На правах рукописи

Лукин Михаил Андреевич

ВЕРИФИКАЦИЯ АВТОМАТНЫХ ПРОГРАММ

Специальность 05.13.11 – Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

АВТОРЕФЕРАТ

диссертации на соискание ученой степени
кандидата технических наук

Санкт-Петербург – 2014

Работа выполнена в Санкт-Петербургском национальном исследовательском университете информационных технологий, механики и оптики

Научный руководитель: Доктор технических наук, профессор
Парфенов Владимир Глебович

Официальные оппоненты: **Карпов Юрий Глебович**
доктор технических наук, профессор, заведующий кафедрой «Распределенные вычисления и компьютерные сети» Санкт-Петербургского государственного политехнического университета

Нарвский Андрей Сергеевич
кандидат технических наук, генеральный директор ООО «Эвелоперс»

Ведущая организация: Ярославский государственный университет им. П.Г. Демидова

Защита состоится 26 декабря 2014 г. в 14 часов на заседании диссертационного совета Д 212.227.06 при Санкт-Петербургском национальном исследовательском университете информационных технологий, механики и оптики по адресу: 197101, Санкт-Петербург, Кронверкский пр., д.49., конференц-зал центра интернет-образования.

С диссертацией можно ознакомиться в библиотеке Санкт-Петербургского национального исследовательского университета информационных технологий, механики и оптики по адресу: 197101, Санкт-Петербург, Кронверкский пр., д.49 и на сайте fppo.ifmo.ru.

Автореферат разослан « ____ » ноября 2014 года.

Ученый секретарь диссертационного совета
кандидат технических наук, доцент Лобанов Игорь Сергеевич

Общая характеристика работы

Актуальность

Автоматное программирование, основанное на применении конечных автоматов, всё шире используется при разработке программного обеспечения (ПО). В частности, оно применяется для построения программ управления ответственными объектами, для которых особую важность имеет правильность их работы. Обеспечение качества ПО достигается за счёт использования соответствующих методов проектирования программ, а также тестирования и верификации. Настоящая работа посвящена вопросу верификации автоматных программ на основе метода проверки моделей (*model checking*), который может быть автоматизирован.

Цель диссертационной работы

Цель работы – разработка метода верификации автоматных программ. Для достижения этой цели в диссертации решаются следующие задачи:

- выбор математических моделей управляющих автоматов;
- разработка алгоритма построения *Spin*-модели управляющего автомата;
- разработка алгоритма построения *Spin*-модели системы вложенных управляющих автоматов (иерархического автомата);
- разработка алгоритма построения *Spin*-модели параллельно работающих иерархических автоматов;
- разработка метода верификации автоматных программ: построение Spin-модели, преобразование расширенных LTL-формул в *Spin*-формулы, преобразование контрпримера в путь в системе автоматов;
- разработка подхода к верификации программ в случаях, когда модель внешней среды нельзя представить в виде автомата, например, для клетчатого поля неограниченных размеров;
- разработка инструментального средства, поддерживающего предлагаемый метод.

Научная новизна

В работе получены следующие научные результаты, которые выносятся на защиту:

- обоснован выбор математических моделей управляющих автоматов;
- разработаны алгоритмы построения *Spin*-моделей для одного автомата, системы вложенных автоматов (иерархического автомата) и системы параллельно работающих иерархических автоматов;
- разработан метод верификации автоматных программ: построение *Spin*-модели, преобразование *расширенных* LTL-формул в *Spin*-формулы, преобразование контрпримера в путь в системе автоматов;

- разработан подход для верификации программ в случаях, когда модель внешней среды нельзя представить в виде автомата, например, для клетчатого поля неограниченных размеров.

Методы исследования

В работе использованы методы дискретной математики, построения и анализа алгоритмов, теории автоматов, теории верификации, объектно-ориентированного программирования.

Достоверность

Достоверность научных положений, выводов и практических рекомендаций, полученных в диссертации, подтверждается корректным обоснованием постановок задач, точной формулировкой критериев, компьютерным моделированием, а также результатами использования методов, предложенных в диссертации, на практике.

