Automated Failure Analysis in Model Checking based on Data Mining

Ning Ge, Marc Pantel, Xavier Crégut

Automated Failure Analysis in Model Checking Based on Data Mining*

Ning Ge[1], Marc Pantel[2], and Xavier Crégut[2]

[1] LAAS-CNRS, France
Ning.Ge@laas.fr
[2] University of Toulouse, IRIT-INPT, France
{Marc.Pantel,Xavier.Cregut}@enseeiht.fr

Abstract. This paper presents an automated failure analysis approach based on data mining. It aims to ease and accelerate the debugging work in formal verification based on model checking if a safety property is not satisfied. Inspired by the `Kullback-Leibler Divergence` theory and the `TF-IDF` (Term Frequency - Inverse Document Frequency) measure, we propose a suspiciousness factor to rank potentially faulty transitions on the error traces in time Petri net models. This approach is illustrated using a best case execution time property case study, and then further assessed for its efficiency and effectiveness on an automated deadlock property test bed.

Keywords: Model checking, Failure analysis, Data mining, Time Petri net.

1 Introduction

Generating counterexamples in case a logic formula is violated is a key service provided by model checkers. Counterexamples are expected to display some unwanted but possible behaviors of the system and to help the user(s) in correcting the faulty design. Counterexamples often stand for error traces, which represent sequences of states and transitions and are therefore usually lengthy and difficult to understand. It is usually an exhausting work to understand the origin of failure and to extract useful debugging information using counterexamples. The origin of failure might be anywhere along these traces, thus requiring a lengthy analysis by designers. Based on the above understanding, we conclude that feeding back counterexamples in model checking provides limited help in understanding the origin of errors and in improving the model design. Our ultimate goal is to provide the designer with the suspicious ranked faulty elements by analyzing the error traces in the model checking results. The fact is, although model checking has been developed as a mature and heavily used formal verification technique, the automated failure analysis relying on model checking results is still mostly an open challenge.

* This work was funded by the French ministries of Industry and Research and the Midi-Pyrénées regional authorities through the FUI P and openETCS projects.

Failure analysis in model checking is difficult due to the use of abstractions. At the time of writing, the conflict between model precision and verification cost is a key issue in model checking. The abstraction is a must for model checking to reduce the size of state space. It eliminates property-irrelevant semantics and may also combine property-relevant semantics. But this semantics may help failure analysis.

According to the survey from [1], existing automated fault localization techniques in model checking usually produce a set of suspicious statements without any particular ranking. [2] proposed to analyze fault localization using one single counterexample that violated the expected properties in a particular case. Whenever a counterexample was found, the approach compared the error trace derived from the counterexample to all the correct traces that conformed to the requirement. On the observed error and correct traces, the transitions that led to the deviation from correct traces are marked as suspicious transitions. [3] proposed to rely on multiple counterexamples. It defined the concepts of positive trace and negative trace. The negative traces start from initial states and ended with error states. The transitions in positive traces are not prefix to any negative traces. It distinguished the transitions that existed in all positive traces; the transitions that appeared in all negative traces; the transitions that existed in one of positive traces but not in any negative traces; and the transitions that appeared in one of negative traces, but not in any positive traces. The algorithm then used the above marked transitions to identify the origin of failure. [4] proposed to define a distance between the error trace and the successful traces. The distance was then used to find the closest successful trace to the counterexample. The causes of error were then derived from the comparison results between the closest successful trace and the counterexample.

In this paper, we will improve the effectiveness of failure analysis in model checking by providing a suspiciousness factor, when a safety property is not satisfied. The safety property asserts that nothing bad happens [5]. Examples of safety properties include mutual exclusion, deadlock freedom, partial correctness, and first-come-first-serve [6]. They can be satisfied when no reachable error states (erroneous behavior) or deadlock states (no outgoing transitions) exist in the reachability graph. Otherwise, some unwanted states are detected, called violation states. Inspired by the `Kullback-Leibler Divergence` theory and the `TF-IDF` (Term Frequency - Inverse Document Frequency) measure in data mining, the suspiciousness factor is proposed to rank the suspicious faulty transitions. We construct error traces in the reachability graph using all the violation states. The suspiciousness factor is defined as the fault contribution of each transition on all the error traces. It is computed using the entropy and differential entropy of transition. We apply this approach to Time Petri net (`TPN`) model relying on observers to provide all the faulty execution traces and the violation states in the reachability graph preserving markings. This verification approach was studied in our previous work [7]. The proposed failure analysis method is illustrated using a `TPN` case study where the `BCET` (Best Case Execution Time) property

is verified, and then further assessed for its efficiency and effectiveness on an automated test bed where the deadlock property is verified.

This paper is organized as follows: Section 2 gives some preliminaries; Section 3 introduces the core idea of the proposed approach; Section 4 details the proposed automated failure analysis approach using a BCET case study; Experimental results derived from a set of test cases are presented in Section 5 to assess the effectiveness and efficiency; Section 6 summarizes the contributions of this work.

2 Preliminaries

2.1 Time Petri Net

Time Petri nets [8] extend Petri nets with timing constraints on the firing of transitions. Time Petri nets are widely used to capture the temporal behavior of concurrent real-time system in a formal way. Our work relies on TINA (TIme petri Net Analyzer)[1] as the verification toolset.

