УДК 519.68/69

А. М. Андреев, И. А. Козлов

ПОДХОД К ВЕРИФИКАЦИИ МОДЕЛЕЙ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ С ПОМОЩЬЮ МЕТОДА MODEL CHECKING

Рассмотрена задача построения алгоритма верификации систем реального времени. Предложен подход к проверке таких систем на основе метода Model Checking. Описаны основные шаги верификации, области применения разработанного подхода и приведены примеры проверки различных моделей.

E-mail: arkandreev@gmail.com

Ключевые слова: *система реального времени, проверка модели, временной автомат.*

Одной из важнейших задач, решаемых при создании программного обеспечения (ПО) для ответственных систем, является обеспечение его качества. Основным способом решения этой проблемы является верификация программ [1]. Следует выделить два основных подхода к верификации: динамический и статический.

Динамический подход подразумевает проверку программы в процессе исполнения и используется при тестировании программ. Однако тестирование ПО не может доказать, что система, алгоритм или программа не содержит никаких ошибок и дефектов и удовлетворяют определенному свойству. Это можно сделать с помощью формальной верификации.

Формальная верификация позволяет доказать соответствие программы своей спецификации. При этом под спецификацией программы понимаются формализованные требования к ее поведению. Одним из основных направлений в рамках этого подхода является верификация модели программы (model checking) [2].

Верификация модели программы требует решения ряда задач: необходимо построить по программе ее модель с конечным числом состояний и формально описать требования к программе в терминах одного из видов темпоральных логик [3, 4]. В результате верификации модели либо подтверждается соответствие модели формализованным требованиям, либо строится контрпример. В последнем случае нужно определить причину некорректности: либо ошибка присутствует в исходной программе, либо она появилась в результате некорректного построения модели. Также часто требуется решить задачу переноса контрпримера в исходную программу.

Системы реального времени имеют особенности, которые делают невозможной их проверку с помощью верификаторов общего назначения: при создании таких систем учитываются ограничения на временные характеристики функционирования, а значит, и верификатор

должен позволять проверять соответствие системы этим ограничениям. Для верификации программ реального времени необходим особый подход, включающий в себя подходящие математические модели и алгоритмы их построения и проверки. В качестве таких моделей может выступать взвешенный граф, вершинам которого сопоставляется время выполнения отдельных действий, или временной автомат.

В настоящей статье рассматривается подход к верификации систем реального времени, основанный на пошаговом преобразовании исходной модели программы к виду временно́го автомата и последующей проверке полученной структуры методом Model Checking.

Описание алгоритма верификации систем реального времени. Верификация моделей программ включает в себя несколько основных этапов: преобразование модели в структуру Крипке и построение требований к модели; собственно процесс верификации (отработку алгоритмов на полученных моделях); построение подтверждающих трасс (контрпримеров) в модели Крипке, а также представление контрпримеров, записанных в терминах модели Крипке, в виде путей в исходной модели. Общая схема процесса верификации приведена на рис. 1.

Для описания требований к системам, состояние которых изменяется во времени, используются темпоральные логики. Эти логики представляют собой классическую логику высказываний, расширенную некоторыми темпоральными операторами, которые позволяют описать отношения между моментами времени, в которые наступили те или иные события. Обычно неважно, в какие именно моменты времени в системе происходят те или иные события, имеет значение лишь порядок их наступления. Но отсутствующее количественное представление о времени существенно для спецификации систем, критичных ко времени, а потому для этой цели нельзя использовать основные темпоральные логики — LTL и CTL.

Одним из наиболее важных аспектов является выбор семантики временно́й области. Поскольку в природе время имеет непрерывную структуру, наиболее естественным выбором является непрерывная временна́я область, такая как вещественные числа. Одной из темпоральных логик, которые позволяют осуществлять проверку моделей в терминах непрерывной временной области (R^+ — неотрицательные вещественные числа), является ветвящаяся темпоральная логика реального времени Timed CTL [5]. Базовая идея TCTL состоит в использовании временных ограничений в качестве параметров обычных темпоральных операторов CTL.

Для проверки соответствия модели спецификации, выраженной в виде TCTL-формулы, необходимо построить дерево ее подформул, операторами каждой из которых являются подформулы более низкого уровня (рис. 2).

