ISSN 1816-0301 (Print)
ISSN 2617-6963 (Online)

ЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ

LOGICAL DESIGN

УДК 681.32

Поступила в редакцию 06.11.2018
Received 06.11.2018

Принята к публикации 26.11.2018
Accepted 26.11.2018

Верификация систем с параллелизмом поведения на основе графа достижимых состояний

Ю. В. Поттосин✉, В. И. Романов, Л. Д. Черемисинова

Объединенный институт проблем информатики
Национальной академии наук Беларуси, Минск, Беларусь
✉*E-mail: pott@newman.bas-net.by*

Аннотация. Рассматривается задача верификации систем управления на основе моделей их поведения, которая состоит в проверке соответствия поведения системы требованиям, предъявляемым спецификацией на ее проектирование. Тестирование предполагает выполнение экспериментов, заключающихся в моделировании исследуемой системы, в ходе которого она проверяется на вход-выходное соответствие модели. Тестовая последовательность генерируется на основе модели, описывающей желаемое поведение системы. Предлагается метод построения тестовой последовательности для верификации схемной (или программной) реализации системы управления с параллелизмом поведения, который основан на обходе графа состояний, достижимых при функционировании системы. Описывается метод построения множества достижимых полных состояний для параллельного алгоритма описания поведения системы управления и получения тестовых наборов. Полагается, что описание функционирования системы, заданное спецификацией на проектирование, корректно; проверке подлежит схемная (или программная) реализация, которая должна соответствовать этой спецификации.

Ключевые слова: параллельный алгоритм, верификация, параллельный автомат, граф достижимых состояний, спецификация на проектирование

Благодарности. Работа выполнена при финансовой поддержке БРФФИ (проект Ф17АРМ-008).

Для цитирования. Поттосин, Ю. В. Верификация систем с параллелизмом поведения на основе графа достижимых состояний / Ю. В. Поттосин, В. И. Романов, Л. Д. Черемисинова // Информатика. – 2019. – Т. 16, № 2. – С. 62–72.

Verification of systems with behavior parallelism on the basis of the graph of reachable states

Yuri V. Pottosin✉, Vladimir I. Romanov, Ljudmila D. Cheremisinova

The United Institute of Informatics Problems of the National Academy
of Sciences of Belarus, Minsk, Belarus
✉*E-mail: pott@newman.bas-net.by*

Abstract. Considered problem of model based verification of control systems is the checking whether the system behavior satisfies the requirements fixed in the design specification The testing includes the experiments consisting in simulation of investigated system to see input-output correspondence to the model.

The test sequence is generated on the basis of the model that describes the desired behavior of the system. The method to construct a test sequence for verification of hardware (or software) implementation of a control system with behavior parallelism is suggested that is based on traversal of the graph of the states that are reachable in system functioning. A method for constructing the set of reachable global states for a parallel algorithm of the control system behavior and a method to obtain the test sets are described. The description of the system functioning, which is given by the design specification, is assumed to be correct. The hardware (or software) implementation that must conform to this specification is to be verified.

Key words: parallel algorithm, verification, parallel automaton, graph of reachable states, specification for design

Acknowledgements. This work was supported by the BRFFR (project F17APM-008).

For citation. Pottosin Yu. V., Romanov V. I., Cheremisinova L. D. Verification of systems with behavior parallelism on the basis of the graph of reachable states. *Informatics*, 2019, vol. 16, no 2, pp. 62–72.

Введение. Развитие микроэлектроники и средств автоматизации проектирования обеспечило возможность проектирования микроэлектронных управляющих систем значительной сложности. При проектировании и реализации таких систем невозможно избежать ошибок, число которых растет вместе с расходами на исправление и ликвидацию их последствий [1, 2]. Снижение надежности проектирования и реализации современных микроэлектронных систем обусловило тот факт, что неотъемлемой частью процесса проектирования стало тестирование, в частности проверка соответствия поведения устройств требованиям, предъявляемым спецификацией на их проектирование. В русскоязычной литературе эта задача называется верификацией устройства. В англоязычной литературе верификация конкретизируется как соответствие реализации объекта проектирования условиям ее спецификации, или тестирование на основе модели. Тестирование на основе моделей предполагает проверку свойств реального управляющего устройства, связанных с его функциональностью, путем выполнения ряда экспериментов. В ходе экспериментов, заключающихся в моделировании исследуемой системы, проверяется ее функциональность, т. е. правильно ли она реагирует на подаваемые стимулы. Тестовая последовательность генерируется на основе модели, описывающей желаемое поведение системы. Успешное прохождение тестов, сгенерированных надлежащим образом на основе модели, служит достаточной гарантией правильности реализации этой системы.

Ключевым моментом тестирования является заключение о том, корректна ли (в некотором смысле) реализация относительно данной спецификации. Тестирование является одним из наиболее важных и широко используемых методов проверки схемных реализаций и программного обеспечения, на него приходится до половины общих затрат на их разработку. Это мотивирует всевозрастающий интерес к данной задаче [2].

В настоящей работе рассматривается задача верификации систем управления на основе моделей их поведения, или задача проверки системы на вход-выходное соответствие модели. Наиболее разработанным направлением в этой области является верификация на основе моделей, представленных конечными автоматами [3, 4].

Наряду с традиционно организованными системами, которые реализуют чисто последовательное поведение, задаваемое на языках описания конечных автоматов, существует ряд систем, в которых выразительных средств аппарата конечных автоматов оказывается недостаточно. Наиболее важным свойством таких систем управления является присущий им параллелизм происходящих в них процессов. В качестве моделей этих цифровых систем на алгоритмическом уровне используются их представления на языках параллельных алгоритмов управления (в основе которых лежит аппарат сетей Петри) [5–7]. Данные языки позволяют задавать и исследовать параллелизм процессов при функционировании цифровых систем.

