Вероятностная верификации при проектировании вычислительных систем

С.Л. Френкель[1] , В.Н.Захаров[1], В.Г. Ушаков[2]

[1] Институт проблем информатики РАН, Москва, РФ, fsergei@mail.ru, VZakharov@ipiran.ru

[2] Московский государственный университет им.М.В.Ломоносова, Институт проблем информатики РАН, Ushakov@akado.ru

Аннотация - В статье рассматривается возможность снижения трудоемкости решения задачи вероятностной верификации проектов программно-аппаратных систем, подверженных случайным сбоям, и использования результатов верификации для защиты или коррекции элементов системы, функционирование которых может нарушаться при действии сбоев. Ранее предложенный авторами подход к вероятностному анализу устойчивости проектируемой системы, основанный на модели вычисления вероятности того, что некоторый случайный сбой не нарушит ее заданных свойств, сравнивается с широко известной моделью вероятностной верификации Probabilistic Model Checking (PMC). Показывается, что предложенный подход имеет преимущества по сравнению с PMC с точки зрения сложности диагностики причин невыполнения тех или иных требований к надежности проектируемой системы. Рассматривается техника статистической оценки вероятностей событий, используемых при решении задачи верификации.

Keywords: вероятностная верификация; Model Checking; моделирование вычислительных систем.

I. Введение

Одной из самых трудоемких задач, решаемых при проектировании, является задача верификации проекта на определенном этапе проектирования. При этом, как известно, стоимость обнаружения ошибок проектирования на каждом последующем этапе (например, на уровне регистровых передач (RTL - Register Transfer Level) после системного уровня) существенно возрастает [1].

Под “верификацией” проекта мы будем понимать любые действия, направленные на доказательство правильности (в смысле некоторого критерия корректности работы проектируемого устройства), и определение причин некорректности, в случае ее обнаружения. В отличие от “тестирования”, предназначенного в основном для обнаружения неисправностей (ошибок, дефектов) данного класса (и, следовательно, предполагающего наличие “модели неисправности”), задача верификации имеет дело со “свойствами” (property).

К задачам, которые можно рассматривать как верификационные, можно отнести как анализ выполнения свойств относительно спецификации проектируемой системы, так и относительно способности системы выполнять специфицированные функции при тех или иных отступлениях от структурных или функциональных требований, например, под действием помех [2], или при возможном проявлении ранее не обнаруженных дефектов. При таких условиях выводы о выполнении тех или иных свойств могут быть сделаны только с некоторой вероятностью. В статье будет рассмотрена возможность снижения трудоемкости решения задачи вероятностной верификации проектов программно-аппаратных систем, подверженных случайным сбоям, и использования результатов верификации для защиты или коррекции элементов системы, функционирование которых может нарушаться при действии сбоев.

Ранее, в [3], была предложена модель вычисления вероятности того, что некоторый случайный сбой не нарушит заданных свойств проектируемой системы, и было выполнено сравнение предложенной модели с моделью анализа последствий сбоев, основанной на комбинации моделирования с инжекцией неисправностей с формальной верификацией выполнения свойств (Model Checking - “Проверка Моделей”) [4]. Была показана возможность снижения суммарных трудозатрат на проектирование при использовании предложенной в [3] модели, благодаря возможности использования средств проектирования и формализованных описаний проектов, применяемых для решения других задач единого проектного цикла (логический синтез, моделирование, прогон на тестовых эталонах (бенчмарках) и т.д). В настоящей статье предлагается сравнение модели [3] с широко известной моделью вероятностной верификации Probabilistic Model Checking (PMC) [5]. При этом мы также учитываем важность использования формализованных описаний для снижения суммарной стоимости проектирования при решении как можно большего числа задач.

Работа выполнена при частичной поддержке грантов РФФИ № 12-07-00109 и № 13-07-00579, а также проекта № 1.1 фундаментальных исследований Президиума РАН № 16.

II. Факторы неопределенности при верификации

С наиболее обшей формальной точки зрения, неопределенность получения исчерпывающих результатов верификации свойств программы состоит в том, что задача верификации в общем виде является *неразрешимой (*undecidable), т.е. решение связано с ответами "да/нет ответа", но не существует программы, позволяющей получить ответ "да" или "нет".

Как следствие, любая программа формальной верификации, проверяющая выполнение условий "*безопасности*" (safety – "нечто плохое для системы никогда не случится"), способна установить их истинность лишь для некоторого набора *утверждений* (assertion), являющегося подмножеством множества всех таких утверждений. Иными словами, возможна лишь частичная (partial) верификация.