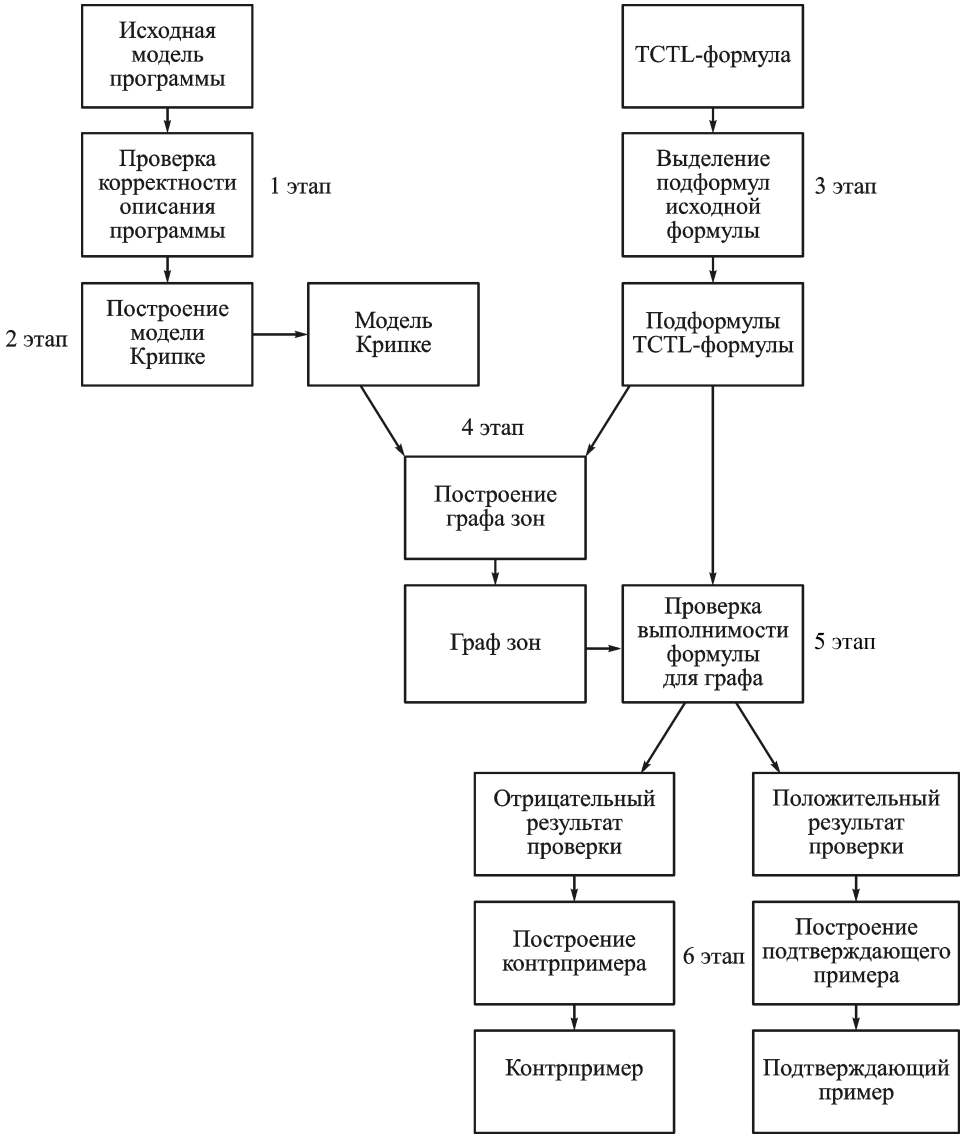

Рис. 1. Схема процесса верификации систем реального времени

Построение верифицируемой модели осуществляется в два этапа: первый шаг аналогичен построению структуры Крипке для верификаторов общего назначения; однако, учитывая специфику систем реального времени, необходимо использовать более сложную модель. В качестве спецификационного формализма для систем реального времени вводится понятие временно́го автомата — расширения конечного автомата, в котором для измерения времени используются часы (рис. 3). Состояние временно́го автомата представляет собой пару $\langle s, v \rangle$, где s — локация (соответствующая позиции в невременно́м конечном автомате), а v — совокупность значений часов. Такой выбор модели позволяет перенести на нее инварианты состояний и ограничения переходов

