Статья опубликована в сборнике докладов конференции молодых ученых и специалистов «Информационные технологии и системы» (ИТиС' 09). М.: ИППИ РАН. 2009, с. 77 – 82.

Совместное применение генетического программирования и верификации моделей для построения автоматов управления системами со сложным поведением

К.В. Егоров, Ф.Н. Царев
Санкт-Петербургский государственный университет информационных технологий, механики и оптики
egorovk@rain.ifmo.ru, tsarev@rain.ifmo.ru

Аннотация

Цель работы заключается в разработке метода совместного применения генетического программирования и верификации моделей для построения автоматов управления системами со сложным поведением. Исходными данными являются тесты (каждый тест состоит из входной последовательности событий и соответствующей ей последовательности выходных действий, которую должен вырабатывать автомат) для системы со сложным поведением и утверждений на языке логики линейного времени (Linear Time Logic, LTL).

В предыдущих работах построение автоматов осуществлялось только на основе тестов. Одним из недостатков такого подхода является то, что с помощью тестов достаточно трудно (такое описание будет слишком громоздким) описать все варианты поведения. Это означает, что построенный таким образом автомат нельзя использовать без дополнительных проверок. В случае обнаружения ошибок в построенном автомате его придется модифицировать вручную, что достаточно трудно, так как зачастую компьютер выделяет состояния и переходы не так, как это делает человек.

Предлагаемый подход исправляет указанный недостаток. Функция приспособленности, используемая в алгоритме генетического программирования, учитывает успешность прохождения тестов и истинность LTL-формул. В случае прохождения всех тестов и выполнения всех LTL-формул можно считать, что автомат с заранее заданным поведением построен.

Приводятся результаты экспериментального исследования на двух задачах – на одной из них применение верификации позволяет построить удовлетворяющий спецификации автомат, который не получается построить только на основе тестов, а на другой применение верификации замедляет построение автомата.

1. Введение

Автоматное программирование – это парадигма программирования, в рамках которой программы предлагается проектировать в виде совокупности взаимодействующих автоматизированных объектов управления [1]. При этом каждый автоматизированный объект управления содержит внутри себя управляющий конечный автомат.

Для многих задач автоматы удается строить эвристически, однако существуют задачи, для которых такое построение автоматов затруднительно. К задачам этого класса относится, в частности, задача об «Умном муравье» [2–4]. Для построения автомата для этой задачи в указанных работах достаточно успешно применялись генетические алгоритмы.

Одним из авторов настоящей работы в работе [5] предложен метод построения автоматов с помощью генетического программирования на основе тестов. Однако, как известно, тесты не могут описывать всего поведения программы, и их выполнимость не может служить критерием правильности. Поэтому нельзя использовать автоматные программы, созданные на их основе, без дополнительных проверок и, в случае неправильности, необходимо вручную менять программу или набор тестов.

Метод проверки того, что программная система соответствует заявленной спецификации (обладает необходимыми свойствами или удовлетворяет определенным требованиям), называется верификацией. Обычно систему верифицируют после ее создания, но это сложный процесс, поэтому верификации, как правило, не уделяют должного внимания. В

настоящей работе предлагается использовать верификацию для создания программ и описывать поведение системы на языке верификатора до создания программы.

Наиболее практичным в настоящее время является метод верификации, названный *Model Checking* [6, 7]. Его использование предполагает преобразование программы в формальную модель с конечным числом состояний для последующей верификации и формальную запись утверждений, которые требуется проверить. Для программ, написанных традиционным путем, возникают сложности при построении модели по программе и при преобразовании контрпримера из терминов модели в термины программы.

Этих сложностей можно избежать, если программа является автоматной [1, 8]. Здесь имеет место та же ситуация, что и при контроле аппаратуры, которая при сложной логике не может быть проверена, если она не спроектирована специальным образом с учетом контролепригодности.

В настоящей работе рассматриваются только автоматные программы. Особенности этого класса программ позволяют избежать преобразований к модели и обратно, так как автоматная программа представляет собой модель, пригодную для верификации.

В ходе работы разработано средство, которое принимает на вход набор тестов и *LTL*-формул, а на выход выдает автоматную программу, удовлетворяющую как тестам, так и *LTL*-формулам.

2. Генерация автоматных программ на основе тестов

В автоматных программах выделяют три типа объектов: поставщики событий, система управления и объекты управления. Система управления представляет собой конечный автомат или систему взаимодействующих автоматов. Автомат – это множество состояний и переходов между ними. Каждый переход помечен событием, при котором он может осуществиться, и условием, выполнимость которого требуется для перехода. Поставщики событий генерируют события, а система управления по каждому событию может совершать переход, считывая значения входных переменных у объектов управления для проверки условия перехода. Такая система называется реагирующей или событийной [9].