Практическое значение

Практическое значение работы состоит в том, что предложенный метод обеспечивает возможность верификации программ, базирующихся на разных типах автоматов, используемых в промышленности.

На основе этого метода автором разработано инструментальное средство *Stater*, которое позволяет графически строить иерархические автоматы, импортировать их из сред для проектирования автоматных программ, генерировать по этим автоматам программный код и проводить верификацию автоматных программ.

Экспериментальные исследования

На примере задачи поиска пути с O(1) памяти на бесконечном клетчатом поле были выполнены экспериментальные исследования предложенного инструментального средства. Исследования проводились на 800 автоматных программах, созданных при помощи генетического программирования. Удалось доказать корректность работы для 231 программы (вопрос о корректности остальных программ остаётся открытым).

Использование и внедрение результатов работы

Результаты работы использованы в ходе выполнения государственного контракта по теме «Разработка технологии верификации управляющих программ со сложным поведением, построенных на основе автоматного подхода» (2007, http://is.ifmo.ru/verification/).

Результаты использованы также при проведении занятий по курсу «Верификация программного обеспечения» для магистрантов кафедры «Компьютерные технологии» в 2012/2013 и 2013/2014 учебных годах, что

подтверждается соответствующим актом. Для поддержки этого курса в 2012 году в Университете ИТМО было публиковано учебное пособие.

Кроме того, по курсу при участии автора был создан на русском и английском языках учебно-методический комплекс, опубликованный в системе дистанционного обучения Университета ИТМО. Акты использования имеются.

Результаты работы внедрены в 2012 году в ООО «Специальный технологический центр» (СТЦ) при разработке редактора описания антенных систем, что подтверждается актом внедрения.

Результаты также внедрены в 2012 году в студии «PointOmega Games» (Санкт-Петербург) при разработке игры Turtleball HD (https://play.google.com/store/apps/details?id=com.pointomega.turtleball), что также подтверждается актом внедрения.

Апробация диссертации

Основные положения диссертационной работы докладывались на следующих научных и научно-практических конференциях: Международная научно-методическая конференция «Высокие интеллектуальные технологии и инновации в образовании и науке» (СПбГПУ, 2008), Конференция молодых ученых Университета ИТМО, 2009, Всероссийская научная конференция по проблемам информатики СПИСОК (Матмех, СПбГУ, 2011), Международная научно-практическая конференция Tools & Methods of Program Analysis (Костромской государственный технический университет, TMPA-2013), IBM Haifa Verification Conference (Haifa, 2014).

Публикации

По результатам диссертации лично автором и в соавторстве опубликовано 11 научных работ, из которых монография (издательство «Наука»), учебное пособие Университета ИТМО (на русском и английском языках), три статьи (все в изданиях из перечня ВАК).

Кроме того, автором получено свидетельство о регистрации программ на описываемое ниже инструментальное средство *Converter*.

Структура диссертации

Диссертация изложена на 185 страницах и состоит из введения, пяти глав, заключения и двух приложений. Список литературы состоит из 87 наименований. Работа иллюстрирована 34 рисунками и содержит 4 таблицы.

Краткое содержание работы

В **первой главе** приведен обзор используемых понятий и математических объектов.

Во **второй главе** выполнен обзор существующих методов верификации автоматных программ.

Сравнение с аналогами приведено в таблице 1.

Таблица 1. Сравнение с аналогами

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1.	LTL	-	+	-	-	-	-	-
2.	LTL	-	+	-	-	-	-	-
3.	CTL	-	+	-	-	-	-	-
4.	CTL	-	+	-	-	-	-	-
5.	LTL	-	+	-	-	-	+	-
6.	LTL	-	+	-	-	-	+	-
7.	-	+	-	+	-	-	?	-
8.	-	+	-	+	-	-	?	-
9.	LTL	+	+	+	+	-	+	-
10.	CTL	+	-	-	-	+	-	+

В столбцах указаны свойства, поддерживаемые соответствующими методами: 1. Темпоральная логика. 2. Поддержка многопоточных программ. 3. Поддержка UML-диаграмм *Statechart*. 4. Поддержка автоматов *Stateflow*. 5. Интерактивность. 6. Возможность передавать события между параллельными автоматами. 7. Bounded Model Checking (ограниченная проверка моделей). 8. Поддержка автоматов стандарта *IEC 61499*.