Definition 1 (Time Petri Net). *A Time Petri Net (TPN) $\mathcal{T}$ is a tuple $\langle P, T, {}^\bullet(.), (.)^\bullet, M_0, (\alpha, \beta)\rangle$, where:*

- $P = \{p_1, p_2, ..., p_m\}$ *is a finite set of places;*
- $T = \{t_1, t_2, ..., t_n\}$ *is a finite set of transitions;*
- ${}^\bullet(.) \in (\mathbb{N}^P)^T$ *is the backward incidence mapping;*
- $(.)^\bullet \in (\mathbb{N}^P)^T$ *is the forward incidence mapping;*
- $M_0 \in \mathbb{N}^P$ *is the initial marking;*
- $\alpha \in (\mathbb{Q}_{\geq 0})^T$ *and* $\beta \in (\mathbb{Q}_{\geq 0} \cup \infty)^T$ *are respectively the earliest and latest firing time constraints for transitions.*

Following the definition of enabledness in [9], a transition t_i is enabled in a marking M iff $M \geq {}^\bullet(t_i)$ and $\alpha(t_i) \leq v_i \leq \beta(t_i)$ (v_i is the elapsed time since t_i was last enabled). There exists a global synchronized clock in the whole TPN, and $\alpha(t_i)$ and $\beta(t_i)$ correspond to the local clock of t_i. The local clock of each transition is reset to zero once the transition becomes enabled. The predicate $\uparrow Enabled(t_k, M, t_i)$ is satisfied if t_k is enabled by the firing of transition t_i from marking M, and false otherwise.

$$\uparrow Enabled(t_k, M, t_i) = (M - {}^\bullet(t_i) + (t_i)^\bullet \geq {}^\bullet(t_k)) \wedge ((M - {}^\bullet(t_i) < {}^\bullet(t_k)) \vee (t_k = t_i)) \quad (1)$$

We use an example (see Ex. 1) to explain the syntax and semantics of time Petri nets.

Example 1 (TPN Example). The example in Fig. 1 models the concurrent execution of a process. The whole net shares a common synchronized clock. P_{init} is the place holding the initial token. When the $fork$ transition is fired, concurrent $task_1$ and $task_2$ start at the same time within respective execution time [11,15]

[1] http://projects.laas.fr/tina/

and [19,27] associated to the transitions. Each transition uses a local clock which starts once the transition becomes enabled. When the control flow reaches the *join* place, the system will exit or restart the whole execution according to the running time.

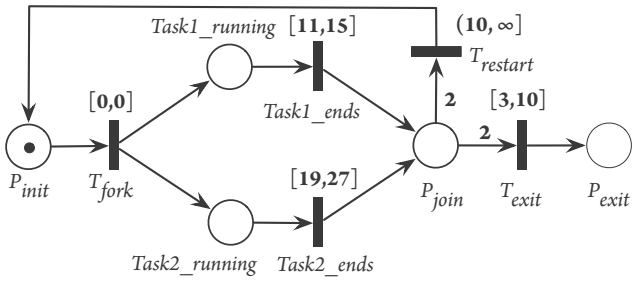

Fig. 1. Time Petri Net Example

2.2 `Kullback-Leibler Divergence` Applied to Textual Documents

The `Kullback-Leibler Divergence` (`KL-Divergence`) is also called information divergence, information gain, or relative entropy [10]. It is a fundamental equation of information theory that qualifies the proximity of two probability distributions. Many statistical procedures for inference use `KL-Divergence` information either directly or indirectly. It is also the theory basis of the `TF-IDF` measure.

Definition 2 (`KL-Divergence`)**.** *The `KL-Divergence` is a measure in statistics that quantifies how close a probability distribution $P = \{p_i\}$ is to a model (or candidate) distribution $Q = \{q_i\}$. The KL-divergence of Q from P over a discrete random variable is defined as*

$$D_{KL}(P \parallel Q) = \sum_i P(i) \ln \frac{P(i)}{Q(i)} \tag{2}$$

Note: In the definition above, $0 \ln \frac{0}{0} = 0$, $0 \ln \frac{0}{q} = 0$, *and* $p \ln \frac{p}{0} = \infty$.

Many successful applications are based on `Kullback-Leibler Divergence`. We give an example about the text classification problem [11]. A textual document d is a discrete distribution of $|d|$ random variables, where $|d|$ is the number of terms in the document. Let d_1 and d_2 be two documents whose similarity we want to compute. This is done using $D_{KL}(d_1 \parallel d_2)$ and $D_{KL}(d_2 \parallel d_1)$.

2.3 Term Frequency – Inverse Document Frequency

Another major application of `KL-Divergence` is the `TF-IDF` (Term Frequency - Inverse Document Frequency) algorithm [12]. `TF-IDF` is a numerical statistic

which reflects how important a term is for a given document in a corpus (collection) of documents. It is often used as a weighting factor in information retrieval and text data mining. Variations of the TF-IDF weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance to a given user query. TF-IDF is the product of two statistics, TF and IDF. Suppose we have a collection of English textual documents and aim to determine which documents are most relevant to the query "the model checking". We might start by eliminating documents that do not contain the three words "the", "model", and "checking", but this still leaves many documents. To further distinguish them, we might count the number of times each term occurs in each document and sum them all together; the number of times a term occurs in a document is called TF. It stands for the frequency of a term in a document, and it reflects how important a term is in this document. However, because the term "the" is so common, this might incorrectly emphasize documents which happen to use the word "the" more frequently, without giving enough weight to the more meaningful terms "model" and "checking". The term "the" is not a good keyword to distinguish relevant and non-relevant documents and terms, unlike the less common words "model" and "checking". Hence IDF factor is incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely.

