Статья опубликована в «Научно-техническом вестнике СПбГУ ИТМО». 2010. № 5 (69), с.81–89.

УДК 004.4’242

ПРИМЕНЕНИЕ ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ ДЛЯ ПОСТРОЕНИЯ АВТОМАТОВ УПРАВЛЕНИЯ СИСТЕМАМИ СО СЛОЖНЫМ ПОВЕДЕНИЕМ НА ОСНОВЕ ОБУЧАЮЩИХ ПРИМЕРОВ И СПЕЦИФИКАЦИИ

Егоров К. В., Царев Ф. Н., Шалыто А. А.

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Предлагается метод машинного обучения, основанный на совместном применении генетического программирования и верификации моделей для построения автоматов управления системами со сложным поведением на основе обучающих примеров. Приводится описание структуры хромосом, генетического алгоритма, операций мутации и скрещивания. Изложены результаты экспериментального исследования на задаче построения конечного автомата управления дверьми лифта.

Ключевые слова: генетическое программирование, машинное обучение, верификация моделей, автоматное программирование

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 совершить переход из рассматриваемого состояния, считывая значения входных переменных у объектов управления для проверки условия перехода в другое состояние.

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 (обучающих примеров) [5]. Однако, как известно, тесты не могут полностью описывать поведение программы, а их выполнимость не является критерием ее корректности.

Цель настоящей работы – расширение возможностей указанного метода построения автоматных программ за счет использования обучающих примеров и верификации в процессе работы алгоритма генетического программирования.

Верификация автоматных программ

Для описания спецификации управляющего конечного автомата будем применять язык логики линейного времени LTL (Linear Temporal Logic). Алгоритм верификации основан на проверке пустоты языка, задаваемого пересечением рассматриваемого конечного автомата и автомата Бюхи, соответствующего отрицанию LTL-формулы, представляющей требование к автомату [7, 8]. Эта проверка осуществляется с помощью алгоритма двойного обхода в глубину.

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 каждый из них содержит последовательность входных событий (входная последовательность, соответствующая i-ому тесту, будет обозначаться как Input[i]) и соответствующую ей последовательность выходных воздействий (в

дальнейшем будет обозначаться, как Answer[i]). Отметим, что в настоящей работе в обучающих примерах не используются входные переменные.

Функция приспособленности основана на редакционном расстоянии [6]. Для этой функции 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 функция FF_1 вида:

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

Здесь ED(A, B) – редакционное расстояние между строками A и B, а Answer[i] – 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.

Функция приспособленности должна зависеть не только от того, насколько «хорошо» автомат работает на тестах, но и числа переходов, которые он содержит. Предлагается ее вычислять 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. При этом отметим, что все рассматриваемые в процессе работы алгоритма генетического программирования автоматы имеют одинаковое наперед заданное число состояний.

Функция приспособленности построена так, что при одинаковом значении функции FF_1, учитывающей «прохождение» тестов автоматом, преимущество имеет автомат, содержащий меньшее число переходов. Учет числа переходов в функции приспособленности необходим, так как минимизация их числа приводит к тому, что в результирующем автомате отсутствуют переходы, неиспользуемые в тестах.

Совместное применение генетического программирования и верификации

Предлагается при вычислении функции приспособленности учитывать как поведение автомата при обработке тестов, так и число выполняемых (верных) для автомата LTL-формул, составляющих спецификацию. При этом, чем больше число выполняемых 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. Для учета указанных особенностей необходимо изменить введенную выше функцию приспособленности:

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

Здесь n_2 – общее число темпоральных формул в спецификации, а n_1 – число формул, которые выполняются 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 состояния и номер начального состояния. Для каждого состояния хранится список переходов. В свою очередь, 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 пометки на переходах (вырабатываемые на них выходные воздействия) расставляются на основе тестов. При этом расстановка пометок происходит таким образом, чтобы получившийся в результате автомат достаточно хорошо 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 переходе, можно определить, какие выходные воздействия должны вырабатываться на переходах, использовавшихся при обработке входной последовательности (рис. 1).

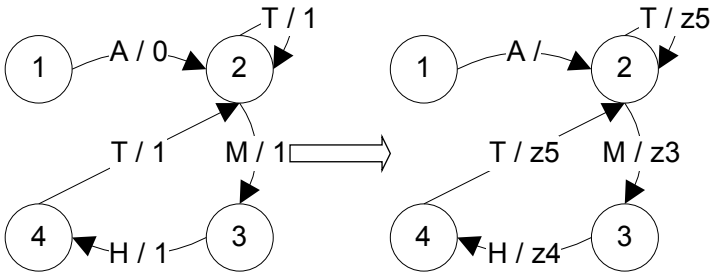

Рис. 1. Применение алгоритма расстановки пометок

На рис. 1 буквами A, H, M и T обозначены события, поступающие на вход конечного автомата, а как z3, z4 и z5 обозначены выходные воздействия.

