

Опубликовано в материалах 2-й межвузовской научной конференции по проблемам информатики СПИСОК-2011, с. 343-350.

К. В. Егоров, Ф. Н. Царев

*Санкт-Петербургский государственный университет
информационных технологий, механики и оптики*

Применение генетического программирования для построения автоматов управления системами со сложным поведением на основе верификации моделей и обучающих примеров

Введение

Автоматное программирование – это парадигма программирования, в рамках которой программы предлагается проектировать в виде совокупности взаимодействующих автоматизированных объектов управления [1]. В автоматных программах выделяют три типа объектов: поставщики событий, система управления и объекты управления. Система управления представляет собой конечный автомат или систему взаимодействующих конечных автоматов. Поставщики событий генерируют события, а система управления по каждому событию может совершать переход, считывая значения входных переменных у объектов управления для проверки условия перехода.

Для многих задач автоматы удается строить эвристически, однако существуют задачи, для которых такое построение затруднительно [2–4]. Одним из авторов настоящей работы был предложен метод построения автоматов с помощью генетического программирования на основе тестов [5]. Однако, как известно, тесты не могут

полностью описывать поведение программы, а их выполнимость не может служить критерием ее корректности.

Целью настоящей работы является расширение возможностей метода построения автоматных программ на основе генетического программирования за счет использования верификации на этапе вычисления функции приспособленности, скрещивания и мутации.

Верификация автоматных программ

Для описания требований к автоматным программам будем применять язык логики линейного времени (*Linear Temporal Logic, LTL*). По любой LTL-формуле можно построить автомат Бюхи. Алгоритм верификации основан на проверке пустоты языка пересечения допускаемого конечным автоматом модели и отрицанием LTL-формулы [7, 8].

Верификатор получает на вход модель автоматной программы и LTL-формулу [9]. После проверки модели верификатор либо сообщает, что формула выполняется, либо приводит контрпример – путь в модели, опровергающий утверждение [10].

Построение управляющих конечных автоматов на основе обучающих примеров с помощью генетического программирования

При использовании метода построения управляющих конечных автоматов на основе обучающих примеров (тестов) каждый тест представляет собой последовательность входных воздействий (входная последовательность, соответствующая i -ому тесту, будет обозначаться как $\text{Input}[i]$) и соответствующую ей последовательность выходных воздействий (в дальнейшем будет обозначаться, как $\text{Answer}[i]$). Под входными

воздействиями понимаются события от поставщиков событий и условия переходов, а под выходными – вызываемые действия объектов управления.

Функция приспособленности основана на редакционном расстоянии [6]. Для ее вычисления выполняются следующие действия: на вход автомату подается каждая из последовательностей $Input[i]$. Обозначим последовательность выходных воздействий, которую сгенерировал автомат на входе $Input[i]$ как $Output[i]$. После этого вычисляется величина FF_1 :

$$FF_1 = \frac{\sum_{i=1}^n \left(1 - \frac{ED(Output[i], Answer[i])}{\max(|Output[i]|, |Answer[i]|)}\right)}{n}$$

Здесь как $ED(A, B)$ обозначено редакционное расстояние между строками A и B , как $Answer[i]$ обозначена эталонная выходная последовательность, которую должен генерировать автомат на входе $Input[i]$. Отметим, что значения этой функции лежат в пределах от 0 до 1, при этом, чем «лучше» автомат соответствует тестам, тем больше значение функции приспособленности.

Итоговое значение функции приспособленности зависит не только от того, насколько «хорошо» автомат работает на тестах, но и от числа переходов, которые он содержит. Она вычисляется по формуле

$$FF_2 = FF_1 + \frac{1}{M} \cdot (M - cnt)$$

Здесь cnt – число переходов в рассматриваемом конечном автомате, а M – некоторое число, большее максимально возможного числа переходов в автомате с заданным числом состояний.

Эта функция приспособленности устроена таким образом, что при одинаковом значении функции FF_1 , отражающей «прохождение» тестов автоматом, преимущество имеет автомат, содержащий меньше

переходов. Учет числа переходов в функции приспособленности необходим, так как минимизация числа переходов приводит к тому, что в результирующем автомате отсутствуют неиспользуемые в тестах переходы.

Совместное применение генетического программирования и верификации

Предлагается при вычислении функции приспособленности учитывать как поведение автомата при обработке тестов, так и число верных для автомата LTL-формул. При этом, чем больше число верных формул и успешно пройденных тестов, тем больше значение функции приспособленности.

