Сборник докладов семинара Go4IT – шаг к новым технологиям Интернета. Москва, Институт системного программирования, 2007. С. 43-48.

Кузьмин Е.В. [1]***, Соколов В.А.*** [2]

О НЕКОТОРЫХ ПОДХОДАХ К ВЕРИФИКАЦИИ АВТОМАТНЫХ ПРОГРАММ

Ярославский государственный университет им. П.Г. Демидова, г. Ярославль
[1] *egorkuz@mail.ru* , [2] *sokolov@uniyar.ac.ru*

Статья посвящена обзору методов и средств, которые могут быть применимы для практической верификации «автоматных» программ.

В последние годы при построении промышленных систем широко применяется синхронное программирование. Несмотря на популярность и повсеместное использование универсальных языков программирования, таких как C++ и Java, активно проводятся работы по созданию специализированных языков и технологий, предназначенных, главным образом, для реализации программных систем управления ответственными объектами. Ряд языков и технологий, таких как Lustre [8], Esterel [6], SyncCharts, Argos [12], тесно связанных с понятием конечного автомата и оперирующих им в декларативной или графической форме, объединяются под общим названием «синхронное программирование» [21]. Эта технология программирования находит применение и активно внедряется в области точного приборостроения, при производстве авиатехники, в судостроении и т. д.

С 1991 года в России под руководством профессора А.А. Шалыто (заведующего кафедрой технологии программирования СПбГУ ИТМО) для указанного класса систем развивается технология программирования с явным выделением состояний (SWITCH-технология), которая в качестве языка спецификации использует графы переходов (взаимодействующие событийные автоматы Мили–Мура). SWITCH-технология, или «автоматное» программирование, не зависит от платформы, операционной системы или языка программирования. При этом в некоторых случаях «автоматное» программирование можно рассматривать как разновидность «синхронного». Технология автоматного программирования является достаточно эффективной при построении программного обеспечения для «реактивных» систем и систем логического управления. Кроме того, она может быть применима для программирования коммуникационных протоколов разных уровней, поскольку структура большинства протоколов в основе своей является автоматной. Эта технология, не исключая других методов построения программного обеспечения «без ошибок», существенно более конструктивна, так как позволяет начинать «борьбу с ошибками» еще на стадии алгоритмизации [18,19,20].

В данной статье «автоматный» подход к программированию (как и само синхронное программирование) оценивается как наиболее подходящее с точки зрения анализа программной корректности. Представляется весьма интересным направление моделирования, спецификации и верификации «автоматных» программ в рамках общей тематики «Моделирование и анализ информационных систем». Несмотря на то, что сама идея синхронного и «автоматного» программирования направлена на построение надёжных программ, задача проверки их правильности попрежнему остаётся актуальной. Причём, если в промышленных языках синхронного программирования эта проблема решается в «жёсткой» привязке к компилятору, то в «автоматном» программировании, как языково-неориентированной технологии, в большей степени вопрос остаётся открытым.

При автоматном подходе к проектированию и построению программ выделяются две части: системно независимая и системно зависимая. Первая часть реализует логику программы и задаётся системой взаимодействующих автоматов Мура–Мили. Проектирование каждого автомата состоит в создании по словесному описанию (декларации о намерениях) схемы связей, описывающей его интерфейс, и графа переходов, определяющего его поведение. По этим двум документам формально и изоморфно может быть построен модуль программы (и затем реализована системно зависимая часть), соответствующий автомату.

Интересным является тот факт, что к «автоматным» программам могут быть успешно применимы (в совокупности) все существующие методы анализа корректности: тестирование, метод доказательства теорем и метод проверки модели. *Тестирование* широко применяется после окончательного написания программы. Но, как известно (Э. Дейкстра), тестирование вовсе не гарантирует отсутствие ошибок в программе. Если при тестировании ошибки найдены не были, это ещё не означает, что их нет. *Метод доказательства теорем* [16] представляется очень трудоёмким методом с сильной привязкой к семантике языка программирования. Но при желании, вполне может быть применим для «автоматных» программ после их полного построения, непосредственно для проверки корректности процедур, соответствующих выходным воздействиям или входным запросам. Каждое выходное воздействие выполняет свою обычно небольшую отдельную задачу, корректность реализации которой может быть проверена. Таким образом, можно говорить о том, что автоматная структура программы благоприятствует применению метода доказательства теорем. Но этот метод оказывается бесполезным при проверке логики программы. С другой стороны для проверки логики «автоматных» программ идеально подходит *метод проверки модели* (model checking) [17]. При этом методе для программы строится формальная конечная модель (т.е. с конечным числом состояний), а

проверяемые свойства задаются с помощью формул темпоральной логики. Проверка выполнимости темпоральных формул, задающих свойства модели, происходит автоматическим образом.

