Software robustness analysis by a probabilistic technique

Technical report

Sergey Frenkel[1], Victor Zakharov[1], Boris Basok[2]

[1]Federal Reserach Center "Computer Science and Control " Russian Academy of Sc., Moscow, Russia, fsergei51@gmail.com, VZakharov@ipiran.ru
[2]Moscow Technological University “MIREA”

Abstract. The report shows the possibility to use a two-dimensional Markov model defined on direct product of state spaces of two finite state machines (FSM), one of which is a program finite automaton model that is running under normal conditions, and second is the same FSM in which at some point in time (Depending on the considered temporal discreteness) there was a failure (e.g., within the time of a single operation, or a program's block execution) for estimation of software robustness to transient hardware failures or rather short-term external attacks. This model previously was proposed for probabilistic verification of hardware systems. The robustness of the program is estimated by probability of the faulty FSM return on the path of transitions of the source machine after termination of the failure.

The model is analyzed in detail for an example of block-diagram of a specific program.

Keywords: fault tolerance, Markov chains, Finite State Machine**,** design verification.

1 Introduction

The problem of accounting for the effect of failures on the operation of computer systems, both hardware and software subsystems, remains one of the very important for the theory and practice of designing computer systems.

Failures can occur both in this or that part of the program memory area, and outside it, both due to some physical effects (e.g., some radiation) and some malicious attacks. They can distort both the data and the contents of the program (values of variables, constants, operation codes, etc.).

Since the main way of the correctness assessment of the complex system operations, including in conditions of failure, is testing for those or other Sets of programs and data (benchmarks, for example, as in [1], or functional tests), it would be very useful for the possible consequences of these failures to be evaluated and verified during the development of the CS.

The report shows the possibility of using the two-dimensional Markov model of finite automaton behavior in order to assess the stability of programs for short-term failures, which has previously been proposed for probabilistic verification of hardware systems [3, 4]. A model is a product of Markov chains corresponding to a finite-automaton model of a program operating under normal conditions, and an automaton in which a momentary (for example, within the execution time of one operation, one machine cycle or some specific time window), failure occurs. Resistance to failures is estimated by the probability of returning the specified product chain to a path corresponding to the normal functioning of the machine.

We are considering this approach to assess the likelihood of a malicious action of failures on a given program being developed as an alternative to the widely used FI technique.

In the FI technique, the application program is emulated by the instruction-by-instruction, while the source operands are randomly modified before the instruction is executed, and the result of the execution of the instruction is also accidentally distorted, thereby simulating the error action. A software counter is also accidentally modified if it is a branch instruction. The resulting sequence of such modifications of the instructions and the counter is traced to determine the level of security of this program to specified failures (soft error rate - SER).

In fact, we simulate the action of the injected fault in violation of the execution of a certain transition in the above-mentioned automaton.

Since the effect of short-term malicious attacks on the computing system is also modeled by injecting a fault, we can apply our approach also to assess the resistance to attacks.

One of benefit of our approach is that in contrast to FI based methods which requires developers to have rather expensive software, which is not used to solve other numerous design problems, in particular verification and functional testing [2], We can consider our method as part of the overall process of designing a computer system using generic design tools.

2. Brief Description of the Program External-caused Failure Model

At the heart of the application program model considered in this article is the finite Mealy automaton, corresponding to the algorithm implemented by this program. In this paper, we will not consider the formal aspects of using this model-we will refer the reader to [6] and the article of the authors [7], but give only explanations necessary for further understanding.

In terms of this automaton model, the execution of the program is modeled as transitions between the states of the corresponding automaton [6]. A directional arc in the diagram (graph) of the automaton represents the sequence of execution of the

program from state to state. The inputs of Mealy model can be flags of events or conditions. Outputs are computed variables.

Based on the finite automaton model of the program, we define the concept of failure. Informally, under the failure of the program we will understand the computational errors caused by random changes in any of the bits of a certain machine word from the binary image of the program, that is, by replacing '0' with '1' or '1' with '0 ', which can be caused by various external causes, including malicious attacks In what follows we will assume that such a change leads to a change in one or another state of the automatic representation of the program, but does not lead to the appearance of new states.

We call the event HS(t) as the self-recovery of the automaton after a malfunction that occurred at the clock t_0, if $t > t_0$ is the first following the automaton clock cycle at which A_f (t) = A(t) and Y_f (t) = Y(t), where A(t) and A_f (t) are the states of the Markov chain of the fault-free faulty automata at the clock t, and Y (t) and Y_f (t) are the corresponding output value vectors.

