ATLANTIS PRESS

Advances in Intelligent Systems Research (AISR), volume 142

2017 International Conference on Information Technology and Intelligent Manufacturing (ITIM 2017)

Research on Error Diagnosis Method Based on Single FSM State Machine Model

MingCheng Qu[1,2],XiangHu Wu[1,2],YongChao Tao[1,2],Ying Liu[1,2]

[1]School of Computer Science and Technology, Harbin Institute of Technology; Harbin 150001 Heilongjiang China

[2]Shenzhen Academy of Aerospace Technology, Shenzhen 518057 Guangdong China

[*]E-mail:qumingcheng@hit.edu.cn

***Abstract*—The shortcomings of the existing single Finite State Machine (FSM) model and diagnosis method are given by analyzing the single FSM model and problem model. Then, according to the shortcomings of the current single FSM error model, a new state machine model and problem model are proposed, that is, the traditional FSM model increase the non-executable conversion, and then increase the two error models, that is, the conversion unexecuted errors and the conversion redundant error. According to the phased diagnosis method, an improved algorithm is proposed based on the existing algorithm to improve the efficiency of the algorithm. Experimental tests show that the proposed improved FSM problem model and the error diagnosis method are correct and effective.**

Keywords—Finite state machine; forward analysis; reverses analysis; judgment error

I. INTRODUCTION

State machine is finite state machine FSM [1] abbreviation, divided into Moore state machine [2] and Mealy state machine [3], Mealy state machine refers to the output is not only related to the state and input-related state machine, The state of the study refers to the Mealy state machine.

At present, there are some achievements in the research based on the state machine diagnosis method, but the model of the state machine is less, the error model is limited to the output error and the conversion error. The designed algorithm is also rough. For the actual state machine model based test, there is no error diagnosis method for the case where the conversion error is not performed or there is an excess conversion, and the existing diagnostic method still needs to be improved and optimized.

Aiming at the research status and existing problems of state machine error diagnosis, this paper presents a new problem model. Based on the new problem model, a complete improved algorithm is proposed on a single state machine. The idea of the algorithm in this paper is a good theoretical guidance for the enrichment and perfection of the existing state machine diagnosis method. Most of the studies on FSM error diagnosis are based on a single FSM model, and this article also examines the error diagnosis method under a single FSM.

II. SINGLE FSM MODEL AND PROBLEM MODEL

As shown in Fig.1, a single FSM test framework, the state machine from the external test channel to receive input, the conversion will output to the external test channel.

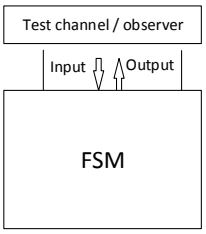

Fig. 1. Single FSM test framework

For a single FSM model, consider the following four errors, that is the error model:

(1) Output error: After the conversion, the end state is the same as expected, but the actual output is not consistent with the expected (including the actual no output situation).

(2) Conversion error: After the conversion, the actual output is consistent with the expected, but the arrival of the end state is different from expected.

(3) Conversion is not implemented error: conversion does not occur, the performance of the step without output; in the state machine is expressed as the output is empty, end state and head state consistent conversion.

(4) Conversion redundant error: the original state machine does not exist in the conversion, the actual implementation of the output (or output is empty), and a state transition (when there is output from the transfer, the output is empty when not self-transfer).

The error model has two situations to note: one situation is when an auto-conversion occurs when the output is empty and an auto-conversion is not performed. When this happens,

although it is impossible to determine which error is specific, wrong conversion; the other case is when the conversion of redundant errors for a self-conversion and output is empty, the situation although the error occurred, but did not produce symptoms, so do not discuss the situation.

A. Pretreatment

The first part is preprocessing, including the following steps:

(1) generates a transition sequence TR and an expected output O for each use case, and generates a conversion set T- that cannot be executed by the generation state.

(2) Compare the expected output and actual output of each use case, generate all the symptom sets SS, and divide the TS into all use case tables TSs with symptoms and all use cases without symptoms.

(3) In the use case with symptoms, the first symptom of each use case is called the initial symptom, denoted as IS, and the initial symptom set for all use cases is recorded as ISS. The initial path before the transition path is called the Conflict Set CS.

B. To determine the error

The second part is to determine the wrong process, according to the wrong phenomenon is different; the decision process in accordance with the output error, conversion is not the implementation of errors, conversion errors and conversion redundant error in the order.

