A Markov Reward Model for Software Reliability *

YoungMin Kwon and Gul Agha
Open Systems Laboratory
Department of Computer Science
University of Illinois at Urbana Champaign
{ykwon4, agha}@cs.uiuc.edu

ABSTRACT

A compositional method for estimating software reliability of many threaded programs is developed. The method uses estimates of the reliability of individual modules and the probability of transitions between the modules to estimate the reliability of a program in terms of its current state. The reliability of a program is expressed using *iLTL*, a probabilistic linear temporal logic whose atomic propositions are linear inequalities about transitions of the probability mass function of a Discrete Time Markov Chain. We then use a Markov reward model to estimate software reliability. The technique is illustrated in terms of an example.

1. INTRODUCTION

It is well known that fixing a fault in a program becomes increasingly expensive in later phases of software development [4]. It is much more cost effective to fix as many faults as possible before releasing a program. Unfortunately, because it becomes harder to detect a fault as the software becomes more reliable, the cost of testing also increases [4]. Thus, at some point, testing is no longer cost-effective and the software has to be released. It has also been observed that modular testing is a good strategy [6]. Moreover, as the lines of code increase, the testing effort required to fix a fault grows superlinearly [4]. Hence, modular testing with fewer lines of code would significantly reduce the overall effort required for testing.

We address the problem of accurately estimating reliability of large-scale software systems and, at the same time, improving the effectiveness of the testing process. We assume that we can estimate the control transition probabilities between modules using operational profiling. Together with estimated module reliabilities, these transition probabilities not only enable us to estimate system reliability, they also help us focus testing on modules that may more effectively increase the reliability of the entire system.

A many threaded program, such as a server program, differs from a single threaded program in that its current state is better abstracted by a *probability mass function* (pmf). A naive representation of the program state as a product of the states of each thread results in a state explosion. Suppose that a program with 100 thread is modeled as a 3 state state-machine; this results in a program with an unwieldy 3^{100} states. Observe that the threads are executed concurrently; thus the current state of each thread execution may be represented as a random variable that has a stochastic behavior. We can use this representation to express the reliability of a program more precisely as a conditional probability given the current pmf of the states of the program.

Our approach uses a tool based on *iLTL*, a probabilistic linear temporal logic, that can check whether a discrete time Markov chain (DTMC) is a model for an *iLTL* specification or not. The atomic propositions of the *iLTL* are linear inequalities about the probability mass function (pmf) transitions of a given DTMC.

There are many probabilistic verification approaches such as probabilistic LTL, pCTL, pCTL*, CSL, Markov reward model and its variations [1, 2, 5, 9, 8, 11]. These approaches assign probability measures to paths of computation and check probabilistic specifications on these paths. We use *iLTL* since our primary interest is not path behavior but reasoning about the pmf transitions to establish (conditional) program reliability. *iLTL* allows us to express specifications such as: "given the current pmf estimation, eventually a certain property will be satisfied." Specifically, we add fault states and check for the probability of transition to these states.

The outline of the paper is as follows. Section 2 defines the Markov Reward Model and discusses the modeling assumptions we make. Section 3 defines a specification logic and provides a verification algorithm for it. Section 4 illustrates our approach by means of an example executed on an *iLTL* checker. The final section summarizes some open problems.

2. A MARKOV REWARD MODEL FOR SOFTWARE RELIABILITY

We first show how we construct a Markov model for software reliability based on the reliability of components and the probability of transitions between them. We make the modeling assumptions explicit and illustrate our approach by means of an example.

2.1 Markov Model

We assume that a program consists of a set of modules and that control flow of a program is represented as a sequence of these modules. The particular control flow depends on the input and the logics of the modules. We may assume that there are different probabilities for different possible inputs. Moreover, for a given input, there is a transition probabilities between modules and we can assume that these transition probabilities are known. This assumption is not unrealistic since we can build an operational profile that provides the transition probabilities between modules [6].

With a set of modules and transition probabilities between modules, we can model a program as a Markov chain. Recall that a

*This research has been supported in part by the DARPA IXO NEST program under contract F33615-01-C-1907, by NSF under grant CNS 05-09321 and by ONR under DoD MURI award N0014-02-1-0715.