Рассматривается также задача построения тестовой последовательности для верификации схемной (или программной) реализации управляющего устройства с параллелизмом поведения. Тестовая последовательность формируется на основе графа состояний, достижимых при функционировании системы с параллелизмом поведения. Предлагается метод построения множества достижимых полных состояний согласно описанию спецификации на проектирование устройства и получения тестовых наборов. Предполагается, что описание функционирования устройства, заданного спецификацией на проектирование, корректно (не содержит ошибок). Проверке

подлежит схемная (или программная) реализация, которая должна соответствовать спецификации на области задания последней.

Язык задания спецификации на проектирование систем с параллелизмом поведения. Одной из важнейших проблем автоматизации производственных процессов в различных отраслях промышленности является проблема проектирования систем управления. При решении задач реализации систем управления приходится иметь дело с параллелизмом, присутствующим в объектах управления. Управление такими объектами заключается в обеспечении согласованной работы взаимодействующих компонентов, работающих параллельно и асинхронно. Параллелизм, присутствующий в объектах управления, отражается в функциональной модели цифровых систем, управляющих данными объектами. Для цифровых систем рассматриваемого класса устройств характерно также и то, что управляющие воздействия и сигналы о состоянии объектов управления описываются булевыми переменными, лишь небольшой процент всей информации является числовым. В настоящее время в качестве языка задания спецификации на проектирование управляющих систем используются сети взаимодействующих конечных автоматов и языки, базирующиеся на формальной модели сети Петри.

Для задания спецификации на проектирование систем с параллелизмом поведения предлагается использовать параллельные алгоритмы логического управления, которые широко применяются при проектировании и тестировании цифровых систем. Одним из языков спецификации систем является язык ПРАЛУ [8] описания простых алгоритмов логического управления. Верификация алгоритма управления на языке ПРАЛУ значительно упрощается при приведении его к стандартному виду, представляемому моделью, названной параллельным автоматом [8]. Алгоритмы в таком виде являются подклассом цветных сетей Петри [5, 6] – расширенных сетей свободного выбора [9]. На языке параллельных автоматов алгоритм представляет собой совокупность стандартных цепочек вида

$$\mu_i : - k_i^1 \to k_i^2 \to \nu_i, \tag{1}$$

где μ_i и ν_i – множества частичных состояний, в которых автомат находится перед и после срабатывания i-го перехода; k_i^1 и k_i^2 – элементарные конъюнкции булевых переменных. В отличие от классического конечного последовательного автомата параллельный автомат может одновременно находиться в нескольких состояниях, называемых частичными состояниями. Все множество частичных состояний, в которых рассматриваемый параллельный автомат находится в некоторый момент времени, составляет полное внутреннее состояние. Смысл приведенной цепочки заключается в следующем. Если автомат находится одновременно в состояниях, составляющих множество μ_i, и булевы переменные приняли значения, обращающие конъюнкцию k_i^1 в единицу, то конъюнкция k_i^2 приобретает значение единицы и автомат переходит из частичных состояний, составляющих множество μ_i, в частичные состояния, составляющие множество ν_i. Любая из операций $- k_i^1$ или $\to k_i^2$ (ожидания или действия) может отсутствовать. Отсутствие операции $- k_i^1$ означает тождественное равенство единице конъюнкции k_i^1. Отсутствие $\to k_i^2$ означает в зависимости от интерпретации данной модели либо то, что все выходные переменные обращаются в нуль, либо то, что значения сигналов на выходе не меняются. Любое описание алгоритма на языке ПРАЛУ легко преобразуется в параллельный автомат [8].

Параллельный автомат задается:

– множеством $S = \{1, 2,\ldots\}$частичных внутренних состояний, входящих в μ_i и ν_i;

– входным и выходным алфавитами X и Y, которые состоят из входных и выходных булевых переменных, входящих соответственно в конъюнкции k_i^1 и k_i^2;

– переходами $\tau_i = (\mu_i \to \nu_i) / (k_i^1 \to k_i^2)$, соответствующими цепочкам (1).

Переход τi автомата срабатывает, когда текущая маркировка N_t включает все состояния из μ_i и переменные принимают значения, обращающие k_i^1 в единицу. После срабатывания перехода переменным из k_i^2 присваиваются значения, обращающие k_i^2 в единицу, а маркировка N_t заменяется на $(N_t \setminus \mu_i) \cup \nu_i$.

В качестве примера приведем описание параллельного автомата, заданного следующим множеством обобщенных переходов:

$\tau_1 = (1 \rightarrow 10) / (x_1\, x_2 \rightarrow y_1\ \bar{y}_2)$,
$\tau_2 = (10 \rightarrow 2.3.4) /(\ \bar{x}_2)$,
$\tau_3 = (2 \rightarrow 5.6) /(\rightarrow \bar{y}_1)$,
$\tau_4 = (3.5 \rightarrow 8) /(x_2)$,
$\tau_5 = (4 \rightarrow 7) /(x_1 \rightarrow \bar{y}_1)$,
$\tau_6 = (4 \rightarrow 9) /(x_1 \rightarrow y_1)$,
$\tau_7 = (7 \rightarrow 9) /(x_2)$,
$\tau_8 = (6.8.9 \rightarrow 11) /(\rightarrow \bar{y}_2)$,
$\tau_9 = (11 \rightarrow 1) /(x_1)$.

Постановка задачи проверки соответствия и метод ее решения. Задача проверки соответствия между спецификацией и ее схемной реализацией существенно отличается от проблем проверки эквивалентности пары схемных реализаций путем их функционального тестирования, верификации и валидации путем установления корректности проектируемой системы (соответствия поведения объектов взаимодействия ряду свойств). Отличие заключается в том, что проверяется не корректность спецификации (предполагается, что описание, отраженное в спецификации, корректно), а соответствие функциональности схемной реализации (или обнаружение неисправностей) функциональности, заданной спецификацией на проектирование. Основным средством тестирования схемной реализации на соответствие спецификации является моделирование, для проведения которого предварительно строится тестовая (проверяющая) последовательность, или просто тест. Под тестом понимается упорядоченная последовательность наборов значений входных сигналов и соответствующая последовательность изменений значений выходных сигналов, которые должны происходить после подачи этих наборов. Тестирование осуществляется на уровне вход-выходных последовательностей путем выполнения экспериментов над реальным устройством или его моделью. При моделировании тестовые наборы подаются на входные полюсы моделируемой системы, определяются изменения значений сигналов, происходящие на ее выходах, которые сравниваются с эталонными значениями.