In this case, the chain has one more absorbing state, which can be determined in one of two ways:

- it corresponds to the distortion of at least one output variable before the return of the trajectory of the transitions of the automaton into a state corresponding to the normal one;
- it corresponds to the distortion of the outputs when the states of the serviceable and faulty automata coincide.

The matrix of transition probabilities is calculated from the given table of automaton transitions and the probabilities of Boolean units of input binary variables of the automaton (since the transition to a given state is determined by one or another operation over the current value of the input variable and the current state). We can determine the number of time slots to possible return to correct functioning by calculating the probability vector of the states into which the two-dimensional Markov chain (MC) falls due to interference for t transitions of the automaton:

$$\vec{p}(t) = \vec{p}(t-1)P^* = \vec{p}(0)(P^*)^t, \quad (1)$$

where the initial distribution $\vec{p}(0)$is determined by the initial states of the serviceable and faulty automata, and P* is the transition matrix of this two-dimensional Markov chain.

If the valid automaton at the initial moment 0 is in the state i_0, and the faulty state is in the state $j_0 \neq i_0$, then $p_{i_0,j_0}(0) = 1$, and the remaining coordinates of the vector $\vec{p}(0)$ are zero.

The components of the vector p (t) are the probabilities of getting into the absorbing state A_0 corresponding to the restoration of the correct functioning after the interruption of the interference, the probability $p_1(t)$ of the absorbing state A_1 (determined by one of the above methods),and the probability of transitions to the rest (transient) states of the MC,in the sum equal to $1 - p_0(t) - p_1(t)$.

A detailed mathematical analysis of the model is carried out in [3]. Here, only an illustrative example is provided that helps to better understand the sensitivity analysis of a particular program described below.

Consider the automaton given in Table. 1, where a_t, a_s are the current and next states of the automaton, $X = (x_1, x_2, x_3)$, $Y = (y_1, y_2, y_3, y_4)$ are binary input and output variables, and the absence of one or another component of the vector X in (0, 1, 1), and also the input vectors (0, 0, 0), (0, 0, 1), (0, 1, 0) corresponding to the cell of table with $\overline{x_1}$, applied to state a_1, lead the automaton to the state a_2 and form the vector (0, 1, 0, 1) at the output, and so on.

Recall that if a valid automaton at the initial moment 0 is in the state i_0, and the faulty state is in the state $j_0 \neq i_0$, then $p_{i_0,j_0}(0) = 1$, and the remaining coordinates of the vector are zero.

Table 1. Automaton with two states

a_t	a_s	X	Y
a_1	a_2	$\overline{x_1}$	y_2y_4
	a_2	$x_1\overline{x_2}$	y_2y_3
	a_1	x_1x_2	y_2y_4
a_2	a_1	$\overline{x_2}$	y_2y_4
	a_1	$x_2\overline{x_3}$	y_1y_4
	a_2	x_2x_3	y_2y_4

For a given automaton, the matrix of transition probabilities is calculated from a given table of automaton transitions and the probabilities of Boolean units of input binary automaton variables.

In this case, depending on the choice of the conditions that determine the two-dimensional MC approach to the absorbing state A1, we can consider different models of the matrix of transition probabilities P. Let us consider, using the example of this automaton, the two possible models.

The model 1. The MC gets into the absorbing state A_1 in all cases of the mismatch between the outputs of the serviceable and faulty automata.

The transition probability matrix of this two-dimensional MC is shown in Table. 2, where $p_i = \text{Prob}(x_i = 1)$, $i = 1, 2, 3$, $q_i = 1 - p_i$.

Table 2 Transition probability matrix of the automaton of Table 1

	A_0	(1,2)	(2,1)	A_1
A_0	1	0	0	0
(1,2)	$q_1p_2p_3$	$p_1p_2p_3$	q_1q_2	$p_1q_2 + p_2q_3$
(2,1)	$q_1p_2p_3$	q_1q_2	$p_1p_2p_3$	$p_1q_2 + p_2q_3$
A_1	0	0	0	1

The states A_0, A_1 (Table 2) correspond to the above absorbing states of a two-dimensional MC. The states of the indicated MC (1, 2), (2, 1) mean that in the initial state the automaton must be in the state a_1 (a_2), but as a result of a malfunction (error, interference) it turns out to be a_2 (a_1). Obviously, if the number of states of the initial automaton is N, then the size of the two-dimensional MC will be N (N-1) + 2, and in this example it will be 4.

Let us explain in detail the construction of the transition probability matrix of two-dimensional MC for the automaton of Table. 1, presenting for greater clarity all the information about the possible behavior of a pair of non-defective and failing automata in the form of table. 3.