1-4244-0910-1/07/$20.00 ©2007 IEEE

Markov process is a stochastic process whose past has no influence on the future, except as it is represented in the present. A *Markov chain* is a Markov process that has a countable number of states. We represent a *Discrete Time Markov Chain* (DTMC) X as a tuple $(S, \mathbf{M})$ where $S = \{s_1, \ldots, s_n\}$ is a set of finite states and $\mathbf{M} \in \mathbb{R}^{n \times n}$ is a Markov transition matrix that governs the transitions of probability mass function (pmf). Since $\mathbf{M}$ is a Markov matrix, its elements are non-negative and its column sums are all one: $0 \leq M_{ij} \leq 1$ for $1 \leq i, j \leq n$ and $\sum_{i=1}^{n} M_{ij} = 1$ for $1 \leq j \leq n$. Let $\mathbf{x} \in \mathbb{R}^{n \times 1} = [x_1(t), \ldots, x_n(t)]^T$ be a pmf of X at time t such that $x_i(t) = P\{X(t) = s_i\}$. Thus,

$$\mathbf{x}(t+1) = \mathbf{M} \cdot \mathbf{x}(t)$$

If a program consists of a set of modules $S = \{s_1, \ldots, s_n\}$ and transition probabilities between modules are represented by a matrix $\mathbf{M}$ then we can model the program by a DTMC $X = (S, \mathbf{M})$, where M_{ij} is the probability that control transfers from module s_i to module s_j : $P\{X(t+1) = s_j | X(t) = s_i\}$. We assume that s_n is a terminating state that every successful execution arrives at. Note that s_n is an absorbing state: $P\{X(t+1) = s_n | X(t) = s_n\} = 1$.

We regard the reliability of a module as the probability that a module does not produce a fault when a control is passed to it. Note that this reliability is independent of the transition probabilities. Moreover, unlike the transition probabilities, reliabilities are usually unknown and we have to estimate them. A simple reliability model is a NHPP (Non-homogeneous Poisson Process) exponential model. This model is based on the following simplifying assumptions:

- The faults in a program are mutually independent, at least from the point of view of failure detection.
- The number of failures detected at any time is proportional to the current number of faults in a program. In other words, the probability of failures that occur due to faults is constant.
- Faults that are detected are isolated and removed prior to further testing.
- Each time a software failure occurs, the software error which caused it is immediately removed, and no new errors are introduced as a result of this removal.

Note that these simplifying assumptions need not hold exactly. However, because we are dealing with probabilities, as long as they are hold with a sufficiently high probability, our reliability estimation technique would nevertheless be realistic.

Under these assumption, the expected number of errors detected by time t is:

$$m(t) = a(1 - e^{-bt}),$$

where a is the expected total number of faults that exist in the software before testing and b is the failure detection rate or the failure intensity of a fault. Given the number of failures and the detection times for a module, we can estimate a and b for that module. Using these parameters, we can estimate the reliability of a module as:

$$\hat{R}(x|t) = e^{-\hat{a}(e^{-\hat{b}t} - e^{-\hat{b}(t+x)})},$$

where the reliability $R(x|t)$ is the probability that a module does not have a failure during the time interval t to $t + x$ [4].

During the interval between between correction of errors, we assume that the reliability of a module over a fixed sampling interval x, $R(x|t)$, remains constant. For example, suppose the software is released to a market. Then, as long as no errors are corrected, the reliabilities of the modules do not change.

We can extend our DTMC model $X = (S, \mathbf{M})$ in order to check the reliability of a program. Let r_i be the reliability of a module s_i. We add one more states: $S' = S \cup \{f\}$ where f represents a *fail* state. Then the extended transition matrix such that $M'_{ij} = r_j \cdot M_{ij}$ for $n \leq i, j \leq 0$, $M'_{(n+1)j} = 1 - r_j$ for $n \leq j \leq 0$, $M'_{i(n+1)} = 0$ for $n \leq i \leq 0$, and $M'_{(n+1)(n+1)} = 0$.

Because a terminating state s_n cannot result in a failure, M_{in} is 0 if $i \neq n$ and 1 if $i = n$ and r_n is 1. The extended matrix $\mathbf{M}'$ is also a probability matrix: each element is non-negative and the sum of each column is 1. One can easily check that $\sum_{i=1}^{n} r_j \cdot M_{ij} + 1 - r_j = 1$ for $1 \leq j \leq n$ since $\sum_{i=1}^{n} M_{ij} = 1$.

The reliability of a program is the probability that a program eventually arrives at the final success state s_n: $P\{X(\infty) = s_n\}$. This probability can be computed by:

$$r = \mathbf{M}_{n*} \cdot \lim_{t \to \infty} \sum_{i=1}^{t} \mathbf{M}_*^i \cdot \mathbf{x}(0),$$

where $\mathbf{M}_*$ is a sub-matrix of $\mathbf{M}$ that comprises the first $n-1$ rows and the first $n-1$ columns of $\mathbf{M}$, $\mathbf{M}_{n*}$ is a n^{th} row vector of $\mathbf{M}$ with first $n-1$ elements, and $\mathbf{x}(0)$ is an initial probability mass function of $X(0)$. Note that the reliability of a program is a function of initial pmf $\mathbf{x}(0)$.

