Journal of Computers Vol. 31 No. 6, 2020, pp. 210-229
doi:10.3966/199115992020123106017

Research on Formal Verification Method of Embedded Software Requirements Analysis Document

Yingjie Wang[1,2,3], Kuanjiu Zhou[1,3*], Xudong Zhang[4], Bin Li[2],
Zhui Sun[2], Mingchu Li[1,3], Jie Pan[1]

[1] Software School, Dalian University of Technology, Dalian 116620, China
wyj_dut@163.com, zhoukj@dlut.edu.cn, mingchul@dlut.edu.cn, 472811776@qq.com

[2] College of Information Engineering, Dalian University, Dalian 116622, China
839732666@qq.com, 1692959384@qq.com

[3] Liaoning Provincial Key Laboratory of Ubiquitous Network and Service Software, Dalian 116620, China

[4] Liaoning Hongyanhe Nuclear Power CO., Ltd, Dalian 116001, China
zhangxudong@cgnpc.com.cn

Received 17 December 2019; Accepted 17 December 2019

Abstract. Software requirement analysis and design documentation are important basis for software development. The documentation quality does directly affect the quality of software development in the subsequent stage. At present, the requirement analysis and design documents are described by a natural language. These documents are usually reviewed by the document walk-through method. The illegibility and ambiguity of the text expression may cause problems such as missing requirements and logical contradictions. This paper introduces the advantages and modeling process of formalized modeling method -- State Transition Matrix (STM) in detail, and proposes to use the Extended State Transition Matrix (ESTM) formalized method to model the requirements analysis and system design documents. According to the transition feature performs symbolic coding and formal description of the properties, the missing requirements and logical contradictions would be discovered in the design document, test cases and testing code could also be generated. The experimental part takes the train control system test as an example, utilized the formal verification method, judged whether there are logic errors in the document. The result shows that the method can better solve a part of logic problems and requirement vulnerabilities that cannot be found by the traditional code test method.

Keywords: formal verification, linear temporal logic (LTL), requirements analysis document, state transition matrix (STM)

1 Introduction

Embedded control software is widely used in security systems such as modern rail transit, aviation and aerospace systems. Therefore, it is very important to ensure the accuracy of its embedded control software. Formal modeling and verification and software testing adopted in large-scale software development are two effective methods to ensure software quality. The common means of ensuring software quality is to thoroughly test the software. Traditional software testing can find many problems in the software, but it is difficult to find all hidden defects. The main reason is that requirements analysis documents as test standards are inevitably ambiguous and fuzzy due to the use of natural language description [1]. In addition, the use of document walk-through methods, not only can not guarantee its correctness, but also introduce new errors due to understanding bias. Therefore, more and more people choose to use formal methods to describe software requirements and logical structures in the software

[*] Corresponding Author

development process, and use model detection [2-3] and other technical means to ensure software logic correctness. Although this method can effectively guarantee the reliability of software, but for large software design, there are also defects such as modeling difficulty and state space explosion [4], which fails to achieve the effect of simple and easy to use.

Software testing and verification is the basic and most important means to ensure software correctness and improve software reliability [5]. Software testing includes testing of the program code and testing of the document [6], so the quality level of the program and the document together determine the quality of the entire software. The key work in testing is to make sure the documentation is correct and generate valid test cases from the documentation. The test of the program is based on the program source code to design test cases, and the coverage is judged whether the test is sufficient; The test of the document is based on the requirements design specification to generate test cases, by determining whether the demand conditions are fully tested as the termination conditions for the test completion. This makes the testing of the document somewhat limited due to the inability of the document to run. Usually, the document is tested by means of meeting review. The method can effectively find the description errors, data errors, and so an in the document. However, it is difficult to find the logic class between the modules of the document, especially in the case of large document size and complicated structure, such as the design of the demanding column control system.

Wang [7] proposed a document testing method for different error types by analyzing the Bug types of user class documents; Ji et al. [8] proposed a method to measure the quality of software documents by using quality measurement model and comprehensive evaluation model, aiming at the shortcomings of manual inspection; Chen et al. [9] implemented different software document strategies according to different process characteristics in the software life cycle, so as to improve the quality of software documents; Yan et al. [10] proposed the concept of document state, and managed the document process by using document roles, document activities and inter-document dependencies based on document state, so as to effectively guarantee software quality; Huang [11] et al. proposed a formal engineering method for the modeling of airborne control software requirements. Based on the formal method and the basic principles of software requirements engineering, the engineering staff is guided to complete the construction of the requirements specification from the original requirements and the evolutionary process. In order to confirm the accuracy of the software requirements specification, this method gives a graph-based static review and model-based dynamic simulation technology.

Software credibility has been an urgent and important issue, but the evaluation of software credibility has not been a systematic and objective standard. Wang [12] discusses the influence of process entity, process behavior and process products on the credibility of software products from the perspective of software development process. And establish a software process credibility model based on evidence covering the entire life cycle of the software. A credible evaluation method for software processes based on this model is given. Usually in the requirements verification phase, people pay attention to the correctness, security, integrity and reliability of the demand model. The common methods to verify the reliability and security of system requirement specification are review, simulation, test and formal verification. The formal method is the most reliable, but more complex. The reliability of the method based on review mainly depends on the experience of the reviewer, and the reliability guarantee degree is the lowest. The reliability of the simulation and test methods is in the middle segment [13].

In this paper, the formal method is applied to document validation of software requirement analysis. This paper designs an embedded software requirement analysis document formal engineering modeling method to realize the modeling process from natural language requirement to formal demand model evolution. An extended state transition matrix (ESTM) formalization method is proposed to model the requirements analysis and system design documents. The variable dependencies and state transition relationships are extracted from the formalized demand specification, and the formalized description of symbolic coding and properties according to the migration characteristics of the model is presented in tabular form. Thus, the missing requirements and logical contradictions in the design document are found.

2 Model Description

Everything in the world is connected, and so are user needs and product knowledge. This paper starts with the association network (knowledge map) that builds the system requirements knowledge, and builds the user demand knowledge map. It is possible to link the internal and external relationships between demand knowledge, requirements and product knowledge and visually visualize their relationships. This makes it easier to understand user needs in a more comprehensive and macro-level [14], and to identify missing and logical problems as early as possible.

2.1 Formal Method Selection

As an example of demand analysis, according to the requirements, extract scope, trigger events, behavior, and next state (SEAT), deploy in the state transition matrix, and find out the missing parts of the system requirements and requirements.

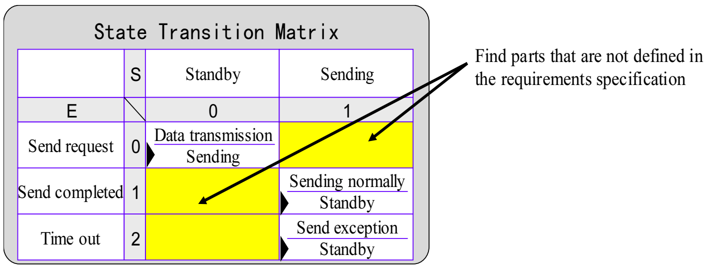
The task specifications are as follows:

(1) According to the delivery request of Host, the message of the delivery request is conveyed.

(2) Delivery completed, inform Host delivery successful.

(3) If a timeout notification occurs before the delivery is completed, the Host is notified that the delivery is abnomal.

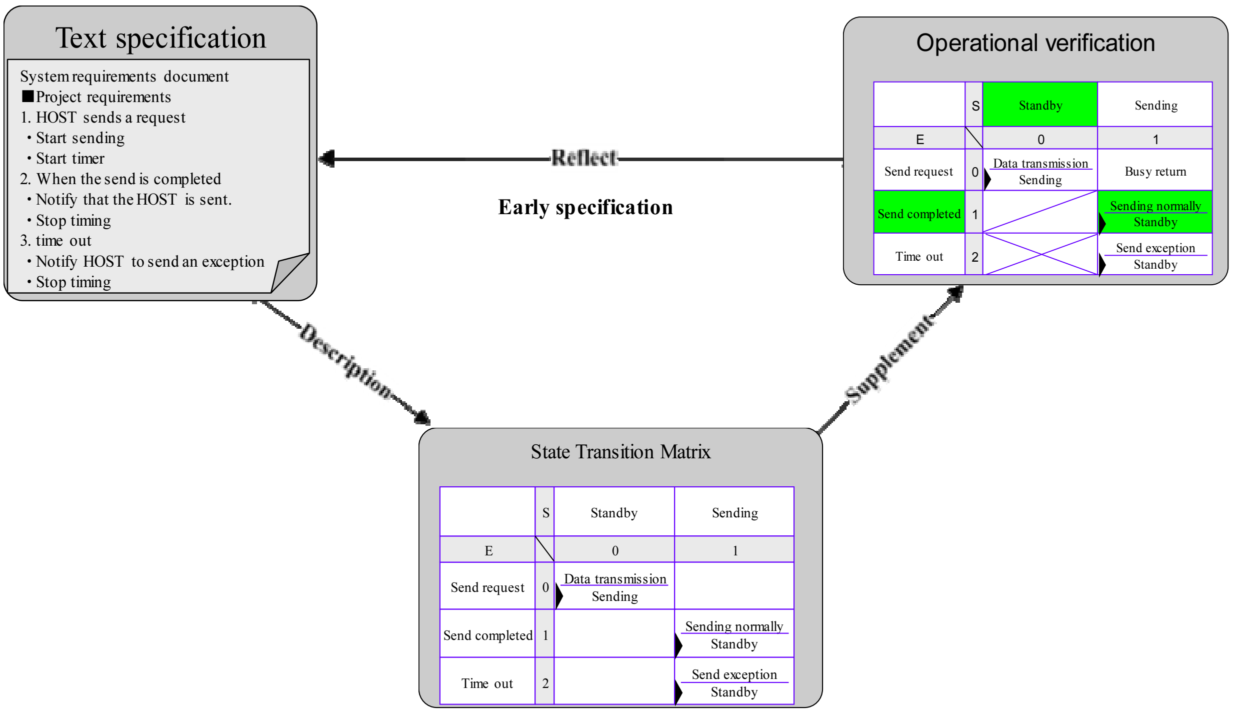
STM can discover parts that are not defined in the specification, discover the scope of system requirements, missing parts of the requirements, and consider verifying early leaks. As shown in Fig. 1.