Автоматный подход к программированию с точки зрения моделирования и анализа программных систем имеет ряд преимуществ по сравнению с традиционным подходом. При построении модели для программы, написанной традиционным способом, возникает серьёзная проблема адекватности этой программной модели исходной программе. Модель может не учитывать ряд программных свойств или порождать несуществующие свойства. При автоматном программировании такая проблема исключена, поскольку набор взаимодействующих автоматов, описывающий логику программы, уже является адекватной моделью, по которой формально и изоморфно строится программный модуль. И это является бесспорным плюсом автоматной технологии. Более того, модель имеет конечное число состояний, что является необходимым на практике условием для успешной автоматической верификации, поскольку model checking представляет собой переборный метод. К тому же свойства программной системы в виде автоматов формулируются и специфицируются естественным и понятным образом, легко соотносятся со взаимодействующими автоматами, которые задают логику «автоматной» программы. Если в качестве примеров рассмотреть системы управления кофеваркой, банкоматом или лифтом, можно привести такие свойства, как «кофеварка не может варить кофе без воды», «нагревательный элемент кофеварки не должен перегреваться», «кабина лифта не должна двигаться при открытых дверях» или «при нормальной работе банкомат вернёт пользователю карту, если она была вставлена». Все эти свойства связаны с автоматами управления и легко задаются с помощью темпоральных логик (таких, как CTL и LTL), так как элементами управляющих автоматов являются либо чётко выраженные состояния объекта управления, либо понятные действия над ним. Проверка свойств осуществляется в терминах, которые естественно вытекают из автоматной модели программы. Элементарные высказывания в рамках свойств определяются над элементами модели – событиями, входными и выходными воздействиями и состояниями.

Одними из наиболее популярных темпоральных логик для спецификации и верификации свойств программных систем являются логика CTL (branching-time logic или computation tree logic) и логика линейного времени LTL (linear-time logic). Для целей верификации автоматных программ логика LTL заслуживает особого внимания, поскольку любая формула в рамках этой логики, по сути, представляет собой автомат Бюхи, описывающий (принимающий) бесконечные допустимые пути структуры Крипке, которая в свою очередь задаёт поведение (все возможные исполнения) проверяемой на корректность «автоматной» моде-

ли. Что позволяет при спецификации и верификации «автоматных» программ оперировать в основном таким простым понятием, как «автомат».

Таким образом, перечисленное выше позволяет говорить об эффективном применении для верификации «автоматных» программ (для анализа корректности логики «автоматных» программ) метод проверки модели. Для этого целесообразно использовать уже существующие пакеты прикладных программ-верификаторов, которые разрабатываются и поддерживаются ведущими научными лабораториями и центрами на протяжении довольно длительного времени (более десяти лет). Среди программных средств верификации классическим методом проверки модели можно выделить, например, SPIN[14] и SMV[13].