It is known that an arbitrary matrix $\mathbf{M}$ with the property $\lim_{t \to \infty} \mathbf{M}^t = \mathbf{0}$ satisfies the equation [7]:

$$\lim_{t \to \infty} \sum_{i=0}^{t} \mathbf{M}^i = (\mathbf{I} - \mathbf{M})^{-1}.$$

Moreover, if all elements of a matrix $\mathbf{M}$ satisfy $0 \leq M_{ij} < 1$ and each column sum of $\mathbf{M}$ is less than 1 then $\lim_{t \to \infty} \mathbf{M}^t = \mathbf{0}$ [7].

Assuming that the reliability r_i of each module is less than one, the sub-matrix $\mathbf{M}_*$ satisfies the previous condition: all elements are non-negative and less than one, and the column sum is less than one since $\mathbf{M}$ is a Markov transition matrix. So, the reliability of a program r can be rewritten as:

$$r = \mathbf{M}_{n*} \cdot (\mathbf{I} - \mathbf{M}_*)^{-1} \cdot \mathbf{x}(0).$$

Note that since the transition probability from s_n to s_n is 1, the reliability r computed above should be equal to the n^{th} element of the following vector.

$$\mathbf{M}_*^{\infty} \cdot \mathbf{x}(0)$$

2.2 Example

Figure 1 shows a reliability diagram of a program with three modules. In addition to these three modules, a *success* state is added to indicate that a program has successfully terminated. Figure 1 also shows the transition probabilities between these modules and the *success* state. If we regard r_A, r_B and r_C as 1, then the number at the arrows connecting these states represent the transition probabilities that have been obtained from an operation profile. We extend this diagram to check the reliability of a program.

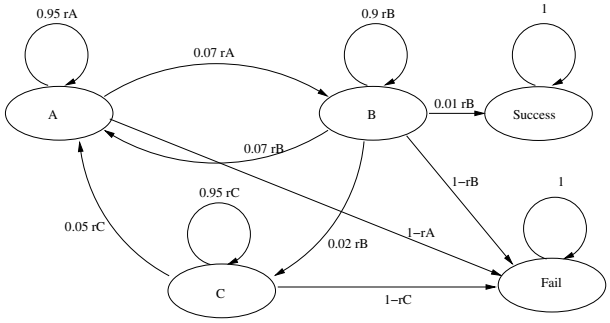

Figure 1: a module reliability diagram

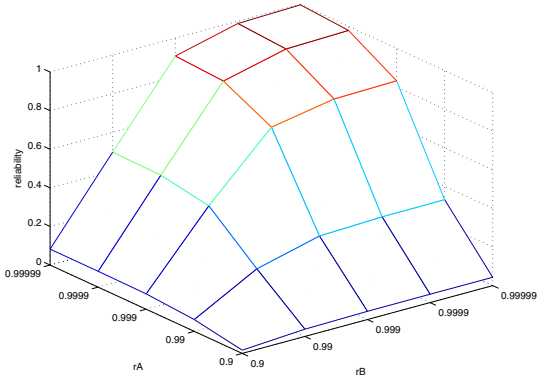

Figure 2: a reliability of a program as a function of module reliabilities r_A and r_B with $r_C = 1 - 10^{-5}$.

Let r_A, r_B and r_C be the reliabilities of the modules A, B and C. We can model the reliability of the program by a DTMC $X = (S, \mathbf{M})$ where $S = \{A, B, C, success, fail\}$ and:

$$\mathbf{M} = \begin{bmatrix} .95r_A & .07r_B & .05r_C & .00 & .00 \\ .05r_A & .90r_B & .00r_C & .00 & .00 \\ .00 & .02r_B & .95r_C & .00 & .00 \\ .00 & .01r_B & .00 & 1.0 & .00 \\ 1-r_A & 1-r_B & 1-r_C & .00 & 1.0 \end{bmatrix}.$$

Thus, given $r_A = r_B = r_C = 0.9$ and $\mathbf{x}(0) = [1, 0, 0, 0, 0]^T$, the reliability of X is 0.1655. This agrees with the probability with large t $P\{X(t = 10^5) = success\} = (\mathbf{M}^{10^5} \cdot [1, 0, 0, 0, 0]^T)_4 \simeq 0.1655$.