Автоматные программы строятся генетическим алгоритмом на основе тестов. Каждый тест для программы представляет собой последовательность входных и соответствующих ей выходных воздействий. Под входными воздействиями понимаются события от поставщиков событий и условия переходов, а выходные – вызываемые действия объектов управления.

Построение автоматов на основе тестов осуществляется на основе алгоритма генетического программирования. Он содержит такие элементы, как создание начальной популяции, подсчет функции приспособленности, скрещивание, мутация.

Начальная популяция создается случайным образом – генерируются автоматы, содержащие одинаковое число состояний. В каждом из них начальное состояние выбирается случайным образом, число переходов из каждого состояния выбирается случайно, события, по которым выполняются переходы, также назначаются случайным образом.

Функция приспособленности вычисляется исходя из успешности выполнения тестов для автомата. Вычисление функции приспособленности основано на редакционном расстоянии (расстоянии Левенштейна) [10]. Функция приспособленности основана на редакционном расстоянии. 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

$$\mathrm{FF}_1 = \frac{\sum_{i=1}^{n}\left(1 - \frac{\mathrm{ED}(\mathrm{Output}[i], \mathrm{Answer}[i])}{\max(|\mathrm{Output}[i]|, |\mathrm{Answer}[i]|)}\right)}{n}$$

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.

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

$$\mathrm{FF}_2 = \begin{cases} 0.5 \cdot T \cdot \mathrm{FF}_1 + 0.01 \cdot (100 - \mathrm{cnt}), \mathrm{FF}_1 < 1 \\ T + 0.01 \cdot (100 - \mathrm{cnt}), \mathrm{FF}_1 = 1 \end{cases}$$

В этой формуле как *cnt* обозначено число переходов в автомате, а как *T* обозначена «стоимость» прохождения всех тестов (при

проведении экспериментов было выбрано значение $T=100$).

Эта функция приспособленности устроена таким образом, что при одинаковом значении функции FF_1, отражающей «прохождение» тестов автоматом, преимущество имеет автомат, содержащий меньше переходов. Кроме этого, автомат, который «идеально» проходит все тесты, оценивается выше, чем автомат, проходящий тесты не идеально.

Учет числа переходов в функции приспособленности необходим по двум причинам. Во-первых, минимизация числа переходов приводит к тому, что в результирующем автомате отсутствуют неиспользуемые в тестах переходы – так как они не используются, то могут быть удалены из автомата без ущерба для его поведения на тестах. Во-вторых, чем меньше в автомате переходов, тем более «общее» поведение он задает. Таким образом, частично решается проблема «переобучения», заключающаяся в том, что автомат демонстрирует правильное поведение только на тестовых входных последовательностях. Две указанные особенности должны учитываться при построении набора обучающих тестов.

Кроме этого, при проверке всех тестов помечаются переходы, участвующие в обработке каждого из них.

При скрещивании используется стратегия элитизма, то есть в следующее поколение переходят несколько лучших особей без изменений. В скрещивании участвуют все особи, причем скрещивание может быть как случайным, без учета прохождения тестов, так и с сохранением вершин, помеченных при вычислении функции приспособленности.

Мутация заключается в случайном изменении события перехода, вызванных действий или конечного состояния перехода.

Благодаря стратегии элитизма и скрещиванию «по тестам» обеспечивается рост максимального значения функции приспособленности по поколению и, в итоге, выполнение всех тестов. Скрещивания, выполняемые традиционным образом, и мутации позволяют избежать преждевременной сходимости к локальным максимумам, в которых выполняются не все тесты.

3. Верификация автоматных программ

В настоящей работе верифицируется не вся автоматная программа, а только ее модель, представленная конечным автоматом. При этом поставщики событий и объекты управления рассматриваются в качестве «внешней среды», которая ничего не помнит о последовательности переходов рассматриваемого автомата и вызванных действиях. Таким образом, в любой момент времени может быть совершен любой переход из текущего состояния автомата [11].

Для описания требований к автоматным программам будем применять язык *LTL*, в котором время линейно и дискретно. Синтаксис *LTL* включает в себя пропозициональные переменные Prop, булевы связки $(\neg, \wedge, \vee)$ и темпоральные операторы. Последние применяются для составления утверждений о событиях в будущем.