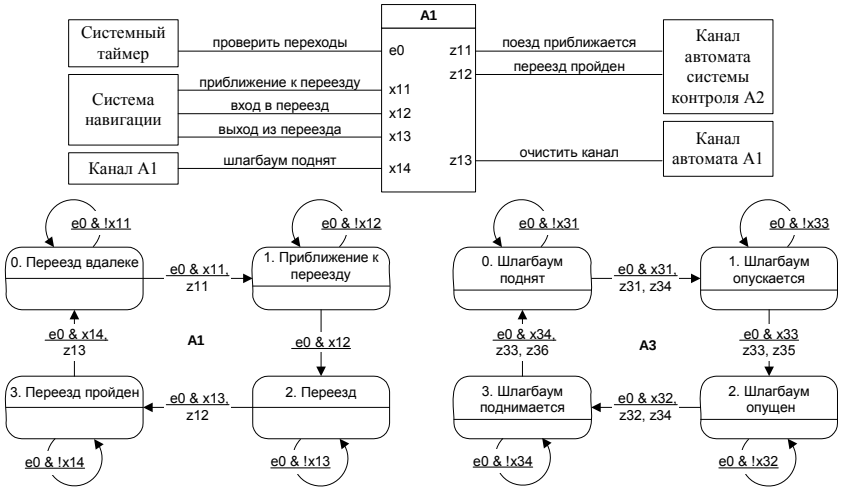
Однако при построении реактивных систем реального времени с использованием автоматного подхода к программированию классического метода проверки модели недостаточно для анализа сложных временны*х* темпоральных свойств (свойств с ограничениями по времени). В этом случае необходимо применять существующие методы и средства верификации систем реального времени, представляющие собой развитие метода model checking. Основной акцент при построении и моделировании временны*х* систем ставится на интерпретации времени. При автоматном программировании синхронных систем реального времени целесообразно строить и верифицировать модели с дискретным временем. Спецификация в этом случае осуществляется на языке темпоральной логики реального времени. Расширенная с учётом дискретного реального времени темпоральная логика позволяет выражать такие свойства, как «всегда верно, что за p последует q не позднее чем через 3 единицы времени». Примером такой темпоральной логики является логика RTCTL (real-time CTL), которая используется для спецификации свойств в верификаторе VERUS [7]. Непрерывное время, с другой стороны, является естественной моделью для асинхронных систем, поскольку промежуток времени, разделяющий события, может быть сколь угодно мал. Стандартным формализмом для моделирования и анализа асинхронных систем реального времени являются временны*е* автоматы [1, 2]. Рассмотрим пример того, каким образом временны*е* автоматы могут быть использованы для верификации «автоматных» программ, реализующих реактивные системы реального времени.

Рассмотрим систему управления железнодорожным переездом. Пример системы был взят из [2]. Система состоит из трех работающих асинхронно компонентов, взаимодействующих между собой посредством передачи сообщений через каналы связи. Компоненты условно назовём следующим образом: «Поезд», «Шлагбаум» и «Контроллер». При приближении к переезду «Поезд» сообщает об этом «Контроллеру», который в свою очередь должен опустить шлагбаум до того, как поезд войдёт в переезд, и поднять его после выхода поезда из переезда. В сти-

ле автоматного программирования эта задача может быть решена, как показано на рис. 1. Пример очень упрощён, но достаточен для демонстрации идей автоматного программирования. На рисунке отражена лишь логика работы каждого из компонентов и оставлены без программного описания реализации событий (**e**), входных запросов (**x**), выходных воздействий (**z**) и т.д. Правила переходов из одного состояния в другое для каждого автомата, представленного на рисунке, следующие. После наступления события **e**, которое может быть обработано в текущем состоянии, автомат опрашивает параметры объекта управления (входные запросы **x** или в отрицательной форме **!x**), затем (в зависимости от результата опроса параметров) производит последовательность выходных воздействий **z** и после этого переходит в новое состояние (которое в случае петли может быть тем же самым).

Исходя из логики взаимодействия компонентов программная система, представленная на рис. 1, может быть смоделирована с помощью параллельной композиции конечных автоматов, синхронизирующихся по одинаковым меткам переходов. Эта модель изображена на рис. 2. В модели переходы двух автоматов, помеченные *a*! и *a*?, срабатывают одновременно, т.е. происходит синхронизация автоматов по метке *a*. Для построенной модели может быть применим классический model checking при проверке свойств, которые непосредственно касаются требований к системе. Например, с успехом может быть проверено следующее истинное для системы свойство: если поезд далеко от переезда (т.е. автомат *A*1 находится в состоянии «0. Переезд вдалеке»), то переезд открыт (т.е. автомат *A*3 находится в состоянии «0. Шлагбаум поднят»). Также может быть проверено свойство: шлагбаум должен опуститься до того (автомат *A*3 должен находится в состоянии «2. Шлагбаум опущен»), как поезд войдет в переезд (автомат *A*1 окажется в состоянии «2. Переезд»). В отличие от предыдущего это свойство не выполняется для модели, представленной на рис. 2. Но это совсем не значит, что автоматная программа на рис. 1 реализована ошибочно. Дело в том, что с одной стороны, параллельная композиция конечных (синхронизирующихся) автоматов является слишком грубой моделью, а с другой стороны, программная автоматная система на рис. 1 не предоставляет достаточно полной информации для моделирования. Дополним автоматную систему описанием, а точнее, деталями, с учётом которых она создавалась. При проектировании предполагалось, что система навигации сообщит поезду о его приближении к переезду не менее чем за 2 мин. до входа поездом в переезд. Более того, известно, что поезду требуется не более 5 мин. для полного прохождения переезда после того, как поступил сигнал о приближении. Также известно, что время опускания