Table 3. Possible transitions of pairs of states of non-faulty and faulty automata

a_t \ X	$x_1 x_2 x_3$	$x_1 x_2 \bar{x}_3$	$x_1 \bar{x}_2 x_3$	$x_1 \bar{x}_2 \bar{x}_3$	$\bar{x}_1 x_2 x_3$	$\bar{x}_1 x_2 \bar{x}_3$	$\bar{x}_1 \bar{x}_2 x_3$	$\bar{x}_1 \bar{x}_2 \bar{x}_3$
a_1	$a_1/\ y_2y_4$	$a_1/\ y_2y_4$	$a_2/\ y_2y_3$	$a_2/\ y_2y_3$	$a_2/\ y_2y_4$	$a_2/\ y_2y_4$	$a_2/\ y_2y_4$	$a_2/\ y_2y_4$
a_2	$a_2/\ y_2y_4$	$a_1/\ y_1y_4$	$a_1/\ y_2y_4$	$a_1/\ y_2y_4$	$a_2/\ y_2y_4$	$a_1/\ y_1y_4$	$a_1/\ y_2y_4$	$a_1/\ y_2y_4$
(1, 2)	(1, 2)	A_1	A_1	A_1	A_0	A_1	(2, 1)	(2, 1)
(2, 1)	(2, 1)	A_1	A_1	A_1	A_0	A_1	(1, 2)	(1, 2)

Here, in contrast to Table. 1, all the components x_1, x_2, x_3 of the input vector are specified, including those that do not affect this transition.

Suppose that at the initial instant of time, because of the action of (undetected) interference, the automaton (Table 1) instead of the state is in a state. The general CM Zt, describing the joint functioning of the serviceable and faulty automata, has states: transient (1, 2), (2, 1) and absorbing A_0 and A_1. Here (1, 2) – fault-free and faulty automata are respectively in the states and, (2, 1) - in the states and, A_0 - the event "by the moment the automaton's trajectory has been restored (it was in the correct state) and the output is not Were distorted ", A_1 - malfunctioning has already manifested itself in the output signal. In contrast to the absorbing states, transitions are possible for states (1, 2) and (2, 1).

We calculate the transition probabilities of the CM Z_t (Table 2). The states A_0 and A_1 are absorbing, and therefore the probability of remaining in each of these states is 1.

From the state (1, 2) to the state A_0 it is possible to get only in the event that the signal (with probability) arrives. Then both automata (fault-free and faulty) will go to the same state, and the output signal for both automata. From the state (1, 2) to the state (1, 2) it is possible to hit only in the event that the signal (with probability) arrives. Then the serviceable machine remains in the state, the faulty state is in the state, and the output signal for both automata is y_2y_4. The probability of a transition from the state (1, 2) to the state (2, 1) is calculated similarly. From state (1, 2), state A_1 can be accessed with different input signals. First, if a signal arrives (with probability, the third component of the input signal can be any), then the output signal of the serviceable automaton is, for the faulty one, y_2y_4 (while the serviceable state machine goes from state to state, faulty - from state to state).

Secondly, if there is a signal (with probability), then the output signal of the serviceable automaton is, for the faulty one (both automata get into the state). Thirdly,

if a signal (with probability) arrives, then the output signal of the serviceable automatic device is, for the faulty one.
Let P1 = 0.2; P2 = 0.4; P3 = 0.25.

For the initial state vector $\vec{p}^*(0) = (0, 1, 0, 0)$ after the first step

$$\vec{p}^*(1) = \vec{p}^*(0)P^* = (0.08, 0.02, 0.48, 0.42)$$

Accordingly, the probability $p_0(1)$ of falling into the absorbing state A_0, at which the automaton, upon removal of the short-term perturbation that caused the malfunction, will return to the trajectory of the transitions of the serviceable automaton before the change in at least one output y1, y2, y3, y4, 0.08, the probability p1 (1) to get into the absorbing state A1 (the output was distorted before returning to the correct trajectory) is 0.42 (the product of the row of initial states on the 4th column of the transition probability matrix corresponding to the hit in A1), and the chain Markov remains in the subset of ergodic states (1, 2), (2, 1) with probability

$\vec{p}^*(1) = 0.02 + 0.48 = 0.5$.

After second step:

$\vec{p}^*(2) = \vec{p}^*(1)\ P^* = (0{,}1200;\ 0{,}2308;\ 0{,}0192;\ 0{,}6300)$.

and so on in accordance with the Table 4.