Другой источник неопределенности при верификации состоит в самой вероятностной природе целого ряда задач, решение которых может оказаться необходимо при проектировании. Сюда относятся как задачи предсказания поведения проектируемой системы в присутствии возможных сбоев, так и задачи предсказания поведения системы при проявление ошибок проектирования на данном уровне при не устранении тех или иных ошибок на предыдущих этапах.

Поскольку точно локализация и функциональные последствия наличия таких ошибок неизвестны, их можно попытаться характеризовать вероятностью проявления в результатах работы, и, соответственно, говорить о *вероятностной верификации*.

III.Вероятностная верификация

Если говорить о формальных методах верификации, то данную задачу призваны решать инструменты, основанные на Вероятностном (Probabilistic) Model Checking. Применение Probabilistic Model Checking (PMC) основано на модели проектируемой системы как системы помеченных переходов, с заданными вероятностями этих переходов [6]. В рамках данной концептуальной модели можно определить две математические модели – цепь Маркова (ЦМ) с дискретным временем (Discrete Time Markov Chain (DTMC), и с непрерывным временем (Continuous Time Markov Chain CTMC). CTMC позволяет отразить скорости переходов (*transition rates*), и поэтому может быть использована для оценки производительности.

Здесь мы будем рассматривать только DTMC в качестве основного инструмента для выражения вероятностей тех или иных проявлений некорректного поведения в проектируемой системе.

DTMC определяют как тройку $D=(S,P,L)$, где S – конечное множество состояний, $P : S\times S \rightarrow [0,1]$ – матрица вероятностей переходов, $\sum_{s'\in S} P(s,s')=1$ для всех $s\in S$, и $L : S \rightarrow 2^{AP}$ – разметка системы переходов, представляющая собой множество логических атомарных предложений (*atomic propositions* (AP), т.е. неделимых на более мелкие предложения), которые каждому состоянию *s* сопоставляют те или иные атомарные предложения, верные в данном состоянии s. AP определяются над множеством переменных (например, *x, s)*, числовых (0, 1, 2, …) или символьных (например, "true" ,"false") констант, функций (скажем, *max*(), min()) и тех или иных предикатов (например, $x = 2$, $x <= 0$, …), которые необходимы для описаний свойств проектируемой системы. 2^{AP}, как обычно, означает множество подмножеств множества *AP*.

Свойства задаются формулами *Probabilistic CTL* (PCTL), которые являются расширением CTL (Computational Tree Logic) посредством квантификации CTL-формул значениями вероятностей их выполнения, например $P_{<p}$(Ffail), (где Fq оператор временной логики (CTL,LTL) "Формула q будет истина когда-нибудь в будущем"), означающее, что вероятность того, что система в будущем будет в неисправном состоянии, меньше чем p.

Вопрос вычисления вероятности выполнения тех или иных специфицированных свойств может касаться, например, проблемы определения переменных программы или запоминающих элементов аппаратной части вычислительного устройства, подверженных сбоям, как из-за действия кратковременных внешних помех ("мягких"), так и из-за внутренних причин, не идентифицированных при проектировании (разумеется, такой моделью можно представить лишь часть возможных ошибок проектирования, которые приводят к однократному изменению того или иного перехода - несколько подробнее об этом будет сказано ниже).

"Мягкие" (soft) ошибки (сбои), соответствующие принятой в современной литературе модели "Soft Event Upset" (SEU), неформально можно описать как ошибки вычислений, вызываемые случайными изменениями любого из битов некоторого машинного слова, представляющего то или иное переменное, т.е. заменой '0' на '1', или '1' на '0'.

Более формально, для системы (S, T, S_0), где S – множество состояний, кодируемых булевыми векторами $s_1,s_2,..,s_n \in S$, $T \subseteq S\times S$ *– способ вычисления переходов как функций* $s_i := f_i(V)$, где f_i (V) – некоторые функции подмножеств $V \subseteq S$ [4], если булевы переменные начального состояния s_{0i} из множества S_0 *возможных начальных состояний* искажаются, то *модель переключает значение* в следующем цикле как $s_{0i} := \neg f_i(V)$. Заметим, что данная система переходов есть обобщение конечных автоматов (FSM - Final State Machine), и ее недетерминизм позволяет при выполнении алгоритма model checking моделировать влияние SEU на s_i на произвольном такте.