Будем использовать следующие темпоральные операторы:

- **X** (ne**X**t) – «Xp» – в следующий момент выполнено p;
- **F** (in the **F**uture) – «Fp» – в некоторый момент в будущем будет выполнено p;
- **G** (**G**lobally in the future) – «Gp» – всегда в будущем выполняется p;
- **U** (**U**ntil) – «pUq» – существует состояние, в котором выполнено q и до него во всех предыдущих выполняется p;
- **R** (**R**elease) – «pRq» – либо во всех состояниях выполняется q, либо существует состояние, в котором выполняется p, а во всех предыдущих выполнено q.

Множество *LTL*-формул таково:

- пропозициональные переменные Prop;
- True, False;
- φ и ψ – формулы, то:
  - $\neg\varphi$, $\varphi\wedge\psi$, $\varphi\vee\psi$ – формулы;
  - $X\varphi$, $F\varphi$, $G\varphi$, $\varphi U\psi$, $\varphi R\psi$ – формулы.

Верификатор, используемый в данной работе, на вход получает модель автоматной программы и *LTL*-формулу. После проверки модели, верификатор либо сообщает, что формула выполняется, либо приводит контрпример – путь в модели, опровергающий утверждение. Подробнее об этом можно прочитать в работах [7, 12, 13].

4. Совместное применение генетического программирования и верификации

Как отмечалось выше, создание автоматов только на основе тестов не может гарантировать правильное поведение программы. Поэтому предлагается использовать *LTL*-формулы для описания заранее заданного поведения и их использование для генерации конечного автомата.

Проверка *LTL*-формул будет выполняться в процессе работы алгоритма генетического программирования для автоматов, которые рассматриваются в процессе его работы. Заметим, что утверждения об автоматах ограничиваются утверждениями о переходах и действиями над объектами управления, например о последовательности обработанных событий и выработанных при этом действиях. Это ограничение объясняется тем, что мы заранее не можем ничего знать о состояниях.

Предлагается также генетическими алгоритмами строить программу, но при этом учитывать результат верификации. Для каждой особи из текущей популяции при вычислении функции приспособленности верифицировать модель (конечный автомат) особи и учитывать результат для вычисления приспособленности. Чем больше число успешно выполненных *LTL*-формул для конкретной особи, тем больше должна быть функция приспособленности. Это позволяет выживать особям, для которых выполняется большее число *LTL*-формул. Приведем выражение для функции приспособленности: $\mathrm{FF}_3 = \mathrm{FF}_2 + F \cdot \frac{n_1}{n_2}$. Эта функция приспособленности основана на функции приспособленности, которая использовалась при генерации автоматных программ на основе тестов. В указанной формуле как *F* обозначена «стоимость» выполнения всех формул (при проведении вычислительных экспериментов было выбрано значение $F=10$), как n_1 – число *LTL*-формул, которые выполняются для автомата, а как n_2 – общее число *LTL*-формул, специфицирующих поведение системы со сложным поведением.

Однако выполнимость или невыполнимость *LTL*-формул позволяет только отбирать «лучшие» особи, но не позволяет «улучшать» популяцию путем скрещивания или мутации, так как этот процесс был бы случайным и не гарантировал бы увеличения числа верных утверждений. Это обусловлено тем, что мы не учитывали бы влияния структуры автомата (вершин и переходов) на выполнимость или невыполнимость формул. Для преодоления этого предлагается учитывать контрпример построенный верификатором. Такой контрпример представляет собой список вершин и переходов автомата, которые опровергают *LTL*-формулу. Благодаря такому списку мы можем увеличить вероятность скрещивания или мутации по одному или нескольким переходам из контрпримера. Например, мы можем случайно заменить конечное состояние перехода, входное воздействие и выходные. Благодаря предложенным действиям перестанет существовать данный контрпример для *LTL*-формулы, и увеличатся шансы новой особи соответствовать большему числу *LTL*-формул.

Необходимость применения верификации при автоматическом построении модели программы можно рассмотреть на примере автомата управления дверьми лифта [11]. Поставщики событий генерируют следующие события: e11 – открыть двери, e12 – закрыть двери, e2 – успешное открытие или закрытие дверей, e3 – препятствие мешает закрыть двери, e4 – двери сломались. У объекта управления есть следующие методы: z1/z2 запуск открытия/закрытия дверей, z3 – звонок в аварийную службу. Предположим, что автомат строится только на основе тестов, приведенных в таблице.

Таблица. Набор тестов для автомата управления дверьми лифта.