Автомат навигации поезда



Автомат управления шлагбаумом

Автомат системы контроля

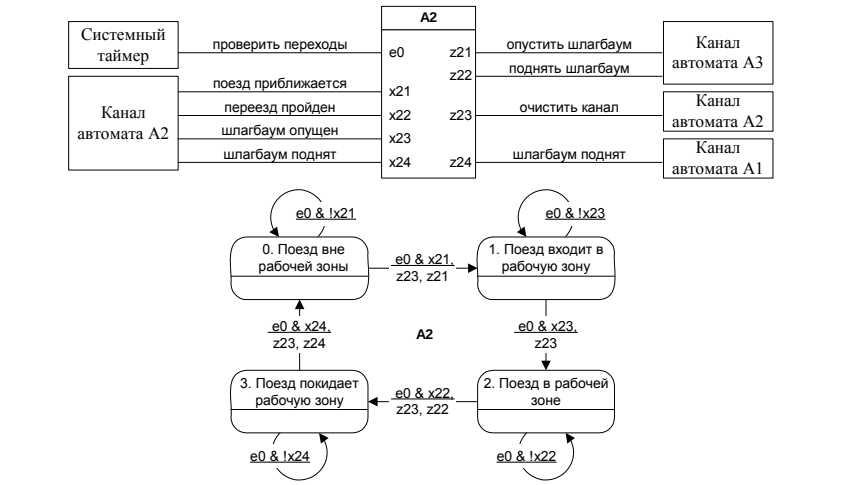


Рис. 1. Схемы связей и графы переходов автоматов, реализующих компоненты «Поезд», «Шлагбаум» и «Контроллер»

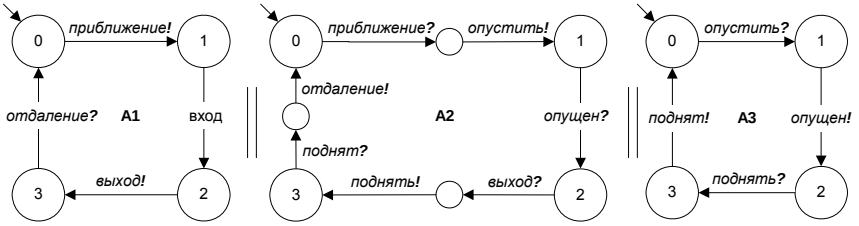


Рис. 2. Модель автоматной системы в виде параллельной композиции синхронизирующихся конечных автоматов

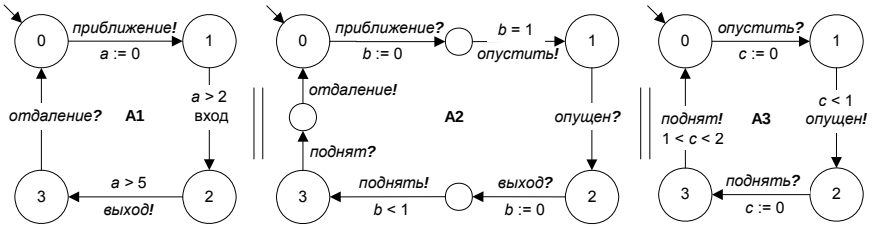


Рис. 3. Модель автоматной системы в виде параллельной композиции синхронизирующихся временны*х* автоматов

шлагбаума составляет не более 1 мин., а поднятия – от 1 до 2 мин. Кроме того, для «Контроллера» время отклика на сигнал о приближении поезда равно 1 мин., а на сигнал о выходе поезда из переезда – не более 1 мин. Эти условия вполне могут быть учтены при моделировании системы в виде параллельной композиции временны*х* автоматов так, как это показано на рис. 3.