На рис. 1 показаны форматы приведенного описания параллельного автомата в программном комплексе LOCON (LogicalControl) [10].

```
TITLE pott1
FORMAT PRL
AUTHOR Bibilo
DATE 28.04.2017
PROJECT LOCON_VHDL
DCL_PIN
EXT
INP
x1 x2
OUT
y1 y2
INTER
END_PIN
BLOCK pott1main
1: -^x1*x2 > y1*^y2 > 10;
10: -^x2 > 2.3.4;
2: > ^y1 > 5.6;
3.5: -x2 > 8;
4: -^x1 > ^y1 > 7;
   -x1 > y2 > 9;
7: -^x2 > 9;
6.8.9: >^y2 > 11;
11: -x1 > 1;
END_BLOCK pott1main
END_pott1
```

а)

```
TITLE pott1
FORMAT  PA
AUTHOR Bibilo
DATE 08.11.2017
PROJECT LOCON_VHDL
DCL_PIN
EXT
INP
x1 x2
OUT
y1 y2
INTER
END_PIN
BLOCK pott1main
1>10: ^x1*x2>y1*^y2;
10>2.3.4: ^x2>;
2>5.6: >^y1;
3.5>8: x2>;
4>7: ^x1>^y1;
4>9: x1>y2;
7>9: ^x2>;
6.8.9>11: >^y2;
11>1: x1>;
END_BLOCK pott1main
END_pott1
```

б)

Рис. 1. Форматы представления алгоритмов логического управления:
а) ПРАЛУ-алгоритм; *б)* параллельный автомат

Рассматриваемая задача проверки соответствия (рис. 2) включает этапы:
– генерации проверяющей последовательности на основе заданной спецификации;
– подачи тестовых наборов на входы устройства;
– наблюдения поведения устройства;
– определения, соответствует ли реализация исходной спецификации.

Рис. 2. Проверка соответствия между спецификацией и ее схемной реализацией

По способу проверки соответствия между реализацией и моделью, заданной спецификацией, можно выделить следующие варианты тестирования схемы:

1. *Проверку соответствия внутренних состояний схемы и модели*. В этом случае предполагается, что состояния, в которых находится схема, наблюдаемы. Проверяющая последовательность строится по модели и представляет собой последовательность троек (S_i^b, X_i, S_i^e), задающих S_i^b и S_i^e – начальное и конечное множества состояний, X_i – входной стимул.

2. *Проверку соответствия выходных откликов схемы и модели*. Проверяющая последовательность строится в виде пар (X_i, Y_i), где Y_i – изменение значений переменных при подаче на вход схемы входного набора X_i. Предполагается, что схема не допускает наблюдаемость состояний, но возможен перевод ее в начальное состояние, начиная с которого проводится эксперимент, путем сброса значений триггеров. В противном случае, если сброс невозможен, выполняется процедура инициализации схемы предварительной подачей на ее входы установочной последовательности.

3. *Проверку соответствия состояний и выходных откликов схемы и модели*. Проверяющая последовательность строится в виде четверок (S_i^b, X_i, S_i^e,Y_i), что обеспечивает наиболее полное тестирование схемы на соответствие спецификации.

Задача синтеза проверяющего теста заключается в построении конечного множества воздействий на систему, по реакции на которые можно определить правильность ее функционирования. Можно выделить два основных способа генерации входных воздействий в проверяющей последовательности:

– генерацию псевдослучайных наборов значений входных переменных. Минусы такого способа заключаются в том, что число возможных входных воздействий в каждом состоянии схемы (и ее модели) экспоненциально зависит от числа переменных и, главное, не все такие наборы попадают в область определения модели (а значит, и реализации);

– генерацию наборов значений входных переменных и проверяющей последовательности в целом исходя из описания модели. Этот случай не только обеспечивает наиболее полное тестирование схемы на области, определяемой спецификацией на ее проектирование, но и позволяет сократить длину проверяющей последовательности.

При тестировании параллельных управляющих систем будет рассматриваться задача генерации проверяющей последовательности в виде последовательности четверок (S_i^b, X_i, S_i^e, Y_i). Каждая четверка может порождаться переходами исходного алгоритма.

Основным инструментом, лежащим в основе методов анализа поведенческих свойств управляющей системы с параллелизмом поведения, служит граф достижимых состояний параллельного алгоритма управления, являющегося спецификацией на проектирование этой системы. Вершинам ориентированного графа достижимых состояний соответствуют полные состояния параллельного автомата, задаваемые разметками N_t, а дугам – переходы между полными состояниями. Дуга помечается символом τ_i перехода автомата и соединяет разметки N_t, такие, что сеть переходит от первой разметки ко второй при срабатывании перехода τ_i.

Граф достижимых состояний представляет собой ориентированный мультиграф, в котором могут быть петли и кратные дуги, различающиеся присвоенными им метками. Кроме того, очевидно, что каждый переход алгоритма управления может повторяться много раз в качестве метки дуг графа достижимости. Граф достижимости рассматриваемого выше параллельного алгоритма имеет 12 вершин (рис. 3).