В строках приведены известные инструментальные средства и методы верификации автоматных программ.

1. *Unimod.Verifier* (http://is.ifmo.ru/verification/_jaminov.pdf).
2. *Automata Verificator* (http://is.ifmo.ru/papers/automataverificator).
3. *FSM Verifier* (http://is.ifmo.ru/download/2008-02-25_politech_verification_kurb.pdf).
4. *CTL Verifier* (http://is.ifmo.ru/download/2008-02-25_politech_verification.pdf).
5. *Latella D., Majzik I., Massink M.* Automatic verification of a behavioral subset UML stetechart diagrams using the SPIN model-checker (http://home.mit.bme.hu/~majzik/publicat/fac99.pdf).
6. *Васильева К. А., Кузьмин Е. В.* Верификация автоматных программ с использованием LTL // Моделирование и анализ информационных систем. 2007. № 1, с. 3–14. (http://is.ifmo.ru/verification/_LTL_for_Spin.pdf).
7. *Chen C., Sun J., Liu Y., Dong J., Zheng M.* Formal Modeling and Validation of Stateflow Diagrams. (http://link.springer.com/article/10.1007%2Fs10009-012-0235-0#page-1).
8. *Miyazawa A., Cavalcanti A.* Refinement-based verification of sequential implementations of Stateflow charts. (https://www.academia.edu/3531758/Refinement-based_verification_of_sequential_implementations_of_Stateflow_charts).

9. *Pingree P. J., Mikk E., Holzmann G. J., Smith M. H., Dams D.* Validation of mission critical software design and implementation using model checking [spacecraft]. (http://ieeexplore.ieee.org/xpl/abstractAuthors.jsp%3Farnumber%3D1067982&rct=j&q=&esrc=s&sa=U&ei=OJ8hVPj_D8PmywORpIDQBg&ved=0CBMQFjAA&sig2=w8kMGk_oXI3Ptc6MFdmCNg&usg=AFQjCNHnIrnNTGE8BmOQVcZ3hQOvPXBQVw).
10. *Вяткин В. В., Дубинин В. Н.* Верификация приложений IEC 61499 на основе метода Model checking. (http://cyberleninka.ru/article/n/verifikatsiya-prilozheniy-iec-61499-na-osnove-metoda-model-checking).

В результате выполненного анализа установлено, что в настоящее время отсутствует метод верификации автоматных программ, модели которых предложены в третьей главе диссертации.

В **третьей главе** описан предлагаемый метод для разработки и верификации автоматных программ.

Описание автоматных моделей

В работе используется распределенная система взаимодействующих иерархических конечных автоматов. При этом каждый иерархический автомат работает в отдельном потоке.

1. Математическая модель управляющего автомата

Модель управляющего автомата – это кортеж $<S, s_0, T, E, X, Z, B, \delta, \lambda>$, где:

- S – конечное множество состояний;
- s_0 – начальное состояние;
- $T \subseteq S$ – множество допускающих состояний (возможно, пустое);
- E – множество входных воздействий (событий);
- X – множество переменных;
- Z – множество выходных воздействий;
- B – множество логических функций от переменных из множества X (условий на переменные из множества X);
- δ: $S \times E \times B \to S$ – отношение переходов;
- $\lambda = \lambda_1 \cup \lambda_2 \mid \lambda_1: S \times E \times B \to Z$, $\lambda_2: S \to Z$ – отношение выходных воздействий.

Каждое множество одинаковых автоматов назовем *автоматным типом*.