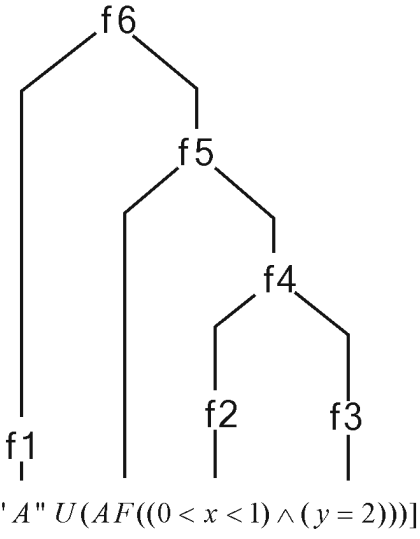

Рис. 2. Дерево подформул TCTL-формулы

исходной системы. Также на этом этапе решается еще одна задача: в случае использования в программе нескольких параллельно действующих автоматов, необходимо построить их параллельную композицию.

Затем осуществляется дальнейшее преобразование модели: по временно́му автомату строится граф временных зон. Каждой вершине графа соответствует одна из локаций автомата, построенного на предыдущем этапе, а также временна́я зона — область временно́го пространства, все элементы которой эквивалентны относительно условий некоторой конкретной задачи. Зоны выбираются так, чтобы интерпретации часов удовлетворяли заданным временным ограничениям (рис. 4).

В результате преобразования получим граф временны́х зон, проверка которого на соответствие TCTL-формуле может быть проведена посредством того же алгоритма, который используется для модели Крипке и нетаймированной CTL-логики. Его базовая идея состоит в пометке каждого состояния подформулами, которые верны в этом состоянии. Эта процедура выполняется итеративно и завершается пометкой состояний, в которых выполняется сама формула φ. Считается, что формула выполняется для данной модели, если в число помеченных состояний входит начальное.

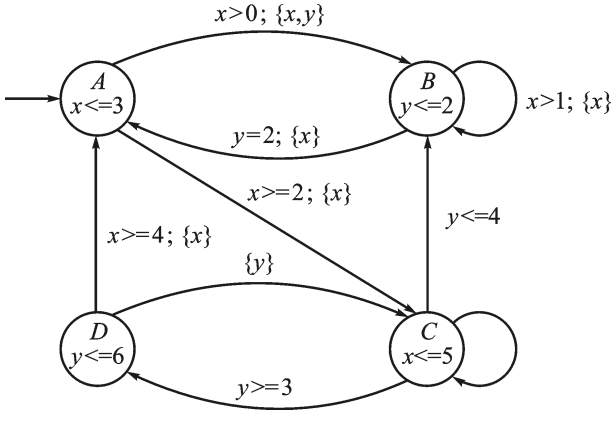

Рис. 3. Временной автомат

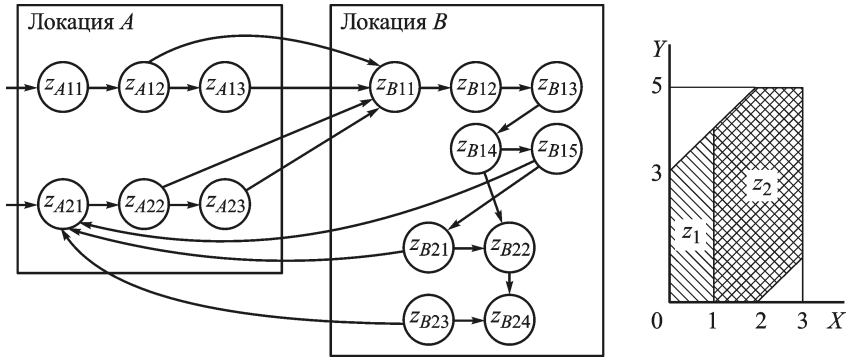

Рис. 4. Граф временных зон (слева); зоны Z_1 и Z_2 во временно́м пространстве с двумя часами (справа)

Наконец, после завершения процедуры верификации строится пример, подтверждающий правильность проверки системы. После этого подтверждающая трасса (контрпример) представляется в терминах исходной программы.

