Modeling Input Space for Testing Scientific Computational Software: A Case Study

Sergiy A. Vilkomir[1], W. Thomas Swain[1], Jesse H. Poore[1], and Kevin T. Clarno[2]

[1] Software Quality Research Laboratory,
Department of Electrical Engineering and Computer Science, University of Tennessee,
Knoxville, TN 37996, USA
{vilkomir,swain,poore}@eecs.utk.edu
[2] Reactor Analysis Group, Nuclear Science and Technology Division,
Oak Ridge National Laboratory, Oak Ridge, TN 37923, USA
clarnokt@ornl.gov

Abstract. An application of a method of test case generation for scientific computational software is presented. NEWTRNX, neutron transport software being developed at Oak Ridge National Laboratory, is treated as a case study. A model of dependencies between input parameters of NEWTRNX is created. Results of NEWTRNX model analysis and test case generation are evaluated.

1 Introduction

Testing scientific computational software has been the subject of research for many years (see, for example, [3, 4, 6, 7]). Because of the increase in size and complexity of such software, automation of scientific software testing is very important. A part of this task is test case selection from a large input space.

For test automation, model-based approaches are most effective. In particular, we use Markov chain models to support automated statistical testing [13, 14]. The method and tools were initially applied to systems where sequences of discrete stimuli cause software responses and changes in the state of use. However, the behavior of computational software is more typically a function of a large multi-parameter static input space rather than sequences of discrete stimuli. Often all input parameters are entered as a batch, and then the software computes results with no further interaction with users.

Although other testing methods (combinatorial testing [2, 5], etc.) are applicable to this problem, they are not as directly supportive of automated testing as the method based on directed graphs and Markov chains. We considered this problem in [12] where a method based on dependency relations encoded into Markov chain models was described. When mutual dependencies limit the set of valid input combinations, the method captures only the valid combinations. In this paper, we consider an application of this method to scientific software[1]. The case study is NEWTRNX [1] – a neutron transport simulation developed at Oak Ridge National Laboratory (ORNL)[2].

[1] This research is supported by the University of Tennessee Computational Science Initiative in collaboration with the Computing and Computational Sciences Directorate of Oak Ridge National Laboratory.
[2] Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No. DE-AC05-00OR22725.

M. Bubak et al. (Eds.): ICCS 2008, Part III, LNCS 5103, pp. 291–300, 2008.

For NEWTRNX, as in most multiphysics simulations, the large number of input parameters and values makes manually selecting an arguably sufficient set of test cases very challenging. As a first step towards test automation, a model of dependencies among NEWTRNX input parameters is created. Then special tools can be used to generate test cases automatically from the model. In our test automation, we used JUMBL [10] – the J Usage Model Builder Library developed in the Software Quality Research Laboratory (SQRL) at the University of Tennessee.

Sec. 2 presents a brief review of the method for modeling the input space. In Sec. 3, we describe the neutron transport software in very general terms. In Sec. 4, we apply our method to this specific case to create a model of dependencies among NEWTRNX input parameters. Sec. 5 contains results of model analysis and test case generation for the model from Sec. 4, using the JUMBL. Conclusions are discussed in Sect. 6.

2 Modeling Input Space for Software Test Case Generation

Modeling an input space for the purpose of test case generation was presented in [11] for independent input parameters and in [12] for the situations when dependencies between input parameters exist. The basis of the method is representation of the input space as a directed graph, where nodes represent input parameters and incoming arcs represent specific values of the parameters. Any path through the graph (a sequence of arcs) corresponds to a test case, so the graph can be used as a model for test case generation.

From a usage point of view, there is some probability (relative frequency) that the parameter will have a particular value. We associate these probabilities with corresponding arcs of the graph, creating a probability distribution over the exit arcs of each node. This creates a model which satisfies the probability law of Markov chains, allowing analysis of the testing process via well-known mathematical techniques and software.

Input space modeling can be illustrated with the following small example. Consider a system with three input parameters i_1, i_2, and i_3, which can take the following values:

- $i_1 \in \{1, 2, 7, 9\}$
- $i_2 \in \{2, 5, 8\}$
- $i_3 \in \{3, 6, 9\}$

If these parameters are independent, then all combinations are possible. One model of the input space for this situation is shown in Fig. 1.