(1) To determine the output error

The output error object is the transition that exists in the state machine implementation, including the case where the output is empty. Using forward analysis, when an output error occurs, the arrival of the state does not change, it will not affect the other expected conversion sequence. Since there is only one error, all use cases produce the same symptoms, and their corresponding conversions are the same and unique, which we call the unique symptom transition (ust). So the algorithm needs to determine whether all the corresponding output and conversion of the symptoms are equal. If not, then the error may be excluded, if so, then continue. Then use the reverse analysis, because the output error will produce symptoms, so in all cases without symptoms, there must be no error conversion. So the algorithm needs to use the case in the absence of symptoms to determine whether the existence of the suspicious conversion. If it is present, it is possible to exclude the error. If not, the error may be established and the suspicious conversion will be recorded. After confirming the suspicious conversion set, it is not necessary to verify all use cases as long as it verifies that the output of the suspicious conversion is verified in the use case that contains the suspicious conversion to verify its output backwards. In addition, if the output is empty error, suspend conversion that step corresponds to the same conversion, the output is empty.

According to the above analysis, the output error judgment algorithm is described as follows:

TABLE I. OUTPUT ERROR DECISION ALGORITHM

Algorithm input: state machine FSM, all symptom set SS, all conversion sets TR Algorithm output: ustset record output conversion error

Procedure ust-processing(FSM，SS，TR)
Flag=True，ustset=∅;
For ∀s ∈ SS Do /* S for symptoms */
While Si≠Sj or t(Si) ≠t(Sj) Do
/* Symptoms or symptoms corresponding to the conversion is not the same */
Flag=False;
Exit;
End
ust←t(s); /* T (s) is the corresponding conversion of the symptoms */
For ∀tc ∈ TS-s and t ∈ TR Do
While t=ust Do
Flag=False;
Exit;
End
If（Flag=True）
ustset=｛ust｝

In addition, if the output is empty error, suspend conversion that step corresponds to the same conversion, the output is empty.

The original algorithm for all use cases are verified conversion and all the output results are checked, this step, the use case is divided into use cases with symptoms and use cases without symptoms, is conducive to the type of error type, and in the verification of whether the output error, Just traverse all use cases with symptoms, without traversing all use cases.

(2) To determine the conversion did not perform the error

Using forward analysis, if the conversion does not perform an error, the initial symptoms of the symptom with the use case are the same as the conversion and the output must be "-". Therefore, the algorithm should first determine the use case with symptoms, the initial symptoms are output is empty and the corresponding conversion t_k is the same, otherwise the error may be excluded, it is to continue. In a reverse analysis, there is no suspicious conversion in a symptom-free use case, if the conversion is not performed, a symptom is generated. Then the algorithm should be in the case with no symptoms to determine whether the existence of the t_k, if there is to exclude the error may be, there is no to continue. Finally, the t_k into the use case with symptoms, from the initial symptoms back to verify, the output is consistent with the actual, then record the t_k, otherwise it will be excluded.

Since forward analysis and reverse analysis only exclude nonconformities, in the final verification itself, including the compliance verification that is to ensure the correctness of the algorithm. Since no previous part of the proposed algorithm, so no need for efficiency analysis.

(3) To determine the conversion error

To determine the conversion error, not only to determine the wrong conversion, but also to determine the wrong state changed, the wrong end state. We have the initial diagnostic set ITC, in order to solve the suspicious conversion and suspicious end state too much time complexity of the problem, we first need to ITC about reduced. After narrowing the

suspicious transformation and the corresponding set of end states, the possible errors are substituted into the state machine model, and the scope of validation should be verified in all use cases that contain the suspicious conversion errors, this is because the conversion error does not necessarily produce symptoms.

The main work of this algorithm is to optimize the set of suspicious states. The optimization process has been expressed in the algorithm analysis. The latter part of the algorithm is verified from the position where the suspicious conversion of the suspicious conversion case is used for the first time. If the output is the same as the actual output, the possible error information is recorded and the possibility is excluded if it is not consistent. This verifies the correctness of the algorithm after verifying the reduced set of suspicious conversions, and also improves the efficiency.