Множество Z состоит из двух множеств Z_1 и Z_2. Множество Z_1 – это множество выходных воздействий, которые состоят из имён вызываемых подпрограмм. Множество Z_2 состоит из выходных воздействий, которые состоят из произвольного программного кода. В модель на языке *Promela* этот код переносится без изменений.

Переменные из множества X могут иметь целочисленные типы (включая массивы).

Эти переменные могут иметь следующие модификаторы: *volatile* (переменная может быть использована в любом месте программы); *external* (переменная может быть использована другим автоматом); *param* (переменная является параметром автомата).

2. Математическая модель вложенных автоматов

Рассмотрим автомат типа A_i. Пусть S_{A_i} – множество его состояний. Пусть M – множество автоматов в системе. Определим отношение вложенности как множество отношений $\nu : \{\nu_i \mid \nu_i : S_{A_i} \to M\}$.

Отметим, что автомат может иметь вложенные автоматы любого типа, кроме собственного, так как иначе будет бесконечная рекурсия. Циклическая рекурсия также запрещена.

Иерархическим автоматом будем называть систему вложенных автоматов.

3. Математическая модель параллельно работающих автоматов

Модель параллельно работающих автоматов – это тройка $<\sigma, \rho, \omega>$, где

- $\sigma : \{\sigma_i \mid \sigma_i : S_{A_i} \to M\}$ – множество отношений, описывающих запуск автоматов ;
- $\rho = \{\rho_i \mid \rho_i : S_{A_i} \times E \times B \to M \times E\}$ – множество отношений, описывающих коммуникацию между автоматами;
- $\omega = \{\omega_{i,j} \mid \omega_{i,j} : X_i \to X_j\}$ – множество отношений, описывающих совместно используемые переменные.

Перейдём к рассмотрению *Spin*-моделей. В диссертации предложены *Spin*-модели для каждого из рассмотренных типов автоматов. На рисунке 1 приведена *Spin*-модель для параллельных иерархических автоматов.

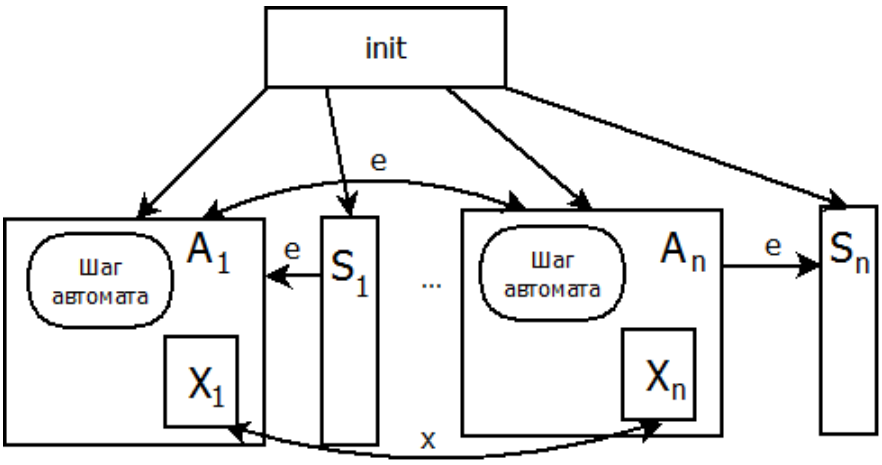

Рисунок 1. *Spin*-модель параллельных иерархических автоматов

Основные компоненты модели:

- A_i – процессы, содержащие автоматы. Каждый процесс содержит один автомат. Процесс состоит из цикла, который принимает события при помощи *каналов Spin* и запускает функцию, реализующую шаг автомата. Каждая функция реализует один автоматный тип и принимает на вход автомат и событие.
- *init* – процесс, запускающий процессы A_i.

- S_i – процессы-модели поставщиков событий.
- X_i – переменные автоматов.

Изложим предлагаемый **метод верификации автоматных программ**.