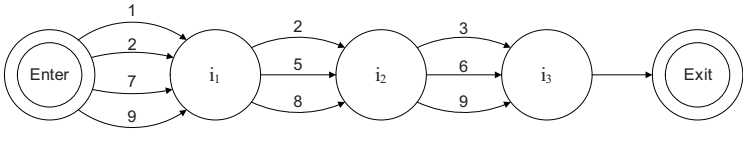

Fig. 1. Model for independent parameters

When dependencies among parameters exist, we modify the model by splitting nodes and merging nodes [12]. Splitting nodes is used for a dependency between two parameters. For example, consider the following dependency between i_2 and i_3: $P_1(i_2) \Rightarrow i_3=6$, where $P_1(i_2)$ is the characteristic predicate for set {2, 8}. In other words, if i_2 takes value 2 or value 8, then i_3 shall take only the value 6. To encode this dependency, we split i_2 node into to nodes: (i_2, P_1) and (i_2, P_2), as shown in Fig. 2, where P_2 is the characteristic predicate for set {5}. The model in Fig. 2 now satisfies the Markov probability law. The model can then be analyzed as a Markov chain and used for test case generation.

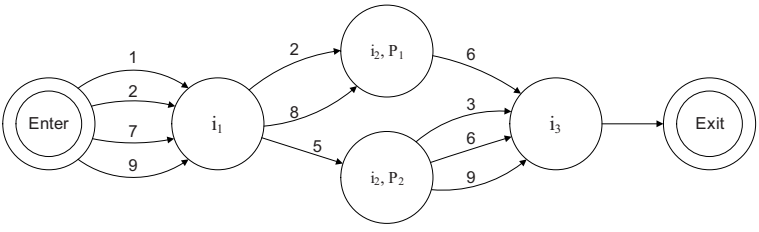

Fig. 2. Model of dependency between i_2 and i_3

Merging nodes is used for dependencies among several parameters. For example, suppose the pair (i_2, i_3) is dependent on i1:

- $P_3(i_1) \Rightarrow (i_2, i_3) \in \{(5, 9), (8, 3), (8, 9)\}$, where $P_3(i_1)$ is the characteristic predicate for set {2, 9}.
- $P_4(i_1) \Rightarrow (i_2, i_3) \in \{(2, 3), (2, 6), (2, 9), (5, 3), (5, 6), (8, 6)\}$, where $P_4(i_1)$ is the characteristic predicate for set {1, 7}.

This dependency can be modeled in three steps (Fig. 3):

- Creating a new derived parameter (i_2, i_3) by merging parameters i_2 and i_3,
- Splitting parameter i_1 according to predicates P_3 and P_4,
- Establishing arcs between i_1 and (i_2, i_3) based on the possible values of (i_2, i_3).

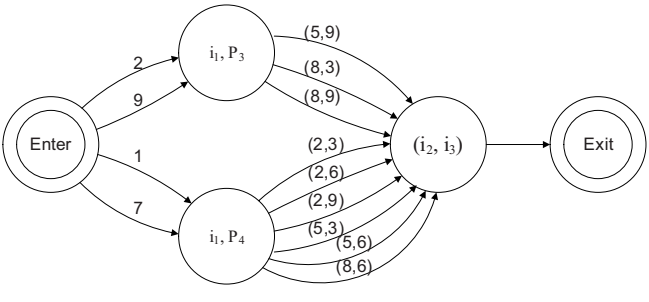

Fig. 3. Model of the dependency between i_1 and (i_2, i_3)

Detailed information about this method, including the application of the method to different types of dependencies, results of test case generation, and more examples can be found in [12]. The purpose of the current paper is to apply this approach to an existing computational science software application.

3 A Case Study: Neutron Transport Software (NEWTRNX)

The NEWTRNX transport solver [1] was developed at ORNL to provide proof-of-principle software for high-fidelity modeling of nuclear fission energy systems. The primary function of the solver is to provide the heat generation distribution of neutrons.