Хотя вопрос моделирования ошибок проектирования не является целью настоящей статьи, поясним, что указанная выше модель "мягкой ошибки" может моделировать многие виды возможных ошибок в проекте на достаточно высоком уровне проектирования (до RTL, по крайней мере), в частности тот широкий класс проектных ошибок (как на программном, так и на аппаратном уровне), которые могут моделироваться "инжекцией ошибок",

выполняющих “неисправную трансформацию” проектируемых объектов [7].

При этом в структурной автоматной модели (получаемой из абстрактного автомата двоичным кодированием переменных) эти трансформации неизбежно приводят к указанным выше изменениям в битах переменных. Известна также модель, в которой ошибки могут вводиться непосредственно в свойства, представляемые с помощью assertions [4, 7].

Ввиду того, что данные сбои – кратковременные (например, действуют в пределах одного машинного цикла), не все ошибки будут оказывать влияние на результаты работы программы, выполняемой проектируемой системой, и указанные выше феномены непроявления неисправностей могут трактоваться как самовосстановление (или, пользуясь современной терминологией, self-healing) [8, 9] после прекращения действия сбоя. В этом случае разработчик может решать вопрос о том, есть ли необходимость введения избыточности для обеспечения надежности, например, применяя репликацию в программах и в аппаратуре, или используя помехоустойчивые коды [8].

Очевидно, что практичность использования PMC зависит от его вычислительной сложности, и от его диагностических возможностей.

Пусть вероятностная система переходов, используемая на данном этапе проектирования, содержит N состояний и M переходов.

Будем рассматривать PCTL в качестве языка временной логики. Как известно [10, 11], алгоритм PCTL model checking имеет полиномиальную сложность по числу состояний используемой цепи Маркова и линейную по размеру проверяемой (для данной DTMC) PCTL формулы φ, описывающей верифицируемые свойства, где под размером формулы понимают число булевых конъюнкций и число термов во временных операторах CTL (“F”, “X”, ‘pUq” и т.д.). В настоящее время получены различные оценки сложности [11], в частности, весьма конструктивной на наш взгляд является оценка в виде $O(poly(size(D)))\cdot K\cdot|\varphi|$, где $size(D)$ - число состояний модели, $|\varphi|$ - размер формулы, специфицирующей проверяемые свойства, величина K зависит от способа реализации алгоритма вычисления формулы “until” $\varphi_1 U \varphi_2$ (в основном используется т.н. bounded model checking [12], в которой K - параметр, ограничивающий число итераций). Полином в приведенной оценке обычно третьей степени, что соответствует сложности известных методов вычисления вероятностей для цепи Маркова.

Отметим, однако, что хотя сложность полиноминальна по числу состояний, само число состояний может быть неприемлемо высоко, завися экспоненциально, например, от числа входных переменных. Существенно, что оценка вычислительной сложности Model Checking CTL-специфицированных систем равна $O(N)|\varphi|$, что означает, что вычислительная сложность PMC возрастает по сравнению с традиционным Model Checking как квадрат числа состояний модели. К этому вопросу мы еще вернемся.

Следующий важный вопрос состоит в том, как использовать в проектировании полученные вероятности невыполнения тех или иных условий (например, в результате сбоя или возможной ошибки). Чтобы диагностировать причину такого невыполнения свойства (в указанном выше вероятностном смысле), разработчику, помимо быстродействия используемого инструмента, требуется также иметь средства интерпретации полученных результатов, в частности, вероятностей выполнения верифицируемых свойств в терминах поведения системы в реальных условиях эксплуатации. При использовании для верификации PMC, пользователь получает вероятность выполнения требуемого свойства в виде вероятности достижения данного множества состояний. При этом считается, что свойство нарушено, если вероятность попадания в состояния, не отвечающие условию “безопасности” (*safety),* больше некоторого выбранного порога (с точки зрения неформальных критериев качества функционирования, формулируемых разработчиками). При применении Model Checking, если некоторое проверяемое свойство не выполняется, программа, реализующая Model Checking, формирует *контрпример,* представляющий собой трассу (последовательность переключения состояний или *путь*) на конкретном наборе входных данных, на которых данное свойство не выполняется.