State Transition Matrix
Find parts that are not defined in the requirements specification

	S	Standby	Sending
E		0	1
Send request	0	Data transmission Sending	🟨
Send completed	1	🟨	Sending normally Standby
Time out	2	🟨	Send exception Standby

Fig. 1. Uses the state transition matrix to find missing requirements

Through the state transition matrix we confirm the logic of each cell and verify that the logic of each blank cell is not required or is vulnerable. A cell is found to have a vulnerability. When the data is being sent, the request to send the data again will prompt to be busy and return; There are two cell contents that are not needed. As shown in Fig. 2.



System requirements document
■Project requirements
1. HOST sends a request
・Start sending
・Start timer
2. When the send is completed
・Notify that the HOST is sent.
・Stop timing
3. time out
・Notify HOST to send an exception
・Stop timing

	S	Standby	Sending
E		0	1
Send request	0	Data transmission Sending	Busy return
Send completed	1		Sending normally Standby
Time out	2		Send exception Standby

	S	Standby	Sending
E		0	1
Send request	0	Data transmission Sending	
Send completed	1		Sending normally Standby
Time out	2		Send exception Standby

Fig. 2. Discovering the requirements vulnerability, supplemental requirements text specification process

The above requirements analysis has some omissions, such as the use of flowcharts, state transition diagrams and sequence diagrams in Fig. 3 have not been found. However, these vulnerabilities can be easily found by using the state transition matrix proposed in this paper. Therefore, the formalization of the state transition matrix in the system requirements analysis phase can detect logic problems early and reduce the software testing cost.

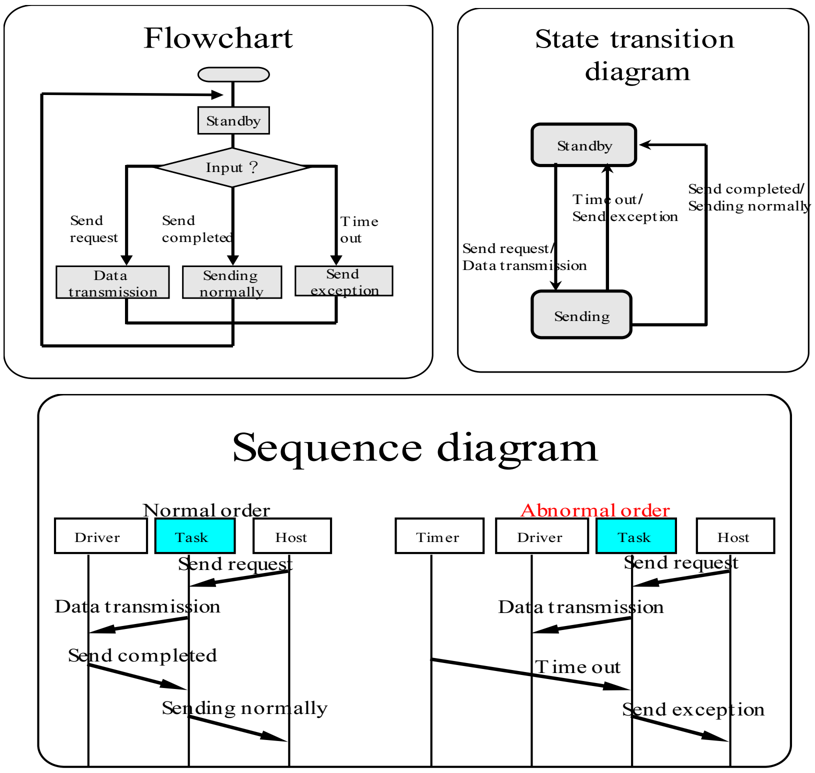


Fig. 3. No omissions were found in flowchart, state transition diagram and sequence diagram

2.2 State Migration Matrix (STM) Basic Structure

The basic structure of the ZIPC model is a two-dimensional table consisting of event, state, and action processing, which is the state transition matrix. The state transition matrix (STM) can be defined as a triple <S, E, C>, where S represents a state set, E represents an event set, and C is a set of action processing units. The action processing unit includes three types: a normal unit, a negligible unit “/”, and an unreachable unit “×”. In the STM (shown in Fig. 4), the row indicates the state that exists in the software; the column indicates the event that will occur in the software; The row and column intersection unit indicates the processing that the software needs to perform when an event occurs in a certain state. Processing consists of two parts: one is normal transaction processing, and the other is software state migration.

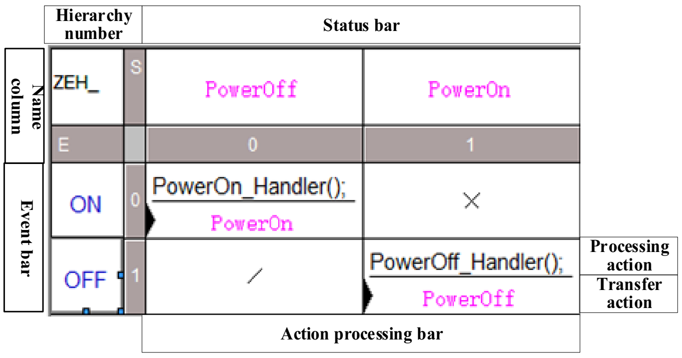


Fig. 4. STM structure

The STM structure not only preserves the formal semantics of the software model, but also facilitates subsequent software development (Software requirements and design can be done through STM and the software code framework can be generated automatically). STM supports hierarchical design that breaks down large software into subsystems. Each subsystem consists of several STM tables, and each table can communicate via mechanisms such as shared variables, messaging, or table calls. In addition, STM can explicitly represent software design flaws. (Use the “Error (×)” unit and the “Ignore (/)”unit in Fig. 4. For example, in the PowerOn state, what happens when the ON event occurs; in the PowerOff state, what happens when the OFF event occurs.) These defects have a great impact on software security, and it is difficult to be effectively discovered through traditional software modeling methods.

2.3 An Extended State Transition Matrix (ESTM)

The basic state transition matrix (STM) structure is too simple to solve complex practical problems, so STM needs to be expanded. The expansion method is as follows:

The extended state transition matrix ESTM is defined as a quintuple, H=(S, S_0, L, E, T):

- S is a finite state set consisting of an atomic state and a group state, and has only a single activation state at any given time. There are two cases in S:

(1) Atomic state: does not include other states;

(2) Group status: group status can include atomic status and group status. A group state is a sequence of states consisting of an atomic state and a group state.

Let S be an $n(n \geq 0)$ -order state tree. If one node in T has a 0 indegree and the other nodes have an indegree of 1, then the node is called the initial state S_0. Nodes with an input degree of 1 and an output degree of 0 are called atomic states, and nodes with an input degree of 1 and an output degree of not 0 are called group states. {L | indegree, outdegree} Indicates the hierarchical relationship between states.

- S_0 is the initial state of ESTM, and can belong to the atomic state or to the group state.
- E is a finite event set. Each event has a unique id identifier. Events are the factors that trigger the processing. They can be divided into four types: variables, interrupts, inmail, and function calls.
- T is the cell type, including the normal unit T_{normal}, the ignorable unit T_{ignore} and the error unit T_{errorr}:

(1) The normal unit $T_N \in T_{normal}$ is a quintuple (s, e, g, a, s’), where source(T_N)=s is the state corresponding to T_N, event(TN)=e is the event that triggers TN, guards(TN)=g is the enabling condition and the cell transfer function, actions(T_N)=a is the performed action, target(T_N)=s’ is the migration target state. The unit T_N specifies that when H is in the state source(T_N) and an event(T_N) arrives, when the enabling condition in guards(T_N) is satisfied, the action behavior of H is actions(T_N).

(2) The ignorable unit $T_i \in T_{ignore}$ is a triple (s, e, /), where s, e is defined the same as T_N, and the symbol “/” means blank.

(3) The error unit $T_e \in T_{error}$ is a triple (s, e, ×), where s, e is defined the same as T_N, and the symbol “×” indicates an error has occurred.

2.4 Introducing Knowledge Graph in ESTM

The Knowledge Graph is a graph-based data structure consisting of a node and an edge. In the Knowledge Graph, each node represents the “entity” existing in the real world. Each edge is the “relationship” between the entity and the entity, and the entity and relationship have their own “attribute”. Entities, relationships, and attributes form the core three elements of the knowledge map. Knowledge Graph are the most effective representation of relationships. In another words, the Knowledge Graph is a network of relationships that combines all kinds of different information (Heterogeneous Information). The Knowledge Graph provide the ability to analyze problems from the perspective of “relationships”.

In the ESTM, the state changes with the acceptance event or internal processing changes, so the knowledge map can be introduced to reconstruct the STM. The state name and function are shown in the Fig. 5.

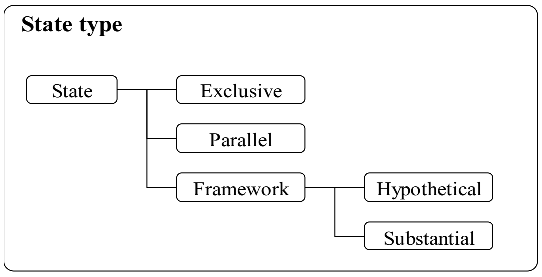


Fig. 5. Status Name and Function Diagram

・**State relationship definition:**

(1) Activation: The initial state is activated by a series of transitions to a state. Both the parent state and the child state can be activated. After the parent state is activated, the child state can be disabled, but the child state must be activated to activate the parent state.

(2) Exclusive: There are $S_1, S_2, \cdots\cdots, S_n$ in the system, a total of n brother states. $\exists 1\ \text{active}(S_1) = true(i \le n)$, other state $\text{active}(S_j) = false(j \ne i,\ j \le n)$. These states are mutually exclusive, and there can only be one active state at a time.