Применение системы верификации. *Автоматные программы.* В основе разработанной системы лежит метод верификации моделей программ. Поэтому она наилучшим образом подходит для верификации программ, построение моделей которых просто автоматизировать, обеспечивая при этом максимальную адекватность построенной модели оригиналу. Этим требованиям отвечают программы, созданные с использованием автоматного подхода [6, 7]. Такие программы можно верифицировать с помощью проверки моделей проще и эффективнее, чем программы других классов. Поэтому при разработке системы верификации была предусмотрена возможность проверки таких программ — при описании входной модели можно использовать все компоненты, используемые при автоматном подходе: события (инициирующие переходы между состояниями), входные воздействия (условия переходов), выходные воздействия (действия, выполняемые при попадании в определенное состояние или при успешном переходе по определенному ребру), условия на состояния внешних автоматов.

Временны́е автоматы. Разработанная система верификации изначально рассматривалась как средство проверки автоматных программ. Однако при проектировании подобной системы конечной целью должна быть возможность проверки программ общего вида. Выясним, осуществима ли такая проверка в созданной системе.

Очевидно, что она не предназначена для верификации программ, написанных с помощью традиционных языков программирования, в исходном виде необходимо преобразование их к форме, понятной верификатору. Для программ, критичных ко времени, такой формой может являться временно́й автомат — это стандартный формализм для моделирования и анализа асинхронных систем реального времени.

Такая структура состоит из пяти компонентов:

• Множество состояний, включающее в себя начальное состояние. Этот компонент присутствует в разработанной системе.

• Множество локальных часов. Часы также используются в системе — как явно описанные пользователем, так и формульные.

• Множество действий (включая пустое действие). Во временно́м автомате каждому переходу может соответствовать максимум одно действие, которое является условием перехода по ребру. Аналогом в данной системе является событие.

• Инвариант, присутствующий в некоторых локациях.

• Множество переходов. Каждый переход помечен действием, временно́й защитой перехода и сбрасываемыми часами. Использование всех этих пометок возможно в системе.

Таким образом, на вход системы можно подать временно́й автомат. При этом не используются входные и выходные воздействия. Также нужно учесть, что во временно́м автомате переход представляет собой единое целое и разбивать его на отдельные участки не следует. Поэтому задержку в состоянии, соответствующему любому событию, нужно сделать нулевой.

При рассмотрении возможности верификации с помощью разработанной системы программ общего вида (представленных в виде временно́го автомата) возникает вопрос: к каким классам систем возможно применение данного верификатора?

Другие классы систем. Итак, данная система может использоваться для верификации временны́х автоматов, с помощью которых могут быть представлены программы общего вида. Однако существует множество видов автоматов, которые используются в различных классах программных систем. Рассмотрим некоторые из этих классов.

Реактивные системы. Это основное направление, в котором используются автоматные программы. Поэтому в системах этого класса возможно наиболее полное использование возможностей как самого автоматного программирования, так и разработанного верификатора.

Отдельно рассмотрим одну из традиционных областей применения автоматов — задачу логического управления. Системы этого подкласса имеют ряд особенностей: не задействованы события, используются автоматы Мура, большое значение имеет параллелизм, широко используется взаимодействие обменом состояний. Автоматы, применяемые в таких системах, могут быть проверены с помощью разработанной системы. При этом не используются события и выходные воздействия на переходах.

Трансформирующие системы, например компиляторы, — это традиционная сфера применения конечных автоматов.

• Конечный автомат — преобразователь должен содержать множество состояний, входной и выходной алфавиты, а также отношение переходов и выходов. В качестве входного алфавита используются события. Роль выходных символов играют выходные воздействия в состояниях (для автомата Мура) или на переходах (автомат Мили). Входные воздействия не используются.

• Конечный автомат — распознаватель, в отличие от преобразователя, включает в себя множество допускающих состояний, но не имеет выходного алфавита. Для верификации автоматов этого класса совсем не используются выходные воздействия.