Для вычисления функции приспособленности конечный автомат, задаваемый рассматриваемой особью, запускается на всех тестах и проверяется на соответствие всем темпоральным формулам, составляющим спецификацию. После этого вычисляется величина

$$FF = FF_1 \cdot \left(1 + \frac{n_1}{n_2}\right) + \frac{1}{M} \cdot (M - cnt)$$

Здесь как n_2 обозначено общее число темпоральных формул в спецификации, а как n_1 – число формул, которые являются верными для рассматриваемого конечного автомата.

Структура хромосомы в алгоритме генетического программирования

Конечный автомат в алгоритме генетического программирования представляется в виде объекта, который содержит описания переходов для каждого из состояний и номер начального состояния. Для каждого из состояний хранится список переходов. Каждый переход описывается событием, при поступлении которого этот переход

выполняется и числом выходных воздействий, которые должны быть сгенерированы при выборе этого перехода.

Таким образом, в особи кодируется только «скелет» управляющего конечного автомата, а конкретные выходные воздействия, вырабатываемые на переходах, определяются с помощью алгоритма расстановки пометок, который аналогичен предложенному в работе [11].

Идея алгоритма расстановки пометок состоит в том, что генетическим алгоритмом строится только «скелет» конечного автомата, а пометки на переходах – вырабатываемые на них выходные воздействия – расставляются на основе тестов. При этом расстановка пометок происходит таким образом, чтобы получившийся в результате автомат как можно лучше «соответствовал» тестам.

Опишем более формально алгоритм расстановки пометок на переходах, применяемый в настоящей работе. Подадим на вход конечному автомату последовательность событий, соответствующую одному из тестов, и будем наблюдать за тем, какие переходы выполняет автомат. Зная эти переходы и информацию о том, сколько выходных воздействий должно быть сгенерировано на каждом из переходов, можно определить, какие выходные воздействия должны вырабатываться на переходах, использовавшихся при обработке входной последовательности (рис. 1).

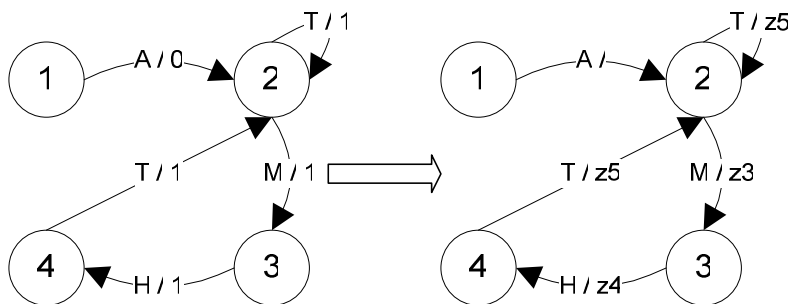


Рис. 1. Применение алгоритма расстановки пометок

Для случая нескольких тестов этот принцип можно обобщить следующим образом. Для каждого перехода T и каждой последовательности выходных воздействий zs вычисляется величина $C[T][zs]$ – число раз, когда при обработке входной последовательности, соответствующей одному из тестов, на переходе T должны быть выработаны выходные воздействия, образующую последовательность zs . Далее, каждый переход помечается той последовательностью zs_0 , для которой величина $C[T][zs]$ максимальна.

Операции скрещивания и мутации

Операция мутации может выполняться двумя способами – традиционным и учитывающим результат верификации. Традиционный способ описан в работе [5] – он используется в методе построения управляющих автоматов на основе обучающих примеров. Операция скрещивания может быть осуществлена тремя способами – традиционным, с учетом тестов и с учетом результата верификации. Первые два метода также описаны в работе [5].

Опишем методы мутации и скрещивания, учитывающие результат верификации. Напомним, что алгоритм верификации основан на двойном обходе в глубину автомата модели и автомата Бюхи, построенного по отрицанию LTL-формулы. При использовании такого алгоритма, та часть модели, которая была посещена в процессе первого обхода в глубину, удовлетворяет LTL-формуле и может быть использована в процессе скрещивания точно так же, как в методе скрещивания с учетом тестов (помеченные переходы копируются в новые особи напрямую). Иными словами, подграф переходов,

которые обошел верификатор в процессе верификации, может перейти без изменений в новую особь.

В то же время, мы можем не только сохранять часть модели, на которой выполняется темпоральное свойство, но и удалять те переходы, которые входят в контрпример, возвращаемый верификатором. Такой контрпример представляет собой путь в модели, поэтому при мутации мы можем либо удалить переход из этого пути, либо изменить его конечное состояние, число генерируемых выходных воздействий или событие, инициирующее переход.

Экспериментальное исследование

Экспериментальное исследование предлагаемого метода машинного обучения проводилось на задаче построения автомата управления дверьми лифта. Эта система содержит пять возможных входных событий ($e11$ – нажата кнопка «Открыть двери»; $e12$ – нажата кнопка «Закрыть двери»; $e2$ – открытие или закрытие дверей успешно завершено; $e3$ – препятствие мешает закрыть дверь; $e4$ – дверь сломалась) и три выходных воздействия ($z1$ – начать открытие дверей; $z2$ – начать закрытие дверей; $z3$ – позвонить в аварийную службу).