(3) Parallel: There are $S_1, S_2, \cdots\cdots, S_n$ in the system, a total of n brother states. $\text{active}(S_i) \wedge \text{active}(S_j) \wedge \cdots \wedge \text{active}(S_k) = true(i \ne j \ne k, i \le n,\ j \le n,\ k \le n)$. Multiple states of a sibling relationship can be represented as active at the same time;

(4) Hypothetical state framework: The parent state does not have a state number and does not have an action sequence, but performs a substate action through an event;

(5) Substantial state framework: The parent-child state has a state number at the same time, has an action sequence, and performs an action of the parent-child state through an event.

In Fig. 6, there are two parent states “power OFF” and “power ON”. These two states are exclusive states, and only one state can be activated at the same time; There are two parallel states “Telephone” and “TV”, the two brother states can be activated at the same time, and the following sub-states can accept events together and process them synchronously; In the illusion state frame, the parent state “power ON” does not have an action sequence, but performs a substate action through an event; in the substantive state frame, the parent and child states simultaneously have an action sequence, and the parent and child state actions are performed through the event.

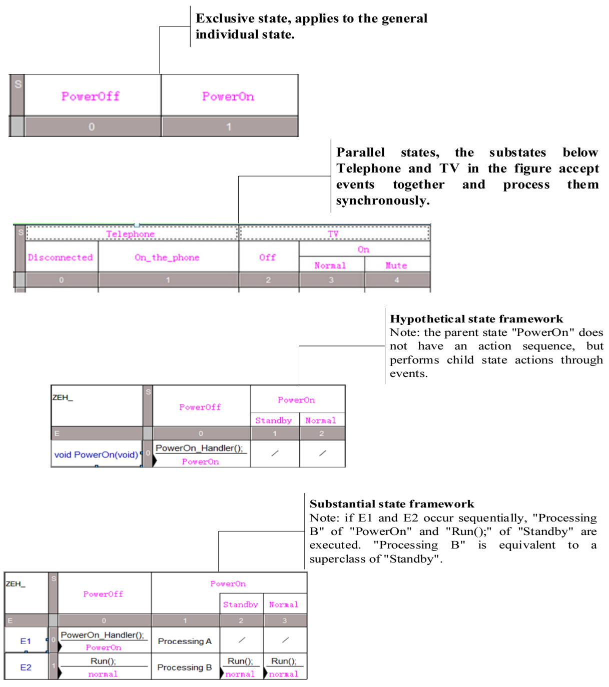

Exclusive state, applies to the general individual state.
S
PowerOff
PowerOn
0
1
Parallel states, the substates below Telephone and TV in the figure accept events together and process them synchronously.
Telephone
TV
Disconnected
On_the_phone
Off
On
Normal
Mute
0
1
2
3
4
Hypothetical state framework
Note: the parent state "PowerOn" does not have an action sequence, but performs child state actions through events.
ZEH_
PowerOff
PowerOn
Standby
Normal
E
void PowerOn(void)
PowerOn_Handler();
PowerOn
Substantial state framework
Note: if E1 and E2 occur sequentially, "Processing B" of "PowerOn" and "Run();" of "Standby" are executed. "Processing B" is equivalent to a superclass of "Standby".
ZEH_
PowerOff
PowerOn
Standby
Normal
E1
E2
PowerOn_Handler();
PowerOn
Processing A
Run();
normal
Processing B
Run();
normal
Run();
normal

Fig. 6. Introducing Knowledge Graph into ESTM

2.5 ESTM Initial Modeling

Modeling with ESTM. First, the requirements document is analyzed, the concept and state hierarchy are extracted, and the state tree is generated. Then, the basic ESTM model is obtained according to the state tree.

(1) Generating a State Tree

Give a concrete example of the functional requirements of the music player. From the abstraction of the concept and hierarchical relationship to generate a state tree, the state tree is shown in Fig. 7.

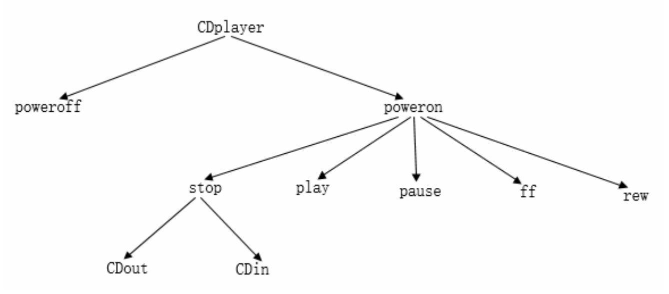


Fig. 7. Status tree of the music player

Music Player Instance - Functional Requirements:

①There is a "Power" button, the initial state of CDPlayer is set to "power off";

②When the "Power" button is pressed, the status of the CDPlayer should be "stop" in the "power on" state. When pressing the "Power" button in the "power on" state, you should switch to "power off";

③The CDPlayer contains a tray that enables CD loading and ejection when the "Eject" button is pressed. (If CDPlayer is playing during pop-up, stop playing and eject the CD.)

④When the "play" button is pressed, the CDPlayer enters the "play" state;

⑤In the "stop" state, only the "Power", "play" and "Eject" buttons are valid, and other buttons are invalid;

⑥There is a "stop" button, when the button is pressed, CDPlayer stops playing;

⑦There is a "pause" button. When the button is pressed, if it is in the "play" state, it will pause playback; press it again to resume playback.

⑧There is a "fast forward" button, fast forward when pressed, and play normally when the button is raised;

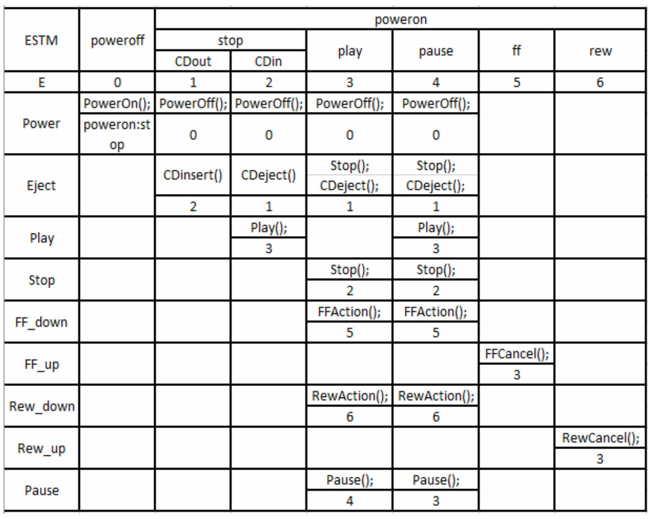
⑨There is a "Rewind" button, which is rewinded when pressed, and plays normally when the button pops up.

(2) Get the ETEM model state hierarchy from the state tree

First of all, it is easy to get the hierarchical relationship between states by the state tree. The abstracted hierarchy and the OR relationship are the stratification of ESTM. The upper level is described in detail, and the lower level is described in detail. Therefore, a complex state tree can be represented by an extended state transition table (ESTM). Hierarchy is divided into event stratification and state stratification.

In event stratification, the summary event is captured in the upper state transition table. The upper state transition table passes the event to the lower state transition table, and the detailed event is captured in the lower state transition table. Build events from summary to detailed relationships, using a basic state transition table to form a huge system. State stratification is called from a state cell. In the state hierarchy ESTM, only the ESTM under the same root can be called, and the ESTM under other roots cannot be called.

Combining the functional requirements of the music player instance with the state tree of Fig. 7, the ESTM diagram of Fig. 8 can be obtained. The hierarchical relationship between states can be clearly seen from the figure. "power on" is the root node of a subtree in the state tree, and the corresponding state becomes the group state in the ESTM. The substates corresponding to it are "stop", "play", "pause", "ff", "rew". The "stop" is also a group state, and its corresponding sub-states are "CDout" and "CDin" respectively. At the same time, using natural language to describe requirements will be ambiguous, which will easily lead to problems such as missing requirements and logical contradictions. After modeling with the extended state transition matrix (ESTM), not only can the above problems be solved, but the requirements can be simplified and clear in the form.



ESTM	poweroff	poweron					
		stop		play	pause	ff	rew
		CDout	CDin				
E	0	1	2	3	4	5	6
Power	PowerOn(); poweron:stop	PowerOff(); 0	PowerOff(); 0	PowerOff(); 0	PowerOff(); 0		
Eject		CDinsert() 2	CDeject() 1	Stop(); CDeject(); 1	Stop(); CDeject(); 1		
Play			Play(); 3		Play(); 3		
Stop				Stop(); 2	Stop(); 2		
FF_down				FFAction(); 5	FFAction(); 5		
FF_up						FFCancel(); 3	
Rew_down				RewAction(); 6	RewAction(); 6		
Rew_up							RewCancel(); 3
Pause				Pause(); 4	Pause(); 3		

Fig. 8. Music player example ESTM diagram

2.6 ESTM Migration Design

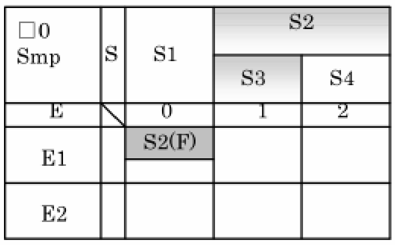
Migration refers to moving from one state to another. It can specify the type of migration when migrating to a parent state with a child state. The type of migration refers to which substate is activated when migrating to the parent state. There are three types of migration: fixed, memorized, and deep-memorized. When transitioning to the status box based on these types, it is determined which of the sub-states of the status box is active. When the type of the migration target state is not set, the default is “deep memory migration”.

(1) Fixed migration type

The fixed migration format is to add “(F)” after specifying the migration target.

Migration target status name (F)

In a fixed migration, the default state is usually set to active. The default state has a state of ▼ or the leftmost state. An example of a fixed migration is shown in Fig. 9. When the “E1” event occurs in the “S1” state, it is migrated to “S2(F)”. When migrating to “S2”, since the sub-states “S3” and “S4” have no ▼ mark, they are migrated from “S2” to “S3”.



□0 Smp	S	S1	S2	
			S3	S4
E	\	0	1	2
E1		S2(F)		
E2				

Fig. 9. Migration example of the fixed migration

(2) Memory migration

The format of the memory migration is to add “(M)” after the specified migration target.

