УДК 004.4'242

ПРИМЕНЕНИЕ МЕТОДОВ МАШИННОГО ОБУЧЕНИЯ ДЛЯ АВТОМАТИЗИРОВАННОГО ПОСТРОЕНИЯ УПРАВЛЯЮЩИХ АВТОМАТОВ В ВЫСОКОУРОВНЕВЫХ СРЕДСТВАХ ПРОЕКТИРОВАНИЯ СИСТЕМ

Н.В. Ведерников
Университет ИТМО
Россия, 197101, Санкт-Петербург, Кронверкский пр., 49
E-mail: vedernikovnv@gmail.com

В.Ю. Демьянюк
Университет ИТМО
Россия, 197101, Санкт-Петербург, Кронверкский пр., 49
E-mail: chavit92@gmail.com

П.А. Кротков
Университет ИТМО
Россия, 197101, Санкт-Петербург, Кронверкский пр., 49
E-mail: krotkov.pavel@gmail.com

В.И. Ульянцев
Университет ИТМО
Россия, 197101, Санкт-Петербург, Кронверкский пр., 49
E-mail: ulyantsev@rain.ifmo.ru

А.А. Шалыто
Университет ИТМО
Россия, 197101, Санкт-Петербург, Кронверкский пр., 49
E-mail: shalyto@mail.ifmo.ru

Ключевые слова: системы управления, управляющие автоматы, *Stateflow*, *MATLAB*/*Simulink*, машинное обучение, идентификация систем

Аннотация: В последнее время при проектировании систем управления все чаще применяются управляющие конечные автоматы. Так, все более широкое распространение получают высокоуровневые средства проектирования управляющих систем, ключевыми элементами в которых являются управляющие автоматы. Построение данных автоматов производится специалистом вручную, после чего генерация кода и программирование целевого устройства производятся автоматизировано. Настоящая работа направлена на повышение уровня автоматизации и уменьшение влияния человеческого фактора на этапе построения управляющих автоматов в указанных средствах проектирования. В качестве примера проводится автоматизированное построение управляющих автоматов сре-

ды *Stateflow*, входящей в пакет *MATLAB/Simulink*, по экспертным сценариям работы при помощи существующих методов машинного обучения.

1. Введение

Для построения управляющих систем, как в индустрии, так и в некоммерческих целях все чаще применяются среды высокоуровневого проектирования. Примерами таких сред являются *MATLAB/Simulink*[1], *LabVIEW*[2], *VisSim*[3]. После визуального моделирования управляющих автоматов в данных средах можно провести автоматизированную симуляцию работы созданной реактивной системы, автоматически преобразовать ее в код для множества целевых платформ на таких языках, как *C*, *C++*, *VHDL*, *Verilog*.

Перечисленные среды предоставляют возможность проектировать управляющие элементы в виде взаимодействующих управляющих автоматов, что соответствует парадигме автоматного программирования, при использовании которой программа или ее фрагмент осмысливается как модель какого-либо формального автомата [1]. Основными преимуществами такого подхода являются наглядность представления логики системы со сложным поведением, а также возможность формальной верификации системы [2], которая затруднительна при применении других подходов к ее программированию.

Для решения многих задач управляющие автоматы удается построить эвристически [3], однако существуют задачи [4, 5], для которых построение автоматов вручную затруднительно. Для автоматизированного построения управляющих автоматов, являющихся решениями таких задач, в последнее время разработано множество методов машинного обучения. Входными данными для данных методов являются или экспертные данные, накладывающие ограничения на идентифицируемую систему, или функция приспособленности управляющего автомата.

На данный момент авторам не известны примеры применения указанных методов машинного обучения для автоматизированного построения управляющих автоматов в средах высокоуровневого проектирования. **Целью настоящей работы** является демонстрация возможности применения разработанных методов для автоматизации проектирования управляющих автоматов в данных средах. Авторами разработано программное средство, позволяющее идентифицировать, загрузить и корректно отобразить управляющий автомат в среде *Stateflow*, входящей в пакет *MATLAB/Simulink*.