Table 4. Probabilities to reach absorbing and ergodic states through t steps

	$t=0$	$t=1$	$t=2$	$t=3$	$t=4$	...	$t=28$
$p_{(0)}(t)$	0,000000	0,080000	0,120000	0,140000	0,150000	...	0,160000
$p_{(1)}(t)$	0,000000	0,420000	0,630000	0,735000	0,787500	...	0,840000
$\dot{p}(t)$	1,000000	0,500000	0,250000	0,125000	0,062500	...	0,000000

The model 2. The occurrence of the Markov chain in the absorbing state A_1 only in the case of a mismatch between the outputs of the serviceable and failing automata when the failing automaton hits a state that coincides with the state of the faulty automaton.
This model corresponds to the transitions in Table. 5, in which the absorbing state A_1 corresponds to the mismatch of the outputs when two automata are in the same state (a1 in this example).

Table 5. Possible transitions of pairs of states of non-faulty and faulty automata (Model 2)

a_t \ X	$x_1 x_2 x_3$	$x_1 x_2 \bar{x}_3$	$x_1 \bar{x}_2 x_3$	$x_1 \bar{x}_2 \bar{x}_3$	$\bar{x}_1 x_2 x_3$	$\bar{x}_1 x_2 \bar{x}_3$	$\bar{x}_1 \bar{x}_2 x_3$	$\bar{x}_1 \bar{x}_2 \bar{x}_3$
a_1	$a_1/\ y_2y_4$	$a_1/\ y_2y_4$	$a_2/\ y_2y_3$	$a_2/\ y_2y_3$	$a_2/\ y_2y_4$	$a_2/\ y_2y_4$	$a_2/\ y_2y_4$	$a_2/\ y_2y_4$
a_2	$a_2/\ y_2y_4$	$a_1/\ y_1y_4$	$a_1/\ y_2y_4$	$a_1/\ y_2y_4$	$a_2/\ y_2y_4$	$a_1/\ y_1y_4$	$a_1/\ y_2y_4$	$a_1/\ y_2y_4$
(1, 2)	(1, 2)	A_1	(2, 1)	(2, 1)	A_0	(2, 1)	(2, 1)	(2, 1)
(2, 1)	(2, 1)	A_1	(1, 2)	(1, 2)	A_0	(1, 2)	(1, 2)	(1, 2)

The probabilities of achieving absorbing and ergodic states through t steps (similar to Table 4) are presented in Table 6.

Table 6. Probabilities of attaining absorbing and ergodic states through t steps (Model 2)

	t = 0	t = 1	t = 2	t = 3	t = 4 …		t = 28
p0(t)	0,000	0,08	0,1488	0,208	0,2589	0,563	
p1(t)	0,000	0,06	0,1116	0,156	0,1941	0,423	
p'(t)	1	0,86	0,7396	0,636	0,547	0,014	

Depending on the purpose of the systems and the approaches to their design, one or the other of the two semantics of the behavior of systems under fault conditions may be more preferable. This issue is discussed in more detail below on a concrete example.

Let us see how we can adapt the above models to assess the likelihood of incorrect program behavior due to malicious attacks, where the FSMs are models of system calls trace corresponding to the program execution like in [11]. We will consider the product of two above normal and faulty ("malicious") FSMs, and corresponding two-dimensional Markov chain as well. That is the models assume that when the attacker injects a fault into the system, an erroneous result will be observed at the output. We consider, that every event in the system (e.g., network) is applied to finite state machine instances which finally results in transition. An attack will be occurred when the machine reaches a state with different outputs**,** like in the models mentioned above. In particular, if we consider binary outputs and states (codewords), then the error can be expressed as an XOR operation (˜Y=Y $\oplus$ e), states A=A+e, where e is a given Boolean vector, describing the bits corruption. Thereby, if we consider a malware with obfuscated codes, the Model 2 could be more relevant, as the states coinciding can be result of the obfuscation.

The system calls may be considered as input sequence of the FSM and the probabilities of the different transitions may be based on the frequency of certain system call sequences during a healthy execution (see Sections 4,5).

3. Example of Program Stability Analysis for Failures

Consider some utility program designed to process successive (ASCII) symbols transmitted through a transmission link. Let the information be structured by strings(blocks) and there are spaces between the blocks in order to define a condition of now information in this data recording medium. This utility program should remove the spaces from character strings according to the following conditions:

- leading (at the beginning of the string) spaces are suppressed;
- if the string ends with spaces, then they are also deleted;
- within a string there can not be a sequence containing more than one space in a row, extra spaces are deleted.