Migration target status name (M)

In the memory type migration, the subordinate first level of the migration target state activates the state of the last activation. Below the second level is the activation default state. That is, the memory type memorizes only the first layer substate immediately following, and the second layer is the same as the fixed type migration operation. An example of a memory migration is shown in Fig. 10. Assume that the current state is “S2-S4-S6”. After the event “E1” occurs, it is migrated to “S1”. Next, if the “E2” event occurs, move to "S2". Since the memory migration is set, the substate of "S2" is migrated to the previously activated "S4". The next substate of "S4" is migrated as a fixed type, so it is migrated to "S5".

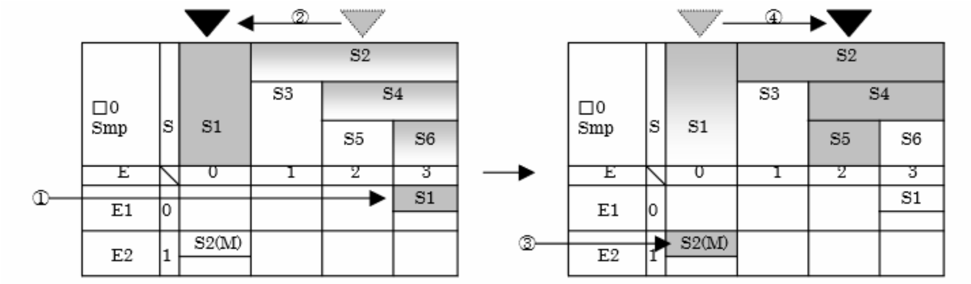


Fig. 10. Migration example of the memory migration

(3) Deep memory migration

The format of deep memory migration is to add "(D)" after the specified migration target.

Migration target status name (D)

In deep memory type migration, all sub-states that were previously activated are activated. Memory-type migration not only memorizes the first layer of sub-states, but all sub-states are memorized. An example of deep memory state transition is shown in Fig. 11. Assume that the current state is "S2-S4-S6", and after the event "E1" occurs, move to "S1". Next, if the "E2" event occurs, move to "S2". Since the deep memory type migration is set, the substate of "S2" migrates to the previously activated "S4". The next sub-state of "S4" is also memorized, so it is migrated to "S6".

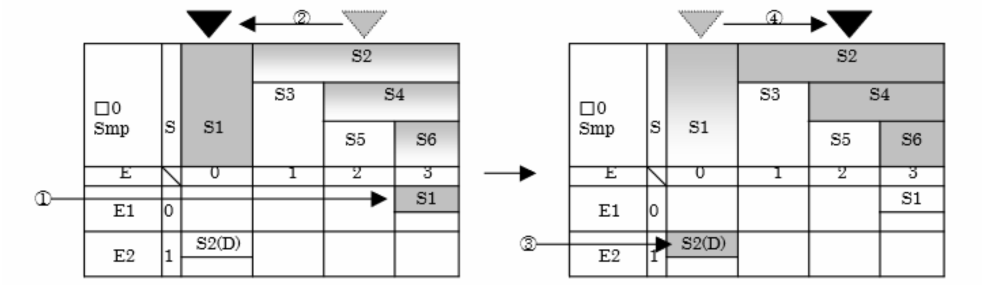


Fig. 11. Migration example of the deep memory migration

2.7 Two Driving Methods of ESTM

The ESTM driver includes two drive modes, an event type (E type) drive mode and a status type (S type) drive mode. The E-type driver is an event-driven, waiting for an event in one place. When the event occurs, the action of the event and the current state intersecting the cell is performed. The S-type drive is the state drive, and the state drive is an event that monitors each state. State driven when there are multiple concurrent states, and event driven when there is no concurrent state.

Event type (E type) drive mode: detects status-independent events, handlers, and condition status. Feature: If no event is detected, processing is not performed. Since all events are treated as monitoring objects, it is impossible to distinguish the event monitoring objects of each state. Therefore, when an unnecessary event is detected in a certain state, even if the condition is satisfied, the process is called an ignore behavior. See Algorithm 1.

Algorithm 1. Event-based migration algorithm

Input: Event

Output: Migration result

```
1. int main() {
2.   stateNo = 0; /* Initialization function */
3.   while (1){
4.  scanf ("%s", Event); /* Insert a monitor system call corresponding to the event wait */
m0if (Event); /* Execution event resolution */}}
void act (void){
 unsigned short i;
 struct Matrix *pmatrix;
 i = eventNo * STATEMAX + stateNo;
 pmatrix = (struct Matrix*)&m0[0][0];
 if (!pmatrix){
    return;}
 if (pmatrix [i].process)
    (pmatrix [i].process)(); /* Processing execution */
 if (pmatrix [i].code == ZNORMAL)
    stateNo = (unsigned short) pmatrix [i].trans; /* Transfer execution */
 else  No-transfer}
void m0if (char * name){
 if (play Button){
   eventNo = 0;
   act();/*ESTM operation function*/
 }else if (STOP Button)){
    eventNo = 1;
    act();/*ESTM operation function */
 }else if (Pause Button)){
    eventNo=2;
    act();/*ESTM operation function */}}
end procedure
```

State type (S type) drive mode: The process is designed according to the activation state. Since the design process is written in the cell, it contains the event detection itself. Therefore, in the case where the processing unit is ignored, the event detection itself is not necessary and it cannot be executed. The state active processing operation continues to be performed, and the monitoring events can be distinguished for each state. See Algorithm 2.

Algorithm 2. State Migration Algorithm

Input: Specific status and events
Output: Migration result

```
int main() {
 scanf ("%d", stateNo) /* Specify the current active state */
 while(1) {
    m0if (stateNo); /* Event detection in a specific state */}}
void m0if (int stateNo){
 if (status 0) {
   scanf("%d", eventNo); /* Enter events in a specific state */
   act();/*ESTM operation function */
 }else if (status 1){
  scanf ("%s", eventNo); /* Enter events in a specific state */
  act();/*ESTM operation function */
 }else if (status 2){
    scanf("%s", eventNo); /* Enter events in a specific state */
    act();/*ESTM operation function */}
void act(void){
 unsigned short i;
 struct Matrix *pmatrix;
 i = eventNo * STATEMAX + stateNo;
 pmatrix = (struct Matrix*)&m0[0][0];
 if(!pmatrix) {
    return;}
 if (pmatrix [i].process)
    (pmatrix [i].process)(); /* Processing execution */
 if (pmatrix [i].code == ZNORMAL)
    stateNo = (unsigned short) pmatrix [i].trans; /* Transfer execution */
 else  No-transfer}
end procedure
```

The difference between ESTM two driving methods: The event-driven mode is called by the event and the current state, and the state-driven mode detects the corresponding event through the state and calls it. The event-driven mode first detects an event, and the event is detected in units of events; The state-driven mode first determines the state, and the event-detected state is the unit. The event-driven mode can be called by the action cell, and the state-driven mode cannot be called by the action cell.

2.8 Proof of Theorem

Theorem 1. Any state diagram can be modeled using ESTM.

(1) When n=1, the state diagram has only one state, and the state diagram is as shown in Fig. 12:

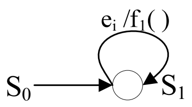
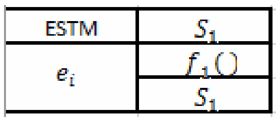


Fig. 12. State diagram with a state

The state diagram of Figure 12 can be modeled using ESTM, as shown in Fig. 13:



ESTM	S_1
e_i	$f_1()$
	S_1

Fig. 13. Modeling Fig. 12 using ESTM

When n=2, there are two states in the state diagram, the situation is relatively simple, as shown in Fig. 14:

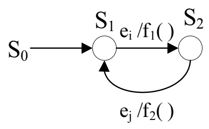
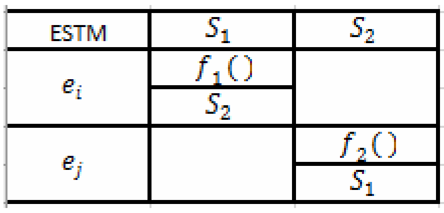


Fig. 14. State diagram with two states

The state diagram of Fig. 14 can be modeled using ESTM, as shown in Fig. 15:



ESTM	S_1	S_2
e_i	$f_1()$	
	S_2	
e_j		$f_2()$
		S_1

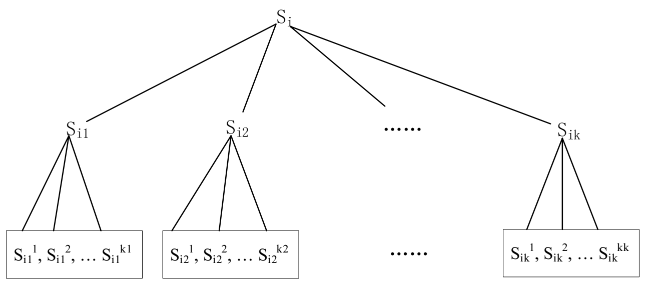
Fig. 15. Modeling Fig. 14 using ESTM

(2) Let n = k be established, that is, for a state diagram G, including K states of $S_1, S_2, \cdots\cdots, S_k$. These states may include atomic states as well as group states, $S = \{S_1, S_2, \cdots\cdots, S_K\}$.

Without loss of generality, here is assumed $S_A = \{S_1, S_2, \cdots\cdots, S_M\} \subset S$. When $\forall s \in S_A$, s is the atomic state;

Assumed $S_G = \{S_{M+1}, S_{M+2}, \cdots\cdots, S_K\}$ when $\forall S \in S_G$, S is the group status.