Realistic nuclear reactor simulation is important because it requires an accurate understanding of the interactions of multi-scale, multi-physics phenomena through complex models that may only be tractable with leadership-class computing facilities. The distribution of neutrons within the nuclear vessel is described by the seven-dimensional isotropic source-driven or forward eigenvalue neutral-particle Boltzmann transport equation (equations 1-3) [8], where the fundamental unknown, ψ, is the time-dependent space-velocity distribution of neutrons in the system.

$$\frac{1}{v}\frac{\partial \psi}{\partial t} + \overline{\Omega} \bullet \overline{\nabla}\psi + \sigma_t \psi = \frac{1}{4\pi}\left\{ \begin{array}{ll} \int_0^\infty dE' \sum_{p=0}^{P} \sigma_s^p \sum_{q=-p}^{p} \phi^{pq} Y_{pq} + Q & isotropic \\ \int_0^\infty dE' \sum_{p=0}^{P} \sigma_s^p \sum_{q=-p}^{p} \phi^{pq} Y_{pq} + \left[\frac{1}{k}\right] \int_0^\infty dE'\, \sigma_f \phi^{00} & forward \end{array} \right\} \quad (1)$$

$$\phi^{pq} = \int_0^{4\pi} d\Omega'\ \psi\left(\overline{r}, E', \overline{\Omega'}, t\right) Y^*_{pq}\left(\overline{\Omega'}\right) \quad (2)$$

$$\begin{aligned} &\overline{r} = space\ (x, y, z) \\ &\left(E, \overline{\Omega}\right) = velocity\ \left(energy\ of\ the\ neutron, direction\ of\ travel\right) \\ &\psi\left(\overline{r}, E, \overline{\Omega}, t\right) = \text{"}angular\ flux\text{"} \\ &\phi^{pq}\left(\overline{r}, E', t\right) = pq^{th}\ angular\ moment\ of\ the\ \text{"}angular\ flux\text{"} \\ &\sigma_t\left(\overline{r}, E, t\right) = total\ cross\ section \\ &\sigma_s^p\left(\overline{r}, E' \rightarrow E, t\right) = p^{th}\ harmonic\ moment\ of\ the\ scattering\ cross\ section \\ &\sigma_f\left(\overline{r}, E' \rightarrow E, t\right) = energy\ distribution\ of\ fission\ cross\ section \\ &Y_{pq}\left(\overline{\Omega}\right) = pq^{th}\ spherical\ harmonic\ moment \\ &v\left(E\right) = speed\ of\ the\ neutron \\ &Q\left(\overline{r}, E, \overline{\Omega}, t\right) = external\ source\ of\ neutrons \\ &k = largest\ eigenvalue\ \left(real\ and\ positive\right) \end{aligned} \quad (3)$$

Solving for the space-velocity distribution for even simple geometries requires a large number of degrees of freedom (dof) for an accurate solution (traditionally 10^9 dof per time-step for small single-processor calculations). To solve the equation, the direction of neutron travel is discretized into a set of uncoupled discrete-ordinate directions for a given neutron energy. Similarly, the energy (speed) of the neutron utilizes a piecewise-constant ("multigroup") discretization. This leads to a multi-level iterative solver where "outer" iterations solve for the coupling of the energy terms for all space and angular moments and "inner" iterations solve for the coupling of all directions for a given energy "group" [8].

There are various methods to solve the inner and outer iterations, as well as possible communications strategies. Many of these options are dependent on other options. For example, there are many ways to choose the discrete-ordinate directions, such as a 3d level-symmetric or a 2d/1d product quadrature set [8]. The type of quadrature set and number of directions constrain the number of angular moments that can be accurately computed. This leads to a large and inter-dependent input space that must be regularly tested for robustness and accuracy. This testing task necessitates automation.

4 Modeling Dependencies Among Input Parameters of NEWTRNX

One important aspect of NEWTRNX is the ability to specify different problem definitions and various mathematical methods for solving each defined problem. It allows the user to choose various input options by setting values of the following input parameters:

- `mode` – type of the problem (isotropic or forward)
- `nSpherical` – number of harmonics moments used to represent the anisotropic scattering source
- `quadrature` – type of quadrature set utilized (level-symmetric or product)
- `nSn` – level-symmetric quadrature set order (even numbers)
- `nAziProd` – azimuthal (2d) portion of the product quadrature set
- `npolar` – polar (1d) portion of the product quadrature set (3d is the product of `nAziProd` and `npolar`)

The parameters `mode` and `nSpherical` can be assigned their values independently, but there are dependencies between `quadrature` and other parameters:

- If `quadrature` equals `symmetric`, then parameter `nSn` is used and `nAziProd` and `npolar` are not used.
- If `quadrature` equals `product`, then parameters `npolar` and `nAziProd` are used but `nSn` is not used.