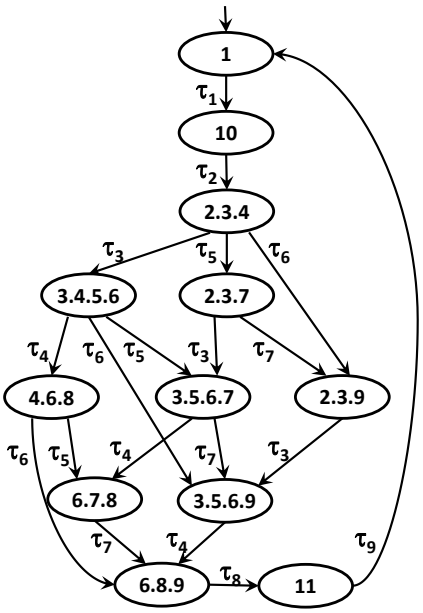

Рис. 3. Граф достижимых состояний

Построение множества полных состояний параллельного автомата. Для построения графа достижимости состояний предварительно получается множество всех полных состояний автомата. Рассматривается «скелет» параллельного автомата, подобный α-сети [8], который задается множеством частичных состояний $S = \{s_1, s_2, \ldots, s_n\}$ и множеством переходов $T = \{\tau_1, \tau_2, \ldots, \tau_k\}$, где $\tau_i = (S_i, F_i)$ и $S_i, F_i \subseteq S$, $i = 1, 2, \ldots, k$. Задается также начальное полное состояние $P_1 \subseteq S$. Переход τ_i происходит следующим образом: если $S_i \subseteq P_g$, где P_g – текущее полное состояние автомата, то полным состоянием в следующий момент времени будет $P_h = (P_g \setminus S_i) \cup F_i$.

Перед запуском алгоритма построения графа достижимости выполняется последовательное получение полных состояний $P = \{P_1, P_2, \ldots, P_l\}$, начиная с заданного начального состояния P_1. Далее происходит просмотр всех заданных переходов $\tau_i = (S_i, F_i)$ и, если $S_i \subseteq P_1$, по формуле $P_j = (P_1 \setminus S_i) \cup F_i$ формируется новое множество P_j, которое объявляется достижимым полным состоянием. Этот процесс повторяется для каждого из вновь внесенных в P множеств P_j. Процесс заканчивается, когда не получается новых множеств такого вида, отличных от уже полученных.

Алгоритм получения множества P полных состояний автомата:

1) $|P| := i := j := 0$, $P := P \cup \{S_1\}$;
2) $i := i + 1$; если $i \le |P|$, $j := 0$, перейти к п. 3,
   иначе перейти к п. 5;
3) $j := j + 1$, если $j \le k$, перейти к п. 4,
   иначе перейти к п. 2;
4) если $S_j \subseteq P_i$, $P := P \cup \{(P_i \setminus S_j) \cup F_j\}$, перейти к п. 3,
   иначе перейти к п. 3;
5) конец.

Результатом работы алгоритма для параллельного автомата *pott1* (см. рис. 1) является следующее множество, состоящее из 12 полных состояний: {{1}, {10}, {2, 3, 4}, {3, 4, 5, 6}, {2, 3, 7}, {2, 3, 9}, {4, 6, 8}, {3, 5, 6, 7}, {6, 8, 7}, {3, 5, 6, 9}, {6, 8, 9}, {11}}.

Построение графа достижимых состояний параллельного автомата. Вершинами ориентированного графа достижимых состояний являются полные состояния в виде множеств $P_1, P_2, \ldots, P_t$, которые получаются в результате выполнения описанного выше алгоритма. Из вершины P_g исходит дуга, заходящая в вершину P_h, если в исходном задании имеется переход $\tau_i = (S_i, F_i)$, такой, что $P_h = (P_g \setminus S_i) \cup F_i$). Метод, лежащий в основе данного алгоритма,

заключается в поиске пар вида (P_g, P_h), которым соответствует указанный переход $\tau_i = (S_i, F_i)$. Каждой такой паре соответствует дуга искомого графа. Представлением получаемого графа является перечень дуг и приписанных им меток. Это представление удобно для описания алгоритма. Оно является также довольно компактным, поскольку графы данного вида обладают сравнительно небольшим числом дуг.

Исходными данными для алгоритма служат следующие объекты:

– множество частичных состояний $S = \{s_1, s_2, \ldots, s_n\}$;

– совокупность троек (S_1, F_1, n_1), (S_2, F_2, n_2), …, (S_s, F_s, n_s), задающих переходы $\tau_i = (S_i, F_i)$, где $n_i = i$ – номер перехода автомата;

– множество полных состояний $P = \{P_1, P_2, \ldots, P_t\}$.

В результате строится ориентированный граф, заданный перечислением дуг: $X = \{x_1, x_2, \ldots, x_r\}$ – номера начал дуг, $Y = \{y_1, y_2, \ldots, y_r\}$ – номера концов дуг, $N = \{n_1, n_2, \ldots, n_r\}$ – номера переходов, являющихся метками дуг.

В работе алгоритма используются множества U, V и W для представления промежуточных результатов.

Алгоритм построения графа достижимых состояний:

1) $X := Y := \varnothing$, $i := 0$;

2) $i := i + 1$; если $i \le t$, $j := 0$, перейти к п. 3,
иначе перейти к п. 7;

3) $j := j + 1$; если $j = i$, перейти к п. 3,
иначе, если $j \le t$, перейти к п. 4,
иначе перейти к п. 2;

4) $W := P_i \cap P_j$, $U := P_i \setminus W$, $V := P_j \setminus W$, $l := 0$, перейти к п. 5;

5) $l := l + 1$; если $l \le s$, перейти к п. 6,
иначе перейти к п. 3;

6) если $U = S_l$ и $V = F_l$, $X := X \cup \{i\}$, $Y := Y \cup \{j\}$, $N := N \cup \{l\}$, перейти к п. 3,
иначе перейти к п. 3;

7) конец.

При применении алгоритма к автомату *pott1* (см. рис. 1) получается граф достижимости (см. рис. 3), имеющий следующее множество дуг: ({1}, {10}, τ_1), ({10}, {2,3,4}, τ_2), ({2,3,4}, {3,4,5,6}, τ_3), ({2,3,4}, {2,3,7}, τ_5), ({2,3,4}, {2,3,9}, τ_6), ({3,4,5,6}, {4,6,8}, τ_4), ({3,4,5,6}, {3,5,6,7}, τ_5), ({3,4,5,6}, {3,5,6,9}, τ_6), ({2,3,7}, {3,5,6,7}, τ_3), ({2,3,7}, {2,3,9}, τ_7), ({4,6,8}, {6,7,8}, τ_5), ({4,6,8}, {6,8,9}, τ_6), ({3,5,6,7}, {6,7,8}, τ_4), ({3,5,6,7}, {3,5,6,9}, τ_7), ({2,3,9}, {3,5,6,9}, τ_3), ({6,7,8}, {6,8,9}, τ_7), ({3,5,6,9}, {6,8,9}, τ_4), ({6,8,9}, {11}, τ_8), ({11}, {1}, τ_9).