Figure 2 shows how the reliability of a program X changes as a function of module reliabilities r_A and r_B. In the figure, we assume that $r_C = 1 - 10^{-5}$ and $\mathbf{x}(0) = [1, 0, 0, 0, 0]^T$. Observe that the overall program reliability depends not only on the reliability of the modules but is significantly affected by the transition probabilities between the modules.

2.3 Markov Reward Model

A *Markov reward process* is a triple $(\rho, S, \mathbf{M})$ where $(S, \mathbf{M})$ is a DTMC and $\rho : S \to \mathbb{R}$ is a reward function for each state. We consider only constant rewards. So we represent the reward function as a constant row vector $\mathbf{r} = [\rho(s_1), \ldots, \rho(s_n)]$. The expected reward at time t is:

$$\sum_{i=1}^{n} \rho(s_i) \cdot P\{X(t) = s_i\} = \mathbf{r} \cdot \mathbf{x}(t) = \mathbf{r} \cdot \mathbf{M}^t \cdot \mathbf{x}(0).$$

Later, we will transform our DTMC representation of a program to a Markov reward model so that we can use *iLTL* to reason about the reliability of a program.

3. SPECIFICATION LOGIC

Since we are interested in the temporal behavior of a probability mass function (pmf), our specification logic should be able to express properties of the transitions of the pmf. The sort of properties we are interested in compare a probability that a DTMC will be a particular state with a constant, or with another such probability at a different time. We use linear inequalities over pmf vectors as atomic propositions of our specification logic.

3.1 Syntax and informal semantics

The syntax of the specification logic is:

$$\begin{array}{rcll} \psi & ::= & T \mid F \mid ineq \mid & \text{atomic propositions} \\ & & \neg\psi \mid \psi \vee \phi \mid \psi \wedge \phi \mid & \text{logical operators} \\ & & \mathtt{X}\,\psi \mid \psi\,\mathtt{U}\,\phi \mid \psi\,\mathtt{R}\,\phi & \text{temporal operators} \\ ineq & ::= & \sum_{i=1}^{n} a_i \cdot P\{X = s_i\} < b, & \end{array}$$

where $X = (\{s_1, \ldots, s_n\}, \mathbf{M})$, $a_i \in \mathbb{R}$ and $b \in \mathbb{R}$. Recall that $\Diamond$ represents *eventually*, $\Box$ represents *always*, $\mathtt{X}$ is the next operator, $\psi\,\mathtt{U}\,\phi$ means ϕ eventually becomes true and before ϕ becomes true ψ is true, and $\psi\,\mathtt{R}\,\phi$ means until ψ first becomes true ϕ is true.

Observe that the comparison between two probabilities at different times can be expressed by the linear inequalities of the form *ineq*. For example, given the DTMC $X = (\{s_1, \ldots, s_n\}, \mathbf{M})$, the probability that X is in state s_i at time $t + k$ is given by:

$$P\{X(t + k) = s_i\} = x_i(t + k) = \mathbf{M}_i^k \cdot \mathbf{x}(t),$$

where $\mathbf{M}_i^k$ is the i^{th} row of $\mathbf{M}^k$ and $\mathbf{x}(t)$ is the pmf at time t.

Predicates about a Markov reward process[3] can also be expressed by linear inequalities. We consider only a constant reward function $\rho : S \to \mathbb{R}$ for each state. A performance metric is an accumulated reward over time. The expected accumulated reward is:

$$\begin{array}{l} \sum_{k=0}^{T} \sum_{s_i \in S} \rho(s_i) \cdot P\{X(t + k) = s_i\} \\ \quad = \mathbf{r} \cdot \left(\sum_{k=0}^{T} \mathbf{M}^k\right) \cdot \mathbf{x}(t) \\ \quad = \mathbf{r} \cdot \mathbf{S} \cdot \left(\sum_{k=0}^{T} \mathbf{\Lambda}^k\right) \cdot \mathbf{S}^{-1} \cdot \mathbf{x}(t) \end{array}$$

where $\rho(s_i)$ is a reward function associated with a state s_i, $\mathbf{r}$ is a row vector $[\rho(s_1), \ldots, \rho(s_n)]$, $\mathbf{M} = \mathbf{S} \cdot \mathbf{\Lambda} \cdot \mathbf{S}^{-1}$ with $\mathbf{\Lambda}$ a diagonal matrix of eigenvalues of $\mathbf{M}$ and the T on the summation is an upper bound of the accumulation interval. Note that the accumulation interval can be ∞ if the reward vector $\mathbf{r}$ is orthogonal to the steady state pmf vector.