Determine the conversion of redundant errors the specific algorithm is described as follows:

TABLE II. TRANSFORM THE REDUNDANT ERROR DETERMINATION ALGORITHM

Algorithm input: state machine FSM, initial candidate set ITC Algorithm Output: Final Candidate Set FTC, Error End State Endstate, Error Output Output

Procedure NeedlessTr-processing(FSM，ITC)
For all tk∈ITC and tk∈T- Do
Flag=True
For all tc and tk∈tc Do
See whether tk first appears when the output is equal;
If not equal, will be excluded tk, check the next suspicious conversion;
If equal and empty, the implementation of the algorithm TranFault-processing（FSM，tk）;
If（Flag=True）Then
FTC= FTC∪{tk};
Endstatek= Endstatek∪｛s｝;
If it is equal and the output is ok, check to check if it contains tk;
If included, will be tk excluded;
If not included, then o(tk) ←o;
Execute the algorithm TranFault-processing（FSM，tk）;
If（Flag=True）Then
FTC= FTC∪｛tk｝;
Outputk=ok ;
Endstatek= Endstatek∪｛s｝;
End
End

The algorithm first eliminates the case where the suspicious conversions are not equal for the first time according to the wrong features, and then the output is empty and non-empty discussion, which ensures the integrity of the discussion scope. In this way, after determining the suspicious conversion type, call the conversion error decision algorithm, find suspicious end state. Since the extra conversion must have been generated by the conversion of the state machine, and the excess conversion is to ensure that the state machine is defined, the state has an output of up to one input, so the excess conversion must be from the "T-" Change from.

C. Identification error

If the result is not unique after the above steps, the following processing is performed.

If there is an output error or the conversion is not possible, the possible error conversion is unique. And for conversion errors and conversion of redundant errors, the final candidate error may not be unique. Moreover, there may be a possibility of another type of error if there is one type of error at the same time. So the result is not unique, indicating that in a single error assumptions and existing test cases, the possibility of a number of errors, the next need to take the initiative to test to increase the test cases used to identify these possible, and thus locate the only mistake. The method of generating the use case in this step is only used to verify whether the error is the only error of the target state machine, and the idea of generating the algorithm using the test case.

For the four types of errors, we divide the error into two categories:

(1) A class is wrong to convert that step to produce symptoms, the output error, the conversion of the implementation of the error and the excess error of the second case belong to this type of error;

(2) The other is the wrong conversion after a step conversion to produce symptoms, conversion errors and conversion of the wrong case of the first case belong to this type of error, and the conversion of the first case of excess error we also classified as a conversion error.

When verifying the first type of suspect error t_k, the method of designing the use case should start from the initial state and find a conversion sequence that arrives at t_k, so that the path does not go through other suspect error conversions. When such a path is found, the corresponding use case is generated. As the entire path only t_k is a suspicious error conversion, so it is based on t_k output can determine whether the wrong tk. Test the use case to see the output, if consistent with the expected, then the suspicious error is not an error, the error may be removed; if inconsistent with the expected, the diagnosis of the error is the only error.

When the path to reach t_k will pass through other suspicious conversion t_n, regardless of t_n belong to the first type of error or the second type of error, according to the principle of the order of the path, from front to back in turn verify the suspicious conversion, verify the conversion method is as follows.

For the second type of suspicious error verification, that is, verify the conversion error, assuming that the error may be t_k: $S_i \xrightarrow{x/y} S_k$ error into $S_i \xrightarrow{x/y} S_j$, then the next to determine whether the conversion is wrong, we designed the use of cases divided into two steps, the first step is Generates a transition sequence TR1 from the initial state to the state, requiring that the path of the sequence is not subjected to other suspicious conversions. If you need to go through other suspicious conversions, a also in accordance with the order of succession in turn. The next step is to find out the different transformations (or conversion sequences) TR2 of S_k and S_j's next output at the same input, called "difference conversion (sequence)", which draws on the UIO method and DS generated by the test case Method of thinking.

Similarly, the process of finding TR2 also requires that the path of the sequence does not go through other suspicious conversions. After finding such a TR2, a use case is generated with the TR1 generated in the first step. For the use case test, if the output is inconsistent with the expected, then the final positioning of the error, if consistent, then the possibility of excluding the error.

If you cannot find such a TR2, try to verify other possible errors to narrow the error range. If all errors are handled and cannot be verified at all, the given diagnostic result cannot determine the unique error, giving the remaining error after the step may be the result of the diagnosis.

Here we have the method proposed in this chapter to make an error diagnosis of the instance.