В общем случае граф достижимости является ориентированным мультиграфом, так как он может содержать кратные дуги, различающиеся присвоенными им метками.

Предложенные алгоритмы построения графа достижимых состояний параллельного автомата были программно реализованы на языке C++ в среде кроссплатформенного программирования Qt на основе использования разработанных инструментальных средств логического проектирования [11], включающих классы булевых матриц и векторов для представления множеств, а также класс представления параллельных автоматов, которые были модернизированы после переноса в среду Qt.

Построение тестовой последовательности для функциональной верификации. Задача синтеза проверяющего теста заключается в построении конечного множества воздействий на систему, по реакциям на которые можно определить правильность ее функционирования. Разработка алгоритмов синтеза тестов требует использования математических моделей, позволяющих отобразить поведение системы. В связи с практической значимостью и теоретическим интересом наиболее изученной областью верификации на основе моделей является тестирование конечных автоматов. Первые работы в этой области появились около 50 лет назад. Тестовая последовательность формируется путем объединения нескольких подпоследовательностей, и, как правило, подпоследовательности имеют перекрытия. В литературе встречается достаточное число работ, в которых предлагаются эвристики для сокращения числа перекрытий с целью

уменьшения общей длины тестовой последовательности. Тестовая последовательность строится следующим образом:

– заранее, до начала процесса тестирования, в виде единого маршрута по графу, проходящего через все дуги графа;

– заранее в виде множества маршрутов, начинающихся в начальной вершине (состоянии) и покрывающих в совокупности все дуги графа;

– в процессе моделирования системы, в этом случае тестовые наборы генерируются динамически при обходе графа в зависимости от откликов схемы на поданные наборы.

Очевидно, что построение тестовой последовательности в виде единого маршрута возможно, если граф достижимых состояний является сильно связным, т. е. если из каждой его вершины достижима любая другая вершина. Если граф достижимости не является сильно связным, то возможно построение тестовой последовательности в виде множества маршрутов из начальной вершины графа. Если граф не является сильно связным, но все его вершины достижимы из начальной, то при моделировании управляющей системы требуется ее рестарт (сброс триггеров блока памяти) при переходе от одного теста к другому. После рестарта производится выбор следующей тестовой последовательности.

Задача построения тестовой последовательности на основе графа достижимых состояний заключается в построении такого кратчайшего ориентированного маршрута (чередующейся последовательности вершин и дуг) на орграфе $G = (V, E)$, который проходит через каждую дугу графа по крайней мере один раз (в общем случае не один раз). В этой постановке задача построения кратчайшего ориентированного маршрута аналогична задаче китайского почтальона [12], в которой ищется кратчайший путь, проходящий через все дуги заданного орграфа. В силу трудоемкости решения такой задачи нахождение точного решения (с минимумом числа возможных повторных прохождений дуг графа достижимости) в общем случае не представляется возможным.

Для решения задачи обхода графа достижимых состояний можно использовать один из известных методов обхода дуг ориентированного графа, разработанных для случая детерминированных автоматных моделей [3, 4, 13–16]. По последовательности переходов алгоритма управления, соответствующих меткам найденной последовательности дуг ориентированного графа, можно определить входные стимулы тестовой последовательности для подачи на входы тестируемой реализации системы управления. Следует заметить, что таким образом для каждого достижимого состояния системы определяются только те стимулы, которые заданы в реализуемой спецификации, т. е. тестирование проводится только на заданной спецификацией области определения. Если в реализации допустимы какие-то другие стимулы, то это никак не влияет на процесс тестирования. Фактически это означает, что для заданной реализации исходного модельного алгоритма управления тестируется некоторая часть ее функциональности, заданная переходами по входным стимулам и состояниям модельного алгоритма, достижимым из начального состояния.

Обход дуг графа достижимости, изображенного на рис. 3, задается следующим циклическим маршрутом, начинающимся и заканчивающимся в начальном состоянии 1:

1, τ_1, 10, τ_2, 2.3.4, τ_3, 3.4.5.6, τ_4, 4.6.8, τ_6, 6.8.9, τ_8, 11, τ_9,
1, τ_1, 10, τ_2, 2.3.4, τ_3, 3.4.5.6, τ_4, 4.6.8, τ_5, 6.7.8, τ_7, 6.8.9, τ_8, 11, τ_9,
1, τ_1, 10, τ_2, 2.3.4, τ_3, 3.4.5.6, τ_6, 3.5.6.9, τ_4, 6.8.9, τ_8, 11, τ_9,
1, τ_1, 10, τ_2, 2.3.4, τ_3, 3.4.5.6, τ_5, 3.5.6.7, τ_4,6.7.8, τ_7, 6.8.9, τ_8, 11, τ_9,
1, τ_1, 10, τ_2, 2.3.4, τ_3, 3.4.5.6, τ_5, 3.5.6.7, τ_7, 3.5.6.9, τ_4, 6.8.9, τ_8, 11, τ_9,
1, τ_1, 10, τ_2, 2.3.4, τ_5, 2.3.7, τ_3, 3.5.6.7, τ_4, 6.7.8, τ_7, 6.8.9, τ_8, 11, τ_9,
1, τ_1, 10, τ_2, 2.3.4, τ_5, 2.3.7, τ_3, 3.5.6.7, τ_7, 3.5.6.9, τ_4, 6.8.9, τ_8, 11, τ_9,
1, τ_1, 10, τ_2, 2.3.4, τ_5, 2.3.7, τ_7, 2.3.9, τ_3, 3.5.6.9, τ_4, 6.8.9, τ_8, 11, τ_9,
1, τ_1, 10, τ_2, 2.3.4, τ_6, 2.3.9, τ_3, 3.5.6.9, τ_4, 6.8.9, τ_8, 11, τ_9, 1.