3.2 Verification Algorithm

Let $s_X(\mathbf{x}(0))$ be a string whose alphabet is $\Sigma = 2^{AP}$ and its i^{th} alphabet is $\{ineq \in AP : ineq(\mathbf{M}^i \cdot \mathbf{x}(0))\}$ where X is a DTMC, $\mathbf{x}(0)$ is an initial pmf and AP is a set of inequalities. Let $L_X \subseteq \Sigma^*$ be a set of strings $s_X(\mathbf{x}(0))$ for all $\mathbf{x}(0)$. Then our model checker checks whether $L_X \subseteq L_\psi$ where L_ψ is a language accepted by the Büchi automata built from an LTL formula ψ. More specifically, for a given specification ψ, it checks whether any $s_X \in L_X$ is in $L_{\neg\psi}$.

Figure 3 shows a block diagram of the *iLTL* model checker. Given an *iLTL* specification, it computes a search depth using the Markov model and the inequalities used in the specification logic. With

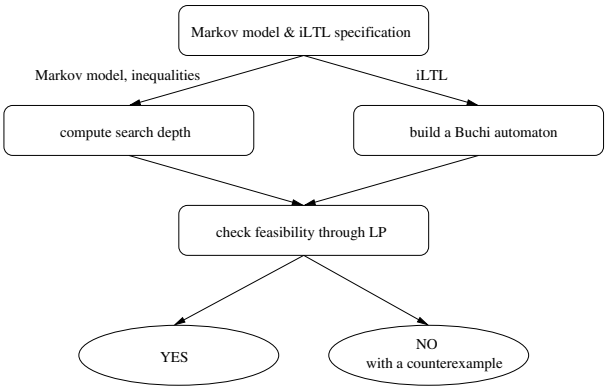

Figure 3: a block diagram of an *iLTL* model checking algorithm

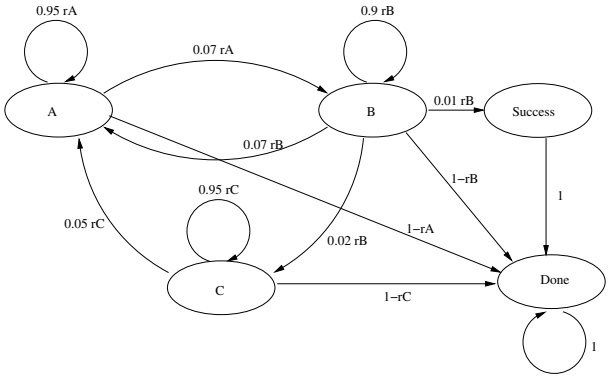

Figure 4: modified reliability state transition diagram

the specified *iLTL*, which essentially is an LTL, we build a Büchi automata by the expand algorithm [10]. Using the search depth and the Büchi automata we check the feasibility of a set of inequalities collected using the Büchi automata.

Our model checking algorithm has two steps. First, we build a Büchi automaton for the negated normal form of a given LTL specification ψ using the *expand* algorithm [10]. Second, we check the feasibility of the initial pmf $\mathbf{x}(0)$ against the set of inequalities collected along finite paths obtained from the automaton. From the set of inequalities, if there is a feasible solution, then a counterexample that does not satisfy the specification ψ is found. Otherwise, the DTMC X satisfies the given specification.

We now provide the details of our algorithm and the technical justification for it. The rest of this section is purely technical and may be skipped without loss of continuity.

Observe that given the linear inequalities of an LTL formula ψ and a Markov matrix $\mathbf{M}$, we can compute an upper bound N on the number of time steps after which the atomic propositions of ψ become constant. Given a DTMC $X = (S, \mathbf{M})$, an initial pmf $\mathbf{x}(0)$ and an LTL formula, because we can compute the bound after which the truth value of the inequalities in the LTL formula become constants, after a finite expansion of the LTL formula, we can evaluate it. Recall that the 'until' and 'release' operators may be rewritten as:

$$\begin{aligned} \phi \,\mathtt{U}\, \psi &\equiv \psi \wedge (\phi \vee \mathtt{X}\,(\phi \,\mathtt{U}\, \psi)) \\ \phi \,\mathtt{R}\, \psi &\equiv (\phi \wedge \psi) \vee (\phi \wedge \mathtt{X}\,(\phi \,\mathtt{R}\, \psi)). \end{aligned}$$

More detailed discussion and proofs about model checking algorithm can be found in [12].