3 Core Idea

In the TF-IDF algorithm, each term in the documents will contribute to the keyword semantics. Some terms are considered as significant if they are more relevant to the keyword semantics. This is similar to the fault contribution caused by a given transition in an error trace in model checking. Fig. 2 compares the similarity between semantic contribution of terms in documents and fault contribution of transitions in error traces. Some terms in documents have closer semantic relation to the keywords, the occurrence of these terms provide more semantic contributions to the occurrence of keywords. Similarly, the fault propagation depends on the topology of error traces, the occurrence of some transitions will provide more fault contributions to the occurrence of violation states.

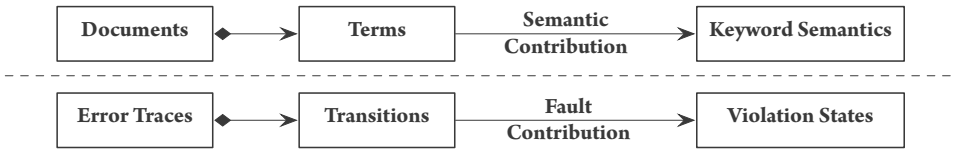

Fig. 2. Comparison to TF-IDF

The semantic contribution of a term in documents is measured by TF-IDF, where TF is the contribution of a term in single document, and IDF is the contribution of a term in a collection of documents. The fault contribution to the

violation states caused by a transition on error traces can also be evaluated by a similar measure defined as Fault Contribution.

Definition 3 (Fault Contribution). *Fault Contribution (C_F) is a suspiciousness factor to evaluate a transition's suspicion level. It is used to rank the suspiciousness of transitions.*

4 Ranking Suspicious Faulty Transitions

Inspired by the TF-IDF algorithm, we propose a probabilistic failure analysis approach based on data mining. The relevance weight $C_F(t)$ is computed to assess the fault contribution of each transition t in error traces.

4.1 BCET Property Case Study

Before presenting the failure analysis algorithm, a BCET case study (see Ex. 2) is provided to help illustration.

Example 2 (Failure Analysis Example). Fig. 3 is a TPN model with 10 transitions $\{t_0, t_1, ..., t_9\}$. It has two main execution paths (respectively through t_1 and t_2), both have a loop with a bound of 2. The expected time property is: system's BCET is bounded within a given time T, i.e. BCET $> T$. We aim to automatically identify the potentially faulty transitions, and to rank them according to their fault contributions to the violation states. We first analyze the case when $T = 10$, then give the analysis results for $5 \leq T \leq 50$.

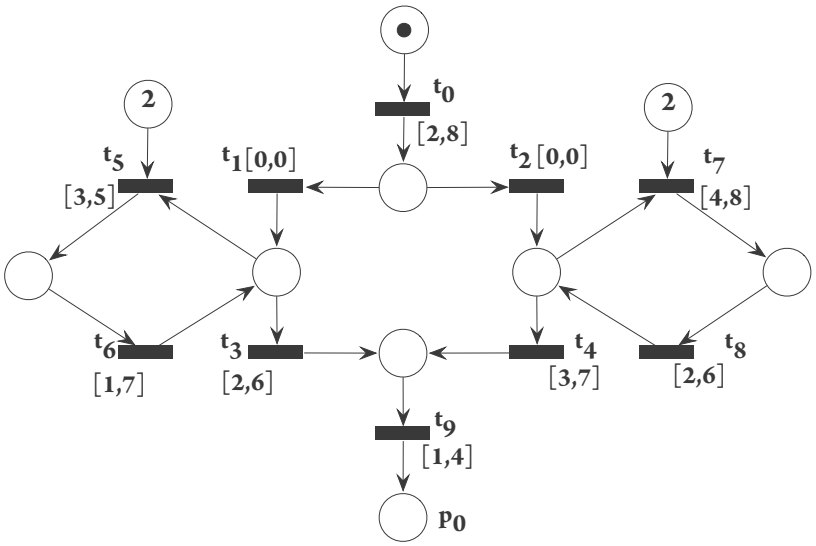

Fig. 3. Failure Analysis Case Study

4.2 Reachability Graph and Violation States

Reachability graphs are used to solve reachability problems in model checking. They contain all the states in the execution of a system and all the transitions between these states. In the `TINA` toolset, depending on the selected options, `tina` builds reachability graphs of different abstractions, expressed as Kripke transition systems (`ktz`). When a safety property is not satisfied, violation states can be found in the reachability graph.

According to the observer-based model checking approach for `TPN` presented in our work [7], we use the state class graph preserving markings as the reachability graph and turn the quantitative problem into a reachability problem. A `TPN` state can be seen as a pair (M, D), in which M is a marking, and D is a set of vectors called the firing domain. The reachability assertions are used to check the marking existence, such as $(M_P = 1)$ or $(M_P = 0)$, where M_P is the marking in the observation place P. Once the given reachability assertion is violated, the set of violation states in the reachability graph is built.