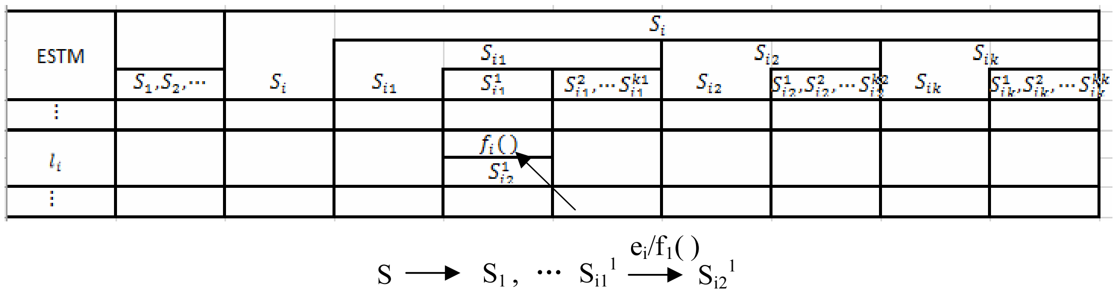
$\forall s \in S_i, S_j \in S_G, S_i$ corresponding to a state set $S_i = \{S_{i1}, S_{i2}, \cdots\cdots, S_{ik}\}$. When $S_i \subset S$, S_j corresponding to a state set $S_j = \{S_{j1}, S_{j2}, \cdots\cdots, S_{jl}\}$. $S_j \subset S$, and $S_i \bigcup S_j = \phi$, that is $\forall S_i \in S$. It can only and at most belong to a group state, so the S state can form a state forest.



S_i
S_{i1}
S_{i2}
……
S_{ik}
S_{i1}^1, S_{i1}^2, … S_{i1}^{k1}
S_{i2}^1, S_{i2}^2, … S_{i2}^{k2}
……
S_{ik}^1, S_{ik}^2, … S_{ik}^{kk}

Fig. 16. Status tree with k states

The ESTM formed by this state tree (see Fig. 16) is shown in Fig. 17.



ESTM
S_i
S_{i1}
S_{i2}
S_{ik}
$S_1, S_2, \cdots$
S_i
S_{i1}
S_{i1}^1
$S_{i1}^2, \cdots S_{i1}^{k1}$
S_{i2}
$S_{i2}^1, S_{i2}^2, \cdots S_{i2}^{k2}$
S_{ik}
$S_{ik}^1, S_{ik}^2, \cdots S_{ik}^{kk}$
⋮
l_i
⋮
$f_i()$
S_{i2}^1
$S \longrightarrow S_1, \cdots S_{i1}^1 \xrightarrow{e_i/f_1(\,)} S_{i2}^1$

Fig. 17. Modeling Fig. 16 using ESTM

(3) Consider k+1 status cases

① In the first case, consider adding a state S_{k+1} to the original state diagram G. If S_{k+1} is not in any group state including the set, S_{k+1} is an independent atomic state, then only one state is added, which is relatively simple.

② In the second case, S_{k+1} belongs to the group state, and there are four cases:

(i) S_{k+1} is the atomic state in the group state, and ESTM is shown in Fig. 18.

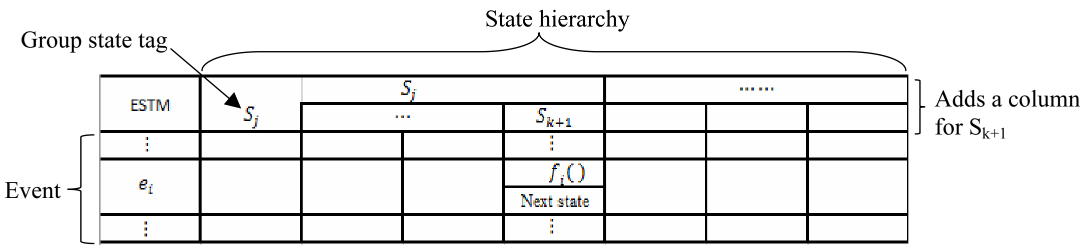
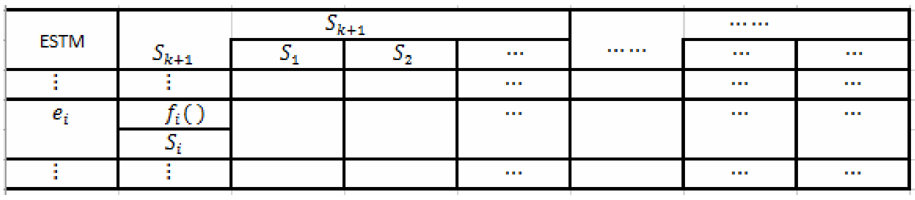


Fig. 18. S_{k+1} is the ESTM diagram of the atomic state in the group state

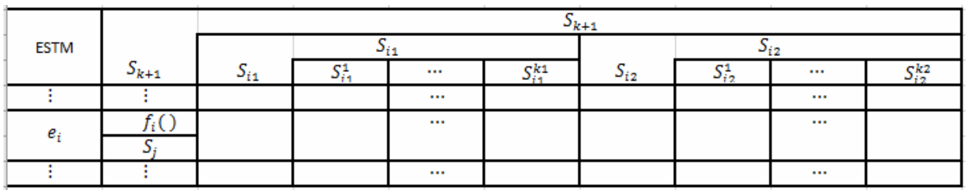
(ii) S_{k+1} is a group state, and its corresponding sub-states are all atomic states, and ESTM is as shown in Fig. 19.



ESTM		S_{k+1}			……	……	
	S_{k+1}	S_1	S_2	…		…	…
⋮	⋮			…		…	…
e_i	$f_i()$			…		…	…
	S_i						
⋮	⋮			…		…	…

Fig. 19. Corresponding substates are all atomic state ETEM diagrams

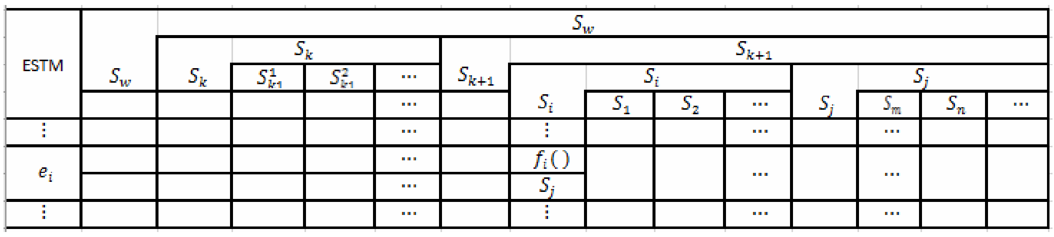
(iii) S_{k+1} is a group state, and its corresponding sub-states are all group state sets, and ESTM is as shown in Fig. 20.



ESTM		S_{k+1}							
			S_{i1}				S_{i2}		
	S_{k+1}	S_{i1}	S_{i1}^1	…	S_{i1}^{k1}	S_{i2}	S_{i2}^1	…	S_{i2}^{k2}
⋮	⋮			…				…	
e_i	$f_i()$			…				…	
	S_j								
⋮	⋮			…				…	

Fig. 20. S_{k+1} corresponding substates are all group state ETEM diagrams

(iv) S_{k+1} is a substate of a group state, and states S_i and S_j are substates of state S_{k+1}. And S_i and S_j are concurrent states, that is, "and" states. ESTM is shown in Fig. 21. The so-called "and" state means that multiple states can be activated simultaneously. There is only one active state in a traditional state machine, but multiple states exist in the actual system at the same time. For example, in the state machine of the car driving process, there are multiple activation states for fueling and gearbox addition and subtraction, and these states are "and" states.



ESTM		S_w												
		S_k				S_{k+1}								
	S_w	S_k	S_{k1}^1	S_{k1}^2	…	S_{k+1}		S_i				S_j		
					…		S_i	S_1	S_2	…	S_j	S_m	S_n	…
⋮					…		⋮			…		…		
e_i					…		$f_i()$			…		…		
					…		S_j							
⋮					…		⋮			…		…		

Fig. 21. S_{k+1} ESMF diagram with substates in concurrent state

In summary, any state diagram can be modeled using ESTM, and the proof is completed.

3 Instance Model

Introducing formalization and model simulation in demand analysis, which can detect missing requirements as early as possible. Traditionally, natural language is used to describe the requirements analysis and system design methods. The reliability of the entire software system depends only on the late code test to find the problem. This approach has many drawbacks and often leads to failure of large software development projects. At present, software design mostly uses UML, Petri nets, flow charts, and timing diagrams for state system analysis and design. However, these models are semi-structured models and do not support subsequent simulation verification and automatic code generation. This paper uses the extended state transition matrix (ESTM) formal modeling method to increase the group state tuples, making the hierarchical relationship between states clearer and describing more complex state relationships. And support for subsequent simulation verification and automatic code generation. Introducing formalization during the system design phase enables early detection of logic problems to reduce software testing costs.

This paper takes the common braking module of the train brake control system document as an example to establish the ESM hierarchical model.

Common brake refers to the brake applied to control the train speed or stop under normal train operation. It is characterized by slow braking and the braking force can be adjusted according to actual needs. The requirements for common brake modules include:

(1) If a module requests a common brake, asks the driver to confirm and does not need to be stationary (or requires static and no driver confirmation), apply the usual brake and notify the driver.

(2) During the application of the usual braking process, if there is no common brake remaining, and the current train speed meets the national value (the common brake is allowed to be relieved immediately), and the common brake is relieved without the remaining emergency braking request, the common brake is removed. All common brake requests and remove text messages sent to the driver.

(3) When the train is in a non-stationary state, if a new braking request is received and the mitigation condition is that the train is stationary, the driver is informed that the normal braking is not allowed to be relieved.

(4) If the brake mitigation condition is for driver confirmation, after the driver confirms the information, the common brake is relieved, all common brake requests are deleted, and all prompts sent to the driver are removed.

(5) If the train is currently at a standstill and there are no remaining common brake requests, the driver is notified to allow the usual brakes to be relieved, after the usual brakes are alleviated, all common brake requests are removed, and the prompt text sent to the driver is removed.

According to the requirements of the common braking modules mentioned above, the state machine shown in Fig. 22 can be obtained, in which S1 represents the non-braking state, S2 represents the braking waiting for the driver to confirm the state, and S3 represents the braking waiting for the train to rest. The state transition condition ST1 indicates that there is a common brake request and the mitigation condition requires driver confirmation. ST2 represents a common brake request and the mitigation condition requires the train to be stationary. ST3, ST4 represent no common braking request and the national value allows for emergency braking to be relieved. ST5 represents the mitigation condition requiring the train to be stationary and the train to be stationary. ST6 represents a common brake request and the mitigation condition is that the train is stationary. The ST7 represents the driver's confirmation of the mitigation of the usual brakes and the driver has confirmed the mitigation. ST8 stands for mitigating common braking conditions requiring the train to be stationary and the train to be stationary, with no other braking requests.