1. Построение *Spin*-модели при помощи предложенных в диссертации алгоритмов.
    1. Если внешняя среда может быть задана темпоральными формулами, то она задаётся в спецификации.
    2. Если внешняя среда может быть задана при помощи автоматов, то она добавляется в модель.
    3. Пример модели внешней среды, которую нельзя задать при помощи автоматов, рассматривается в главе 5.
2. **Расширение** LTL-формул следующими атомарными высказываниями:
    1. `machineName.exx` означает, что в автомат *machineName* пришло событие *exx*.
    2. `machineName.sxx` означает, что автомат *machineName* перешел в состояние *sxx*.
    3. `machineName.Function` означает, что автомат *machineName* вызвал выходное воздействие *Function*.
    4. `machineName->nestedMachine` означает, что автомат *machineName* дошел до вложенного автомата *nestedMachine*.
    5. `machineName||forkMachine` означает, что автомат *machineName* запустил автомат *forkMachine*.
3. Преобразование расширенных LTL-формул:
    1. Разбор формулы и выделение указанных выше атомарных высказываний.
    2. Создание макроса для каждого атомарного высказывания.
    3. Замена атомарного высказывания макросом.
4. Запуск верификатора *Spin*.
5. Преобразование контрпримера:
    1. Внесение *пометок* вывода на экран в код на языке *Promela*:

```
printf("machine%d.state = AManager.state0\n", machine.ID);
```

    2. Запускается парсер *Spin*-контрпримера.
    3. На выходе парсера получается текст контрпримера в терминах автоматов.

Инструментальные средства

Первоначально автором было разработано инструментальное средство *Conveter*[1], которое позволяет верифицировать автоматные программы, базирующиеся только на вложенных автоматах.

[1] http://is.ifmo.ru/diploma-theses/_lukin_master.pdf

В ходе написания диссертации функциональность этого средства была расширена на возможность верификации параллельных автоматных программ. Это инструментальное средство, названное *Stater*, поддерживает предложенный автором метод верификации и позволяет:

- задать графы переходов системы параллельных иерархических автоматов;
- импортировать эти графы из *StateFlow* и стандарта *IEC 61499*;
- верифицировать систему автоматов при помощи *Spin*;
- сгенерировать программный код на языке *C#* по верифицированной системе автоматов.

Графический пользовательский интерфейс *Stater* показан на рисунке 2.