При использовании обычного (не вероятностного)) Model Checking с использованием спецификации LTL-формулой *φ,* в качестве контрпримера используется один из таких путей, на котором истинной будет отрицание формулы *φ (¬φ).* Однако для PMC контрпримеры формируются по множеству путей, для которых вероятности не удовлетворяют заданному порогу, число которых растет экспоненциально. Действительно, вероятность попадания в момент T в некоторое подмножество состояний S цепи Маркова, такое, что $s|=\phi$ (“формула ϕ истинна в состоянии s”) с матрицей переходных вероятностей P(s, s’) вычисляется как $Prob(True\ U_{\le T}\ \phi) = \Sigma\ s_{|=}\phi\ P(s,T)$, где, напомним, $s_{|=}\phi$ означает истинность формулы CTL ϕ (задающей проверяемые свойства) в состоянии s, а P(s,T) – вероятность попадания цепи в состояние *s* в момент времени T, которая вычисляется для данной DTMC по всем путям перехода в *s* за T переходов [13]. Таким образом, определить причины высокой вероятности невыполнения свойств можно только путем перебора всех возможных состояний, в которых формула ϕ истинна, с их вероятностями. Примеры трудностей построения контрпримеров при использовании PMC можно найти в [14].

В настоящее время известны различные подходы к преодолению указанной трудности, в частности, используя SAT-based bounded model checking [13]. Некоторое сокращение перебора обеспечивает использование хорошо известных Binary Decision Diagram (BDD) и Multi-terminal BDD (MTBDD) для символьного

представления DTMC и логических функций, выражающих PCTL properties [15] при генерации контр-примеров (counterexamples), соответствующим трассам выполнения DTMC, на которых заданные свойства не выполняются. При этом ищется наиболее вероятный путь в DTMC. Однако этот метод (как и многие, основанные на этой идее, методы) требуют значительных затрат памяти.

В связи указанными трудностями использования PMC покажем, как предложенный авторами метод [16], который назовем CLNM (Combined Logical-Numerical Method), может быть использован для вероятностной верификации и анализа на ее основе необходимости защиты тех или иных элементов системы (переменных, программы, ячеек памяти), искажение информации в которых может привести к ошибкам функционирования системы. Поясним, что данное название связано с тем, что метод основан как на Model Checking, так и на вычислении вероятности проявления кратковременного сбоя в результатах работы системы оценки надежности в присутствии SEU.

В рамках данного подхода, вероятности нарушения правильного функционирования конечного автомата в результате действия кратковременного сбоя, задаваемого описанной выше моделью “мягкой” ошибки, вычисляются как вероятности попадания двумерной цепи Маркова, состояния которой представляют собой пары состояний цепи, не подверженной действию помехи, и цепи, в которой произошел указанный сбой, в поглощающее состояние, соответствующее возвращению на траекторию, образуемую только правильными переходами цепи. При этом цепь имеет еще одно поглощающее состояние, соответствующее искажению хотя бы одной выходной переменной до возвращения траектории переходов автомата в состояние, соответствующее исправному. Матрица переходных вероятностей вычисляется по заданной таблице переходов автоматов и вероятностям булевых единиц входных двоичных переменных автомата (поскольку переход в данное состояние определяется той или иной булевой операцией над текущим значением входной переменной и текущим состоянием). При этом можно определить число тактов до возможного возвращения к правильному функционированию, а именно, вектор вероятностей состояний, в которые попадает двумерная цепь Маркова при действием помехи за t тактов:

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

где начальное распределение $\vec{p}(0)$ определяется начальными состояниями исправного и неисправного автоматов, и P – матрица переходов указанной двумерной Цепи Маркова.

Если исправный автомат в начальный момент 0 находится в состоянии i_0, а неисправный – в состоянии $j_0 \neq i_0$, то $p_{(i0,j0)}(0) = 1$, а остальные координаты вектора $\vec{p}(0)$ равны нулю.

Таблица 1. Автомат с двумя состояниями

a_t	a_s	$X(a_t, a_s)$	$Y(a_t, a_s)$
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

Например, для автомата с двумя состояниями, заданного таблицей 1, где a_t, a_s-текущие и следующие состояния автомата, $X=(x_1,x_2,x_3)$,$Y=(y_1,y_2,y_3,y_4)$ – входные и выходные переменные, у которых каждая компонента принимает булевское значение, матрица переходов указанной двумерной цепи Маркова имеет вид Таблицы 2, где $p_i = \text{Prob}(x_i=1)$, i = 1,2,3, $q_i = 1- p_i$.

Таблица 2. Матрица вероятностей переходов автомата

	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

Состояния A_0, A_1 (Таблица 2) соответствуют указанным выше поглощающим состояниям двумерной цепи Маркова. Состояния указанной цепи Маркова (1,2), (2,1) означают, что в начальном состоянии автомат должен быть в состоянии a_1 (a_2), но в результате сбоя (ошибки, помехи) оказывается в a_2 (a_1).