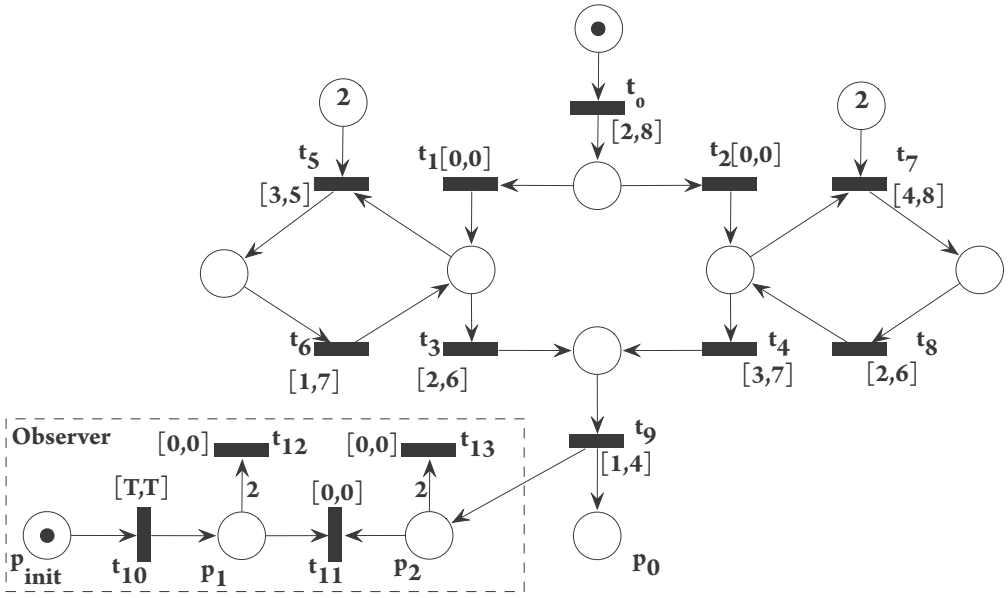

Fig. 4. Failure Analysis Case Study with `BCET` Observer

A `BCET` observer structure in Fig. 4 (in dotted line box) is thus associated with the end transition of the whole system t_9. The observer is linked from t_9. This connection ensures that the observer works in a read-only manner, thus cannot impact system's original behavior. We observe the tokens in places P_1 and P_2. The place P_{init} ensures that the observer structure starts at the same time with the observed system.

To explain how to check `BCET` property, we provide an erroneous scenario in Fig. 5. When the running time is less than T, t_{10} is not yet enabled, the place P_1

is empty. Meanwhile, if t_9 is already fired, P_2 has a token. In this scenario, the execution time of the system is less than T, then the BCET must be less than T. Therefore, the property BCET $> T$ is not satisfied. This property can be formally expressed by the formula $\neg(\neg p_1 \wedge p2)$. Then the assertion $N(\neg(\neg p_1 \wedge p2)) = N_A$ is used to check this property, where $N(\neg(\neg p_1 \wedge p2))$ is the number of states satisfying $\neg(\neg p_1 \wedge p2)$, N_A is the total number of states in system execution.

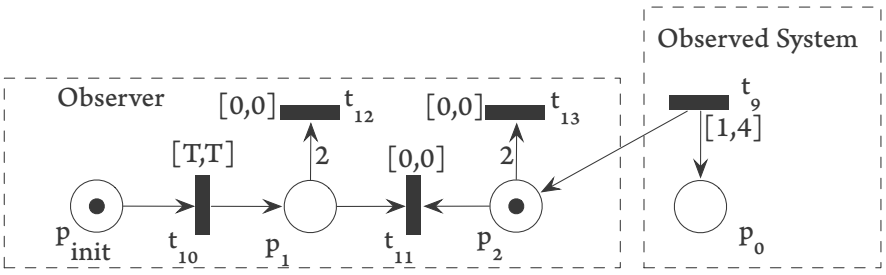

Fig. 5. Failure Scenario

When $T = 10$, we generate the reachability graph (Fig. 6) for TPN model with associated observer. The state number in this reachability graph (N_A) is 39. All the states are labeled with a number from 0 to 38. The checking result for $N(\neg(\neg p_1 \wedge p2)) = N_A$ is $False$, because $N(\neg(\neg p_1 \wedge p2))$ is 37. Therefore, there exist two violation states in the reachability graph. The violation states are those satisfying the formula $\neg p_1 \wedge p2$. They are directly found by the muse model checker in the TINA toolset, i.e. violation states are s_{11} and s_{23}.

4.3 Error Traces

We aim to compute the fault contribution of each transition in the error traces. The error traces are constructed using the violation states in the reachability graph.

Definition 4 (Error Trace). *For all the states $\{s_i\}$ on each path from the initial state s_0 to a violation state s_v in the reachability graph, all the outgoing transitions of s_i are gathered in a sequence called error trace π.*

We consider not only the transitions on the path that leads from s_0 to s_v but also the direct outgoing transitions of all the states in the execution traces that lead to correct states. Indeed, in TPN, the transitions outgoing from the same place can mutually influence each other. A correct transition can impact the firing of a faulty transition if they are both outgoings from the same place. The correct transition will diminish the C_F of the faulty transition.

