\]

In Figure 1, the target state is that the state A and E can reach the states by the jump
condition q. The set of target states of state A and E in the surroundings is \( Succ_{A,E} \).

Equation 2 shows the \( Succ_{A,E} \)

\[
Succ_{A,E} = \{(q_A,q_E) \in Q_A \times Q_E \mid \exists a \in \Sigma, A \rightarrow Q_A \land E \rightarrow Q_E \}
\]

(2)

Equation 1 defines the set of jump-out matching state pairs in the surrounding of
target states. In the same way, we define the set of jump-in matching state pairs
\( Prec_{A,B} \) (see equation 3).

\[
Prec_{A,B} = \{(q_A,q_E) \in Q_A \times Q_E \mid \exists a \in \Sigma, Q_A \rightarrow A \land Q_E \rightarrow E \}
\]

(3)

The set of matching state pairs in the surrounding of target state, \( Surr_{A,E} \), consists
of \( Succ_{A,E} \) and \( Prec_{A,E} \). Equation 4 indicates \( Surr_{A,E} \). Therefore, the local similarity
degree is shown in equation 5.

\[
Surr_{A,E} = Succ_{A,E} \cup Prec_{A,E}
\]

(4)

\[
L(A,E) = \frac{|Surr_{A,E}|}{|\sum A - \sum E| + |\sum E - \sum A| + |Surr_{A,E}|}
\]

(5)

GLOBAL SIMILARITY DEGREE

In the comparison of the state diagram, it is necessary to quantify the global
similarity degree of the state pair in the structure of the whole software running state
diagram. For the state pair A and E, the states in the surroundings are more similar,
they are also more similar. Therefore, the similarity degree of state pair should add the similarity degree of matching state pair in the surroundings. The global similarity degree extends the local similarity degree.

In order to give a higher weight to the state that is closer to the state to be measured and take the importance of the local similarity into account, we make the following adjustments.

1) We use the average value to replace the sum of similarity degrees of the surrounding matching state pairs.

2) We introduce the index k (0<k<1). So, the influence of local similarity can also be adjusted by k. The k is larger and the influence is larger.

After the adjustments, we get the equation 6.

\[ L(A, E) = (1 - k) \frac{| \text{Surr}_{A,E} |}{\sum A - \sum E + \sum E - \sum A + | \text{Surr}_{A,E} |} + k \sum L(q_1, q_2) \]

\[ \text{(6)} \]

Combined with the completely different and incompatible situation of the state to be tested, equation 7 shows the global similarity degree.

\[ L(A, E) = \begin{cases} 
(1-k) \frac{| \text{Surr}_{A,E} |}{\sum A - \sum E + \sum E - \sum A + | \text{Surr}_{A,E} |} + k \sum L(q_1, q_2) & \text{Similar and compatible} \\
0 & \text{Not similar but compatible} \\
-1 & \text{incompatible} 
\end{cases} \]

\[ \text{(7)} \]

**Solving the Similarity Equation**

Given two running state diagram X and Y, for every two-tuple \((q_x, q_y) \in S_x \times S_y\) we can obtain the equations \(SC(q_x, q_y)\) according to the global similarity degree. The square state represents the deny state node and the circular state represents the accepting state node. Figure 2 illustrates two software operating state diagrams. The corresponding matrix is the coefficient matrix of equations obtained by global similarity degree. The first row of the matrix can be obtained by calculating the global similarity of state A and E, \(\text{Surr}_{A,E} = \{(B,F)\}\). In addition, \(| \text{Surr}_{A,E} | = 1, |\sum A - \sum E| = 0, |\sum E - \sum A| = 1\). So, we add these conditions to the equation 6 and the result is as following.

\[ L(A, E) = \frac{1}{0+1+1} (1-k) + kL(B, F) \]

\[ L(A, E) - kL(B, F) = \frac{1}{2} (1-k) \]

The unknown numbers in the equations are the global similarity degree of the state pairs. The coefficients and constants of the equations can be obtained by the local information of each state pair \(\text{Surr}_{A,E} \cdot \sum A - \sum E\) and \(k\) is defined by ourselves.
TABLE II. AUTOMATIC MACHINE COMPARISON SOLUTION.

<table>
<thead>
<tr>
<th></th>
<th>AE</th>
<th>AF</th>
<th>BE</th>
<th>BF</th>
<th>CE</th>
<th>CF</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>AE</td>
<td>1</td>
<td>-k</td>
<td>(1-k)/2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AF</td>
<td>1</td>
<td>-k</td>
<td>(1-k)/5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BE</td>
<td>1</td>
<td>-k</td>
<td>-k</td>
<td>(1-k)/3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BF</td>
<td>-k</td>
<td>-k</td>
<td>-k</td>
<td>1-k</td>
<td>(1-k)/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CE</td>
<td>-k</td>
<td>-k</td>
<td>-k</td>
<td>-k</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CF</td>
<td>-k</td>
<td>-k</td>
<td>-k</td>
<td>-k</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TABLE II describes the comparison solution of automatic machine. We can according to the global similarity degree calculate the first four rows in the matrix. In Figure 2, the state pair (C, E) and (C, F), the state C into the deny state and the state E, F into the accept state through the same jump condition a. Therefore, the state pair (C, E) and (C, F) are incompatible. $L(C, E) = -1$ and $L(C, F) = -1$.