На рис. 3 каждый автомат снабжается своими часами и переменной, с помощью которой отслеживается время или же сбрасывается до нуля конструкцией вида $a := 0$. Временной автомат может совершать переходы по состояниям лишь в том случае, когда выполняются условия на дугах с участием этой переменной (если, конечно, такое условие есть). Переходы совершаются мгновенно, однако течение времени не останавливается, пока автомат находится в своём состоянии.

Для автоматной программы с помощью модели, представленной на рис. 3, могут быть заданы и проверены более сложные свойства, связанные с реальным временем. Например, является истинным следующее свойство: шлагбаум не может быть опущен (переезд не может быть закрыт) более чем на 10 мин., т.е. состояние автомата *A*3 «0. шлагбаум поднят» достигается как максимум через 10 мин. после

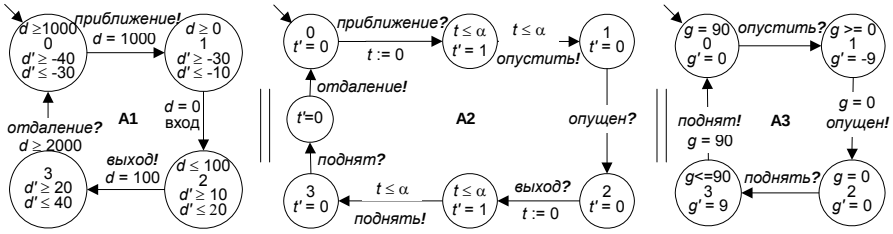


Рис. 4. Модель автоматной системы в виде параллельной композиции синхронизирующихся линейных гибридных автоматов

попадания им в состояние «2. Шлагбаум опущен». Более того, в случае модели с временными автоматами становится истинным свойство, которое не выполнялось для модели, состоящей из композиции не временных автоматов: когда поезд войдёт в переезд, шлагбаум уже должен быть опущен.

Для верификации автоматных программ с использованием моделей на основе временных автоматов могут быть, например, применимы такие программные средства, как UPPAAL [15] и KRONOS [11].

И наконец, для моделирования автоматных программ также могут быть использованы линейные гибридные автоматы [3, 4] (обобщение временных автоматов). Рассмотрим автоматную систему на рис. 1 с другим описанием деталей реализации. Предполагается, что система навигации сообщает поезду о приближении за 1000 м. до переезда. Кроме того, считается, что поезд покинул переезд и можно поднимать шлагбаум, если поездом пройдено 100 м. после переезда. В зависимости от расстояния от переезда поезд меняет свою скорость, например, при приближении к переезду его скорость должна лежать в диапазоне от 10 до 30 м/с. Шлагбаум опускается и поднимается со скоростью 9° в сек. Время отклика «Контроллера» на любой сигнал составляет не более α секунд. Модель в виде параллельной композиции линейных гибридных автоматов для рассматриваемой автоматной системы представлена на рис. 4; здесь автоматы в своих состояниях имеют не только условия (неравенства с участием некоторых переменных величин), при которых они остаются в этих состояниях, но и скорости изменения значений переменных в единицу времени при том, что время в состояниях по прежнему протекает непрерывно. В рамках этой модели могут быть выражены свойства достижимости состояний или свойства безопасности (недостижимости «плохих» состояний), а также может быть осуществлён параметрический анализ, когда, например, требуется узнать максимально допустимое время отклика «Контроллера» на поступающие команды, при котором шлагбаум будет опущен, если поезд подойдет к переезду на расстояние 10 метров. Необходимо отметить, что проблемы дости-

жимости и поиска параметров являются в общем случае неразрешимыми, так как, например, к ним могут быть легко сведены неразрешимые (но частично разрешимые) проблемы достижимости произвольной конфигурации и ограниченности трехсчетчиковой машины Минского соответственно. Для решения указанных задач используются лишь частичные алгоритмы. Однако, как указывается в руководстве к системе верификации линейных гибридных автоматов HyTech [10], на практике в большинстве случаев при проверке реальных примеров частичные алгоритмы анализа, применяемые в HyTech, останавливались. Более того, в работе [9] описан широкий класс линейных гибридных систем, для которых итерационные методы и процедуры анализа разрешимы (всегда сходятся). Таким образом, по всей видимости, линейные гибридные автоматы представляют собой граничный «сверху» по выразительности формализм, который ещё может быть применим для моделирования и анализа автоматных программ.