Example 3 (Error Trace Example). In Fig. 7, s_0 is the initial state, and s_v is a violation state. In the execution trace from s_0 to s_v, there exist four states $\{s_0, s_1, s_2, s_3\}$ (apart from s_v). The states s_5, s_6, s_7, s_8, s_9 do not lead to the

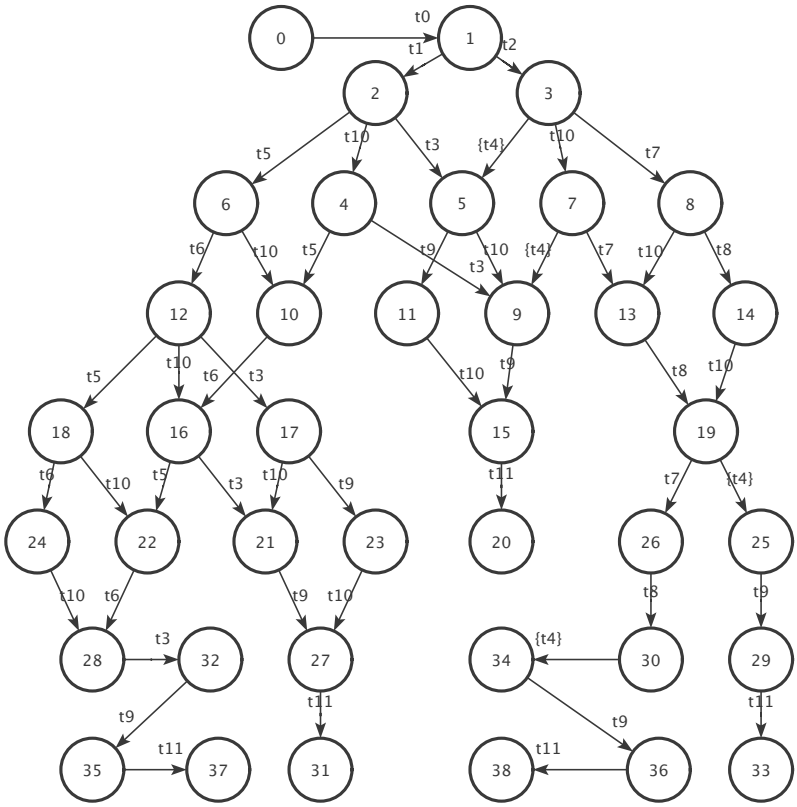

Fig. 6. Reachability Graph of Case Study (T=10)

violation state s_v. They are in the correct traces. When the system is in state s_2, it is possible to transit to s_7 or to s_3. If s_7 (t_4) is removed from the graph, s_3 (t_2) will have higher probability (fault contribution) for the occurrence of s_v. Therefore, transition t_4 should be included in the error trace, although it does not lead to s_v. Similarly, the transitions leading to other correct states should also be included in the error trace. The outgoing transitions of these four states are considered as error traces π, i.e., $\pi = \{t_0, t_1, t_2, t_1, t_5, t_4, t_2, t_3, t_4\}$.

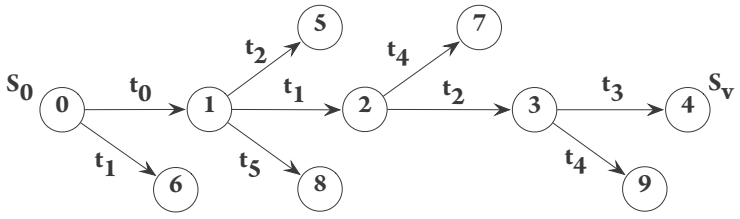

Fig. 7. Error Trace Example

The algorithm for enumerating all the error traces in the reachability graph is trivial, but the impact of state cycles in error traces needs to be discussed. The reachability graph in Fig. 8 contains a state cycle C_s ($s_1 \xrightarrow{t_1} s_3 \xrightarrow{t_4} s_4 \xrightarrow{t_3} s_1$).

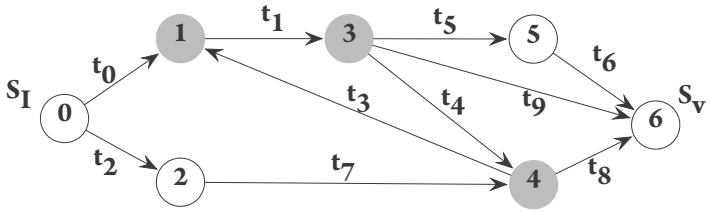

Fig. 8. Error Trace Example

The error traces passing through s_1 may loop in C_s. Take one error trace as an example, the trace passing through states s_0, s_1, s_3, s_4, s_6 is

$$s_0 \xrightarrow{t_0} \{s_1 \xrightarrow{t_1} s_3 \xrightarrow{t_4} s_4 \xrightarrow{t_3} s_1\}_n \xrightarrow{t_8} s_6$$

where n represents the times that the cycle repeats. The repetition of a cycle will not increase the fault contribution of the transitions in the cycle, because the system behavior is restricted to these states. Therefore, the cycle can be treated as a point with a chain of transitions (here $t1, t4, t3$). In other words, n is taken to be 1.

In the BCET case study, we construct error traces using the reachability graph in Fig. 6 and violation states s_{11} and s_{23}. The error traces are as follows:

$$\pi_1 : \{t_0, t_1, t_2, t_5, t_{10}, t_3, t_9, t_{10}\}$$
$$\pi_2 : \{t_0, t_1, t_2, t_4, t_{10}, t_7, t_9, t_{10}\}$$
$$\pi_3 : \{t_0, t_1, t_2, t_5, t_{10}, t_3, t_6, t_{10}, t_5, t_{10}, t_3, t_{10}, t_9\}$$

4.4 TC-ITC Algorithm

Fault contribution of the transition in error traces is measured by two factors, transition contribution and inverse trace contribution.