In the equations, we give the $k = 0.5$ and get the similarity degree of every state pair (see equation 8).

$$
\begin{align*}
L(A, E) &= -1/120 \\
L(B, E) &= -19/120 \\
L(A, F) &= -61/120 \\
L(B, F) &= -31/60 \\
L(C, F) &= -1 \\
L(C, F) &= -1
\end{align*}
$$

We can come to conclusions in the following combined the actual structure of the automaton with the results of the equations.

(1) In Figure 2(2), the second automaton does not contain a deny state, so it is an
indeterminate automaton and the deny state is as infinity. The similarity between the nodes with a large distance from the rejected state is also large. For example, \( L(C, E) \) and \( L(C, F) \) is -1, \( L(B, E) \) and \( L(B, F) \) is lower than \( L(A, E) \) and \( L(A, F) \). Therefore, the state is closer to D state and the similarity is lower, which is in line with the real situation.

(2) It is important to select a proper threshold \( k \). In this paper, \( k = 0.5 \) causes all the similarity degree is negative because the similarity of the surrounding state has a too great effect on the similarity of the measured state.

By analyzing the equations derived from the global similarity degree formula, we can find the state pairs with the highest degree and the highest matching degree. These state pairs are regarded as key state pairs.

It is necessary to select key state pairs from the state of the two automatons. In order to find out the key nodes, we need two thresholds to select. The first threshold is a common measure, and only the state pairs that are higher than the first threshold selected. The second threshold is to exclude the combination of the candidate key state pairs according to the apparent peak.

**Select Key State Pairs**

In the comparison process of the state diagram, the state nodes with the highest similarity degree are used as the key state node. FSM Diff algorithm is used to compare and select the key state pairs in the FSM. The initial key set is the key nodes set of two automata and the rest state nodes are compared based on initial key set. In FSM Diff, we first calculate the set of key state pairs. Then for the state nodes out of set with higher similarity are added to the set and do union operation. In the last, the set of key state pairs is considered to be the matching part of the two FSM. Figure 3 indicates the process of FSM Diff algorithm.

The state that is not in the set of key state pairs Matches in the state diagram to be measured is add to Redundant and the state that is not in the set of key state pairs Matches in the standard state diagram is add to Deficiency. Finally, FSM algorithm return Redundant, Deficiency and Matches. Redundant, Deficiency and Matches are used in the next step.

**Calculate the Accuracy and Coverage**

Precision parameters and Recall parameters are present by Van Rijsbergen 14 and used to compare the two similar objects. The two objects to be compared are treated as standard objects and target objects. The Precision parameter and the Recall parameter indicate the difference between the target object and the standard object. REL is the set of objects to be compared in the standard object, and RET is the set of objects to be compared in the target object. Equation 9 and equation 10 indicates the calculation of Precision and Recall.

In FSM Diff, Redundant and Deficiency are obtained, which represent the specific differences in the structure of two automatons. Precision and Recall can quantify the differences.
Figure 3. Process of FSM algorithm.

\[
\text{Precision} = \frac{|REL \cap RET|}{|RET|}
\]

\[
\text{Recall} = \frac{|REL \cap RET|}{|REL|}
\]

We assume that the FSM Diff inputs the standard running state diagram A and the target running state diagram E. Then, \( REL = \Delta A \), \( RET = \Delta E \) indicates the jump set or state set of A and E. The union set of REL and RET is got by FSM Diff algorithm. Finally, the Precision and Recall can be calculated in equation 11.

\[
REL \cap RET = \{ \delta \in (T_A \cup T_E) \mid \delta \in (\text{Redundant} \cup \text{Deficiency}) \}
\]
EXPERIMENT RESULTS AND DATA ANALYSIS

In order to examine the validity of the FSM Diff algorithm for state diagram comparison. This paper builds the extended Precision and Recall scores between the target behavior model and the standard behavior model, analyzes the matching relation and compares the state diagram based on the obtained data and result.