Вывод

Из всего указанного выше можно заключить, что «автоматная» программа является исключительно удобным объектом для верификации. В частности, если после верификации методом проверки модели тестирование выявит ошибку, то вид этой ошибки скорее всего будет относиться к некорректной программной реализации выходных воздействий, а не к нарушению логики программы, что при исправлении не потребует глобальной перестройки «автоматной» программы (всё сведётся к локальным исправлениям внутри одной или нескольких отдельных процедур).

Список литературы

1. *Alur R.*, *Dill D.L.* A theory of timed automata // Theoretical Computer Science 126, 1994. P. 183–235.
2. *Alur R.*, *Dill D.L.* Automata-theoretic verification of real-time systems // In Formal Methods for Real-Time Computing, Trends in Software Series, John Wiley & Sons Publishers, 1996. P. 55–82.
3. *Alur R.*, *Coucoubetis C.*, *Henzinger T.A.*, *Ho P.-H.*, *Nicollin X.*, *Olivero A.*, *Sifakis J.*, *Yovine S.* The algorithmic analysis of hybrid systems // Theoretical Computer Science 138, 1995. P. 3–34.
4. *Alur R.*, *Coucoubetis C.*, *Henzinger T.A.*, *Ho P.-H.* Hybrid Automata: An algorithmic approach to the specification and verification of hybrid systems. // In Hybrid Systems, LNCS 736, 1993. P 209–229.
5. *Andre C.* Representation and Analysis of Reactive Behaviors: A Synchronous Approach // CESA'96, Lille, France, IEEE-SMC, 1996.
6. *Berry G.*, *Gonthier G.* The Esterel synchronous programming language: Design, semantics, implementation // Sci. Comput. Program., vol. 19,

1992. P. 87-152.

7. *Campos S.*, *Clarke E.*, *Marrero W.*, *Minea M.* Verus: a tool for quantitative analysis of finite-state real-time systems // Workshop on Languages, Compilers and Tools for Real-Time Systems, 1995.
8. *Caspi P.*, *Pilaud D.*, *Halbwachs N.*, *Plaice J. A.* LUSTRE: A declarative language for programming synchronous systems // In ACM Symp. Principles Program. Lang. (POPL), 1987. P. 178–188.
9. *Henzinger T.A.*, *Kopke P.W.*, *Puri A.*, *Varaiya P.* What's decidable about hybrid automata? // Proceedings of the 27th Annual Symposium on Theory of Computing (STOC), ACM Press, 1995. P. 373–382
10. HyTech – http://embedded.eecs.berkeley.edu/research/hytech/
11. KRONOS – http://www-verimag.imag.fr/TEMPORISE/kronos
12. *Maraninchi F.* The Argos language: Graphical representation of automata and description of reactive systems. // Presented at the IEEE Workshop Visual Lang., Kobe, Japan, 1991. 7 p.
13. Symbolic Model Verificator. Carnegie Mellon University. – http://www.cs.cmu.edu/~modelcheck/smv.html
14. SPIN – http://spinroot.com/spin/whatispin.html
15. UPPAAL – http://www.uppaal.com
16. *Грис Д.* Наука программирования /Д. Грис; пер. с англ. – М.: Мир, 1984. – 416 с.
17. *Кларк Э.М.*, *Грамберг О.*, *Пелед Д.* Верификация моделей программ: Model Checking. – М.: МЦНМО, 2002. – 416 с.
18. *Шалыто А.А.* Switch-технология. Алгоритмизация и программирование задач логического управления. – СПб.: Наука, 1998. – 628 с. – http://is.ifmo.ru/books/switch/1/
19. *Шалыто А.А.* Алгоритмизация и программирование для задач логического управления и «реактивных» систем // Автоматика и телемеханика. Обзоры. 2001. №1. С. 3–39. (http://is.ifmo.ru, «Статьи»).
20. *Шалыто А.А.*, *Туккель Н.И.* SWITCH-технология – автоматный подход к созданию программного обеспечения «реактивных» систем // Программирование. 2001. № 5. С. 45–62. – http://is.ifmo.ru/works/switch/1/
21. *Шопырин Д.Г.*, *Шалыто А.А.* Синхронное программирование // Информационно-управляющие системы. 2004. №3.С. 35–42.