На основе рассмотренных особенностей использования верификатора для разных классов систем можно классифицировать системы следующим образом

Классификация систем

Признак классификации	X	E	Z	Zs	Часы	||	Y
Класс автомата:							
Мура		gray	red	green			
Мили	gray	gray	green	red	gray	gray	gray
смешанный	gray	gray			gray	gray	gray
без выходного преобразователя	gray	gray	red	red	gray	gray	gray
Критичность ко времени:							
система реального времени	gray	gray	gray	gray	green	gray	gray
система, некритичная ко времени	gray	gray	gray	gray	red	gray	gray
Метод взаимодействия:							
обмен сообщениями	gray	gray	gray	gray	gray	green	red
взаимодействие по номерам состояний	gray	gray	gray	gray	gray	red	green
смешанное	gray	gray	gray	gray	gray	green	green
Использование событий:							
да	gray	green	gray	gray	gray	gray	gray
нет	gray	red	gray	gray	gray	gray	gray
Использование входных воздействий:							
да	green	gray	gray	gray	gray	gray	gray
нет	red	gray	gray	gray	gray	gray	gray

Примечание. X — входные воздействия; E — события; Z — выходные воздействия на переходах; Zs — выходные воздействия в состояниях; || — параллелизм; Y — условие на номер состояния другого автомата.

Зеленый цвет (green) ячейки означает, что соответствующий компонент автоматной программы используется в данной системе, крас-

ный (red) — что не используется, серый (gray) означает, что наличие/отсутствие этого компонента для рассматриваемого признака не принципиально.

На основе полученной классификации можно сделать вывод, что разработанная система отличается высокой универсальностью, выражающейся в способности верифицировать не только временные автоматы, но и множество других моделей, применяемых в различных классах программных систем.

Пример верификации систем различных классов. *Автоматная программа*. Верификация автоматной программы общего вида будет демонстрироваться на примере автомата, управляющего дверьми лифта (рис. 5).

Для этого автомата проверяются следующие утверждения.

При отсутствии ошибки состояние Opened посещается бесконечное число раз. Результат верификации данного утверждения — "Выполняется".

$\Phi_1 = \mathrm{AG}(\mathrm{AF}(\mathrm{Error} \vee \mathrm{Opened}))$.

При отсутствии ошибки состояние Closed посещается бесконечное число раз. Результат верификации — "Нарушается".

$\Phi_2 = \mathrm{AG}(\mathrm{AF}(\mathrm{Error} \vee \mathrm{Closed}))$.

Рис. 5. Схема связей (*а*) и граф переходов (*б*) автомата AElevator

Приведем результат проверки, осуществляемой разработанной системой.

[AG(AF(Error|Opened))]

Subformulas:

#sub1= Error
#sub2= Opened
#sub3= #sub1|#sub2
#sub4= AF #sub3
#sub5= / #sub4
#sub6= E(#sub0 U #sub5)
#sub7= / #sub6

	#sub1	#sub2	#sub3	#sub4	#sub5	#sub6	#sub7
State 36:	0	0	0	1	0	0	1

Result of verification is positive for the initial state of the graph (state 36)

[AG(AF(Error|Closed))]

Subformulas:

#sub1= Error
#sub2= Closed
#sub3= #sub1|#sub2
#sub4= AF #sub3
#sub5= / #sub4
#sub6= E(#sub0 U #sub5)
#sub7= / #sub6

	#sub1	#sub2	#sub3	#sub4	#sub5	#sub6	#sub7
State 36:	0	1	1	1	0	1	0

Result of verification is negative for the initial state of the graph (state 36)

Результат проверки, выполненной с помощью системы верификации, совпадает с полученным теоретически.

Временно́й автомат. Для проверки работы системы с временны́ми автоматами используем пример, приведенный в работе [8].

Для временно́го автомата, приведенного на рис. 6, проверяется формула $\Phi = AG_{<=1}n$. В работе [8] доказывается, что данная формула не выполняется для этого автомата. Приведем результат проверки, осуществляемой разработанной системой.