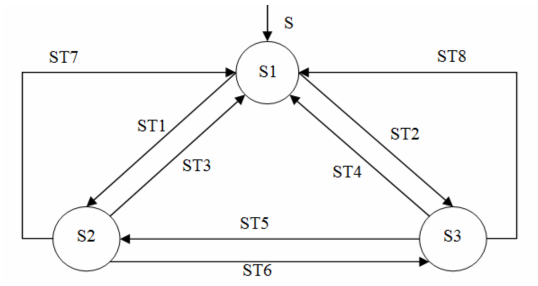
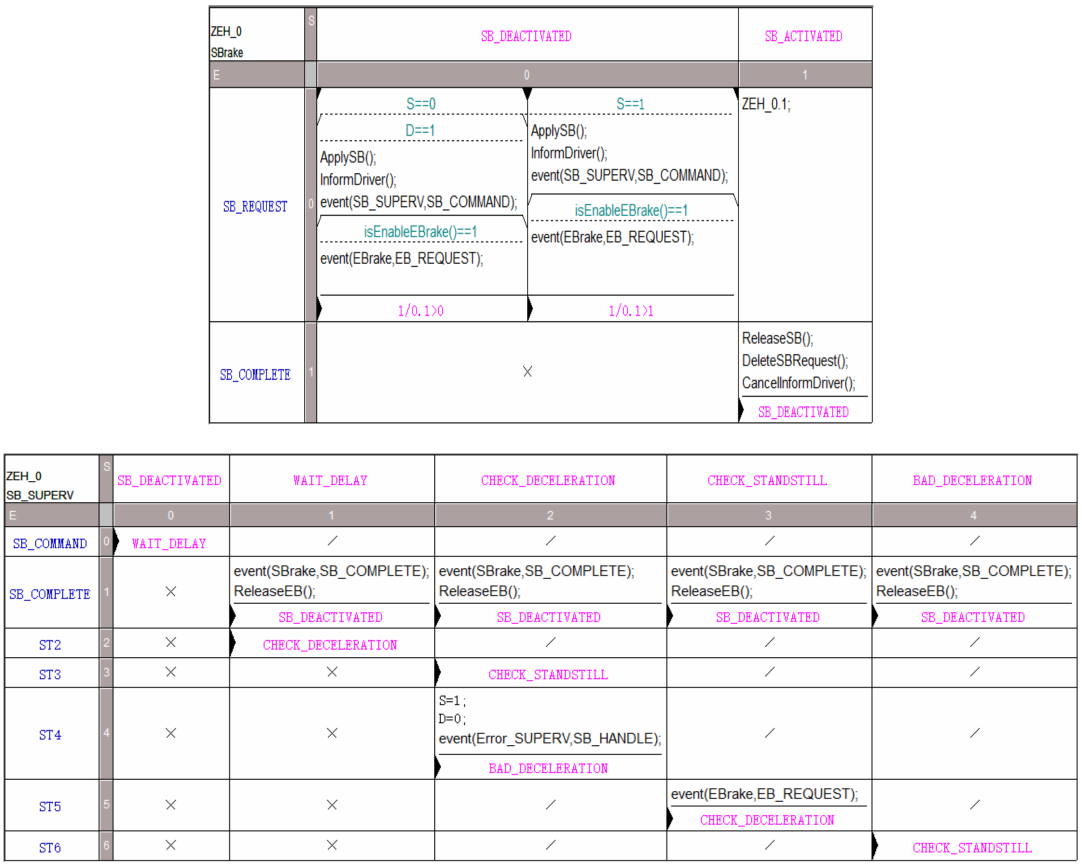


Fig. 22. Common Brake State Machine

The ZIPC model based on the common braking requirements is shown in Fig. 23. The model includes the root table "SBrake" and the child table "ChildSBrake". The subtable is triggered when the root table event "SB_EQUEST" is true and is in the state "SB_ACTIVATED". When the common brake model is in the braking state, the common brake deceleration monitoring model begins to monitor the brake deceleration.



ZEH_0 SBrake	S	SB_DEACTIVATED		SB_ACTIVATED
E		0		1

SB_REQUEST	0	S==0
		D==1
		ApplySB();
		InformDriver();
		event(SB_SUPERV,SB_COMMAND);
		isEnableEBrake()==1
		event(EBrake,EB_REQUEST);
		1/0.1>0
			S==1
			ApplySB();
			InformDriver();
			event(SB_SUPERV,SB_COMMAND);
			isEnableEBrake()==1
			event(EBrake,EB_REQUEST);
			1/0.1>1
				ZEH_0.1;

SB_COMPLETE	1	×		ReleaseSB();
				DeleteSBRequest();
				CancelInformDriver();
				SB_DEACTIVATED

ZEH_0 SB_SUPERV	S	SB_DEACTIVATED	WAIT_DELAY	CHECK_DECELERATION	CHECK_STANDSTILL	BAD_DECELERATION
E		0	1	2	3	4
SB_COMMAND	0	WAIT_DELAY	/	/	/	/

SB_COMPLETE	1	×	event(SBrake,SB_COMPLETE);
			ReleaseEB();
			SB_DEACTIVATED
				event(SBrake,SB_COMPLETE);
				ReleaseEB();
				SB_DEACTIVATED
					event(SBrake,SB_COMPLETE);
					ReleaseEB();
					SB_DEACTIVATED
						event(SBrake,SB_COMPLETE);
						ReleaseEB();
						SB_DEACTIVATED

ST2	2	×	CHECK_DECELERATION	/	/	/
ST3	3	×	×	CHECK_STANDSTILL	/	/

ST4	4	×	×	S=1;
				D=0;
				event(Error_SUPERV,SB_HANDLE);
				BAD_DECELERATION
					/	/

ST5	5	×	×	/	event(EBrake,EB_REQUEST); CHECK_DECELERATION	/
ST6	6	×	×	/	/	CHECK_STANDSTILL

Fig. 23. ZIPC model constraint verification results

4 Characterization and Formal Representation of the Model

4.1 Characterization of Model Properties

The system property is the constraint condition which the system model needs to satisfy. The characterization of properties is usually carried out in a particular logical form. Sequential logic is a kind of logic suitable for characterizing the properties of state transition systems. This logic uses atomic propositions and Boolean operators to form complex expressions to describe the properties of systems. Sequential logic consists of two sub-logic: linear sequential logic LTL and computational tree logic CTL [15]. For software systems, linear temporal sequence logic (LTL) is usually used to represent the change in system behavior caused by the arrival of a specific time or event. The completeness of property characterization is an important problem in the process of describing properties. Model detection technology can effectively verify whether the system satisfies the corresponding properties, but this method cannot guarantee that the characterization of the properties accurately expresses all the properties satisfied by the system to be verified. In this paper, ZIPC tool is used to establish requirements document model and LTL is used to describe the nature of column control documents to be verified. The model and the LTL formula to be verified are input into the solver GarakabuII for verification. Through the analysis, a counterexample is obtained, and the logical problems existing in the document are found.

4.2 Symbolic Coded Representation of Model Properties

Many software testing problems can be attributed to the problem of path-oriented test data generation, and the existence of unreachable path is a difficult problem in path testing. No input data can pass through these unreachable paths. Whether the path is reachable affects the test efficiency, and accurately detecting these unreachable paths can not only reduce the test cost, but also improve the test efficiency [16]. There are two ways to detect unreachable paths: dynamic methods and static methods. The dynamic method generally cannot directly determine the unreachable path. Usually, some means are used in the later stage of the test, such as limiting the depth of the path coverage and the number of searches to determine the unreachable path. Static methods mainly include symbolic valuation methods and methods for detecting branch correlation. The symbolic valuation method uses symbols as input variables to the program, but it is difficult to determine which symbol plays a key role in the result; The method of detecting branch correlation is only for a small number of conditional predicate types, which makes the branch coverage low [17].

This paper uses ZIPC and uses a hierarchical ESTM model to model system requirements documentsA formalized description of symbolic coding and properties based on the migration characteristics of the model. It is proposed to perform state compression by transforming the LTL formula into an automaton, which improves the automation level of document testing to some extent. At the same time, the ZIPC model is used for formal definition and symbolic coding, and it is tested as an input condition.

This paper adopts the symbolic coding and verification method based on BMC method [18-19]. Use the variable b to represent the longest path that the model detects to perform, i.e. the model validation boundary. If the path of the test property in which the negation is established is found within the range of b, the system is considered to be unsatisfied. In the process of symbolic encoding, the variable 0≤r≤b is used to indicate the position of the system execution, x[r] is used to represent the variable to the rth step, and exp[r] is the expression to be executed to the rth step. estmt[r] represents the statement executed to step r.

The constraint relationship between hierarchical ETEM models can be divided into three categories [20]: state reachability relationship, ESTM static constraint relationship and ESTM dynamic constraint relationship. The nature of the three relationships is described as follows:

(1) State reachability relationship

The canonical description of the state reachability relationship can be proved by counterexamples. If you enter the unreachable state of the system, there is a problem with the operation of the model, and the system may have errors. The ESTM model M uses the symbol “x” to indicate the unreachable state. Use $active(M_i, g)$ to indicate an active state of the model M_i, $C_{invalid}$ for unreachable cells, $source(c)$ for the current state, and $event(c)$ for the event that caused the state change. g represents the global reachable state, and the negation property specification of the state can be formally expressed as:

$$\forall c \in M_i \bullet C_{invalid}, \forall g \in G, \ \neg((active(M_i, g) = source(c) \wedge event(c))) \tag{1}$$

(2) Static constraint relationship

The static constraint relationship of a model refers to a state that exists between multiple ESTMs when they are statically related. For example, when M1 is in state s1, M2 is in state s2, and there is a fixed static constraint relationship between the two models. $status(i)$ represents the state of the model i, and this property can be formally described as:

$$\forall g \in G, active(M_i, g) = status(i) \Rightarrow active(M_j, g) = status(j) \tag{2}$$

(3) Dynamic constraint relationship

The dynamic constraint relationship of the model means that when an ESTM model M1 is performing a certain migration action, another ESTM model M2 must also perform a specific migration process. This dynamic constraint property can be formalized as:

$$\begin{gathered}\forall g \in G, active(M_i, g) = status(i) \wedge active(M_i, g') = status(i') \\ \Rightarrow active(M_j, g) = status(j) \wedge active(M_j, g') = status(j')\end{gathered} \tag{3}$$

5 Experiment

5.1 Experimental Environment

The experimental computer configuration is Windows 10 (64-bit) operating system, the 4.00GB memory (RAM), and 2.40 GHz AMD Athlon(tm) II X4 610e Processor. The modeling tool used in the experiment is ZIPC V10. The model verification tool used in this paper is Garakabu II, which is a SMT solver-based model detection tool developed for the ZIPC embedded software CASE tool. Garakabu II can detect pre-set property formulas. Using the LTL formula, the entire verification process can be completed in five phases, and each phase of execution can be performed without warning or error in the previous phase. The software acquires the ESTM model established by the ZIPC CASE tool, and on this basis verifies the input property formula. This further increases the level of automation of model validation.