In order to simplify the analysis, we will consider a block diagram of the program for processing only one string (S1), meaning that the program includes the means of transition to reading and processing the next string from the entire set of strings processed by this program text.
Note that this program can be considered as a model of various programs designed to remove some service symbols in the stream of transmitted symbol information, for example, the signs of the beginning and the end of the transmission of a data block in telecommunication applications (We consider the alphabet as $L_D \cup L_S$, Where D-symbols are used for data transmission, S-some service symbols or their combinations used for). For example, in protocols such as IBM'S SDLC where a string "1" bits is broken by an inserted '0' bit to avoid confusing data and SYN characters. Once received, the inserted 0 is removed.

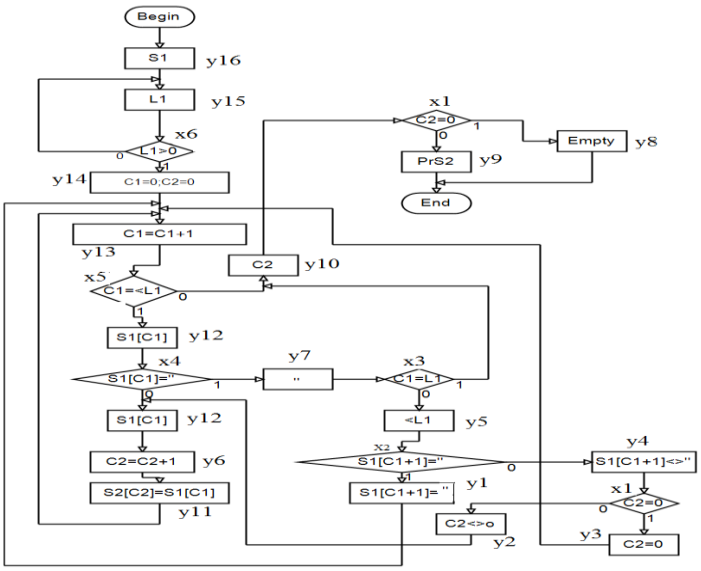

Figure 1. ASM block diagram of the utility program

According to the purpose of this article formulated in the introduction, let us consider the problem of calculating the probabilistic estimation of the possibility of self-recovery of the program module in question after a failure - in this case the question is about the recoverability of the operation when the operation is performed incorrectly.
This obviously depends not only on the structure of the module, but also on the probability of performing certain transitions, that is, on the input information.

Imagine an algorithm that implements this program as a diagram of the Algorithmic State Machine (ASM) [8] (the flowchart in Figure 1).
In the above block diagram, S1 is an array containing the processed string, from which control characters that are not included in the alphabet A are deleted, the length of this array is L1 (the mechanism of reading and buffering is not considered here, considering the whole process of buffering by one operation corresponding to the

rectangle S1 block -scheme). If the length of the string is zero (for example, contain only special control characters (including some end-of-line character) and there is not one character from the alphabet A), then the next string is entered. C1 is the index in array S1, C2 is the index in array S2, in which characters of a string without deleted spaces are written and in which the result of program work is formed (Denoted as PrS2). In case the string consisted of only spaces, the user receives an Empty message.
In this flow chart, the rectangles correspond to the execution of some operation, and the rhombs correspond to the transition conditions. With the help of the Abelite program [8], it is possible to obtain a finite-automaton representation of a block diagram in the form of a Mealy automaton, in which the binary input variables correspond to the path selection conditions in the block diagram of Fig. 1 (from the tops of the rhombus "0" or "1"), and each execution of operations is associated with the transition from state to state (for more information about the Abelite program and its methodology for designing digital systems, see [7, 8]). Note that the program Abelite itself determines by ASM-diagram of the state of the machine being built. However, it is necessary to take into account the following circumstances:

1.Since the model of the two-dimensional Markov chain of the product of automata described in the preceding section assumes the independence of the random inputs of automata, and in the automata obtained from the ASM diagrams, the input variables correspond to conditional transitions, it is necessary that the indicated transitions be performed according to the conditions determined by the independent events.

The automaton built by the Abelite program is shown in Fig. 2.

a1	a13	1	y16
a2	a11	1	y13
a3	a9	1	y12
a4	a2	x1	y3
a4	a3	~x1	y2
a5	a2	x2	y1
a5	a4	~x2	y4
a6	a2	1	y11
a7	a8	x3	y10
a7	a5	~x3	y5
a8	a1	x1	y8
a8	a1	~x1	y9
a9	a6	1	y6
a10	a7	x4	y7
a10	a9	~x4	y12
a11	a10	x5	y12
a11	a8	~x5	y10
a12	a2	x6	y14
a12	a12	~x6	y15
a13	a12	1	y15

Figure 2. Mealy automaton from Figure 3.

2. Since synchronous automata are considered in this fault model, each result of the change in the input data (selection of the transition to ASM) must be recorded in one or another state of the machine, which means that the output of each rhombus is associated with a certain rectangle.