Definition 5 (Transition Contribution (TC)). *TC is a measure of the occurrence frequency of a transition t in an error trace π. It reflects a transition's contribution to violation state s_v in π. It is defined to be*

$$TC(t) = \frac{1}{M} \sum_{i=1}^{M} \frac{Q_i}{L_i} \tag{3}$$

where Q_i is number of occurrences of t in error trace π_i, L_i is the number of states from the initial state to the state before s_v, and M is the total number of error traces.

Definition 6 (Inverse Trace Contribution (ITC)). *ITC is a measure of whether a transition t is common or rare among all the error traces to all the violation states. It is defined to be*

$$ITC(t) = \log_2 \frac{M}{\sum_{i=1}^{M} X_i}, \tag{4}$$

where $X_i = \begin{cases} 1 & \text{if } t \text{ occurs at least one time in an error trace} \\ 0 & \text{otherwise} \end{cases}$ *and* M *is the total number of error traces.*

The weight TC-ITC is the product of the above two measures. In some cases, this product is 0, which does not mean it cannot be the fault source but only implies that the elements make the least contributions to the violation states and have the least probability comparing to the others. These elements usually should be checked at last.

It is expected that the ranking of fault contributions computed by the algorithm corresponds to manual analysis and human intuition. We use the BCET case study to illustrate how they are matched. The analysis results are provided in Fig. 9. The results show the fault contributions (normalized for comparing the trend) of each transition when T varies from 5 to 50. The explanation is provided as follows:

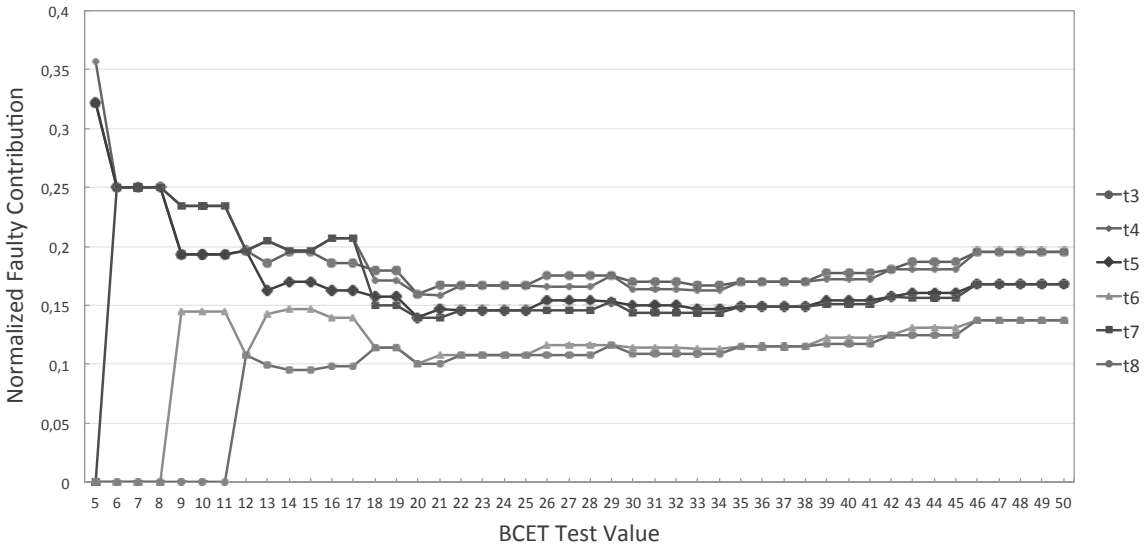

Fig. 9. Feedback of Fault Localization Example

- $1 \leq T < 5$: since the `BCET` of the system is 5, there will not be any violated state and accordingly no failure analysis will be launched.
- $T \geq 47$: since the `WCET` (Worst Case Execution Time) of the system is 47, the reachability graph will not have any change after this threshold, therefore the fault contribution of each transition will preserve the same value as $T = 47$.
- $5 \leq T < 47$: since T represents the expected `BCET` of the system, all execution with time inferior to T will be considered as violation. Without any other information, a reasonable heuristics can then be derived from this assertion: for `BCET` property, the less execution time a transition can contribute/has contributed to the violation of `BCET`, the higher risk it will be the failure cause. Another intuition-valid rule is: when an element holds a more complex function, it has a higher risk to cause design errors. To heuristically quantify the coefficient of these two different types of fault contribution is a subjective measure often context-dependent. In order to avoid this indecisive discussion, each time we encounter this situation in the case study, we will just explain the two aspects without trying to combine them into one score for matching the ranking. The statistical trends are then explained as follows:
  - **Topologically symmetric pair (t_3, t_4) has a higher risk to be the failure cause than (t_5, t_7) and (t_6, t_8)**. This matches the heuristic rule because in whichever execution, t_3 and t_4 will only contribute once to the global execution time (i.e. [2,6] and [3,7] respectively), while t_5, t_6, t_7 and t_8 can at most execute twice and will contribute more (i.e [6,10], [2, 14], [8, 16] and [4, 12] respectively).
  - **In each symmetric pair of above, $t_3 \geq t_4$, $t_5 \geq t_7$ and $t_6 \geq t_8$.** This demonstrates that it is always the one with the smallest execution time that get more risk to be the faulty one.
  - **t_0, t_1, t_2 and t_9 are equally the least suspicious elements.** (To emphasize the other transitions, they are not shown in Fig. 9.) This conforms to the intuition because in all execution paths, whether is good or bad, t_0 and t_9 will always be executed therefore no information added for assessing their faulty risk. For t_1 and t_2 it is the similar situation, because a design fault will either be on the left side or the right side, and in all execution paths of the left (resp. the right) side, t_1 (resp. t_2) will always be executed.
  - **Pair (t_5, t_7) has a higher risk than (t_6, t_8).** Generally since t_6/t_8 has smaller execution time than t_5/t_7, it shall be more possible according to the first heuristic rule. However, since t_5 and t_7 play a role that not only postpone the execution (like t_6/t_8), but also branch the execution path (t_6/t_8 do not have this function), their risk to be the failure cause will be re-distributed and raised as the second rule is engaged.