Очевидно, что если число состояний исходного автомата N, то размер двумерной Цепи Маркова будет N(N-1) + 2, и в данном примере равно 4.

Поясним подробно построение матрицы двумерной цепи Маркова для автомата Таблицы 1.

В начальный момент времени из-за действия (необнаруженной) помехи автомат (Таблица 1) вместо состояния a_1 оказался в состоянии a_2. Построим цепь Маркова (ЦМ) Z_t, описывающую совместное функционирование исправного и неисправного автоматов. Общая ЦМ Z_t имеет 4 состояния: (1,2), (2,1), A_0 и A_1. Здесь (1,2) – исправный и неисправный автоматы находятся, соответственно, в состояниях a_1 и a_2, (2,1) – в состояниях a_2 и a_1, A_0 – к данному моменту произошло восстановление траектории функционирования автомата (он оказался в правильном состоянии) и выходные данные не были искажены, A_1 – неправильное функционирование уже проявилось в выходном сигнале.

Вычислим переходные вероятности ЦМ Z_t (таблица 2).

Состояния $A0$ и $A1$ являются поглощающими, и поэтому вероятность остаться в каждом из этих состояний равна 1.

Из состояния (1,2) в состояние $A0$ можно попасть только в том случае, когда поступит сигнал $(\neg x_1)x_2x_3$ (с вероятностью $q_1p_1p_2$), где "¬" означает логическое отрицание ("НЕ x_1"). Тогда оба автомата (исправный и неисправный) перейдут в одно и то же состояние a_2, а выходной сигнал у обоих автоматов – y_2y_4.

Из состояния (1,2) в состояние (1,2) можно попасть только в том случае, когда поступит сигнал $x_1x_2x_3$ (с вероятностью $p_1p_2p_3$). Тогда исправный автомат остается в состоянии a_1, неисправный – в состоянии a_2, а выходной сигнал у обоих автоматов – y_2y_3.

Из состояния (1,2) в состояние (2,1) можно попасть в том случае, когда поступит сигнал $\neg x1\neg x2$ (с вероятностью q_1q_2, третий компонент входного сигнала может быть любым). При этом исправный автомат переходит из состояния a_1 в состояние a_2, неисправный – из состояния a_2 в состояние a_1, а выходной сигнал у обоих автоматов – y_2y_3.

Наконец, из состояния (1,2) в состояние $A1$ можно попасть при разных входных сигналах. Во-первых, если поступит сигнал $x1\neg x2$ (с вероятностью p_1q_2,третий компонент входного сигнала может быть любым), тогда выходной сигнал у исправного автомата – y_2y_3, у неисправного – y_2y_4 (при этом исправный автомат переходит из состояния a_1 в состояние a_2, неисправный – из состояния a_2 в состояние a_1). Во-вторых, если поступит сигнал $x_1x_2(\neg x_3)$ (с вероятностью $p_1p_2q_3$), тогда выходной сигнал у исправного автомата – y_2y_4, у неисправного – y_1y_4 (при этом оба автомата попадают в состояние a_1). В-третьих, если поступит сигнал $(\neg x_1)x_2(\neg x_3)$ (с вероятностью $q_1p_2q_3$), тогда выходной сигнал у исправного автомата – y_2y_4, у неисправного – y_1y_4.

Структура CLNM представлена на рис.1, где для сравнения также представлен в обобщенном виде Probabilistic Model Checking. Как видно из рисунка, входными данными модели являются описания сбоев, вероятности единичных значений входных булевых переменных (указанные выше значения p_i), формальная спецификация свойств проектируемой системы. При этом, согласно рассмотренной выше марковской модели, сбои задаются как двоичные вектора, у которых единичной является только ячейка с вероятностью появления сбоя в переменной, определяемой индексом данного ненулевого значения в векторе (при некотором выбранном упорядочивании переменных состояний автомата).