Here the first column is the previous state, the second is the next state, the third is the logical values of the inputs x1-x6, which cause the corresponding transition, the last are the outputs of the automaton formed during this transition. The correspondence of the outputs and inputs of the automaton to the flowchart shown in Fig. 1, describes Fig. 3.

Output variables (Micro Operations):

y1 : S1[C1+1]="
y2 : C2<> 0
y3 : C2=0
y4 : S1[C1+1]<>"
y5 : <L1
y6 : C2=C2+1,
y7 : "
y8 : Empty
y9 : PrS2
y10 : C2
y11 : S2[C2]=S1[C1]
y12 : S1[C1]
y13 : C1=C1+1
y14 : C1=0;C2=0
y15 : L1
y16 : S1

Input variables (Logical Conditions):

x1 : C2=0
x2 : S1[C1+1]="
x3 : C1=L1
x4 : S1[C1]="
x5 : C1=<L1
x6 : L1>0

Figure 3. The correspondence between the input and output variables of the automaton to operations and transitions in

Here the <> sign replaces the symbol ≠, the use of which (like some other characters from the MS Windows symbol table) prohibits the interface of the Abelite program [8] used to build the machine.

The states a1-a13 describe the states of the constructed, y1-y16 are the output variables of the automaton, corresponding to the execution of operations in the rectangles of the flowchart consisting either of assigning the values of one variable to another, or of performing the specified actions directly. Variables S1 [C1 + 1] = ",", C2 <> 0 represent the marked "synchronizing" operations, which are inserted into the diagram to separate the changes in the inputs of the machine. These pseudo-operations simply fix the state of the machine to which it has moved after selecting the appropriate transition.

The state of the automaton can easily be interpreted by comparing Fig. 2 and 3. For example, the state a4 is a state where a word is found in the word, followed by a non-blank space. If this occurs at the beginning of a word, then the space is not written to the word S2 being formed, otherwise it is written.
Since malfunctions are treated as events that occur randomly during processing of a set of rows S in a given alphabet, the probabilities of events are specified on the set of all possible strings containing alphabet symbols and on the set of symbol indices inside each of the strings.
To construct the desired Markov model (Section 2), it is necessary to assign each of the automaton (x1, ..., x6) to the probability of the Boolean unit, which corresponds to the probability of activation of The corresponding path in the block diagram of Fig. 1 over the unit branch. These probabilities are estimated by the sets of processed texts ST $\subseteq$ S, considered as a finite sample of N_T from the set S.
For example, the probability that x6 = 1 is evaluated as Prob (L1> 0) = N_{emp}/NT, where N_{emp} is the number of empty strings, N_T is the total number of rows in the test set.

Consider the application of this model in analyzing the sensitivity of the test and the stability of the program to failures, namely by calculating the probability values of the hit of a two-dimensional Markov chain constructed from a pair of automata with different initial conditions, one of which corresponds to a change in the initial state vector as a result of the failure, to the absorbing state. This is an estimation of the probability of the absence of empty strings in the processed stream, for example, consisting only of the beginning and end attributes and not containing any symbol of the used alphabet (the construction of the test set was considered in [4]).

For the considered example, the total number of states of the two-dimensional Markov chain {(i, j)}, where i, j are the indices of the states ai, aj of the serviceable and the automaton subjected to failure (the "failed" automaton), as indicated in the previous section, is N (N - 1) + 2, where N is the number of states of the automaton in Fig. 2, and is 158. The states are ordered as follows:
(1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (1, 9) , (1, 10), (1, 11), (1 , 12), (1, 13), (2, 1), (2, 3), (2, 4), ..., (5, 6) , ..., (13, 11), (13, 12), A_1}.

The vector of probabilities of the initial states $\{p_{ij}\}_{158}$ has a unique unit component (i_0, j_0): if the effective automaton at the initial instant 0 is in the state i_0, and the faulty state is in the state $j_0 \neq i_0$, then $p_{i0,j0}(0) = 1$, and the remaining coordinates of the vector are zero. For example, for a failure of the form (5, 6), the vector will contain one at position (5, 6).

4. Failure Sensitivity Analysis based on Testing Results

Let there be some set of strings S1 representing a set of tests for functional testing of the program. Let probabilities Prob (x1 = 1), ..., Prob (x6 = 1) be defined for a given test set of texts (or a wider set of texts, considered as a training sample).
As noted in Section 3, for each automaton model of the program one can construct two different Markov chains, one of which has an absorbing The state A_1 is determined by the subset of the transitions of the chain Z_t, in which the outputs of the

serviceable and faulty automata do not coincide (Model 1), and in the other Markov chain the state A1 corresponds to transitions in which the outputs coincide when the states coincide (Model 2). The use of this or that model depends on the specific task to be solved.