2. Методы машинного обучения для построения управляющих конечных автоматов

Управляющим конечным автоматом называется детерминированный конечный автомат, каждый переход которого помечен событием, последовательностью выходных воздействий и охранным условием, представляющим собой логическую формулу от входных переменных [1]. Автомат получает события от так называемых *поставщиков событий* (в их роли могут выступать внешняя среда, интерфейс пользователя и т.д.) и генерирует выходные воздействия для объекта управления. При поступлении события автомат выполняет переход в соответствии с охранными условиями и значениями входных переменных. При выполнении перехода генерируются выходные воздействия,

[1] http://www.mathworks.com/products/simulink/
[2] http://sine.ni.com/np/app/main/p/docid/nav-104/lang/ru
[3] http://www.vissim.com/

которыми он помечен, и автомат переходит в соответствующее состояние. Пример управляющего автомата с двумя состояниями приведен на рис. 1.

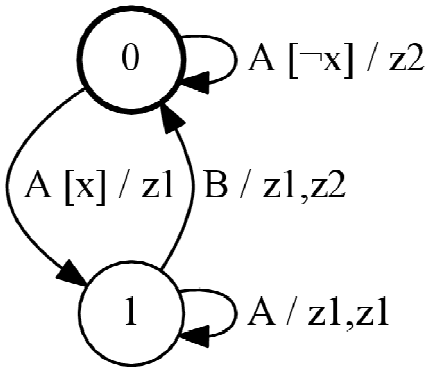

Рис. 1. Пример управляющего конечного автомата.

Вкратце опишем известные подходы к построению конечных автоматов. Отметим, что в общем случае решаются *NP*-трудные задачи построения автоматов, для которых актуальна разработка новых алгоритмов решения. Для построения управляющих автоматов по заданной функции приспособленности используются такие эволюционные алгоритмы, как: эволюционные стратегии [6], генетические алгоритмы [4], а также специализированные муравьиные алгоритмы [5].

В работе [7] решается частная задача идентификации управляющих конечных автоматов по экспертным данным, заданным в виде *сценариев работы программы*. Например, следующие сценарии работы соответствуют автомату, приведенному на рис. 1:

- <*A*; ¬*x*; *z*2>, <*A*; *x*; *z*1>, <*A*; *true*; *z*1,*z*1>;
- <*A*; *x*; *z*1>, <*B*; *true*; *z*1,*z*2>, <*A*; *x*; *z*1>;
- <*A*; *x*; *z*1>, <*B*; *true*; *z*1,*z*2>, <*A*; ¬*x*; *z*2>, <*A*; ¬*x*; *z*2>.

В основе данного метода лежит сведение к задаче о выполнимости булевой формулы (*SAT*), а скорость его работы на несколько порядков превышает скорость работы методов, основанных на эволюционных алгоритмах. При этом метод гарантирует нахождение решения, в случае его существования, в отличие от методов, основанных на эволюционных алгоритмах.

В настоящей работе в качестве метода машинного обучения используется последний из описанных, основанный на методах решения задачи о выполнимости булевой формулы. При помощи метода решается задача построения управляющего конечного автомата, удовлетворяющего следующим требованиям:

- число состояний автомата равно заданному числу *C*;
- автомат удовлетворяет заданному множеству сценариев работы *S*;
- каждый переход автомата подтвержден хотя бы одним сценарием работы.