При проверке соответствия между состояниями и выходными откликами схемной реализации и ее моделью, задаваемой параллельным алгоритмом управления, тестовая последовательность строится в виде четверок (S_i^b, X_i, S_i^e, Y_i), где S_i^b и S_i^e – начальное и конечное множества состояний, X_i – входной стимул, Y_i – изменение значений переменных при подаче на вход схемы

входного набора X_i. Это обеспечивает наиболее полное тестирование схемы на соответствие спецификации. При отображении тестовой последовательности начальное состояние S_i^b (кроме S_1^b) будем опускать, считая его равным S^e_{i-1}:

$(1, x_1 x_2, 10, y_1 \bar{y}_2)$, $(\bar{x}_2, 2.3.4, –)$, $(–, 3.4.5.6, \bar{y}_1)$, $(x_2, 4.6.8, –)$, $(x_1, 6.8.9, y_1)$, $(–,11, \bar{y}_2)$, $(x_1, 1, –)$,

$(x_1 x_2, 10, y_1 \bar{y}_2)$, $(\bar{x}_2, 2.3.4, –)$, $(–, 3.4.5.6, \bar{y}_1)$, $(x_2, 4.6.8, –)$, $(x_1, 6.7.8, \bar{y}_1)$, $(x_2, 6.8.9, y_1)$, $(–, 11, \bar{y}_2)$, $(x_1, 1, –)$,

$(x_1 x_2, 10, y_1 \bar{y}_2)$, $(\bar{x}_2, 2.3.4, –)$, $(–, 3.4.5.6, \bar{y}_1)$, $(x_1, 3.5.6.9, y_1)$, $(x_2, 6.8.9, –)$, $(–, 11, \bar{y}_2)$, $(x_1, 1, –)$,

$(x_1 x_2, 10, y_1 \bar{y}_2)$, $(\bar{x}_2, 2.3.4, –)$, $(–, 3.4.5.6, \bar{y}_1)$, $(x_1, 3.5.6.7, \bar{y}_1)$, $(x_2, 6.7.8, –)$, $(x_2, 6.8.9, y_1)$, $(–, 11, \bar{y}_2)$, $(x_1, 1, –)$,

$(x_1 x_2, 10, y_1 \bar{y}_2)$, $(\bar{x}_2, 2.3.4, –)$, $(–, 3.4.5.6, \bar{y}_1)$, $(x_1, 3.5.6.7, \bar{y}_1)$, $(x_2, 3.5.6.9, –)$, $(x_2, 6.8.9, –)$, $(–, 11, \bar{y}_2)$, $(x_1, 1, –)$,

$(x_1 x_2, 10, y_1 \bar{y}_2)$, $(\bar{x}_2, 2.3.4, –)$, $(x_1, 2.3.7, \bar{y}_1)$, $(–, 3.5.6.7, \bar{y}_1)$, $(x_2, 6.7.8, –)$, $(x_2, 6.8.9, y_1)$, $(–, 11, \bar{y}_2)$, $(x_1, 1, –)$,

$(x_1 x_2, 10, y_1 \bar{y}_2)$, $(\bar{x}_2, 2.3.4, –)$, $(x_1, 2.3.7, \bar{y}_1)$, $(–, 3.5.6.7, \bar{y}_1)$, $(x_2, 3.5.6.9, –)$, $(x_2, 6.8.9, –)$, $(–, 11, \bar{y}_2)$, $(x_1, 1, –)$,

$(x_1 x_2, 10, y_1 \bar{y}_2)$, $(\bar{x}_2, 2.3.4, –)$, $(x_1, 2.3.7, \bar{y}_1)$, $(x_2, 2.3.9, –)$, $(–, 3.5.6.9, \bar{y}_1–)$, $(x_2, 6.8.9, –)$, $(–, 11, \bar{y}_2)$, $(x_1, 1, –)$,

$(x_1 x_2, 10, y_1 \bar{y}_2)$, $(\bar{x}_2, 2.3.4, –)$, $(x_1, 2.3.9, y_1)$, $(–, 3.5.6.9, \bar{y}_1)$, $(x_2, 6.8.9, –)$, $(–, 11, \bar{y}_2)$, $(x_1, 1, –)$.

Если граф достижимости не является сильно связным, но каждое его состояние достижимо из начального и тестируемая схемная реализация допускает рестарт из начального состояния, то можно строить сразу не граф, а усеченное дерево достижимости (рис. 4). Его построение из графа достижимости производится путем обрывания путей из начальной вершины при обнаружении достигнутых ранее разметок. В таком случае искомая тестовая последовательность ищется в виде множества цепей, начинающихся в начальной вершине (состоянии) и имеющих концы в листовых вершинах дерева достижимости. При этом цепи в совокупности должны покрывать все дуги дерева.

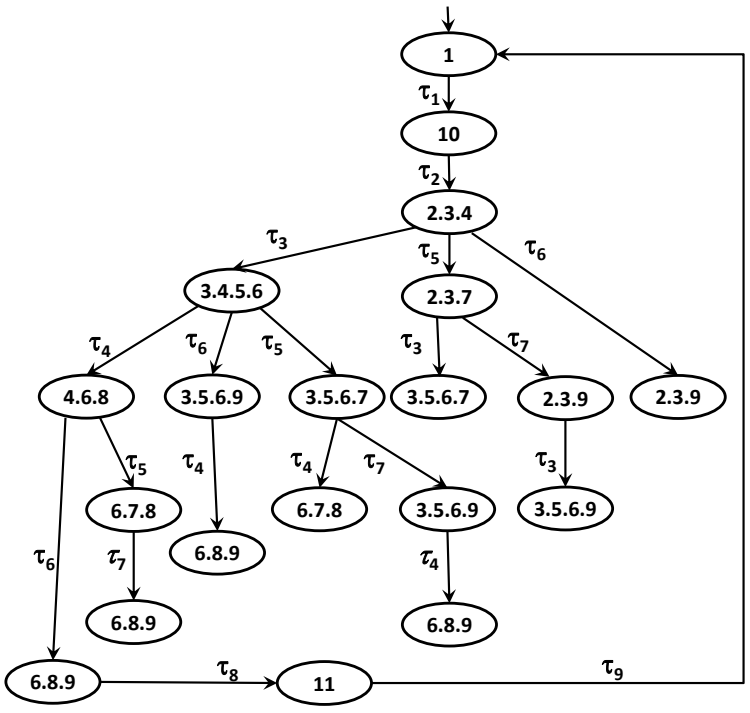

Рис. 4. Дерево достижимых состояний

Получается восемь тестовых последовательностей:

(1, $x_1 x_2$, 10, y_1 $\bar{y}_2$), ($\bar{x}_2$, 2.3.4, –), (–, 3.4.5.6, $\bar{y}_1$), (x_2, 4.6.8, –), (x_1, 6.8.9, y_1), (–, 11, $\bar{y}_2$), (x_1, 1, –);

(1, $x_1 x_2$, 10, y_1 y_2), ($\bar{x}_2$, 2.3.4, –), (–, 3.4.5.6, $\bar{y}_1$), (x_2, 4.6.8, –), (x_1 , 6.7.8, $\bar{y}_1$), (x_2, 6.8.9, y_1);

(1, $x_1 x_2$, 10, y_1 $\bar{y}_2$), ($\bar{x}_2$, 2.3.4, –), (–, 3.4.5.6, $\bar{y}_1$), (x_1, 3.5.6.9,y_1), (x_2, 6.8.9, –);

(1, $x_1 x_2$, 10, y_1 $\bar{y}_2$), ($\bar{x}_2$, 2.3.4, –) ,(–, 3.4.5.6, $\bar{y}_1$), (x_1, 3.5.6.7, $\bar{y}_1$), (x_2, 6.7.8,–);

(1, $x_1 x_2$, 10, y_1 y_2), ($\bar{x}_2$, 2.3.4, –), (–, 3.4.5.6, $\bar{y}_1$), (x_1, 3.5.6.7, $\bar{y}_1$), (x_2, 3.5.6.9,–), (x_2, 6.8.9, –);

(1, $x_1 x_2$, 10, y_1 $\bar{y}_2$), ($\bar{x}_2$, 2.3.4, –), (x_1, 2.3.7, $\bar{y}_1$), (–, 3.5.6.7, $\bar{y}_1$);

(1, $x_1 x_2$, 10, y_1 $\bar{y}_2$), ($\bar{x}_2$, 2.3.4, –), (x_1, 2.3.7, $\bar{y}_1$), (x_2, 2.3.9,–), (–, 3.5.6.9, $\bar{y}_1$–);

(1, $x_1 x_2$, 10, y_1 $\bar{y}_2$), ($\bar{x}_2$, 2.3.4, –),(x_1, 2.3.9, y_1).

Заключение. В работе рассматривается задача анализа системы управления на входвыходное соответствие модели для наиболее сложного и недостаточно изученного случая, когда исходная спецификация описывает систему с параллелизмом поведения. Предложенный метод основан на построении графа достижимости параллельного алгоритма управления и поиска обхода дуг этого графа.

Список использованных источников

1. Валидация на системном уровне. Высокоуровневое моделирование и управление тестированием : пер. с англ. Е. Б. Махияновой / М. Чэнь [и др.]. – М. : Техносфера, 2014. – 296 с.

2. Tretmans, J. Model based testing with labelled transition systems / J. Tretmans // Formal Methods and Testing: Lecture Notes in Computer Science. – Springer, 2008. – Vol. 4949. – P. 1–38.

3. Lee, D. Principles and methods of testing finite state machine – a survey / D. Lee, M. Yannakakis // Proceedings of the IEEE. – 1996. – Vol. 84, no. 8. – P. 1090–1123.

4. Верификация автоматных программ / С. Э. Вельдер [и др.]. – СПб. : Наука, 2011. – 244 с.

5. Питерсон, Дж. Теория сетей Петри и моделирование систем : пер. с англ. М. В. Горбатовой, В. Л. Торхова, В. Н. Четверикова / Дж. Питерсон. – М. : Мир, 1984. – 264 с.

6. Котов, В. Е. Сети Петри / В. Е. Котов. – М. : Наука, 1984. – 160 с.

7. Karatkevich, A. Dynamic Analysis of Petri Net-based Discrete Systems / A. Karatkevich. – Berlin : Springer-Verlag, 2007. – Vol. 358. – 166 p.

8. Закревский, А. Д. Параллельные алгоритмы логического управления / А. Д. Закревский. – Минск : Ин-т техн. кибернетики НАН Беларуси, 1999. – 202 с.

9. Hack, M. Analysis of production schemata by Petri nets / M. Hack // Project MAK-94. – Cambridge, 1972. – 119 p.

10. Experimental system of automated design of logical control devices / A. D. Zakrevskij [et al.] // Proc. of the Intern. Workshop "Discrete Optimization Methods in Scheduling and Computer-Aided Design". – Минск : Ин-т техн. кибернетики НАН Беларуси, 2000. – С. 216–221.

11. Романов, В. И. Разработка инструментальных средств логического проектирования / В. И. Романов // Логическое проектирование. – Минск : Ин-т техн. кибернетики НАН Беларуси, 2001. – Вып. 6. – С.151–170.

12. Thimbleby, H. The directed Chinese Postman Problem / H. Thimbleby // Software Practice and Experience. – 2003. – Vol. 33, no. 11. – P. 1081–1096.

13. Бурдонов, И. Б. Неизбыточные алгоритмы обхода ориентированных графов. Детерминированный случай / И. Б. Бурдонов, А. С. Косачев, В. В. Кулямин // Программирование. – 2003. – № 5. – С. 11–30.

14. Черемисинова, Л. Д. Построение тестов полного перебора для оценки энергопотребления последовательностных схем / Л. Д. Черемисинова // Информатика. – 2017. – № 4. – С. 104–110.