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

Операции мутации и скрещивания

Операция мутации может выполняться двумя способами – традиционным и учитывающим результат верификации. Традиционный способ используется в методе построения управляющих автоматов на основе обучающих примеров [5].

Операция скрещивания может быть осуществлена тремя способами – традиционным, с учетом тестов и с учетом результата верификации. Первые два способа также описаны в работе [5].

Опишем методы мутации и скрещивания, учитывающие верификацию. Как отмечалось выше, алгоритм верификации основан на двойном обходе в глубину автомата, задаваемого особью, 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, должна быть обеспечена возможность 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 можно 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 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 выходные воздействия (`z1` – начать открытие дверей; `z2` – начать закрытие дверей; `z3` – позвонить в аварийную службу).

При построении управляющего автомата использовались девять тестов (табл. 1).

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`

Спецификация управляющего автомата содержит 11 темпоральных свойств (табл. 2).

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

Формула	**Комментарий**
`G( wasEvent(ep.e11) => wasAction(co.z1) )`	Если было событие `e11`, то было вызвано действие `z1`
`G( wasEvent(ep.e12) <=> wasAction(co.z2) )`	Событие `e12` обрабатывается тогда и только тогда, когда вызывается `z2`
`G( wasEvent(ep.e4) <=> wasAction(co.z3))`	Событие `e4` обрабатывается тогда и только тогда, когда вызывается `z3`
`G( wasEvent(ep.e3) => wasAction(co.z1) )`	Если было событие `e3`, то было вызвано действие `z1`
`G( wasEvent(ep.e2) => X(wasEvent(ep.e11) or wasEvent(ep.e12)) )`	Если было событие `e2`, то следующим обработанным событием будет `e11` или `e12`
`G( wasEvent(ep.e11) => X(wasEvent(ep.e4) or wasEvent(ep.e2)) )`	Если было событие `e11`, то следующим обработанным событием будет `e4` или `e2`
`G( wasAction(co.z1) => X(wasEvent(ep.e2) or 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 работы алгоритма генетического программирования (вычисление функции приспособленности, скрещивание и мутация). Для оценки трудоемкости сравниваемых методов было проведено по 1000 экспериментов, и для каждого эксперимента 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.

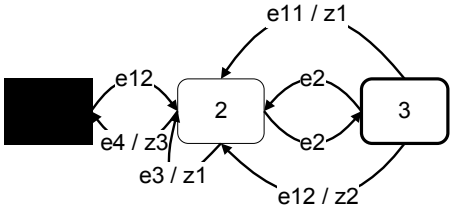

Рис. 2. Автомат управления дверьми лифта, построенный только на основе обучающих примеров

Это 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). Отметим, что некоторые из построенных в этом эксперименте автоматов обладали и другими недостатками.

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

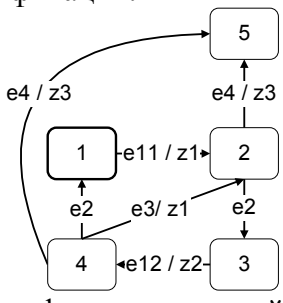

Рис. 3. Автомат управления дверьми лифта, построенный с использованием верификации

При построении конечного автомата управления дверьми лифта только на основе тестов, среднее значение вычислений функции приспособленности оказалось равным $7{,}479 \cdot 10^4$ (минимальное число вычислений – $2{,}184 \cdot 10^4$, максимальное – $2{,}999 \cdot 10^5$, среднеквадратичное отклонение – $2{,}54 \cdot 10^4$).

При использовании верификации моделей совместно с тестами, среднее значение числа вычислений функции приспособленности оказалось равным 7.246×10^5 (минимальное число вычислений – $7{,}054 \cdot 10^4$, максимальное – $5{,}492 \cdot 10^6$, среднеквадратичное отклонение – $7{,}729 \cdot 10^5$).

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 применение предлагаемого в настоящей работе метода оправдывает себя.

Заключение

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 работы алгоритма генетического программирования позволяет говорить об автоматизированном построении автоматов с гарантированным поведением.

Исследование проводится в рамках Федеральной целевой программы «Научные и научно-педагогические кадры инновационной России на 2009-2013 годы», а также финансируется по гранту РФФИ № 10-01-00654а.

Литература

1. Поликарпова Н. И., Шалыто 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. V. 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. Левенштейн В. И. Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академии Наук СССР. 1963. № 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. 2008. № 5, с. 15–21.
11. Lucas S., Reynolds T. Learning Finite State Transducers: Evolution versus Heuristic State Merging // IEEE Transactions on Evolutionary Computation. V. 11. Issue 3. 2007, pp. 308–325.