После работы программного средства, реализующего метод, в случае существования решения будет выведен управляющий автомат в формате *DOT*[4]. Например, автомат, приведенный на рис. 1 будет выведен следующим образом:

```
digraph EFSM {
    node [shape = circle];
    0 [style = bold];
    0 -> 1 [label = " A [x] / z1 "];
    0 -> 0 [label = " A [¬x] / z2 "];
    1 -> 0 [label = " B / z1,z2 "];
```

[4] http://www.graphviz.org/content/dot-language

```
        1 -> 1 [label = " A / z1,z1 "];
    }
```

3. Применение метода машинного обучения для построения моделей среды Stateflow

Преимущества использования управляющих автоматов привели к появлению множества специализированных пакетов или полноценных программных продуктов, предназначенных для визуального построения управляющих автоматов. Одним из наиболее распространенных пакетов является интерактивная среда *Stateflow*, являющаяся частью программного пакета *MATLAB/Simulink*. Среда *Stateflow* предоставляет большой набор разнообразных инструментов, позволяющих спроектировать автоматную модель, промоделировать ее работу на тех или иных входных данных, проверить ее работоспособность с помощью тестирования, преобразовать автоматную модель в программный код и интегрировать его в реальную среду. Опишем процесс построения управляющих автоматов для данной среды – после построения управляющего автомата в формате *DOT* преобразуем его во внутренний формат *Stateflow*.

В качестве целевого формата, в который будет преобразовываться построенный с помощью методов машинного обучения автомат, выбран формат *MDL*, являющийся внутренним форматом программного комплекса *MATLAB/Simulink*.

Основное отличие *MDL* от других форматов хранения данных, используемых средой *MATLAB*/*Simulink*, является представление данных в текстовом, понятном человеку формате. По сути, описание модели, сохраненное в формате *MDL*, является списком всех объектов, участвующих в модели, вместе с перечислением их параметров и значений этих параметров. Таким образом, данный формат является наиболее удобным для автоматического преобразования построенных моделей систем управления.

Множества событий, выходных воздействий и входных переменных извлекаются из сценариев работы, поданных на вход методу машинного обучения. Данные объекты сохраняют свою суть и свойства и при преобразовании их в целевой формат *MDL*. Приведем пример описания события в этом формате:

```
event {
    id                8
    ssIdNumber        51
    name              "A"
    linkNode          [2 0 9]
    scope             INPUT_EVENT
    machine           1
}
```

Формат *MDL* предполагает закрепление на плоскости каждого выводимого на экран элемента. Поэтому наибольшую сложность при преобразовании построенной модели в формат *MDL* представляет собой укладка на плоскости ориентированного графа с пометками на ребрах – именно так представляется управляющий автомат. Главное требование к представлению полученного изображения автомата заключается в том, что по его внешнему виду пользователь (архитектор управляющей системы), должен понять его общую структуру. Это, в свою очередь, необходимо для того, чтобы пользователь мог внести какие-либо модификации в структуру автомата, или же промоделировать его работу на различных входных данных и сделать выводы о закономерностях, которые он наблюдает в процессе работы автомата.

Проблема изображения непланарных графов на плоскости также достаточно хорошо изучена, существуют различные технические решения этой проблемы. Одним из наиболее известных программных продуктов, реализующим наиболее качественные из этих решений, является программный продукт *Graphviz* – открытый пакет утилит для визуализации графов.

К сожалению, результат работы *Graphviz* на управляющих автоматах, полученных описанным выше методом, может оказаться неудовлетворительным с точки зрения наглядности управляющей системы. Одна из специфичных черт, которыми обладают полученные управляющие – наличие большого числа петель (ребер, ведущих из вершины в саму себя). В предлагаемых *Graphviz* вариантах расположения графов на плоскости пометки на ребрах-петлях перекрываются друг с другом (рис. 2). При этом расположение остальных вершин, ребер и пометок является вполне наглядным и может быть использовано для визуального представления автомата.