15. Kanso, B. Compositional testing for FSM-based models / B. Kanso, O. Chebaro // Intern. J. of Software Engineering & Applications (IJSEA). – 2014. – Vol. 5, no. 3. – P. 1–20.

16. Витязь, К. А. Алгоритмы построения функциональных тестов для цифровой схемы на основе автоматной модели ее поведения / К. А. Витязь, В. И. Романов // Танаевские чтения : доклады Восьмой Междунар. науч. конф., Минск, 27–30 марта 2018 г. – Минск : ОИПИ НАН Беларуси, 2018. – С. 52–56.

References

1. Chen M., Qin X., Koo H.-M., Mishra P. *System-Level Validation High-Level Modeling and Directed Test Generation Techniques*. New York, Springer-Verlag, 2013, 250 p.

2. Tretmans J. Model based testing with labelled transition systems. *Formal Methods and Testing: Lecture Notes in Computer Science*. Springer, 2008, vol. 4949, pp. 1–38.

3. Lee D., Yannakakis M. Principles and methods of testing finite state machine – a survey. *Proceedings of the IEEE*, 1996, vol. 84, no. 8, pp. 1090–1123.

4. Vel'der S. E., Lukin M. A., Shalyto A. A., Jaminov B. R. Verifikacija avtomatnyh program. *Verification of Automation Programs*. Saint Petersburg, Nauka, 2011, 244 p. (in Russian).

5. Peterson J. *Petri Net Theory and the Modeling of Systems*. New York, Prentice Hall, 1981, 290 p.

6. Kotov V. E. Seti Petri. *Petri Nets*. Moscow, Nauka, 1984, 160 p. (in Russian).

7. Karatkevich A. *Dynamic Analysis of Petri Net-based Discrete Systems*. Berlin, Springer-Verlag, 2007, vol. 358, 166 p.

8. Zakrevskij A. D. Parallel'nye algoritmy logicheskogo upravlenija. *Parallel Algorithms of Logical Control*. Minsk, Institut tehnicheskoj kibernetiki Nacional'noj akademii nauk Belarusi, 1999, 202 p. (in Russian).

9. Hack M. Analysis of production schemata by Petri nets. *Project MAK-94*, Cambridge, 1972,119 p.

10. Zakrevskij A. D., Pottosin Yu. V., Vasilkova I. V., Romanov V. I. Experimental system of automated design of logical control devices. *Proceedings of the International Workshop "Discrete Optimization Methods in Scheduling and Computer-Aided Design"*. Minsk, Institut tehnicheskoj kibernetiki Nacional'noj akademii nauk Belarusi, 2000, pp. 216–221.

11. Romanov V. I. Razrabotka instrumental'nyh sredstv logicheskogo proektirovayija [Development of Instruments for Logical Design]. Logicheskoe proektirovanie.Vyp. 6 [*Logical Design*. *Issue 6*]. Minsk, Institut tehnicheskoj kibernetiki Nacional'noj akademii nauk Belarusi, 2001, pp. 151–170 (in Russian).

12. Thimbleby H. The directed Chinese Postman Problem. *Software Practice and Experience*, 2003, vol. 33, no. 11, pp. 1081–1096.

13. Burdonov I. B, Kosachev A. S., Kuljamin V. V. Neizbytochnye algoritmy obhoda orientirovannyh grafov. Determinirovannyj sluchaj [Irredandant algorithms for traversal of directed graphs. The determinate case]. Programmirovanie [*Programming*], 2003, no. 5, pp. 11–30 (in Russian).

14. Cheremisinova L. D. Postroenie testov polnogo perebora dlja ocenki energopotreblenija posledovatel'nostnyh shem [Constructing tests of exhaustive search for estimation of power consumption of sequential circuits]. Informatika [*Informatics*], 2017, no. 4, pp. 104–110 (in Russian).

15. Kanso B., Chebaro O. Compositional testing for FSM-based models. *International Journal of Software Engineering & Applications (IJSEA)*, 2014, vol. 5, no 3, pp. 1–20.

16. Vitjaz' K. A., Romanov V. I. Algoritmy postroenija funkcional'nyh testov dlja cifrovoj shemy na osnove avtomatnoj modeli ejo povedenija [Algorithms for constructing functional tests for a digital circuit on the base of automaton model of its behavior]. Tanaevskie chtenija: doklady Vos'moj Mezhdunarodnoj nauchnoj konferencii [*Tanaev Lecturings: Proceedings of the Eighth International Scientific Conference*, *Minsk, 27–30 March 2018*], Minsk, Ob"edinennyj institut problem informatiki Nacional'noj akademii nauk Belarusi, 2018, pp. 52–56 (in Russian).

Информация об авторах

Поттосин Юрий Васильевич, кандидат физико-математических наук, ведущий научный сотрудник, Объединенный институт проблем информатики Национальной академии наук Беларуси, Минск, Беларусь.
E-mail: pott@newman.bas-net.by

Романов Владимир Ильич, кандидат технических наук, ведущий научный сотрудник, Объединенный институт проблем информатики Национальной академии наук Беларуси, Минск, Беларусь.
E-mail: rom@newman.bas-net.by

Черемисинова Людмила Дмитриевна, доктор технических наук, главный научный сотрудник, Объединенный институт проблем информатики Национальной академии наук Беларуси, Минск, Беларусь.
E-mail: cld@newman.bas-net.by

Information about the authors

Yuri V. Pottosin, Cand. Sci. (Phys.-Math.), Leading Researcher, The United Institute of Informatics Problems of the National Academy of Sciences of Belarus, Minsk, Belarus.
E-mail: pott@newman.bas-net.by

Vladimir I. Romanov, Cand. Sci. (Eng.), Leading Researcher, The United Institute of Informatics Problems of the National Academy of Sciences of Belarus, Minsk, Belarus.
E-mail: rom@newman.bas-net.by

Ljudmila D. Cheremisinova, Dr. Sci. (Eng.), Chief Researcher, The United Institute ofInformatics Problems of the National Academy of Sciences of Belarus, Minsk, Belarus.
E-mail:cld@newman.bas-net.by