При построении управляющего автомата использовались девять тестов (табл. 1) и 11 темпоральных свойств (табл. 2).

Таблица 1. Тесты для системы управления дверьми лифта

Входная последовательность	Выходная последовательность
$e11, e2, e12, e2$	$z1, z2$
$e11, e2, e12, e2, e11, e2, e12, e2$	$z1, z2, z1, z2$
$e11, e2, e12, e3, e2, e12, e2$	$z1, z2, z1, z2$

e11, e2, e12, e2, e11, e2, e12, e3, e2, e12, e2	z1, z2, z1, z2, z1, z2
e11, e2, e12, e3, e2, e12, e3, e2, e12, e2	z1, z2, z1, z2, z1, z2
e11, e4	z1, z3
e11, e2, e12, e4	z1, z2, z3
e11, e2, e12, e2, e11, e4	z1, z2, z1, z3
e11, e2, e12, e3, e4	z1, z2, z1, z3

Таблица 2. Темпоральные свойства, составляющие спецификацию системы управления дверьми лифта

Формула	Комментарий
$G(\text{wasEvent}(ep.e11) \Rightarrow \text{wasAction}(co.z1))$	Если было событие e11, то было вызвано действие z1
$G(\text{wasEvent}(ep.e12) \Leftrightarrow \text{wasAction}(co.z2))$	Событие e12 обрабатывается тогда и только тогда, когда вызывается z2
$G(\text{wasEvent}(ep.e4) \Leftrightarrow \text{wasAction}(co.z3))$	Событие e4 обрабатывается тогда и только тогда, когда вызывается z3
$G(\text{wasEvent}(ep.e3) \Rightarrow \text{wasAction}(co.z1))$	Если было событие e3, то было вызвано действие z1
$G(\text{wasEvent}(ep.e2) \Rightarrow X(\text{wasEvent}(ep.e11) \text{ or } \text{wasEvent}(ep.e12)))$	Если было событие e2, то следующим обработанным событием будет e11 или e12
$G(\text{wasEvent}(ep.e11) \Rightarrow X(\text{wasEvent}(ep.e4) \text{ or } \text{wasEvent}(ep.e2)))$	Если было событие e11, то следующим обработанным событием будет e4 или e2
$G(\text{wasAction}(co.z1) \Rightarrow X(\text{wasEvent}(ep.e2) \text{ or } \text{wasEvent}(ep.e4)))$	Если было вызвано действие z1, то следующим обработанным

	событием будет e2 или e4
G(wasEvent(ep.e12) => X(wasEvent(ep.e2) or wasEvent(ep.e3) or wasEvent(ep.e4)))	Если было событие e12, то следующим обработанным событием будет e2, или e3, или e4
G(wasAction(co.z1) => X(U(!wasAction(co.z1), wasAction(co.z2) or wasEvent(ep.e4))))	Если было вызвано действие z1, то оно не будет больше вызвано, пока не будет вызвано z2 или обработано событие e4
G(wasAction(co.z2) => X(U(!wasAction(co.z2), wasAction(co.z1) or wasEvent(ep.e4))))	Если было вызвано действие z2, то оно не будет больше вызвано, пока не будет вызвано z1 или обработано событие e4
!F(wasEvent(ep.e4) and X(F(wasEvent(ep.e11) or wasEvent(ep.e12) or wasEvent(ep.e2) wasEvent(ep.e3))))	Не верно, что в будущем будет после события e4 когда либо будут обработаны e11, e12, e2 или e3 (лифт не может самостоятельно починиться)

Целью экспериментального исследования было сравнение метода построения управляющих конечных автоматов на основе тестов с предлагаемым в настоящей работе методом, использующим верификацию моделей на различных стадиях генетического алгоритма (вычисление функции приспособленности, скрещивание и мутация). Для каждого метода было проведено 1000 экспериментов, для каждого из которых записывалось число вычислений функции приспособленности.

Эксперименты показали, что при построении автоматов только на основе тестов, очень редко (всего в девяти случаях из 1000) результатом являлся автомат, который полностью удовлетворяет спецификации. Пример автомата, построенного только на основе тестов, приведен на рис. 2.