To reflect this dependency, we split parameter `quadrature` into two nodes, as described in Sec. 3.1. The structure of the dependency is presented in Fig. 4 (values of the parameters are not shown).

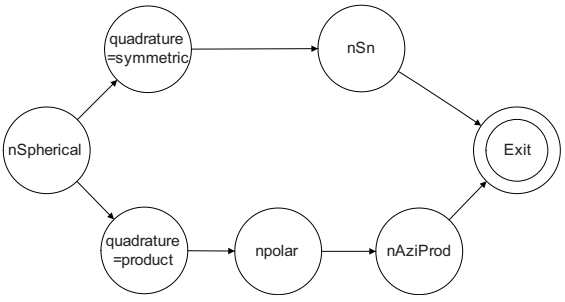

Fig. 4. Structure of the dependency among `quadrature` and `nSn`, `nAziProd`, `npolar`

The dependency among `nSpherical`, `nSn`, `nAziProd`, and `npolar` is shown in Table 1. The left column contains all possible values of `nSpherical`, and the other columns contain corresponding values of `nSn`, `nAziProd`, and `npolar` for every value of `nSpherical`.

Table 1. Dependency among `nSpherical` and `nSn`, `nAziProd` and `npolar`

nSpherical	nSn
{0, 1}	{2, 4, 6, 8, 10, 12, 14, 16}
{2}	{6, 8, 10, 12, 14, 16}
{3}	{10, 12, 14, 16}
{4}	{14, 16}
{5}	-

nSpherical	npolar	nAziProd
{0, 1}	{1, 2, 3, 4, 5, 6}	{1, 2, 3, 4, 5,…, 12}
{2, 3}	{2, 3, 4, 5, 6}	{2, 3, 4, 5,…, 12}
{4, 5}	{3, 4, 5, 6}	{3, 4, 5,…, 12}

Five different groups of values of `nSpherical` correspond to only four groups of values of `nSn` and to three groups of values of `nAziProd` and `npolar`. To reflect this, we split the node for `nSpherical` into five different nodes (Fig. 5). Then for each of these five nodes, we reflect the dependency on `quadrature` by splitting the node for `quadrature` into two nodes.

The model in Fig. 5 shows all dependencies between pairs of NEWTRNX input parameters and can be used for test case generation. Note that the model contains only necessary states (five for `nSpherical`, four for `nSn`, and three for `nAziProd` and `npolar`).

5 Test Case Generation for NEWTRNX

The model in Fig. 5 is a Markov chain model of valid input parameter combinations. We use the JUMBL [10] for the following tasks:

- Analysis of the Markov chain model.
- Test case generation based on the model.

To define a model for input to the JUMBL, the model is described in The Model Language (TML [9]). First the JUMBL is used to check logical consistency of the model and then to produce a model analysis report. General information includes numbers of nodes, arcs, stimuli, etc., as shown in Fig. 6 for the NEWTRNX model. Detailed information includes long run model statistics for nodes, arcs, and stimuli. Of particular interest are statistics for stimuli (Fig. 7) because they directly describe the long run use of different input values represented by the model.

Here "occupancy" is the number of occurrences of a given value (stimulus) divided by the total number of stimuli occurrences. "Mean occurrence" is the average number of times the specific value occurs in a single test case.