#	Входные воздействия	Выходные воздействия
1	e11, e2, e12, e2	z1, z2
2	e11, e2, e12, e2, e11, e2, e12, e2	z1, z2, z1, z2
3	e11, e2, e12, e3, e2, e12, e2	z1, z2, z1, z2
4	e11, e2, e12, e2, e11, e2, e12, e3, e2, e12, e2	z1, z2, z1, z2, z1, z2
5	e11, e2, e12, e3, e2, e12, e3, e2, e12, e2	z1, z2, z1, z2, z1, z2
6	e11, e4	z1, z3
7	e11, e2, e12, e4	z1, z2, z3
8	e11, e2, e12, e2, e11, e4	z1, z2, z1, z3
9	e11, e2, e12, e3, e4	z1, z2, z1, z3

Согласно приведенным тестам получится неправильный автомат (рис. 1), для которого выполняются все тесты. В сгенерированном автомате открытие дверей может быть начато, когда двери уже открыты, и аналогично закрытие, когда двери закрыты. Так же после поломки лифта двери опять могут начать закрываться, хотя этого не предусмотрено.

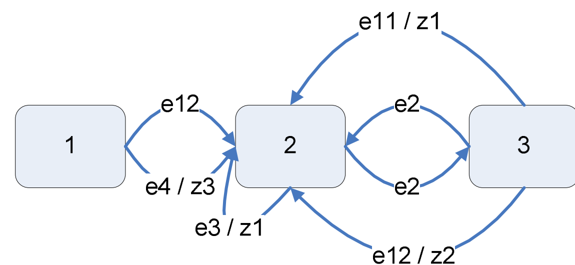

Рис. 1. Автомат, построенный только на основе тестов

Статья опубликована в сборнике докладов конференции молодых ученых и специалистов
«Информационные технологии и системы» (ИТиС' 09). М.: ИППИ РАН. 2009, с. 77 – 82.

Для ограничения такого поведения и требуются *LTL*-формулы. Например, они позволяют делать утверждения о допустимости и недопустимости определенных последовательностей событий неограниченной длины. При помощи тестов можно описывать только последовательности ограниченной длины.

Построение модели не только на тестах, но и на *LTL*-формулах, позволяет получить автомат с правильным поведением. Приведем некоторые из них:

- **G**(wasEvent(ep.e11) => wasAction(co.z1)) – при обработке события открыть двери, запускается открывание дверей;
- **G**(wasEvent(ep.e4) <=> wasAction(co.z3)) – если сломалась дверь, то тогда и только тогда делается звонок в аварийную службу;
- **G**(wasEvent(ep.e3) => wasAction(co.z1)) – если препятствие мешает открыть дверь, то открыть дверь;
- **G**(wasEvent(ep.e11) => **X**(wasEvent(ep.e4) **or** wasEvent(ep.e2))) – после обработки события открыть двери, следующее будет либо успешное открытие, либо событие поломки;
- **G**(wasAction(co.z1) => **X**(!wasAction(co.z1) **U**(wasAction(co.z2) **or** wasEvent(ep.e4)))) – если дверь начала открываться, то она не сможет повторно открыться до закрытия или поломки.

Построенный автомат показан на рис. 2 и он не обладает недостатками, указанными выше.

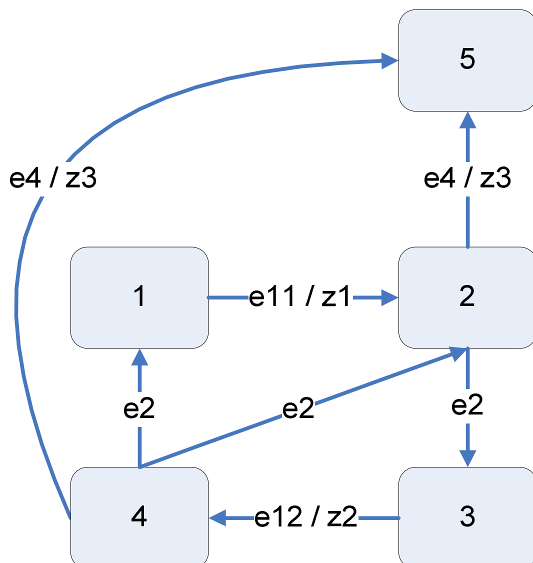

Рис. 2. Автомат, построенный на основе тестов и *LTL*-формул

5. Сравнение методов

Сравнение методов проводилось на примере задачи управления электронными часами с будильником, взятом из работы [1]. В работе [5] для этой задачи автомат был построен только на основе тестов.