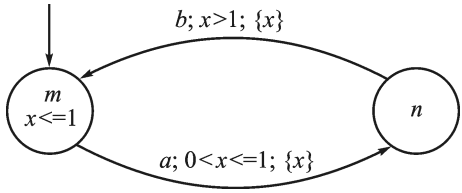

Рис. 6. Модель временно́го автомата

AG(<=1)(n)]

Subformulas:

#sub1= n
#sub2= / #sub1
#sub3= #sub0 | #sub2
#sub4= #z<=1
#sub5= #sub2 & #sub4
#sub6= E(#sub3 U #sub5)
#sub7= / #sub6

	#sub1	#sub2	#sub3	#sub4	#sub5	#sub6	#sub7
State 1:	0	1	1	0	0	0	1
State 2:	0	1	1	1	1	1	0
State 3:	1	0	1	0	0	0	1
State 4:	1	0	1	1	0	0	1
State 5:	1	0	1	0	0	0	1
State 6:	0	1	1	0	0	0	1
State 7:	0	1	1	0	0	0	1
State 8:	0	1	1	0	0	0	1
State 9:	0	1	1	1	1	1	0
State 10:	0	1	1	1	1	1	0

Result of verification is negative for the initial state of the graph (state 10)

Результат проверки, выполненной с помощью системы верификации, совпадает с полученным теоретически.

Невременной конечный автомат-преобразователь. В качестве автомата-преобразователя возьмем модель поведения отца из известного примера:

Опишем поведение отца, сын которого учится в школе и приносит двойки и пятерки. Отец не хочет хвататься за ремень каждый раз, как только сын получит двойку, и выбирает более тонкую тактику воспитания. Граф поведения представлен на рис. 7. Двумя возможными событиями являются принесенная сыном “2” или “5”.

Проверим для этого автомата следующее утверждение: “Даже если сейчас отец наказывает сына, возможно, что в будущем он будет

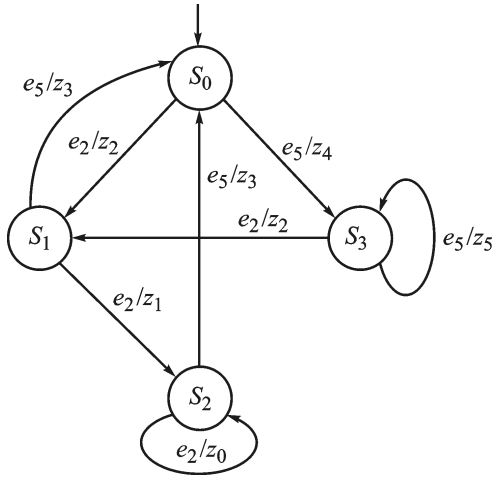

Рис. 7. Модель автомата-преобразователя:
z_0 — брать ремень; z_1 — ругать сына; z_2 — успокаивать сына; z_3 — надеяться; z_4 — радоваться; z_5 — ликовать

ликовать”. Это утверждение можно выразить в виде CTL-формулы: AG(z0 → EF(z5)). После анализа структуры автомата становится ясно, что эта формула для него выполняется. Приведем результат проверки, осуществляемой разработанной системой.

```
[AG(/(z0)|EF(z5))]

Subformulas:

#sub1= z0
#sub2= / #sub1
#sub3= z5
#sub4= E(#sub0 U #sub3)
#sub5= #sub2 | #sub4
#sub6= / #sub5
#sub7= E(#sub0 U #sub6)
#sub8= / #sub7

          #sub1 #sub2 #sub3 #sub4 #sub5 #sub6 #sub7 #sub8
State 34:   0      1      0      1      1      0      0      1

Result of verification is positive for the initial state of the graph (state 34)
```

Результат проверки с помощью системы верификации совпадает с полученным теоретически.

Система логического управления. Верификация системы логического управления будет демонстрироваться на примере системы управления шлагбаумом на железнодорожном переезде (рис. 8–10).