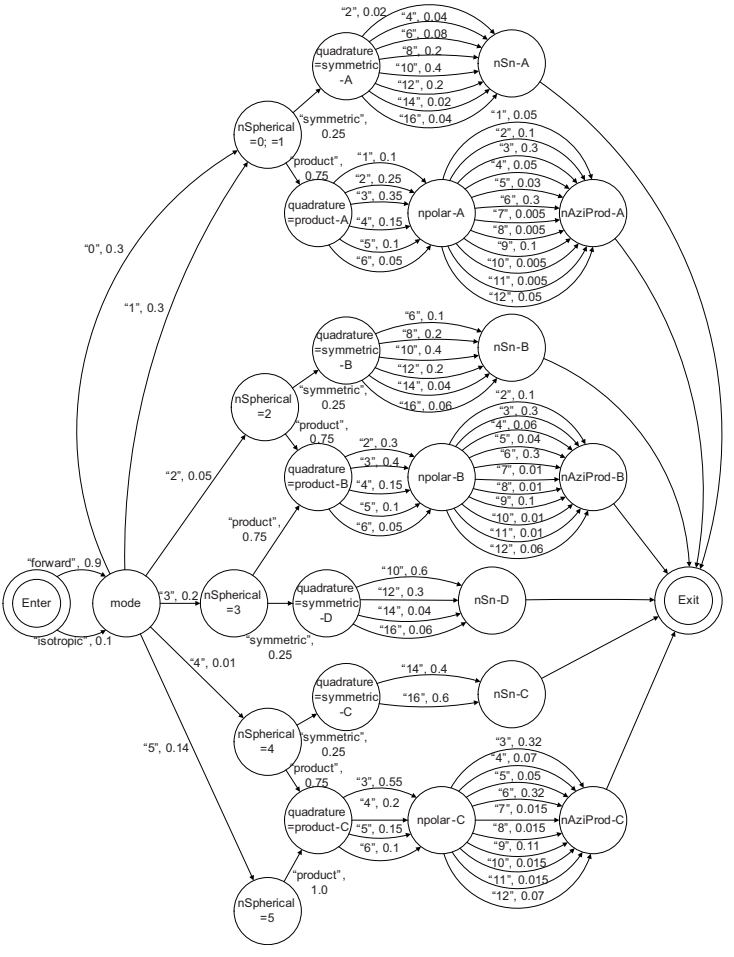

Fig. 5. Dependencies between input parameters of NEWTRNX

Node Count	25 nodes
Arc Count	92 arcs
Stimulus Count	37 stimuli
Expected Test Case Length	5.785 events
Test Case Length Variance	2.534 events
Transition Matrix Density (Nonzeros)	52.8E-3 (33 nonzeros)
Undirected Graph Cyclomatic Number	9

Fig. 6. NEWTRNX model statistics (fragment from JUMBL output)

Stimulus	Occupancy	Mean Occurrence (visits per case)
Exit	0.172860847	1
mode=forward	0.155574762	0.9
mode=isotropic	17.2860847E-3	0.1
nAziProd=1	3.88936906E-3	22.5E-3
nAziProd=10	1.09550562E-3	6.3375E-3
nAziProd=11	1.09550562E-3	6.3375E-3
nAziProd=12	7.61884183E-3	44.075E-3
nAziProd=2	11.019879E-3	63.75E-3

Fig. 7. NEWTRNX stimulus statistics (fragment from JUMBL output)

Table 2. NEWTRNX weighted test cases (fragment)

N	Probability	mode	nSpherical	quadrature	nSn	npolar	nAziProd
1	0.027	forward	3	symmetric	10	-	-
2	0.027	forward	0	symmetric	10	-	-
3	0.027	forward	1	symmetric	10	-	-
4	0.022	forward	5	product	-	3	6
5	0.022	forward	5	product	-	3	3
6	0.021	forward	0	product	-	3	6
7	0.021	forward	0	product	-	3	3
8	0.021	forward	1	product	-	3	6
9	0.021	forward	1	product	-	3	3
10	0.016	forward	3	product	-	3	6

Various types of test cases can be generated from the model including model coverage tests, random tests, and weighted tests. Coverage tests are generated as the minimal set of test cases that cover every arc in the model. Thus, to cover all arcs in the NEWTRNX model, 53 test cases were generated. Random test cases are generated according to the probabilities on the arcs. Weighted test cases are those generated in order of decreasing probability. For random and weighted tests, the number of test

cases can be specified. A separate file is created for every test case. The ten highest probability test cases for the NEWTRNX model are shown in Table 2, with their individual probabilities of occurrence. Statistics for the 53 coverage tests and the ten most likely tests are provided in separate test analysis reports (Fig. 8).

Random test cases can be generated for reliability estimation. The "optimum" reliability represented by a particular set of test cases can be computed prior to test execution by assuming that all test cases will be successful. These values can be used during test planning for estimation of the required number of test cases. When testing is completed and the number of failures is known, operational reliability estimates are included in a test analysis report.