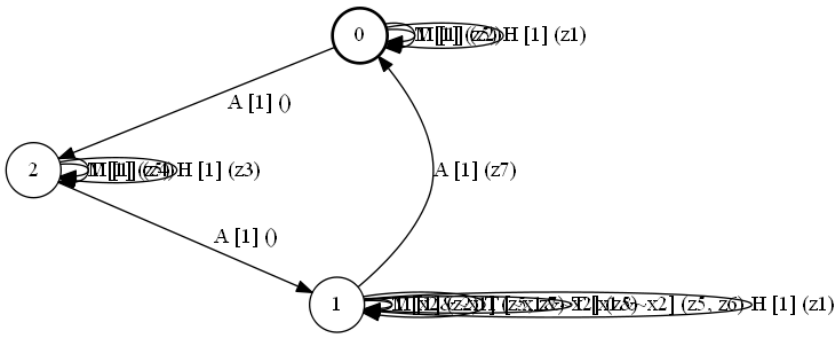

Рис. 2. Пример укладки автомата на плоскость, полученной в результате запуска программы *Graphviz.*

Формат данных *EPS*, являющийся выходным для *Graphviz*, является хорошо документированным форматом хранения изображений в векторном виде, вследствие чего он хорошо подается синтаксическому разбору. Итоговый алгоритм преобразования построенной автоматной модели системы управления из формата *DOT* в формат *MDL* выглядит следующим образом:

- конвертация множеств событий, входных переменных и выходных воздействий;
- построение изображения модели на плоскости с помощью *Graphviz*;
- распознавание и модификация построенного изображения в формате *EPS*: расположение ребер-петель для наглядного представления автомата;
- сохранение управляющего автомата в работающую модель *MATLAB/Simulink* (формат *MDL*) с указанием построенных координат для изображения на плоскости.

Описанный алгоритм был реализован на языке программирования *Java*. Откроем полученный в результате работы программы *MDL*-файл в среде *Stateflow*. Полученное представление управляющего автомата в среде приведено на рис. 3. Сравнив рис. 2 и рис. 3 несложно заметить, что проделанные преобразования повысили наглядность управляющего автомата.

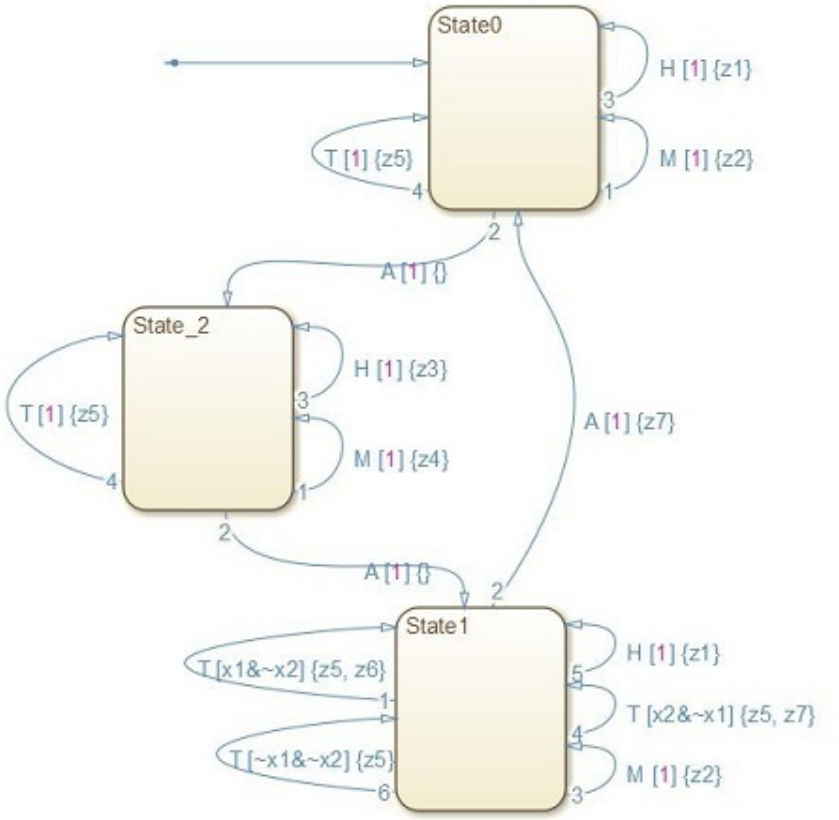

Рис. 3. Представление построенного управляющего автомата в среде *Stateflow* (фрагмент изображения экрана).