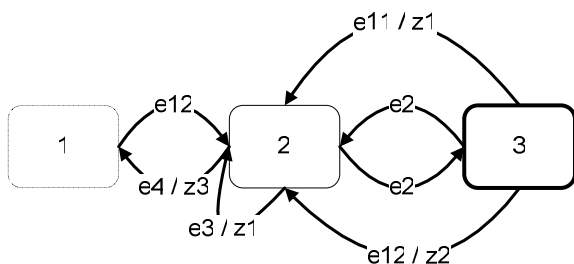


Рис. 2. Автомат управления дверьми лифта, построенный только на основе обучающих примеров

Этот автомат обладает тем недостатком, что может отдать команду на закрытие дверей после того, как они сломаются или же начать открывать (закрывать) двери, когда они уже открыты (закрыты).

При использовании предлагаемого метода (с применением верификации моделей) построение автомата (рис. 3) занимало больше времени, но построенный автомат удовлетворял всем требованиям спецификации.

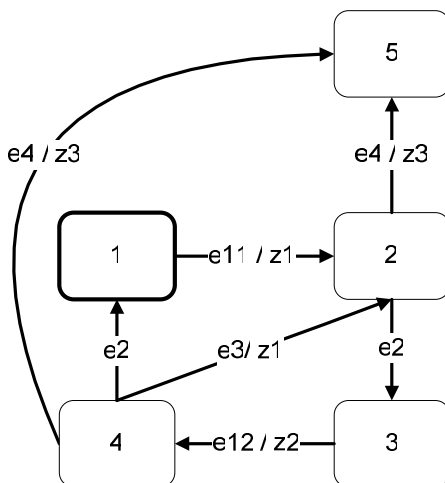


Рис. 3. Автомат управления дверьми лифта, построенный с использованием верификации

При построении конечного автомата управления дверьми лифта только на основе тестов, среднее значение вычислений функции приспособленности оказалось равным 7.479×10^4 (минимальное число вычислений – 2.184×10^4 , максимальное – 2.999×10^5 , среднеквадратичное отклонение – 2.54×10^4).

При использовании верификации моделей совместно с тестами, среднее значение числа вычислений функции приспособленности оказалось равным 8.372×10^5 (минимальное число вычислений – 6.331×10^4 , максимальное – 5.912×10^6 , среднеквадратичное отклонение – 7.57×10^5).

Таким образом, использование верификации хоть и замедляет процесс построения управляющего конечного автомата примерно в десять раз, но если принять во внимание то, что при построении только на основе тестов процент правильно построенных автоматов меньше 1%, то

совместное применение тестов и верификации оправдывает себя.

Заключение

В работе предложен метод машинного обучения для построения управляющих конечных автоматов на основе обучающих примеров. Предложенный метод основан на совместном применении генетического программирования и верификации моделей программ. Применение верификации в процессе построения автомата позволяет говорить об автоматизированном построении автоматов с гарантированным поведением.

Литература

1. *Поликарпова Н. И., Шальто А. А.* Автоматное программирование. СПб: Питер, 2009.
2. *Angeline P. J., Pollack J.* Evolutionary Module Acquisition // Proceedings of the Second Annual Conference on Evolutionary Programming. 1993. <http://www.demo.cs.brandeis.edu/papers/ep93.pdf>
3. *Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C., Wang A.* The Genesys System. 1992. <http://www.cs.ucla.edu/~dyer/Papers/AlifeTracker/Alife91Jefferson.html>
4. *Chambers L.* Practical Handbook of Genetic Algorithms. Complex Coding Systems. Volume III. CRC Press, 1999.
5. *Царев Ф. Н.* Метод построения автоматов управления системами со сложным поведением на основе тестов с помощью генетического программирования / Материалы Международной научной конференции «Компьютерные науки и информационные технологии». Саратов: СГУ. 2009, с. 216–219.

6. *Левинштейн В. И.* Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академии Наук СССР 163.4, с. 845–848.
7. *Кларк Э., Грамберг О., Пелед Д.* Верификация моделей программ: Model Checking. М.: МЦНМО, 2002.
8. *Gerth R., Peled D., Vardi M. Y., Wolper P.* Simple On-the-fly Automatic Verification of Linear Temporal Logic / Proc. of the 15th Workshop on Protocol Specification, Testing, and Verification, Warsaw, 1995, pp. 3–18.
9. Разработка технологии верификации управляющих программ со сложным поведением, построенных на основе автоматного подхода. Второй этап. СПбГУ ИТМО, 2007.
http://is.ifmo.ru/verification/2007_02_report-verification.pdf
10. *Егоров К. В., Шалыто А. А.* Методика верификации автоматных программ // Информационно-управляющие системы. СПб: Политехника, 2008, № 5, с. 15–21.
11. *Lucas S., Reynolds T.* Learning Finite State Transducers: Evolution versus Heuristic State Merging // IEEE Transactions on Evolutionary Computation Volume 11, Issue 3, June 2007, pp. 308–325.