4. MODEL CHECKING OF SOFTWARE RELIABILITY

We are interested in a number of different kinds of program properties related to reliability:

- Recall that the reliability of a program depends on an initial pmf $\mathbf{x}(0)$. Thus we may be interested in finding the initial pmf $\mathbf{x}(0)$ that would result in the lowest reliability.
- If we can estimate the current pmf $\mathbf{x}(t)$, we may want to compute the reliability of a program given the estimated pmf.
- We may want to check the effects on the reliability of a program if different execution constraints are enforced on the program.

These properties can be handled by *iLTL* using the reliability model we considered in Section 2.1. However, we cannot use our *iLTL* model checking algorithm directly on the model we mentioned, because the model violates the eigenvalue constraints of theorem 1. So, we have transform the model slightly.

In theorem 1, our *iLTL* model checking algorithm has two constraints on the Markov transition matrix $\mathbf{M}$. One is the diagonalizability of $\mathbf{M}$ and the other is that $\mathbf{M}$ should have only one eigenvalue whose absolute value is one. The latter condition ensures a unique steady state pmf of a Markov chain. However, the model in Section 2.1 violates the second condition: two of its eigenvalues are 1. One can easily check this from Figure 1. The transition probabilities from *success* to *success* and from *fail* to *fail* are one. That means once a pmf becomes $P\{X(t) = success\} = \alpha$ and $P\{X(t) = fail\} = 1 - \alpha$, it will remain there for any $0 \leq \alpha \leq 1$. Hence, there is no unique steady state pmf in the DTMC of Figure 1. Specifically, for the matrix $\mathbf{M}$ of Section 2.2, the two vectors $\mathbf{x}_1 = [0, 0, 0, 1, 0]^T$ and $\mathbf{x}_2 = [0, 0, 0, 0, 1]^T$ are eigenvectors of it with $\mathbf{x}_1 = \mathbf{M} \cdot \mathbf{x}_1$ and $\mathbf{x}_2 = \mathbf{M} \cdot \mathbf{x}_2$. Hence the two eigenvalues λ_1 and λ_2 are one.

In order to use our *iLTL* model checking algorithm we modify the diagram of Figure 1 to Figure 4. First we replaced the *fail* state by the *done* state. And, we remove the self loop transition of *success* state. Instead we add a transition from state *success* to *done* with a probability one and make the *success* state transient. With this change every successful execution arrives at *done* state through *success* state whereas every unsuccessful execution arrives at *done* state without going through *success* state.

The reliability of a program is the accumulated sum of the probabilities that the *success* state is visited. So our modified DTMC model is $X = (S, \mathbf{M})$ where $S = \{A, B, C, Success, Done\}$ and

$$\mathbf{M} = \begin{bmatrix} .95r_A & .07r_B & .05r_C & .00 & .00 \\ .05r_A & .90r_B & .00r_C & .00 & .00 \\ .00 & .02r_B & .95r_C & .00 & .00 \\ .00 & .01r_B & .00 & .00 & .00 \\ 1 - r_A & 1 - r_B & 1 - r_C & 1.0 & 1.0 \end{bmatrix}.$$

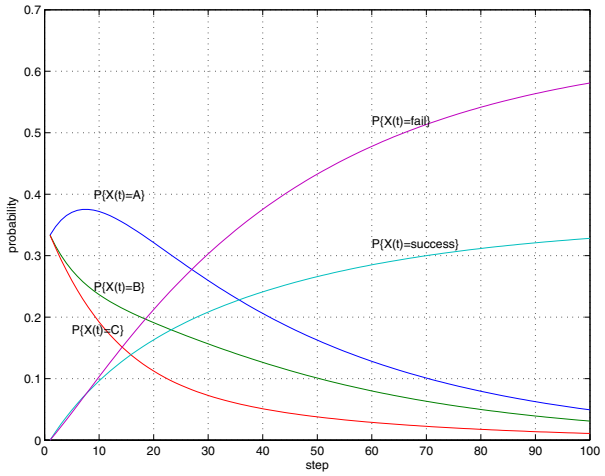

Figure 5: probability transitions of figure 1

The reliability of the program X is:

$$\begin{aligned} r &= \sum_{t=0}^{\infty} P\{X(t) = success\} \\ &= \sum_{t=0}^{\infty} [0,0,0,1,0] \cdot \mathbf{x}(t) \\ &= \sum_{t=0}^{\infty} [0,0,0,1,0] \cdot \mathbf{M}^t \cdot \mathbf{x}(0) \end{aligned}$$

For the example of Section 2.2 with module reliabilities $r_A = .97$, $r_B = .999$ and $r_C = .999$, the reliability of the program can be expressed as $\mathbf{r} \cdot \mathbf{x}(0)$ where $\mathbf{r} = [.2149, .3478, .5036, 0, 0]$. Figure 5 shows how the probabilities of each states change over time and how the reliability of the program ($P\{X(t) = success\}$) is accumulated with module reliabilities r_A, r_B and r_C and initial pmf $\mathbf{x}(0) = [1,1,1,0,0]/3$.