5.2 Property Verification

Using the most commonly used brake module requirements of the column control system documentation, LTL formalization is used to represent the three constraint relationships of the established ZIPC model.

(1) State reachability relationship

$$!((\mathrm{SB\_COMPLETE}=\mathrm{true})\wedge(\mathrm{statusSBrake}=0)) \tag{4}$$

$$!((\mathrm{ACK}=\mathrm{true})\wedge(\mathrm{statusChildSBrake}=1)) \tag{5}$$

$$!((\mathrm{TIMEOUT}=\mathrm{true})\wedge(\mathrm{statusChildSBrake}=0)) \tag{6}$$

(2) tatic constraint relationship

$$[G]((\mathrm{SBrake}==1)=>(\mathrm{SB\_SUPERV}!=0)) \tag{7}$$

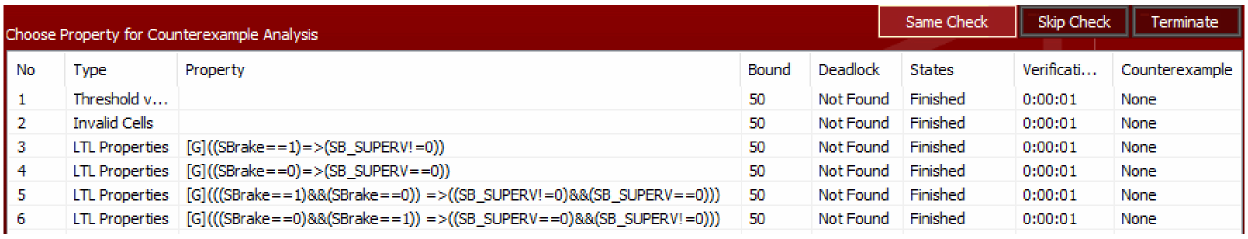
$$[G]((\mathrm{SBrake}==0)=>(\mathrm{SB\_SUPERV}==0)) \tag{8}$$

(3) Dynamic constraint relationship

$$[G](((\mathrm{SBrake}==0)\&\&(\mathrm{SBrake}==1))=>((\mathrm{SB\_SUPERV}==0)\&\&(\mathrm{SB\_SUPERV}!=0))) \tag{9}$$

$$[G](((\mathrm{SBrake}==1)\&\&(\mathrm{SBrake}==0)) => ((\mathrm{SB\_SUPERV}!=0)\&\&(\mathrm{SB\_SUPERV}==0))) \tag{10}$$

The above properties were verified using the model detection tool Garakabu II. Since Garakabu II can automatically verify the unreachable nature of the ZIPC model, there is no need to repeat the formula. The verification results for each property are shown in Fig. 24. Among them, the Invalid Cells term is the unvalidated constraint property of automatic verification; bound is the boundary value detected by the model; Verification time is the model detection elapsed time; Counterexample indicates whether there is a counterexample path. If there is a counter-example path, the step size of the execution of the counter-example path is displayed.



No	Type	Property	Bound	Deadlock	States	Verificati...	Counterexample
1	Threshold v...		50	Not Found	Finished	0:00:01	None
2	Invalid Cells		50	Not Found	Finished	0:00:01	None
3	LTL Properties	[G]((SBrake==1)=>(SB_SUPERV!=0))	50	Not Found	Finished	0:00:01	None
4	LTL Properties	[G]((SBrake==0)=>(SB_SUPERV==0))	50	Not Found	Finished	0:00:01	None
5	LTL Properties	[G](((SBrake==1)&&(SBrake==0)) =>((SB_SUPERV!=0)&&(SB_SUPERV==0)))	50	Not Found	Finished	0:00:01	None
6	LTL Properties	[G](((SBrake==0)&&(SBrake==1)) =>((SB_SUPERV==0)&&(SB_SUPERV!=0)))	50	Not Found	Finished	0:00:01	None

Fig. 24. Common brake system ESTM model

Fully testing the listed documents requires verification of all required conditions. Taking the common brake module as an example, the LTL formula corresponding to the demand condition is as follows:

$$[G](((\mathrm{SB\_REQUEST}==\mathrm{true})\&\&((D==1)\&\&(S==0)))=>[F]((\mathrm{ApplySB}==\mathrm{true})\&\& (\mathrm{InformDriver}==\mathrm{true}))) \tag{11}$$

$$[G](((\mathrm{SB\_NATIONAL\_VALE}==\mathrm{true})\&\&(\mathrm{SB\_REQUEST}==\mathrm{false}))=>[F](\mathrm{ReleaseSB}==\mathrm{true})) \tag{12}$$

$$[G](((D==1)\&\&(\mathrm{DRIVER\_ACK}==\mathrm{true}))=>[F]((\mathrm{ReleaseSB}==\mathrm{true})\&\&(\mathrm{SB\_REQUEST}==\mathrm{false}))) \tag{13}$$

$$[G]((\mathrm{SB\_REQUEST}==\mathrm{true})\&\&(S==1)\&\&(\mathrm{STANDSTILL}==\mathrm{false})=>(\mathrm{ReleaseSB}==\mathrm{false})) \tag{14}$$

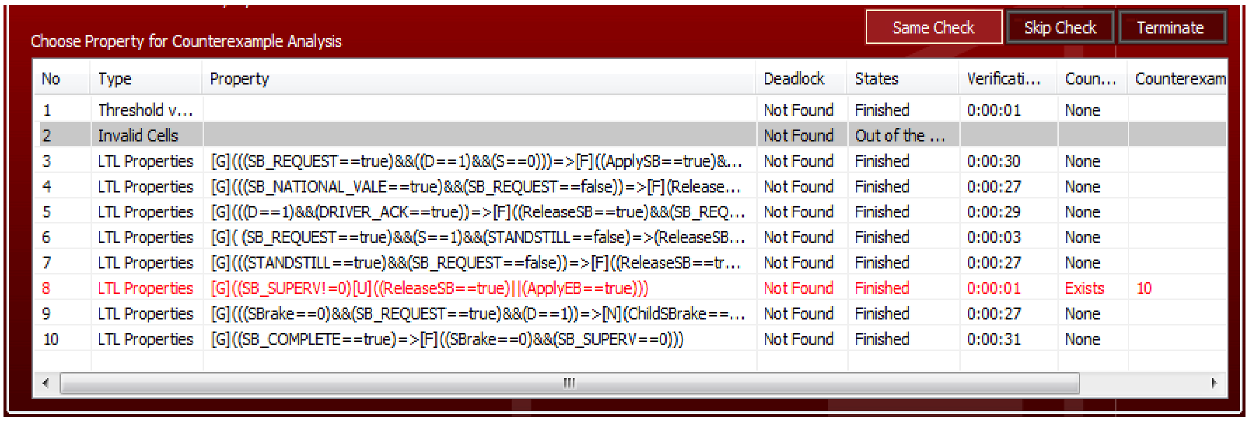
$$[G](((STANDSTILL==true)\&\&(SB\_REQUEST==false))=> [F]((ReleaseSB==true)\&\&(SB\_REQU(EST==false))) \quad (15)$$

$$[G]((SB\_SUPERV!=0)[U]((ReleaseSB==true)||(ApplyEB==true))) \quad (16)$$

$$[G](((SBrake==0)\&\&(SB\_REQUEST==true)\&\&(D==1))=>[N](ChildSBrake==0)) \quad (17)$$

$$[G]((SB\_COMPLETE==true)=>[F]((SBrake==0)\&\&(SB\_SUPERV==0))) \quad (18)$$

Each demand condition was verified using Garakabu II, and the results are shown in Fig. 25. The LTL formula (16) corresponding to the demand condition is not verified. In the 10th step of the verification, find the counterexample, which can be used as a test case for the column control document. Targeted testing of documents during document walkthroughs.