5 Experiments

To assess the success of a fault localization algorithm, many important criteria should be measured, such as effectiveness, precision, informativeness, efficiency,

performance, scalability and information usefulness. In our work, we assess our approach by using two significant criteria: effectiveness and efficiency.

Efficiency. The fault localization techniques in model checking, like other techniques, should terminate in a timely manner, limited by some resource constraints. The efficiency can be assessed by the scalability and the performance.

Effectiveness. An effective fault localization method should point out the origin of failure. The effectiveness can be evaluated by the precision. According to the survey [13], the effectiveness can be assessed by a score called `EXAM` in terms of the percentage of statements that have to be examined until the first statement containing the fault is reached. In this work, the `EXAM` score measures the percentage of transitions that have to be examined until the first faulty transition is found.

In order to assess the effectiveness and efficiency of the proposed method, we have designed an automated deadlock property test bed.

5.1 Automated Deadlock Property Test Bed

The test bed will randomly generate systems that might have deadlocks, then apply the proposed analysis algorithm to detect the introduced deadlocks. The main reason to use the deadlock as test property is because it is relatively easy to create a scalable system with deadlock heuristically. As the analysis in our method is based on the error traces and violation states in the reachability graph, it does not distinguish the property types in the model level. Although the test bed contains only one property, the effectiveness and efficiency evaluations will be meaningful for other safety properties.

For a given `TPN` system $S(P, R, M)$, P are processes which run infinitely and need a resource before the next task (a task is represented by a transition); R are resources shared by processes, but only accessible in an exclusive way; M is a matrix to decide whether process P_i needs to access resource R_j. We rely on the Coffman conditions [14] to build the deadlock test cases. Each process is designed to be moderately consuming the resource, i.e. it will use its resources consequently, always release one before locking another. The order in which a resource is accessed by processes is however random, which establishes the necessary condition of deadlock. In practice, the above conditions can be constructed statically when building test cases, while the circular wait condition (each process in a circular list or chain is waiting for a resource held by the next process in the list) can only be checked dynamically during the system execution. Therefore, the generated `TPN` will not systematically guarantee that a "real" deadlock will occur. To improve the success of creating deadlocks, we introduce another mechanism to enforce deadlocks: randomly let some processes during some tasks forget to release a used resource. These tasks are then considered as the failure cause of deadlocks. With a generated system and its already known faulty transitions (release-forgot tasks), the test bed will apply our method to compute the fault contribution of each task.

5.2 Evaluation of Efficiency

We have generated thousands of test cases by assigning P and R values from 5 to 20, creating 1 to 8 faulty transitions, with all the other parameters totally random. To create systems with deadlocks, we generate 10,000 cases for each fault number from 1 to 8. After examining the circular wait condition, most of these cases are deadlock-free, therefore the number of deadlock system is in fact much smaller than 10,000. The exact number of deadlock test cases is shown as the second column in Table 1.

The tests are performed on a 2,4 GHz Intel Core 2 Duo processor running Mac OS X 10.6.8. The average time for analyzing the deadlock cases is given in the evaluation column of the table. It shows that the approach is efficient.

Table 1. Efficiency Evaluation

System			Evaluation
Fault Num.	**Test Num.**	**Average State/Transition**	**Average Time (s)**
1	400	4949 / 15440	2.9092
2	517	2428 / 7130	1.1244
3	500	9884 / 31237	3.3533
4	402	8811 / 26663	2.5998
5	303	6756 / 18247	1.2196
6	504	27094 / 75808	5.064
7	757	104857 / 304741	15.0072
8	100	112306 / 283004	15.0289

5.3 Evaluation of Effectiveness

To evaluate the effectiveness, the `EXAM` score is calculated. Its value is the percentage of transitions that have to be examined until the first transition causing the deadlock fault is reached. The `EXAM` score measures the improvement of effectiveness with the help of ranking factor. Without ranking factors, in the worst case, the user needs to check the transitions one by one, until finally find the one with error.

We use an example to illustrated the evaluation method in Fig. 10. In a test case, assume there are 20 transitions in the system, and the transition t_2 is the only cause of failure. After applying the proposed failure analysis approach, the value of C_F ranking factor for each transition is calculated. As the transitions t_2, t_3 and t_4 have equal C_F values, the t_2 will be ranked either as the second position or the fourth position in the whole ranking list. The `EXAM` score is different in these two cases, respectively 10% and 20% We thus distinguish them by defining Fig. 10 (a) as a best case, and Fig. 10 (b) as a worst case.