Consider the problem of determining the probable interval of self-healing. In this case, Model 2 is obviously adequate, since Model 1 means checking the mismatch of outputs for each state, which in some cases may not allow the self-healing effect to be realized, since the program can be stopped already at the first clock cycle after a failure, whereas many Algorithms of self-healing and self-correction just suggest the presence of some time for self-healing [9].

Next, we consider several examples of the analysis of the automaton given in Table. 5, for sensitivity and resistance to malfunctions.

Example 1. Consider failure (10, 8) "the automaton instead of the state a10, in which it should be on this clock after the transition at the previous clock cycle, as a result of the malfunction was in the state a8", which means, according to Fig. 2, the execution of operation y10 instead of y12 and, accordingly, the formation of the resulting string S2 can be completed before the exhaustion of the characters in the input string S1.

In this case, the probability table for the recovery of $\vec{p}(0)$, the appearance of a malfunction at the outputs $p_1(t)$ and the finding on an incorrect trajectory of transitions for undistorted output variables ~ p (t) is calculated by formula (1) as the product of the above probability vector of initial states on the t-th degree of the probability matrix of the transitions of the two-dimensional Markov chain, calculated from the transition probabilities, as shown in the example of the automaton of Table. 1-3. For this example, analogous matrices 158×158 are constructed by the program [10].

We take a certain vector of probabilities of the unit values of the inputs x1, ..., x6 (without asking for a connection between their values and the semantics of the program), for example, 0.2, 0.7, 0.2, 0.7, 0.01, 0.2, And calculate the value of the probabilities of self-healing at steps (cycles) t = 1, ..., 19 by formula (1), representing them as a vector:

P0(t) = [0, 0, 0, 0, 0, 0,3136, 0,3136, 0,31395, 0,45906, 0,57226, 0,64495, 0,69991, 0,75712, 0, 78388, 0.82202, 0.85148, 0.87607, 0.89468, 0.91082, 0.992396],

where the number of each position corresponds to the tick number minus 1 (the first element corresponds to the moment of failure and is considered a zero index).

As can be seen from the probability vector of self-healing P0(t), the probability of recovery is already high at the 12th clock after the failure (0.757).

Let us analyze the connection between probabilities x1, ..., x6 with the semantics of the applied problem in Fig. 1. As noted above, the probabilities of the input variables x1, ..., x6 are determined by the distribution of spaces in the texts that form the test set of rows. The probability of the event C2 = 0 (represented by the variable x1) means the probability that at a randomly chosen step of the algorithm (on the whole set of test strings) it turns out that no leading space has been encountered.

In this example, this probability is 0.2, which indicates a sufficiently high density of spaces in the strings of the set. The same is said about the probability value for variables x2, x4, corresponding to the frequency of spaces in the test strings. The probability value for x3 is the probability of encountering at the random step the end

of the string, which in this case is small. Obviously, the variables x5 and x3 ("C1 = L1") are logically related (the greater the probability of encountering the end of the string, the less likely that the string is (relatively) long, at a randomly chosen time moment, the condition C1<L1) which poses the question of the possibility of considering them independent. Note that if, as noted above, to consider this program for analyzing the presence and removal of gaps as a model of a telecommunications program for which spaces correspond to certain service symbols, then the condition L1 = 0 means that there is no information symbols in some premise, and the small probability values L1 > 0 mean that information packages are quite rare.

We give examples of calculating the probabilities of self-healing also for other failures and for different probabilities x1, ..., x6.

Example 2. For the failure (10, 8) and the probabilities of inputs 0.05; 0.7; 0.1; 0.05; 0.9; 0.1

$P_0(t)$ = [0, 0, 0, 0, 0, 0.00405, 0.00405, 0.0040898, 0.062046, 0.062999, 0.066632, 0.068139, 0.069252, 0, 10226, 0.10411, 0.10778, 0.11016, 0.1119, 0.13258, 0.13501].

Example 3. For failure (7, 9) with the same input probabilities, the recovery probability is extremely small:

$P_0(t)$ = [0, 0, 0, 0, 0, 0, 0, 0.00729, 0.00741, 0.010307, 0.010507, 0.0124, 0.01847, 0.018905, 0, 02131, 0.021632, 0.0242, 0.0295, 0.03004, 0.03211].