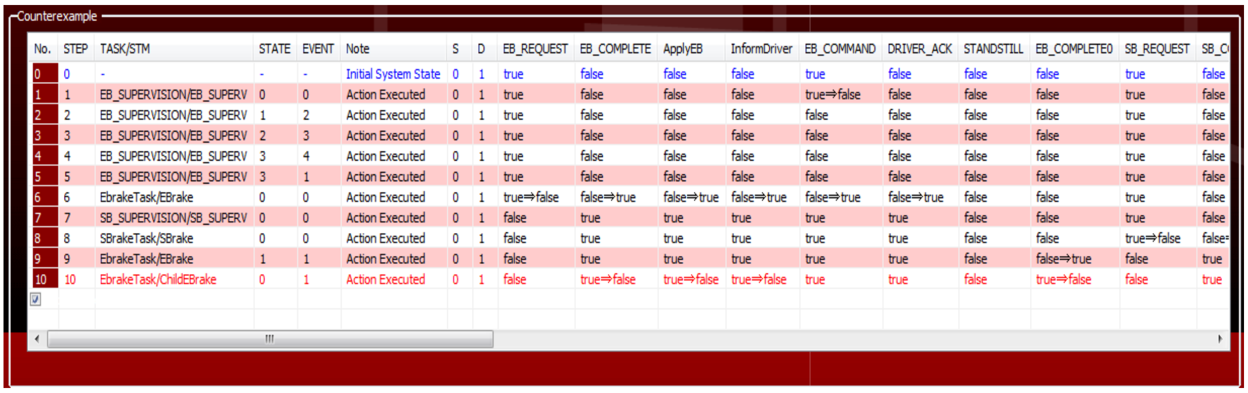

Choose Property for Counterexample Analysis

Same Check | Skip Check | Terminate

No	Type	Property	Deadlock	States	Verificati...	Coun...	Counterexam
1	Threshold v...		Not Found	Finished	0:00:01	None	
2	Invalid Cells		Not Found	Out of the ...			
3	LTL Properties	[G](((SB_REQUEST==true)&&((D==1)&&(S==0)))=>[F]((ApplySB==true)&...	Not Found	Finished	0:00:30	None	
4	LTL Properties	[G](((SB_NATIONAL_VALE==true)&&(SB_REQUEST==false))=>[F](Release...	Not Found	Finished	0:00:27	None	
5	LTL Properties	[G](((D==1)&&(DRIVER_ACK==true))=>[F]((ReleaseSB==true)&&(SB_REQ...	Not Found	Finished	0:00:29	None	
6	LTL Properties	[G]((SB_REQUEST==true)&&(S==1)&&(STANDSTILL==false)=>(ReleaseSB...	Not Found	Finished	0:00:03	None	
7	LTL Properties	[G](((STANDSTILL==true)&&(SB_REQUEST==false))=>[F]((ReleaseSB==tr...	Not Found	Finished	0:00:27	None	
8	LTL Properties	[G]((SB_SUPERV!=0)[U]((ReleaseSB==true)||(ApplyEB==true)))	Not Found	Finished	0:00:01	Exists	10
9	LTL Properties	[G](((SBrake==0)&&(SB_REQUEST==true)&&(D==1))=>[N](ChildSBrake==...	Not Found	Finished	0:00:27	None	
10	LTL Properties	[G]((SB_COMPLETE==true)=>[F]((SBrake==0)&&(SB_SUPERV==0)))	Not Found	Finished	0:00:31	None	

Fig. 25. Common brake demand test results

The verification of the LTL formula (16) corresponding to the requirement is not passed, and the counterexample is found in the 10th step of the verification, and the corresponding counterexample path is as shown in Fig. 26.


Counterexample

No.	STEP	TASK/STM	STATE	EVENT	Note	S	D	EB_REQUEST	EB_COMPLETE	ApplyEB	InformDriver	EB_COMMAND	DRIVER_ACK	STANDSTILL	EB_COMPLETE0	SB_REQUEST	SB_C
0	0	-	-	-	Initial System State	0	1	true	false	false	false	true	false	false	false	true	false
1	1	EB_SUPERVISION/EB_SUPERV	0	0	Action Executed	0	1	true	false	false	false	true⇒false	false	false	false	true	false
2	2	EB_SUPERVISION/EB_SUPERV	1	2	Action Executed	0	1	true	false	false	false	false	false	false	false	true	false
3	3	EB_SUPERVISION/EB_SUPERV	2	3	Action Executed	0	1	true	false	false	false	false	false	false	false	true	false
4	4	EB_SUPERVISION/EB_SUPERV	3	4	Action Executed	0	1	true	false	false	false	false	false	false	false	true	false
5	5	EB_SUPERVISION/EB_SUPERV	3	1	Action Executed	0	1	true	false	false	false	false	false	false	false	true	false
6	6	EbrakeTask/EBrake	0	0	Action Executed	0	1	true⇒false	false⇒true	false⇒true	false⇒true	false⇒true	false⇒true	false	false	true	false
7	7	SB_SUPERVISION/SB_SUPERV	0	0	Action Executed	0	1	false	true	true	true	true	true	false	false	true	false
8	8	SBrakeTask/SBrake	0	0	Action Executed	0	1	false	true	true	true	true	true	false	false	true⇒false	false=
9	9	EbrakeTask/EBrake	1	1	Action Executed	0	1	false	true	true	true	true	true	false	false⇒true	false	true
10	10	EbrakeTask/ChildEBrake	0	1	Action Executed	0	1	false	true⇒false	true⇒false	true⇒false	true	true	false	true⇒false	false	true

Fig. 26. Counterexample path for common braking requirements (16)

The information included in the counter-path path interface: The steps of the model run, the name of the ETM model running in the current step, the status number and event number, and the variable information involved in the model. During the transition of the state, changes in the values of the variables are also reflected. Clicking on the sequence of states can jump to the processing cell of the ETM model, so that the counter-example path can be dynamically presented, which is conducive to the discovery and correction of the problem. By analyzing the counter-example path, when the SBrake model is in the processing cell of state 0 event 0, the conditions of ReleaseSB==false and ApplyEB==true are satisfied, and the values of the two variables do not change in the next stage. However, at this time SB_SUPERV==1, the demand condition in the document is described as: Always monitor the common brake deceleration before canceling the normal brake or applying the emergency brake. After the emergency brake command is applied, it should be in the emergency brake deceleration monitoring state, and SB_SUPERV should be in the unmonitored state, so it can be judged that there is a logic error in the document related to the demand condition.

6 Conclusions and Prospects

In this paper, in view of the shortcomings of traditional document walk-through methods in discovering logical problems, a formal method is proposed to validate software requirements analysis documents. An embedded software requirements analysis document formal engineering modeling method was designed, and the system requirements document is modeled using the extended state transition matrix (ESTM) model. Secondly, according to the migration characteristics of the model, the symbolic coding and the formal description of the properties are proposed. The state compression is proposed by transforming the LTL formula into an automaton. Finally, using the new model detection tool Garakabu II for model verification, the automation level of document testing is improved. Through experiments, it can be found that the formal analysis method of demand analysis document can effectively find the required vulnerability and logic error of the document. Using the obtained counterexample, the document can be tested in a targeted manner.

There are still some improvements in the document testing method proposed in this paper: The verification tool used by Garakabu II is based on the CVC4 solver. Compared to other mainstream solvers such as Z3, the verification rate is slightly insufficient. In the next stage we will improve the efficiency of model validation.

Acknowledgements

The authors would like to thank President Watanabe Kazuhiko of Japan CATS Co., Ltd. and Professor Kong Weiqiang of Dalian University of Technology for their guidance on the software evaluation of the train control system to ensure the smooth completion of the project.

References

[1] Z.-Q. Huang, B.-F. Xu, S.-L. Kan, J. Hu, Z. Chen, Survey on embedded software safety analysis standards, methods and tools for airborne system, Journal of Software 25(2)(2014) 200-218.

[2] J.-J. Yang, K.-L. Su, X.-Y. Luo, H. Lin, Y.-Y. Xiao, Optimization of Bounded Model Detection, Journal of Software 20(08)(2009) 2005-2014.

[3] L.-Y. Zhang, Q.-D. Meng, G.-M. Luo, Optimized symbolic model checking for component-based systems, in: Proc. 2014 International Conference on Cognitive Informatics & Cognitive Computing, 2014.

[4] G. Hou, K.-J. Zhou, J.-W. Yong, L.-T. Ren, X.-L. Wang, Survey of state explosion problem in model checking, Computer Science 40(Z6)(2013) 77-86.

[5] J. Yan, W. Wang, H.-W. Chen, Survey of model-based software testing, Computer Science 31(2)(2004) 184-187.

[6] Y.-Q. Wang, H.-Y. Yu, Overview of computer software testing technology, Electronic Technology and Software Engineering (22)(2017) 57-58.

[7] L. WANG, Research on user class document testing, Journal of Computer Applications 26(10)(2006) 2470-2472.

[8] C.-Y. Ji, X.-Q. Song, Research on measurement method of software document quality, Computer Engineering and Design 28(17)(2007) 4068-4071.

[9] Q.-Y. Chen, C.-Z. Long, L. Zhang, C.-L. Jiang, R.-G. Xu, Software document management strategy based on software life cycle, Modern Computer (6)(2004) 51-54.

[10] H.-H. Yan, J. Hu, L.-F. Zhang, F. He, Process management for software quality assurance based on document status, Journal of Beijing University of Aeronautics and Astronautics 27(4)(2001) 474-477.

[11] Y.-H. Huang, J.-C. Feng, H.-Y. Zheng, W.-K. Miao, G.-G. Pu, A formal engineering method for requirement Modeling of airborne embedded control software, Computer Engineering and Science 41(6)(2019) 1016-1025.

[12] D.-X. Wang, Q. Wang, J. He, Evidence-based software process trustworthiness model and evaluation method, Journal of Software (7)(2017) 1713-1731.

[13] M.-S. Chen, Y.-X. Bao, H.-Y. Sun, W.-K. Miao, X.-H. Chen, T.-L. Zhou, Survey on formal method of trustworthy construction for communication-based train control systems, Journal of Software 28(5)(2017) 1183-1203.

[14] Y.-C. Li, Construction and analysis of user demand knowledge mapping for product iterative innovation, [dissertation] Guangzhou: Jinan University, 2018.

[15] S. Cranen, J.-F. Groote, M. Reniers, A linear translation from CTL* to the first-order modal μ-calculus, Theoretical Computer Science 412(28)(2011) 3129-3139.

[16] H. Han, Research on unreachable base path detection technology based on association analysis, [dissertation] Xuzhou: China University of Mining and Technology, 2014.

[17] C.-Y. Xia, Y. Zhang, L. Song, Evolutionary generation of test data for paths coverage based on node probability, Journal of Software 27(4)(2016) 802-813.

[18] E. Clarke, D. Kroening, J. Ouaknine, O. Strichman, Completeness and complexity of bounded model checking, in: Proc. 2004 Proceeding of the Verification, on-Model Checking, and Abstract Interpretation, 2004.

[19] L.-C. Shan, M.-H. Wu. A brief analysis of model checking technique, Information Technology and Informatization (7)(2014) 125-129.

[20] G. Hou, K.-J. Zhou, J.-W. Chang, J. Wang, M.-C. Li, Software formal modeling and verification method based on time STM, Journal of Software 26(2)(2015) 223-238.