В качестве исходных данных были выбраны 38 тестов, описывающих поведение часов с будильником в различных режимах работы. В этих тестах размер входных последовательностей составлял от трех до 12 событий, а размер выходных последовательностей – от одного до 12 выходных воздействий. При применении метода, использующего верификацию, к этим тестам были добавлены одиннадцать *LTL*-формул.

Авторами настоящей работы было проведено 1000 вычислительных экспериментов с использованием каждого из методов. Во всех случаях результатом построения являлся один и тот же конечный автомат, показанный на рис. 3. Отметим, что этот автомат изоморфен автомату, построенному вручную в работе [1].

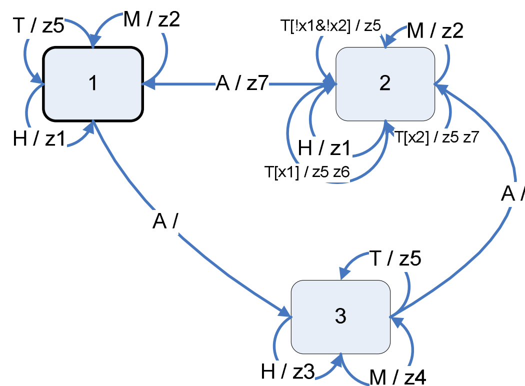

Рис. 3. Автомат управления часами с будильником

На этом рисунке начальным состоянием автомата является состояние с номером «1».

Для каждого из запусков каждого из алгоритмов запоминалось количество вычислений функции приспособленности (оно равно числу просмотренных в процессе работы автоматов) в процессе построения автомата. Для алгоритма, использующего только тесты, минимальное значение этой величины составило 256063, максимальное – 9239523, среднее значение – 1450467.28 (стандартное отклонение – 1106266.586).

При использовании и тестов, и *LTL*-формул минимальное значение этой величины составило 218236, максимальное – 19492104, среднее значение – 2422871.644 (стандартное отклонение – 2312211.574).

Таким образом, добавление верификации замедлило построение автомата. Это можно объяснить тем, что промежуточные особи, проходящие часть тестов, не удовлетворяли *LTL*-формулам, поэтому сложнее было получить «улучшение» популяции.

В то же время, попытка избавиться от контрпримера, найденного верификатором, не всегда дает лучшую особь (особь с большей функцией приспособленности), так как после предлагаемых действий автомат может, как удовлетворять утверждению, так и перестать проходить некоторые тесты.

Для предотвращения такой ситуации планируется учитывать верификацию не за счет устранения плохого пути, а за счет соответствия *LTL*-формуле. Для этого при скрещивании двух особей оставлять неизменной часть автомата, удовлетворяющего формуле, аналогично скрещиванию по тестам. Такую часть автомата можно получить в процессе верификации – когда при обходе в глубину верификатор возвращается в вершину, то на уже просмотренной части автомата *LTL*-формула выполнятся. Сохраняя неизменной эту часть автомат при скрещивании можно действительно получить особь, удовлетворяющую большему числу утверждений и тестов.

6. Заключение

Разработанное средство показало возможность применения верификации и генетических алгоритмов для генерации модели программы. Оно позволяет автоматически строить модель программы по набору тестов и *LTL*-формул. Однако не имеет значения выбор языка логики, с таким же успехом можно было использовать любой другой язык, позволяющий делать утверждения о модели программы. Язык *LTL* был выбран только в качестве примера, имеется возможность перейти на другой верификатор с иным входным языком.

Был продемонстрирован пример, когда применение одних только тестов приводит к построению «неправильного» конечного автомата. В тоже время, совместное применение тестов и *LTL*-формул позволяет добиться правильного поведения автомата.

6. 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. 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] Hoffman L. Talking Model-Checking Technology // Communications of the ACM, 2008, V. 51. № 7, pp. 110–112.

[7] Кларк Э., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. М.: МЦНМО, 2002.

[8] Шалыто А. А. Switch-технология. Алгоритмизация и программирование задач логического управления. СПб: Наука, 1998.

[9] Harel D. et al. Statemate: A Working Environment for the Development of Complex Reactive Systems // IEEE Trans. Software Eng. 1990, № 4, pp. 403–414.

[10] Левенштейн В. И. Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академии Наук СССР 163.4, с. 845–848.

[11] 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

[12] 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.

[13] Егоров К. В., Шалыто А. А. Методика верификации автоматных программ // Информационно-управляющие системы. СПб: Политехника, 2008, № 5, с. 15–21.