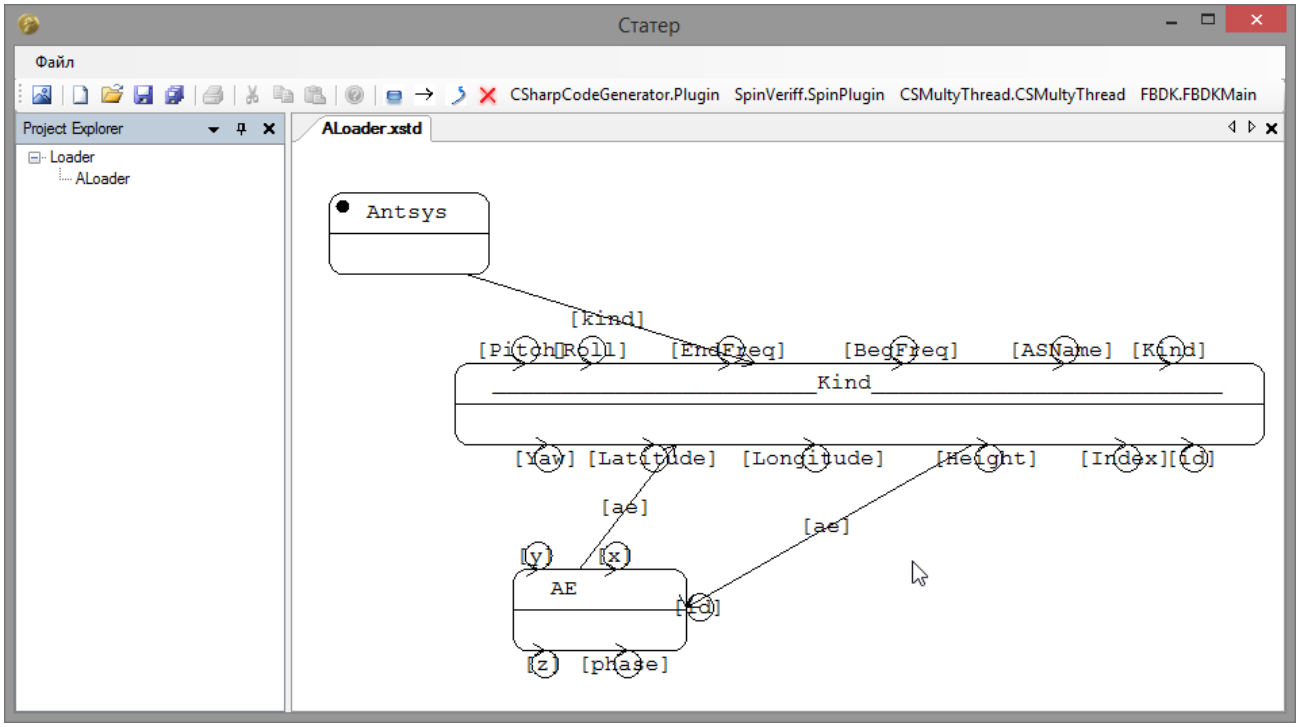

Рисунок 2. Графический пользовательский интерфейс инструментального средства *Stater*

Работа предложенного метода проиллюстрирована в диссертации примером верификации системы управляющих автоматов для гусеничного шасси.

Система состоит из трех автоматов, работающих параллельно. Два автомата *left* и *right* имеют тип *AEngine*, и подают команды на левый и правый двигатели соответственно. На рисунке 3 приведен автомат *AEngine*.

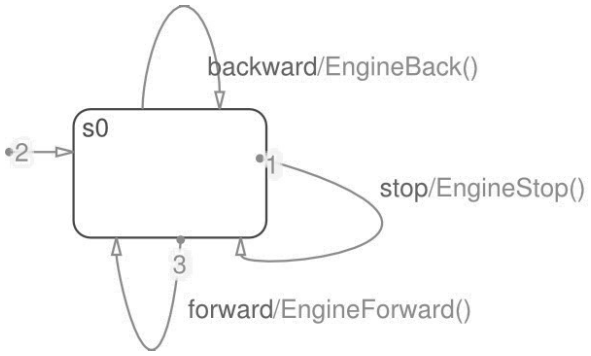

Рисунок 3. Автомат *AEngine*

На рисунке 4 приведен автомат *AManager*. Он получает команды от внешней среды (например, пользователя) и управляет автоматами *left* и *right*.