(1) First of all, the state machine and test sets of results pretreatment. The expected conversion sequence for the use case, the expected output and the actual output are shown in the following table:

TABLE III. EXPECTED CONVERSION SEQUENCE FOR USE CASES, EXPECTED OUTPUT AND ACTUAL OUTPUT

TS	Expect transfer seq	Exp output	Obs output
tc1:abcbab	t2,t4,t3,t1,t16,t9	1121-1	1121-1
tc2:acbbca	t2,t3,t1,t9,t17,t13	1211-1	1211-1
tc3:bacbac	t1,t16,t10,t11,t7,t10	1-1111	1- 111-
tc4:bcbaac	t1,t10,t11,t7,t16,t10	1111-1	11111 -
tc5:cbbaca	t15,t1,t9,t13,t17,t13	- 111-1	- 111-1
tc6:cbaacb	t15,t1,t16,t16,t10,t11	- 1 - -11	- 1 - - 11

(2) Symptom corresponds to the conversion is not unique, it is not the output error, tc3 initial symptoms corresponding to t10, tc4 initial symptoms corresponding to the conversion to t16, inconsistent, it is not a conversion error.

(3)Then verify the conversion error. (Tc3) = {t1, t16, t10, t11, t7}, CS (tc4) = {t1, t10, t11, t7, t16}, it is impossible to convert t16 and take the intersection of two, ITC = {t1 , T10, t11, t7}. First analysis t1, t1 in the TS in the next step is converted to a / -, c / 1 and b / 1. H (a / -) = {s2, s5}, H (c / 1) = {s2, s3}, H (b / 1) = {s0, s1, s2, s5} (T1) = s2, suspicious end set is empty, so t1 removed from the ITC. And then analyze t10, t10 in the next step in the TS converted to b / 1, H (b / 1) = {s0, s1, s2, s5}, remove E (t10) = s5, suspicious end state S = {s0, s1 , S2}, respectively, replace s0, s1 and s2 t10, into the use cases with t10 tc3, tc4 and tc6, verify from s0, s1, s2 as a starting point, input bbc, baac and cb were 11- , 111- and 11. Found that only s2 replaced t10 situation to meet. So FTC10 = {t10}, EndStates10 = {s2}. Then, t11 and t11 are transformed into a / 1, H (a / 1) = {s0, s1, s3, s4} in the next step in TS, and E (t11) = s3 is removed, and the suspicious end state S = {s0, s1 , S4}, respectively, s0, s1 and s4 replaced t11 end state, into the use of t11 with the use cases tc3 and tc4, verify from s0, s1 and s4 as the starting point, input ac and aac were 1 and 11-. Found that only s4 replaced t11 situation to meet. So FTC11 = {t11}, EndStates11 = {s4}. Analysis t7, t7 The next step in TS is converted to a / - and c / -, H (a / -) = {s2, s5}, H (c / -) = {s0, s4}) = S2, take the intersection of suspicious end state S is empty. Remove t7 from ITC.

III. CONCLUDING REMARKS

In this paper, under the assumption of a single error, four error models are proposed on a single FSM state machine model, output errors, conversion errors, conversion errors and conversion errors, and a set of improved error diagnosis methods and algorithms , The method of error determination based on forward analysis and inverse analysis is proposed. The algorithm simplifies the processing step and improves the efficiency compared with the existing algorithm. In addition, the detailed test method is put forward.

The next step can be divided into three aspects, one is in more state machine model to study its error diagnosis method, on the other hand is to further enrich the error model, for example, to increase the state of excess errors or research multiple The wrong method of diagnosis, and on the other hand is the combination of the method and the actual project, the algorithm of this code for mapping, for the actual test.

Acknowledgment

This work is funded by National natural Science Foundation of China (No.61402131); China Postdoctoral Science Foundation Special Fund (2016T90293); China Postdoctoral Science Foundation Item (No.2014M551245); Heilongjiang Provincial Postdoctoral Science Foundation Item (No.LBH-Z13105); Central University Basic Scientific Research Business Expence Special Fund (No.HIT.NSRIF.201651); Harbin Science and Technology Innovator Special Fund (2015RAQXJ047).

References

[1] Author list, paper title, journal name, vol. no. pages-, year

[2] Wagner F, Schmuki R, Wagner T, et al. Modeling software with finite state machines: a practical approach [M]. CRC Press, 2006.

[3] Barkalov A. Principles of optimization of logic circuit of Moore FSM [J]. Cybernetics and System Analysis (1), 1998: 65-72.

[4] Barkalov A, Titarenko L. Logic synthesis for FSM-based control units [M]. Berlin: Springer, 2009.