Разработанный метод возможно применять для преобразования построенных управляющих автоматов с большим числом состояний и переходов, граница применимости значительно превосходит границу применимости рассматриваемых методов машинного обучения (20-30 состояний). В качестве дополнительных примеров приведем изображения управляющих автоматов с пятью (рис. 4) и шестью (рис. 5) состояниями.

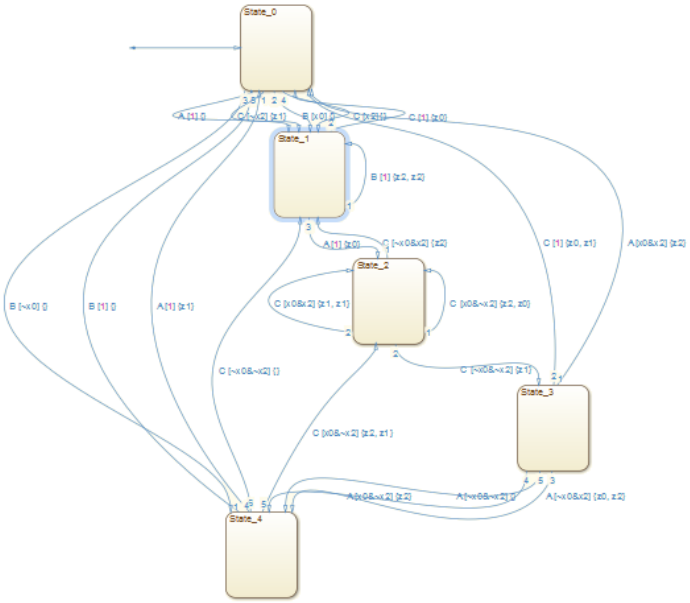

Рис. 4. Пример построенного управляющего автомата с пятью состояниями в среде *Stateflow* (фрагмент изображения экрана).