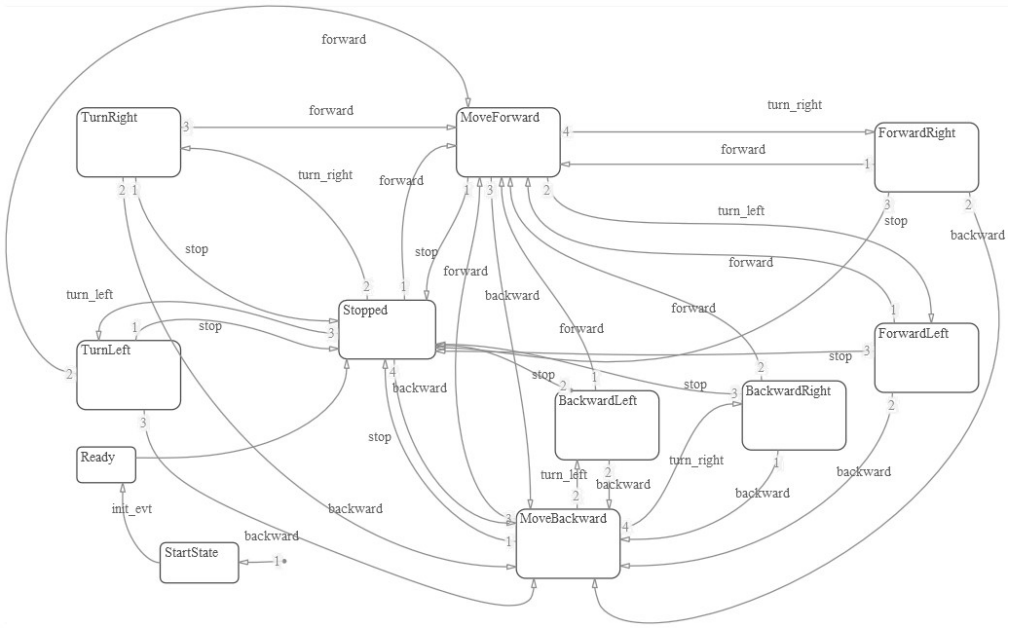

Рисунок 4. Автомат *AManager*

При входе в состояние автомат *AManager* отправляет соответствующее событие автоматам *AEngine*. Ниже приведёт список, включающий название состояния, названия автомата, которому посылается событие, название которого указывается справа от стрелки:

- `Stopped: left ← stop, right ← stop.`
- `MoveForward: left ← forward, right ← forward.`
- `MoveBackward: left ← backward, right ← backward.`
- `TurnRight: left ← backward, right ← forward.`
- `TurnLeft: left ← forward, right ← backward.`
- `ForwardRight: left ← stop, right ← forward.`
- `ForwardLeft: left ← forward, right ← stop.`
- `BackwardRight: left ← backward, right ← stop.`

- `BackwardLeft: left ← stop, right ← backward.`

В качестве примера в диссертации проводится верификация условия, которое в словесной форме записывается как: «в любой момент, если поступила команда «стоп», подается команда на остановку левого двигателя», а в виде темпоральной формулы –
G *(({manager.stop} && !{manager.StartState} && !{manager.Ready} && !{manager.Stopped}) →* ***(F*** *{left.EngineStop}))*

Четвёртая глава содержит описание внедрения полученных результатов на практике.

В рамках работы по государственному контракту[2] была проведена апробация инструментального средства *Converter* на модели банкомата. Были формально верифицированы как истинные свойства банкомата, так и построены контрпримеры для ложных.

Converter и *Stater* используются в учебном процессе. С 2012 года автором на кафедре «Компьютерные технологии» ведется курс по верификации программного обеспечения. В рамках его студенты провели верификацию автоматных программ, выложенных на сайте http://is.ifmo.ru. В одной из программ была найдена ошибка, несмотря на то, что она многократно проверялась до этого.

Инструментальное средство *Stater* было использовано в ООО «СТЦ» для верификации части программы *AntsysEditor*, позволяющей обрабатывать XML-файлы, содержащие информацию о структуре антенн, что подтверждается актом о внедрении.

Кроме того, при помощи *Starter* был использован при верификации логики в игре «Turtleball HD», что также подтверждается актом о внедрении.

В **пятой главе** описан подход к верификации программ, внешняя среда которых не представима в виде автоматов. Подход рассмотрен на примере внешней среды, представляющей собой клетчатое поле неограниченных размеров.

Он состоит из следующих шагов:

1. Построить гипотезу об алгоритме, который реализует программа.
2. Построить *Spin*-модель (возможно, неполную и недетерминированную) программы и внешней среды и множество темпоральных формул, которые вместе с моделью будут использованы верификатором *Spin* для доказательства, что данная программа соответствует гипотезе.
3. Формально доказать как теорему, что если программа соответствует гипотезе, то она корректна.
4. Запустить верификатор *Spin*, используя модель и формулы из шага 2. Программа, которая успешно прошла верификацию, корректна.

Проиллюстрируем подход на примере задачи поиска пути агентом с *O(1)* дополнительной памяти на клетчатом поле.