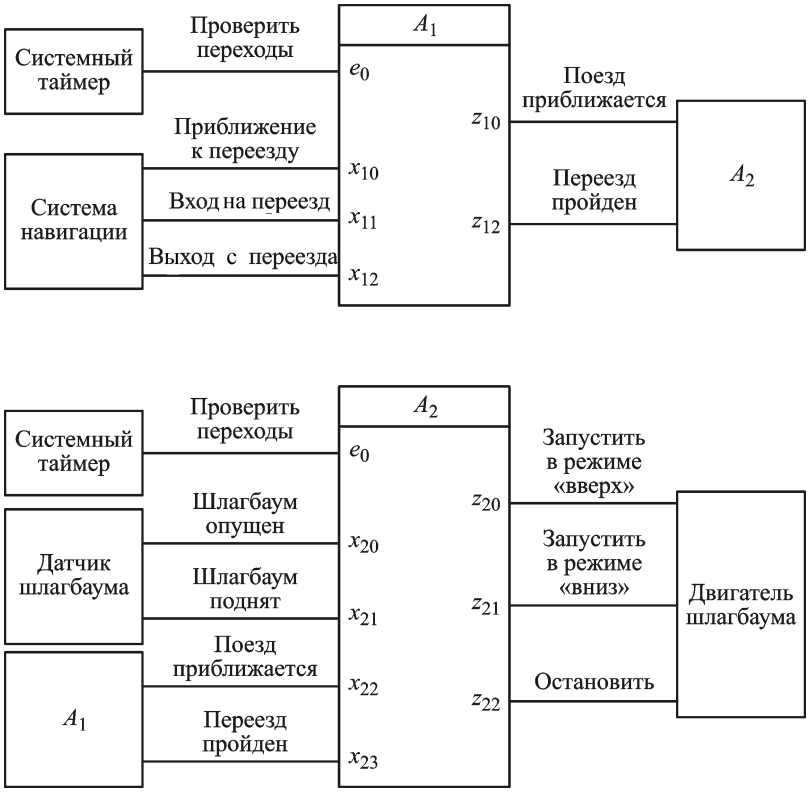

Рис. 8. Схема связей автоматов A1 и A2

Проверим для этого автомата утверждение: “При попадании в состояние s_{12} показания часов c_3 должны превышать 60”. Такое условие должно гарантировать заданный промежуток времени между движением поездов.

Система верификации выдает следующий результат проверки:

```
[AG(/(s12)|(c3>=10))]

Subformulas:
#sub1= s12
#sub2= / #sub1
#sub3= c3>=10
#sub4= #sub2 | #sub3
#sub5= / #sub4
#sub6= E(#sub0 U #sub5)
#sub7= / #sub6

          #sub1 #sub2 #sub3 #sub4 #sub5 #sub6 #sub7
State 58:  0     1     0     1     0     1     0

Result of verification is negative for the initial state of the graph (state 58)
```