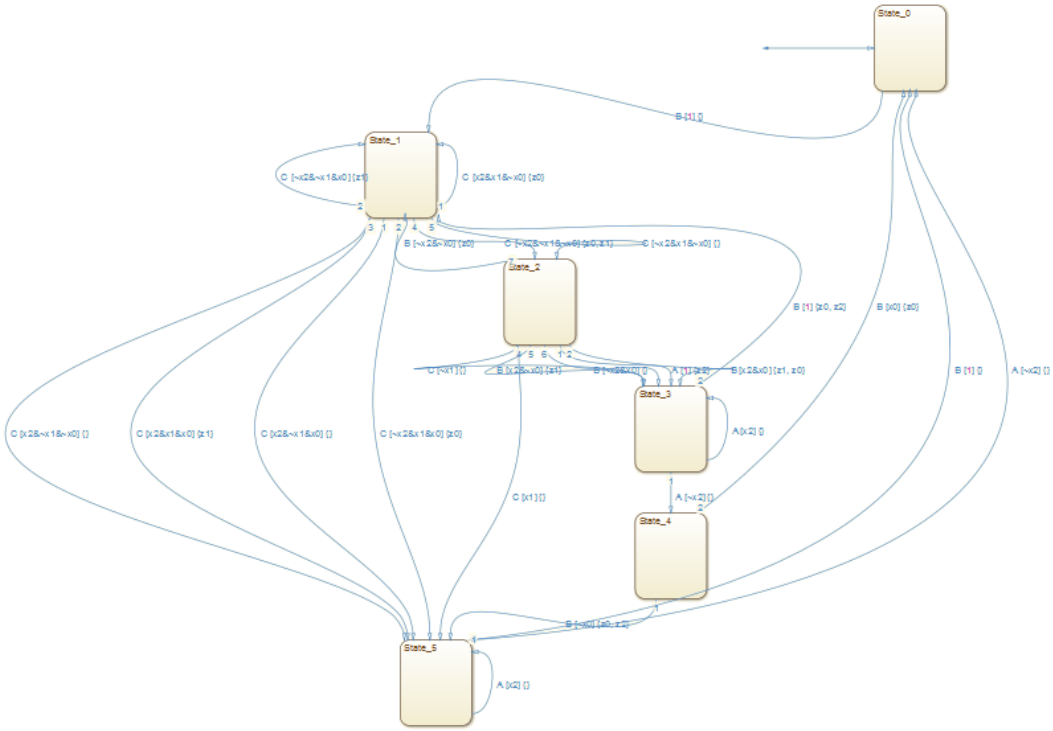

Рис. 5. Пример построенного управляющего автомата с шестью состояниями в среде *Stateflow* (фрагмент изображения экрана).

4. Заключение

В настоящей работе рассмотрена возможность применения методов машинного обучения для автоматизированного построения управляющих автоматов в высокоуровневых средствах проектирования систем. В качестве примера метода машинного обучения используется метод, описанный в [7], а в качестве высокоуровневого средства проектирования систем управления рассматривается среда *Stateflow*, входящая в *MATLAB/Simulink*. Разработан алгоритм, позволяющий загрузить полученный при помощи метода машинного обучения управляющий автомат в среду проектирования. Разумеется, архитектор системы управления волен выбирать для решения задачи как метод машинного обучения, так и средство проектирования.

Таким образом, автоматизацию процесса программирования целевой системы можно поднять на еще одну ступень – у архитектора имеется возможность взамен эвристического построения управляющего автомата задать примеры его поведения, а построенный автомат при желании модифицировать. В дальнейшем, к примеру, среда *Stateflow* предоставляет широкие возможности для моделирования условий, в которых управляющие автоматы будут находиться. Пользователь может поместить построенную модель системы управления в созданную им модель окружающей среды, после чего проверить работоспособность готовой модели при помощи моделирования и тестирования.

Предложенный подход также можно использовать для построения моделей существующих систем управления со сложным поведением. Для этого можно по существующей системе построить сценарии ее корректного поведения в различных ситуациях, после чего подать на вход указанному методу машинного обучения. После этого можно построенный управляющий автомат преобразовать при помощи разработанного в на-

стоящей работе алгоритма в формат *MDL*, открыть при помощи *Stateflow* и провести полноценное тестирование построенной модели.

Работа выполнена при государственной финансовой поддержке ведущих университетов Российской Федерации (субсидия 074-U01).

Список литературы

1. Поликарпова Н. И., Шалыто А. А. Автоматное программирование. 2-е изд. СПб.: Питер, 2011.
2. Вельдер С.Э., Лукин М.А., Шалыто А.А., Яминов Б.Р. Верификация автоматных программ. СПб: Наука, 2011. 242 с.
3. Янкин Ю.Ю., Шалыто А.А. Автоматное программирование ПЛИС в задачах управления электроприводом // Информационно-управляющие системы. 2011. № 1. С. 50- 56.
4. Александров А.В., Казаков С.В., Сергушичев А.А., Царев Ф.Н., Шалыто А.А. Применение эволюционного программирования на основе обучающих примеров для генерации конечных автоматов, управляющих объектами со сложным поведением // Известия РАН. Теория и системы управления. 2013. № 3. С. 85-100
5. Chivilikhin D., Ulyantsev V. Learning Finite-State Machines with Ant Colony Optimization // Lecture Notes in Computer Science. 2012. Vol. 7461. P. 268-275
6. Chivilikhin D., Ulyantsev V., Tsarev F. Test-Based Extended Finite-State Machines Induction with Evolutionary Algorithms and Ant Colony Optimization // Proceedings of the 2012 GECCO Conference Companion on Genetic and Evolutionary Computation. New York: ACM. 2012. P.. 603-606
7. Ulyantsev V., Tsarev F. Extended Finite-State Machine Induction using SAT-Solver // Proceedings of the Tenth International Conference on Machine Learning and Applications ICMLA '2011. Honolulu, HI, USA, 18-21 December 2011. IEEE Computer Society, 2011. Vol. 2. P. 346-349.