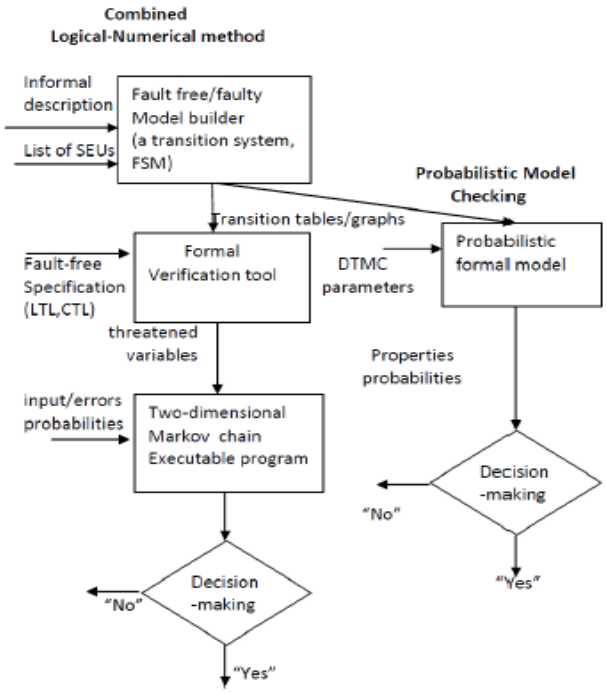

Рис.1. Две вероятностные модели верификации.

Свойства, описанные на CTL или LTL и проверяемые с помощью тех или иных программных инструментов Model Checking (SMV, NuSMV, spin), могут быть дополнены программными модулями инжекции неисправностей [7], описываемыми как указанные "мягкие" неисправности.

Полученный в результате формальной верификации список угрожаемых переменных "Threatened variables" вместе со списком их вероятностей передается в программу генерации и моделирования цепи Маркова.

Результатом является вероятность P_t (формула 1) того, что через t машинных тактов после прекращения действия помехи, эффект которой описывается соответствующим свойством на CTL, автомат восстановит правильное функционирование, а также стационарная вероятность π восстановления автомата ($t\rightarrow\infty$).

Например, вероятность π можно рассматривать как вероятность того, что проектируемая система обладает свойством "живучести" (*liveness)* [6], обязательно рассматриваемым при формальной верификации, т.е. *"если что-то* желательное *должно произойти, то оно обязательно произойдет, независимо от того, что происходит в настоящий момент"*.

Рассмотрим возможные действия разработчика, если полученные результаты (вероятность корректной работы) он рассматривает как неудовлетворительные.

В программных реализациях рассмотренного выше Probabilistic Model Checking, (например, PRISM [17]), где вероятности выполнения свойства (заданного формулами формальной логики) определяются как вероятность достижимости соответствующего подмножества

состояний по полному перебору всех путей переходов, и пространство состояний представляется в терминах BDD, пользователю не доступна в явном виде информация о вероятностях переходов, что затрудняет разработчику анализ причин низкой вероятности того, что проектируемая система отвечает требуемому свойству, в частности, восстановления после прекращения действия того или иного сбоя.

В отличие от этого, мы, как это видно из Рис.1, фактически декомпозируем задачу определения причины невыполнения условий при вероятностной верификации на две независимые задачи, а именно, задачи верификации выполнения проявления неисправности, используя традиционный (не вероятностный) Model Checking (верхние два блока на Рис.1), и, во-вторых, вычисляя вероятность проявления этого свойства по двумерной цепи Маркова, которая строится по исправному автомату и по автомату, у которого начальное состояние изменилось как инверсия булевой переменной, определенной на этапе Model Checking как одна из "Threatened variables" (Рис.1).

При этом матрица вероятностей переходов указанной цепи Маркова представляет собой структуру переходов, вероятности которых явно влияют на вероятность самовосстановления (или его отсутствия), и предоставляют разработчику информацию о возможных причинах ее недостаточного значения, на основании которой можно принять решение о необходимости защиты соответствующих переменных от ошибок. Вычислительная сложность такого двухступенчатого алгоритма может быть оценена как:

$$O((N(N-1)+2)^3)+O(N)|\varphi|,$$

где первое слагаемое – это кубическая сложность вычисления вероятностей для цепи Маркова (решения уравнения (1)) с N(N-1)+2 состояниями, а второе - вычислительная сложность алгоритма Model Checking, линейно зависящая от размера верифицируемой CTL формулы. Эта сложность представляет собой "стоимость" реализации CLNM, которая складывается из вычислительной сложности определения (методом Model Checking)) элементов, потенциально подверженных действию ошибок [3], и сложности решения линейных уравнений для двумерной марковской цепи (не больше чем кубической, как известно).