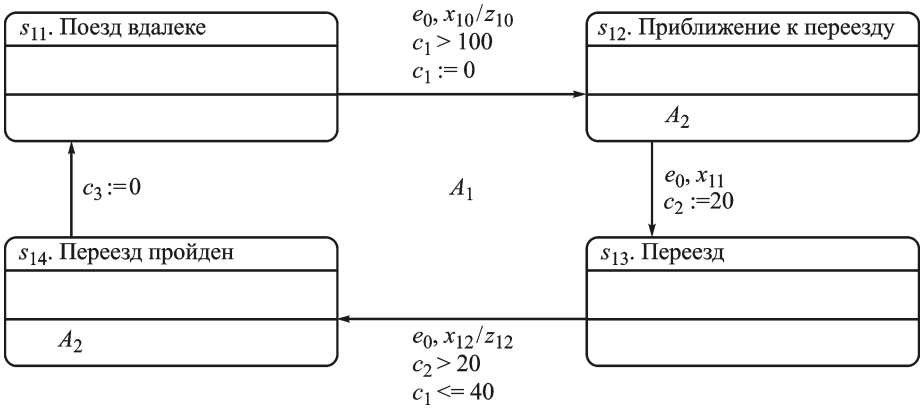

Рис. 9. Граф переходов автомата A1

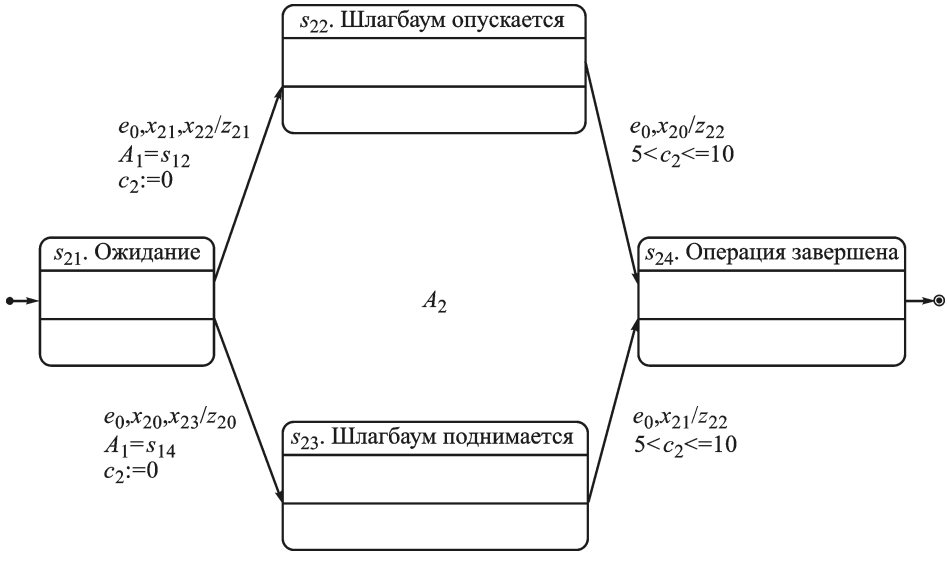

Рис. 10. Граф переходов автомата A2

При анализе результатов верификации становится понятна причина такого результата: при выборе спецификации не было учтено, что автомат A2 может находиться в состоянии s_{21} сколь угодно долго.

Заключение. Предложен подход к верификации систем реального времени на основе метода Model Checking. Данный подход может применяться для выявления ошибок в программах на этапе детализированного проектирования систем реального времени. В соответствии с описанным подходом была разработана система верификации, отличающаяся от существующих средств своей универсальностью. С ее помощью можно проверять временны́е автоматы (модели, используемые для описания поведения асинхронных систем реального времени), автоматные программы, а также множество других моделей, используемых в различных классах программных систем.

К недостаткам подхода можно отнести сложность его применения для больших программных систем. Даже верификаторы общего назначения зачастую плохо справляются с проблемой комбинаторного взрыва в пространстве состояний, возникающего с ростом размера входной модели. Для систем реального времени эта проблема стоит особенно остро, ведь помимо выделения локаций, выполняемого в любом верификаторе, нужно провести разбиение этих локаций на отдельные временны́е зоны. Этот этап усугубляет проблему роста числа состояний, а также значительно увеличивает время выполнения верификации. Описанный в статье подход позволяет отчасти решить первую из этих проблем: итоговое число состояний значительно меньше, чем может быть получено другими методами (такими, как построение графа регионов).

СПИСОК ЛИТЕРАТУРЫ

1. Кузьмин Е. В., Соколов В. А. О дисциплине специализации “Верификация программ” // Докл. II науч.-методич. конф. “Преподавание математики в компьютерных науках”, Ярославль: ЯрГУ. 2007.
2. Кларк Э. М., Грамберг О., Пелед Д. Верификация моделей программ: Model checking. – М.: МНЦМО, 2002.
3. Pnueli A. The temporal logic of programs // Proceedings of the 18th IEEE Symposium on Foundation of Computer Science. 1977.
4. Emerson E. A. Temporal and modal logic // Handbook of Theoretical Computer Science. Chapter 16. 1990.
5. Вельдер С. Э., Лукин М. А., Шалыто А. А., Яминов Б. Р. Верификация автоматных программ. – СПб.: ГУ ИТМО, 2011. – 240 с.
6. Шалыто А. А. Программная реализация управляющих автоматов // Судостроительная промышленность. Сер. Автоматика и телемеханика. – 1991. – Вып. 13.
7. Шалыто А. А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. – СПб.: Наука. 1998.
8. Карпов Ю. Г. MODEL CHECKING. Верификация параллельных и распределенных программных систем. – СПб.: БХВ-Петербург, 2010.

Статья поступила в редакцию 15.12.2011