Node Count	25 nodes
Arc Count	92 arcs
Stimulus Count	37 stimuli
Test Cases Recorded	10 cases
Nodes Generated	19 nodes / 25 nodes (0.76)
Arcs Generated	25 arcs / 92 arcs (0.27173913)
Stimuli Generated	12 stimuli / 37 stimuli (0.324324324)
Nodes Executed	0 nodes / 25 nodes (0)
Arcs Executed	0 arcs / 92 arcs (0)
Stimuli Executed	0 stimuli / 37 stimuli (0)

a) Weighted test cases

Node Count	25 nodes
Arc Count	92 arcs
Stimulus Count	37 stimuli
Test Cases Recorded	53 cases
Nodes Generated	25 nodes / 25 nodes (1)
Arcs Generated	92 arcs / 92 arcs (1)
Stimuli Generated	37 stimuli / 37 stimuli (1)
Nodes Executed	0 nodes / 25 nodes (0)
Arcs Executed	0 arcs / 92 arcs (0)
Stimuli Executed	0 stimuli / 37 stimuli (0)

b) Coverage test cases

Fig. 8. NEWTRNX model test case statistics (fragment from JUMBL output)

6 Conclusions

Specification of all valid test cases from a large input space can be a challenging task, especially when there are dependencies among input parameters. We have presented a method for solving this problem and demonstrated its practical application on the neutron transport software tool NEWTRNX.

The selection of test cases is performed in two steps. First, a Markov chain model of the input space is created, reflecting dependencies among input parameters. Second, the JUMBL library of software tools is used for model analysis and test case generation. Results for NEWTRNX test planning are provided. The case study shows the applicability of model-based statistical testing for testing large scientific computational software systems. The next phase of this effort will investigate methods for automating both test execution and results checking.

References

1. Clarno, K., de Almeida, V., d'Azevedo, E., de Oliveira, C., Hamilton, S.: GNES-R: Global Nuclear Energy Simulator for Reactors Task 1: High-Fidelity Neutron Transport. In: Proceedings of PHYSOR–2006, American Nuclear Society Topical Meeting on Reactor Physics: Advances in Nuclear Analysis and Simulation, Vancouver, Canada (2006)

2. Cohen, D.M., Dalal, S.R., Fredman, M.L., Patton, G.C.: The AETG system: An approach to testing based on combinatorial design. IEEE Transactions on Software Engineering 23(7), 437–444 (1997)
3. Cox, M.G., Harris, P.M.: Design and use of reference data sets for testing scientific software. Analytica Chimica Acta 380(2), 339–351 (1999)
4. Einarsson, B. (ed.): Accuracy and Reliability in Scientific Computing. SIAM, Philadelphia (2005)
5. Grindal, M., Offutt, J., Andler, S.F.: Combination testing strategies: A survey. Software Testing, Verification, and Reliability 15(3), 167–199 (2005)
6. Hatton, L.: The T experiments: errors in scientific software. IEEE Computational Science and Engineering 4(2), 27–38 (1997)
7. Howden, W.: Validation of scientific programs. Comput. Surv. 14(2), 193–227 (1982)
8. Lewis, E., Miller Jr., W.F.: Computational Methods of Neutron Transport. ANS (1993)
9. Prowell, S.: TML: A description language for Markov chain usage models. Information and Software Technology 42(12), 835–844 (2000)
10. Prowell, S.: JUMBL: A Tool for Model-Based Statistical Testing. In: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS 2003), Big Island, HI, USA (2003)
11. Swain, W.T., Scott, S.L.: Model-Based Statistical Testing of a Cluster Utility. In: Sunderam, V.S., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2005. LNCS, vol. 3514, pp. 443–450. Springer, Heidelberg (2005)
12. Vilkomir, S.A., Swain, W.T, Poore, J.H.: Combinatorial test case selection with Markovian usage models. In: Proceedings of the 5th International Conference on Information Technology: New Generations (ITNG 2008), Las Vegas, Nevada, USA (2008)
13. Walton, G., Poore, J.H., Trammell, C.: Statistical Testing of Software Based on a Usage Model. Software: Practice and Experience 25(1), 97–108 (1995)
14. Whittaker, J., Poore, J.H.: Markov Analysis of Software Specifications. ACM Transactions on Software Engineering and Methodology 2(1), 93–106 (1993)