4.1 *iLTL* Checker

In this section, we describe some properties of the example in Section 2.2. We assume that the reliabilities of the modules are $r_A = .97$, $r_B = .999$ and $r_C = .999$ as in the previous section.

Figure 6 describes the *iLTL* Checker description of the reliability model in Figure 4. The *iLTL* Checker has two main blocks. The `model` block describes the DTMC model to be checked. This block begins with the name of the DTMC (`pgm` in this case) followed by a set of states the DTMC has and finally the Markov transition matrix. The `specification` block begins with an optional list of inequalities that will be used in *iLTL* specification. Finally, an *iLTL* formula is specified using the inequalities defined previously.

In Figure 6, the inequalities a, b and c describes whether the reliability of the program `pgm` is less than 0.7, 0.5 and 0.3 each. The inequality d describes that the probability that `pgm` is in states S or D is larger than zero. So, the negation of it means that `pgm` is not in states S or D. The inequality e says that the probability that `pgm` is in state A is at least 0.3 larger than the probability that `pgm` is in state C.

The specification `a` checks whether the reliability of the program `pgm` is less than 0.7 regardless of the initial pmf $\mathbf{x}(0)$. The *iLTL* Checker shows the result as:

```
model:
  Markov chain pgm
    has states :
      { A, B, C, S, D},
    transits by :
      [ .9215, .0699, .05,   .0,  .0;
        .0485, .8991, .0,    .0,  .0;
        .0,    .02,   .9191, .0,  .0;
        .0,    .01,   .03,   .0,  .0;
        .03,   .001,  .001,  1.0, 1.0  ]
specification:
  a : .2149*P{pgm=A} + .3478*P{pgm=B}
                     + .5036*P{pgm=C} < .7,
  b : .2149*P{pgm=A} + .3478*P{pgm=B}
                     + .5036*P{pgm=C} < .5,
  c : .2149*P{pgm=A} + .3478*P{pgm=B}
                     + .5036*P{pgm=C} < .3,
  d : P{pgm=S} + P{pgm=D} > .0,
  e : P{pgm=A} > P{pgm=C} + .3
  a                        # 1)
  #b                        2)
  #e -> b                   3)
  #e -> c                   4)
  #(b /\ ~ d) -> ~ e        5)
  #(b /\ ~ d) -> <> ~ e     6)
```

Figure 6: an iLTL checker description of the reliability model of figure 4

```
Depth:  22
Result: T
```

The first line `Depth: 22` says that the required search depth for this formula is 22. Note that the formula `a` is a state formula (not a path formula). So, in theory the required search depth is zero. However, current implementation of our *iLTL* checker computes a search depth based on the set of inequalities used in the formula and the Markov transition matrix not the formula itself. We plan to improve the tool to avoid excessive search depth in such cases. However, note that the search depth is displayed before actual search begins. So, one can modify specification if the search depth is too large instead of waiting indefinitely. The second line says that `pgm` is a model of the specification `a`.

The second commented specification `b` checks whether the reliability of the program `pgm` is less than 0.5. The *iLTL* checker result is:

```
Depth:  30
Result: F
counterexample:
  pmf(pgm(0)): [ .01247 .0 .98753 .0 .0 ]
```

The result shows that `pgm` is not a model of `b` with a counter example of $\mathbf{x}(0) = [.01247, .0, .98753, .0, .0]$. One can see that $\mathbf{r} \cdot \mathbf{x}(0) = 0.5$. From the first and the second example we know that the maximum reliability of the program `pgm` is in between 0.5 and 0.7.