The effectiveness evaluation is shown in Table 2. For each fault number (from 1 to 8) test cases, we give out `EXAM` score, `EXAM` score variance, rank, and rank variance for the best cases and worst cases, and then show the average `EXAM` score and average rank. The `EXAM` score varies from 2% to 13% for best cases, and varies from 4% to 18% for worst cases. In average, `EXAM` varies from 3% to 16% which

transition	C_F
t1	95%
t2	90%
t3	90%
t4	90%
t5	80%
...	...

(a) best case

transition	C_F
t1	95%
t4	90%
t3	90%
t2	90%
t5	80%
...	...

(b) worst case

Fig. 10. Illustration of Effectiveness Evaluation

Table 2. Effectiveness Evaluation

F. N.	Best Case				Worst Case				Average	
	EXAM	EXAM Var	Rank	Rank Var	EXAM	EXAM Var	Rank	Rank Var	EXAM	Rank
1	0,13335	0,00134	3,25	1,79	0,18603	0,00244	4,33	1,63	0,15969	3,79
2	0,04229	0,00219	1,1	1,75	0,09574	0,00213	2,11	1,75	0,069015	1,605
3	0,02108	0,00106	0,75	1,52	0,05892	0,0009	1,75	1,52	0,04	1,25
4	0,00722	0,0004	0,26	0,49	0,039	0,00042	1,26	0,49	0,02311	0,76
5	0,02044	0,0017	0,83	2,95	0,0478	0,00162	1,83	2,95	0,03412	1,33
6	0,05369	0,00336	2,46	7,36	0,0766	0,0033	3,46	7,36	0,065145	2,96
7	0,08857	0,00372	4,61	10,9	0,10822	0,0037	5,61	10,9	0,098395	5,11
8	0,13091	0,00099	7,3	3,95	0,14905	0,001	8,3	3,95	0,13998	7,8

corresponds to ranking results from 1 to 8. The stability is represented by the variance result. These experimental results shows our approach is effective.

6 Conclusion

Automated failure analysis in model checking is difficult to be computed exactly, due to semantics reduction caused by model abstraction. Yet, it is a key issue, as providing counterexamples is not enough to help designers in debugging faulty designs. It may require a great deal of human effort to locate faulty elements. Some works have provided good results by producing a set of suspicious faulty elements without particular ranking factor.

In our work, inspired by the theory of `Kullback-Leibler Divergence` and its successful application `TF-IDF` in text data mining, we start with comparing the similarity between information retrieval problem for documents and failure analysis problem for model checking, and propose an algorithm to compute the fault contribution of transitions on error traces. The fault contribution is the product of transition contribution (`TC`) and inverse trace contribution (`ITC`). The approach is illustrated using a `BCET` property case study, and then further assessed for its efficiency and effectiveness on a designed deadlock property test bed. The effectiveness is measured by the `EXAM` score.

The automated failure analysis relies on the reachability graph and violations states, thus can be applied to different verification models (`TPN`, timed automata,

etc.) to provide helpful feedback for the assessment of safety properties. The liveness property asserts that something good eventually happens [5]. Examples of liveness properties include starvation freedom, termination and guaranteed service [6]. It will be interesting to study how to apply similar statistical methods to the failure analysis of liveness properties in the future work.

References

1. Alipour, M.A.: Automated fault localization techniques; a survey. Technical report, Technical report, Oregon State University (2012)
2. Ball, T., Naik, M., Rajamani, S.K.: From symptom to cause: localizing errors in counterexample traces. ACM SIGPLAN Notices 38(1), 97–105 (2003)
3. Groce, A., Visser, W.: What went wrong: Explaining counterexamples. In: Ball, T., Rajamani, S.K. (eds.) SPIN 2003. LNCS, vol. 2648, pp. 121–135. Springer, Heidelberg (2003)
4. Groce, A.: Error explanation with distance metrics. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 108–122. Springer, Heidelberg (2004)
5. Lamport, L.: Proving the correctness of multiprocess programs. IEEE Transactions on Software Engineering (2), 125–143 (1977)
6. Alpern, B., Schneider, F.B.: Defining liveness. Information Processing Letters 21(4), 181–185 (1985)
7. Ge, N., Pantel, M.: Time properties verification framework for UML-MARTE safety critical real-time systems. In: Vallecillo, A., Tolvanen, J.-P., Kindler, E., Störrle, H., Kolovos, D. (eds.) ECMFA 2012. LNCS, vol. 7349, pp. 352–367. Springer, Heidelberg (2012)
8. Merlin, P., Farber, D.: Recoverability of communication protocols–implications of a theoretical study. IEEE Transactions on Communications 24(9), 1036–1043 (1976)
9. Berthomieu, B., Diaz, M.: Modeling and verification of time dependent systems using time petri nets. IEEE Trans. Softw. Eng. 17(3), 259–273 (1991)
10. Kullback, S., Leibler, R.A.: On information and sufficiency. The Annals of Mathematical Statistics 22(1), 79–86 (1951)
11. Baker, L.D., McCallum, A.K.: Distributional clustering of words for text classification. In: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 96–103. ACM (1998)
12. Jones, K.S.: A statistical interpretation of term specificity and its application in retrieval. Journal of Documentation 28(1), 11–21 (1972)
13. Wong, W.E., Debroy, V.: A survey of software fault localization. University of Texas at Dallas, Tech. Rep. UTDCS-45-09 (2009)
14. Coffman, E.G., Elphick, M., Shoshani, A.: System deadlocks. ACM Computing Surveys (CSUR) 3(2), 67–78 (1971)