Итак, принципиальным отличием CLNM от PMC является, во-первых, декомпозиция алгоритма на детерминированный Model Checking, и вычисление вероятностей переходов в цепи Маркова, что приводит к аддитивному увеличению вычислительной сложности по сравнению с традиционным Model Checking (в отличие от мультипликативного увеличения для PMC, рассмотренного выше), и, во-вторых, явное использование в модели вероятностей входных переменных для вычисления вероятностей состояний и переходов, что упрощает разработчику принятие решений в процессе проектирования системы. Это обстоятельство ставит вопрос об оценке этих вероятностей.

IV. Измерение распределений вероятностей входных переменных с использованием тестовых эталонов (бенчмарков) для проектируемых систем

Так как, как отмечалось, использование бенчмарков в том или ином виде необходимо при проектировании, можно расширить его результаты для оценки указанных вероятностей, не привлекая дополнительных программных инструментов.

Для того, чтобы для нашей марковской модели самовосстановления оценить вероятности появления входов $x_1,x_2,..x_n$, , можно воспользоваться различными симуляторами архитектуры, широко используемыми при разработке вычислительных систем различного назначения и сложности (от микропроцессоров конкретного прикладного назначения до Systems on Chip (SoC) со всеми архитектурными и программными компонентами компьютера), а также набором бенчмарков, обеспечивающих типичное в среднем поведение проектируемой системы, т.е. задающее ту или иную его *рабочую нагрузку* (workload) при моделировании проекта системы [18]. При этом возможности симуляторов покрывают уровни от командного (instruction set simulator) до потактовоого моделирования (*cycle-accurate simulator*). Широкое применение при проектировании процессоров различного назначения имеет, например симулятор SimpleScalar [19] который позволяет осуществлять прогон бенчмарков, компилируемых для команд различных архитектур.

Например, прогоны бенчмарков используются в методологии верификации фирмы *ST Microelectronics* [20], при проектировании SoCs ("System on Chip"), а именно, совместно используя аппаратный эмулятор и тестовые задачи (тестбенчи) на Verilog, описывающие на автоматном уровне процессы управления, с моделированием на средствах симуляции Mentor Graphics. Аналогично, для верификации контроллеров памяти Hyperstone S5 с использованием наборов бенчмарков для инструкций использовался специальный эмулятор, а протокол I/O моделировался как конечный автомат [21]. Во всех указанных случаях возможно измерение частот переключения входных переменных соответствующих автоматов. Известны также многочисленные наборы бенчмарков, используемые для оценки производительности вычислительных систем [22-24].

Множество операций, соответствующих входам нашей автоматной модели, обеспечивающие статистически значимые оценки вероятностей (как частот переключения), можно выбрать исходя из требуемого размера доверительной области для вероятностей $p_1,p_2,..,p_r$, где r-число входов. Пусть M_p-длина (число операций) рассматриваемой трассы программы P (соответствующей трассе переходов моделирующего автомата). Тогда эта область представляет собой эллипсоид рассеивания для значений частот $f_i,= K_i /M_p$, i=1,..r, K_i – число операций в наборе

бенчмарков, обеспечивающих единичные значений i-го бита входного вектора, определяемого приравниванием критерия хи-квадрат к верхней t-процентной (например, 5-процентной) точке распределения хи-квадрат с r-1 степенью свободы. Если считать числа появлений операций в трассе независимыми событиями (не рассматривать зависимости между отдельными операциями), то распределение векторов, представляющих появления частоты операций в трассах программ, имеет полиномиальное распределение :

$$P(k_{1,\dots}\, k_i \dots k_n) = \frac{n!}{k_1!\, k_2!\dots k_n!}\, p_1^{k1}\, p_2^{k2} \dots p_n^{kn}$$

Соответственно, доверительная область имеет вид:

$$(K_1 - M_P * p_1)^2/K_1 + (K_2 - M_P * p_2)^2/m_2 + \dots + (K_r - M_p * p_N)^2/K_r \le \chi^2(r-1,t) \quad (2)$$

$$p_1 + p_2 + \dots p_r = 1$$

V.Заключение

Мы рассмотрели один из возможных способов снижения трудоемкости решения задачи верификации проектирования программно-аппаратных систем, в которых могут присутствовать либо ошибки проектирования, проявляющиеся как случайные сбои, либо система может быть подвержена сбоям случайной природы. В отличие от Probabilistic Model Checking, этот способ состоит в независимом использовании формальной верификации для анализа потенциальной уязвимости элементов и в вычислении вероятностей ошибок в результате работы устройства вследствие этих сбоев. Общее снижение трудоемкости может быть достигнуто благодаря использованию для верификации тех же программных инструментов, что и для других этапов проектирования, таких как системный синтез (автоматический или полуавтоматический), моделирование, прогон на бенчмарках. Отметим, что высокая эффективность применения бенчмарков при измерении частот (с использованием статистической методики раздела 4) различных значимых событий, характеризующих эффективность проектируемой системы, была нами опробована при решении задачи формальной верификации временных свойств программной системы выполнения программ, разработанных для x86–архитектуры, на RISC-архитектуре [16].