The third commented specification `e -> b` checks whether the reliability of the program is less than 0.5 if the probability that `pgm` is in A state is at least 0.3 larger than the probability that `pgm` is in C state. The *iLTL* checker verifies that it is true. However for the

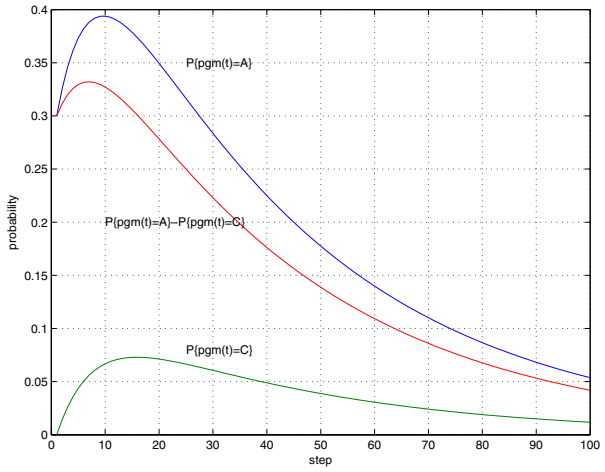

Figure 7: a counterexample of `b -> ¬ e`

fourth commented specification `e -> c`, the model checker proves that it is not true:

```
Depth:  78
Result: F
counterexample:
  pmf(pgm(0)): [ .70523 .0 .294770 .0 .0 ]
```

By comparing the previous example (`b`) and this example (`e -> b`), we know that we should focus more on module A than module C because more probability in A results in decreased reliability of `pgm`.

The fifth example (`b -> ¬ e`) checks whether the fact that the reliability of `pgm` is less than 0.5 implies not `e`. The *iLTL* checker returns a negative answer with a counter example:

```
Depth:  78
Result: F
counterexample:
  pmf(pgm(0)): [ .3 .7 .0 .0 .0 ]
```

However for the sixth example (`b -> <> ¬ e`), if the reliability of a `pgm` is less than 0.5 then eventually the difference between the probability that `pgm` is in sate A and the probability that `pgm` is in sate C will be less than 0.3.

Figure 7 explains the fifth and sixth examples. From step 1 to 15, the probability difference is larger than 0.3. However, eventually after step 15 the difference becomes less than 03.

5. CONCLUSIONS

We have developed a method for estimating software reliability of a program using a Markov reward model. The method uses an operational profile of a program, and the estimated reliability of each module, to estimate the reliability of a program. Using *iLTL*, we show how a variety of reliability properties may be specified and we provide an algorithm for checking these properties. While our technique provides a promising method for rigorous compositional software reliability estimation, empirical studies with real software systems remain to be carried out. Moreover, further research is needed to quantify the effect of deviations from the assumptions used in our model.

6. REFERENCES

[1] Adnan Aziz, Vigyan Singhal and Felice Balarin. It usually works: The temporal logic of stochastic systems. In *LNCS*, volume 939, pages 155–165, 1995.

[2] Andrea Bianco, Luca de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proceedings of Conferenco on Foundations of Software Technology and Theoretical Computer Science*, volume 1026, pages 499–513, 1995.

[3] Gianfranco Ciardo, Raymond A. Marie, Bruno Sericola and Kishor S. Trivedi. Performability analysis using semi-markov reward process. In *IEEE Transactions on Computers*, volume 39, pages 1251–1264, October 1990.

[4] Hoang Pham. *Software Reliability*. Springer, 2000.

[5] Holger Hermanns, Joost-Pieter Katoen, Joachim Meyer-Kayser and Markus Siegle. A markov chain model checker. In *S. Graf and M. Schwartzbach, editors, TACAS'2000*, pages 347–362, 2000.

[6] Jayant Rajgopal and Mainak Mazumdar. Modular operational test plans for inference on software reliability based on a markov model. In *IEEE Transactions on Software Engineering*, volume 28, pages 358–363, April 2002.

[7] J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. Springer-Verlag, 1976.

[8] Marta Kwiatkowska, Gethin Norman and David Parker. Prism: Probabilistic symbolic model checker. volume 2324, pages 200–204. LNCS, Springer-Verlag, April 2002.

[9] Moshe Y. Vardi. Probabilistic linear-time model checking: an overview of the automata-theoretic approach. In *Proc. 5th Int. AMAST Workshop Formal Methods for Real-Time and Probabilistic Systems*, volume 1601, May 1999.

[10] R. Gerth, D. Peled, M.Y. Vardi and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *IFIP/WG*, volume 6.1, pages 3–18, 1995.

[11] Suzana Andova, Holger Hermanns and Joost-Pieter Katoen. Discrete-time rewards model-checked. In *Formal Modeling and Analysis of Timed Systems 2003*, pages 88–104, 2003.

[12] YoungMin Kwon and Gul Agha. Linear inequality ltl (iltl): A model checker for discrete time markov chains. To appear in Int. Conf. on Formal Engineering Methods 2004.