5. Discussion of Results and Conclusion.

Thanks to the proposed approach, one can judge the stability of programs to accidental failures and, if necessary, use additional tools and methods for restoring programs. In this case, the construction of the Markov model (the transition probability matrix) can be performed according to the earlier developed functional tests and does not require additional costs for constructing new tests.

The probability of self-recovery for the program under consideration in the general case may be small, which raises the question of how the information obtained in the self-recovery analysis described can be used by the designer to obtain program variants that are resistant to accidental failures. Let's consider the basic factors of stability to failures which the designer of the software-hardware system, investigated by means of the given model of the program should take into account.

First of all, as it can be seen from the analysis of the considered examples, the applied purpose and functional features of the projected system, which are displayed in the test sets used, are essential. For example, for the program in question, if based on the original design specification of a certain system of which the program is a part, it follows that most of the data at the program input (string S1) does not contain the symbols of the specified (used) alphabet, then the probability of self-recovery is very high.

If this is not the case, and the probabilities of self-healing on the test set under consideration are small (example 3), the designer decides on various methods of providing protection. Depending on the project quality criteria used (cost, energy consumption, weight, dimensions, reliability, etc.), it decides that it is more effective to significantly reduce the intensity of failures and/or apply some self-healing schemes. A simplified assessment of the impact of hardware failures can be presented as follows. Let L_n be the fault list ((10,8), (7,9), etc. in the examples considered).

Then, estimating the probabilities of hardware failures [1] p_i, i = 1, ..., $|L_n|$, which, for example, affect the corresponding areas of program memory responsible for failures from $P_t = \Sigma_{i=1,..,|Ln|} p_i P_o^i$, where P_o^i are the components of the vectors $P_0(t)$ corresponding to the cycles 1, 2, ..., t for each of the faults *i* from the list L_n, examples of which are given in the previous section and which in this case are interpreted as conditional probability of failure, Failure *i*, and the formula given is a formula for the total probability of the event "a failure" occurred.
Having a list of faults and calculated probabilities of self-healing, it is possible to formulate requirements for the equipment to ensure permissible pi. Further, the developer uses some or other approaches to software stability to failures, for example, N-version programming, in which n (where n is an odd number) copies (versions) of the program is written to the memory with the subsequent comparison of the results "by the majority".

Acknowledgement. This work has been partially supported by the Russian Foundation for Basic Research under grant RFBR 15-07-05316.

References

1. *Li X., Adve, S. V., Bose, P., Rivers, J. A.* Softarch: An architecture level tool for modeling and analyzing soft errors. In Proceedings of the International Conference on Dependable Systems and Networks (DSN), Yokohama, Japan, June 2005, pp. 496–505 (2005).

2. Darbar, A., Al-Hashimi, B., Harrod, P., Bradley, D.: A New Approach for Transient Fault Injection using Symbolic Simulation. In: IOLTS 2008: Proceedings of the 14th IEEE International On-Line Testing Symposium. pp. 93–98 (2008)

3. Frenkel, S., Pechinkin, A.: Estimation of self-healing time for digital systems under transient faults.: Informatics and its Applications Journal, 4, 3 pp. 2–8 (2010)

4. Frenkel, S.,. Zakharov, V., Ushakov, V.: Probabilistic verification in the design of computing systems. In: International conference Tools and Methods of Programs Analysis (TMPA-2014): Kostroma, Russia, 14–15 November 2014), pp. 148–155 (2014)

5. Li, X., Yeung, D.: Application-level correctness and its impact on fault tolerance. In: The Sixteenth International Symposium on High-Performance Computer Architecture, Bangalore (India), January 2010. pp. 220–225 (2010)

6. Shalyto, A., Tukkel, N.: SWITCH-technology - Automatic approach to software development of "reactive" systems. Programming, 5,. pp. 45–62 (2001)

7. Baranov, S., Frenkel, S., Zakharov, V.: Semiformal verification for pipelined digital designs based on Algorithmic State Machines. Informatics and its Applications Journal, 4, 3, pp. 49–60 (2010)

8. Baranov, S.: ASMs in high level synthesis of EDA tool Abelite. In: DESDes'09 Int. IFAC Workshop Proceedings. – Valensia, Spain, pp. 195–200 (2009)

9. Lala, P., Kumar, B.: On self-healing digital system design. J. Microelectronics, 37, pp. 353–362 (2001).

10. Frenkel, C., Lyburkin.,D, Program for evaluating the time of self-recovery of a digital system after a failure by its high-level model. Certificate of state registration of the program № 2013661815 of 16.12.2013 (2013)

11. Jacob, G, · Debar, H., · Filiol, E. Behavioral detection of malware: from a survey towards an established Taxonomy, Comput Virol, 4, pp.251–266 (2008).