Литература

1. Pranab K. Nag, Anne Gattiker, Sichao Wei, R.D. Blanton, and Wojciech Maly. Modeling the Economics of Testing: A DFT Perspective // January–February 2002 0740-7475/02 IEEE

2. Baumann R. Soft errors in advanced computer systems // IEEE Design and Test, May-June, 2005. P. 258–266.

3. С.Л. Френкель, В.Н. Захаров, В.Г. Ушаков, Унифицированная высокоуровневая модель программно-аппаратной системы для верификации свойств надежности функционирования // Сборник материалов Международной научно-практической конференции «Инструменты и методы анализа программ» TMPA-2013 (г. Кострома 10 – 12 октября 2013 г.). Кострома, 2013, с. 108-119.

4. S. A. Seshia, W.Li, S. Mitra, Verification-guided soft error resilience // in Conference on Design, Automation and Test DATE07, 2007, p. 1442-1447.

5. M. Kwiatkowska. Quantitative verification: Models, techniques and tools // In ESEC/FSE'07, ACM Press, 2007, p. 449–458.

6. E.M. Clarke O. Grumberg, and D. A. Peled. Model Checking // MIT Press, 2000.

7. Ashish Darbari†, Bashir Al Hashimi, Peter Harrod, and Daryl Bradley. A New Approach for Transient Fault Injection using Symbolic Simulation // IOLTS 2008: p. 93-98.

8. S. Dolev, S.Frenkel, V. Sinelnikov,D. Tamir. Preserving Hamming Distance in Arithmetic and Logical Operations // Journal of Electronic Testing (JETTA), Springer Verlag, vol.29, 2013, p. 903-907.

9. Lala P. K., Kumar B. K. An Architecture for self-healing digital systems // Journal of Electronic Testing: Theory and Applications, 2003. Vol. 19, p. 523-535.

10. Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability // Formal Aspects of Computing, 6(5), 1994, p. 512–535,.

11. C. Courcoubetis M. Yannakakis, The complexity of probabilistic verification // Journal of the ACM, 24(4), 1995, p. 857- 907.

12. Baier, C., Katoen, J. Principles of Model Checking. 2008.

13. H. Aljazzar and S. Leue. Extended Directed Search for Probabilistic Timed Reachability // Tech. Report soft-06-03, 2003, University of Konstanz.

14. M. Schmalz, H. Völzer, and D. Varacca. Counterexamples in probabilistic LTL model checking for Markov chains // Technical Report 627, ETH Zürich, Switzerland, www.inf.ethz.ch/research/disstechreps/techreports, 2009.

15. M. Fujita, P.C. McGeer, and J. C.-Y. Yang. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation // Form. Methods Syst. Des., 10(2-3), 1997, p. 149–169.

16. S. Frenkel. Random Summation and its Application to the Performance Modelling Computer Systems // Proceedings of 17th EUROPEAN SIMULATION MULTICONFERENCE, ESM2003, Nottingham Trent University, England, 9-11 June, 2003, p. 278-283.

17. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems // in 23rd International Conference on Computer Aided Verification (CAV'11), ser. LNCS, vol. 6806. Springer, 2011, p. 585–591.

18. Almasri, I., Abandah, G., Shhadeh, A., Shahrour, A. Universal ISA simulator with soft processor FPGA

implementation // in 2011 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT), p. 1-6.
19. T. Austin, E. Larson, and D. Ernst. SimpleScalar:An Infrastructure for Computer System Modeling // Computer, vol. 35, no. 2, 2002, p. 59-67.
20. Allara, A., F. Brognara. Bringing Constraint Random into SoC SW-driven Verification. // Article presented at DVCon 2013.
21. Arthur Freitas. Hardware/Firmware Co-Verification Using ISS Integration and a SystemVerilog DPI, DVCon, 2007.
22. William L. Oberkampf , Timothy G. Trucano.Verification and validation benchmarks // Nuclear Engineering and Design 238 (2008), p. 716–743.
23. http://www.tomshardware.com/charts/desktop-cpu-charts-2010/benchmarks,112.html
24. C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications // In Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, October 2008.