Experiment Design and Method

Concurrent Version System (CVS) is a code version control software. Because CVS is a relatively mature standard model, this experiment compares the operating state diagrams that are generated by different CVS client software. In this experiment, the standard model is constructed by Roth et al [15] through a program of trace results. Besides, this standard model describes the software behavior performed by the CVS client, including basic operations such as software initialization, connection to server, login, modification of version files, and logout and disconnection. In this experiment, other models compare with this standard model. The automaton generated by the standard model is denoted by $FSM_S$. In addition, the automaton of the model to be tested is denoted by $FSM_{EDSM}$.

In the first step, the running state of standard model and the model to be tested is generated.

Figure 4 illustrates running state diagram of the standard model. The automaton of the standard model is as following (see Figure 6). In Figure 4, there are 16 state nodes, 14 operations and 27 jump processes. $q_0$ is the initial state node and $q_{15}$ is the end state node.

We use EDSM to construct the software behavior model of the model to be tested (see Figure 7). In $FSM_{EDSM}$, the initial state of FSMEDSM is $Q_{E1}$, which is 0 node in state diagram. The end state set contains $Q_{E7}$, $Q_{E10}$ and $Q_{E17}$, which are 6,9,16 nodes in the state diagram. Besides, Figure 5 shows the running state diagram according to $FSM_{EDSM}$. In Figure 5, there are 17 state nodes, 14 kinds of operations, and 22 jump processes.

![Figure 4. Running state diagram of standard CVS.](image-url)
In the second step, we compare two running state diagrams. The Figure 8 shows the process of the comparison. In this step, we calculate the local similarity degree and global similarity degree of the state nodes. The similarity equation is generated according to the global similarity degree. Besides, we assume $k = 0.4$ and solve the similarity equations. In the last, we assume the thresholds $t = 20\%$ and $m = 2$ and use FSM Diff algorithm to select the key state pairs. The key state pairs are input to the next step.
Results and Analysis of Experiment

In the third step, Redundant and Deficiency are generated according the input. After comparing the standard model with the model to be tested, it is found that Redundant, Deficiency and jump are different. The specific results are as following. Redundant is \{5,6,7,8,9,10\} and Deficiency is \{4,5,6,8,9,10\}. Specifically, Redundant(Q) = \{qE5, qE6, qE7, qE8, qE9, qE10\} and Deficiency (Q) = \{qS4, qS5, qS7, qS8, qS9\}. Besides, the corresponding jump is

Redundant(P) = \{ q_{E3} \rightarrow q_{E7} , q_{E7} \rightarrow q_{E3} , q_{E3} \rightarrow q_{E10} , q_{E3} \rightarrow q_{E8} , q_{E8} \rightarrow q_{E9} , q_{E4} \rightarrow q_{E3} , q_{E4} \rightarrow q_{E5} , q_{E5} \rightarrow q_{E6} \}

Deficiency(P) = \{ q_{s3} \rightarrow q_{s6} , q_{s6} \rightarrow q_{s16} , q_{s16} \rightarrow q_{s11} , q_{s11} \rightarrow q_{s10} , q_{s10} \rightarrow q_{s5} , q_{s50} \rightarrow q_{s5} , q_{s5} \rightarrow q_{s11} , q_{s11} \rightarrow q_{s9} , q_{s9} \rightarrow q_{s11} , q_{s11} \rightarrow q_{s4} , q_{s4} \rightarrow q_{s8} , q_{s8} \rightarrow q_{s4} , q_{s8} \rightarrow q_{s8} , q_{s8} \rightarrow q_{s9} \}

The Redundant embodies the redundant states of the target state diagram. So, in the same way, we can get the missing states of the target state diagram by the Deficiency.

In the fourth step, the Precision and Recall of the two state-diagrams are calculated. In this experiment, we get REL=|Qs|=16  RET=|Q_{EDSM}|=17. Then, we use FSM Diff algorithm to get REL∩RET = 11. Therefore, Precision(Q) = 11/17 = 0.6 and Recall(Q) = 11/16 = 0.65.

In this experiment, we use Redundant and Deficiency to calculate Precision and Recall, which quantify the running state diagram. From the state-based structure perspective, the number of missing and increased state nodes in the target diagram is small (6 and 5) and the result of Precision and Recall is higher (0.65, 0.69), which indicates that the bias of the actual operating of the software and the standard is small and the coverage is relatively high.
CONCLUSIONS

This paper studies the software behavior model topology algorithms and proposes the FSM Diff algorithm. In order to monitor the software behavior, we compare the target software operating state diagram with the standard software operating state diagram. Besides, we validate the results by Precision and Recall. In addition, FSM is used to model the software behavior. FSM Diff algorithm selects the key state nodes in FSM.

This paper carries out an experiment based on CVS to examine the validity of FSM Diff. It is found that the FSM Diff algorithm is effective. While there are some limits, in the future, we will do further study and consider the weight of each jump.

ACKNOWLEDGMENT

This work was supported the National Key R&D Program of China under Grant No. 2016YFB0800700.

REFERENCES