---

[2]Государственный контракт № 02.514.11.4048 «Разработка технологии верификации управляющих программ со сложным поведением, построенных на основе автоматного подхода».

Дано двумерное клетчатое поле **неограниченных размеров**. На некоторых клетках есть препятствия, занимающие всю клетку. Связная компонента из смежных препятствий должна быть конечной. Агент также имеет размер в одну клетку. Одна из свободных клеток объявляется целью. Агент должен найти путь к цели. В работе[3] было построено 800 автоматных программ для решения этой задачи при помощи генетического программирования, которое не гарантирует, что решения корректны. **Требуется определить, какие из этих автоматов корректные**. На рисунке 5 приведен пример такого поля.

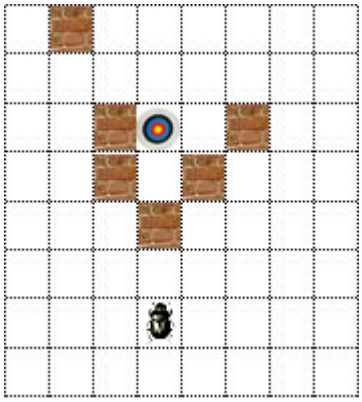

Рисунок 5. Пример поля для задачи о поиске пути («жук» – агент, «мишень» – цель, «кирпичи» – препятствия)

Входные логические переменные агента: «can move forward», «is move forward cool», «is at finish», «is at saved», «is better than saved».

Выходные воздействия агента:

- «move forward»: сделать шаг вперёд на соседнюю клетку;
- «rotate positive»: повернуть на 90 градусов по часовой стрелке;
- «rotate negative»: повернуть на 90 градусов против часовой стрелки;
- «report reached»: завершить работу и вернуть ответ о том, что цель достигнута;
- «report unreachable»: завершить работу и вернуть ответ о том, что цель недостижима;
- «save position»: сохранить текущие координаты и направление в память;
- «do nothing»: ничего не делать.

Основная сложность верификации состоит в том, что внешняя для агента среда не представима в виде автоматов.

Шаг 1. Гипотеза. По результатам тестовых запусков построенных автоматных программ была сформулирована гипотеза, что агенты реализуют модификацию алгоритма *BUG-2*[4] с учетом того, что поле клетчатое.

---

[3] Buzdalov, M., Sokolov, A.: Evolving EFSMs Solving a Path-Planning Problem by Genetic Programming. In: Proceedings of Genetic and Evolutionary Computation Conference Companion. pp. 591–594 (2012)

[4] *Lumelsky V., Stepanov A.* “Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape,” Algorithmica, vol. 2, pp. 403–430, 1987.

Основная идея алгоритма *BUG-2* состоит в следующем. Агенты двигаются по кратчайшей линии к цели, пока это возможно. Когда агент упирается в препятствие, имеются два варианта. Если возможно повернуть и продолжить двигаться по кратчайшей линии к цели (не натыкаясь на препятствия), то агент может это сделать. Кроме того, он может перейти в режим обхода препятствия. Когда можно продолжить движение к цели, агент это делает. Условие, при котором можно продолжить движение к цели: агент ближе к цели, чем точка соприкосновения с препятствием и можно идти в сторону цели.

Шаг 2. Модель агента и поля. Модель состоит из следующих компонентов: информация о поле; пересчёт состояния; модель управляющего автомата, которая была построена при помощи инструментального средства *Stater*; информация о наличии препятствий в восьми соседних клетках; направление агента (вверх, вниз, вправо или влево); пеленг на цель; относительное положение текущего состояния агента по отношению к сохранённому состоянию.

«Проблемным» действием является движение вперед, так как при движении вперед информация о клетках позади автомата забывается, а информация о клетках впереди агента недетерминированно генерируется. Кроме того, когда в модели недостаточно информации о направлении на цель для того, чтобы вычислить это после движения вперед, недетерминированно выбирается один из возможных вариантов направления на цель. Эта модель обладает недостатками, которые не позволяют верифицировать построенные программы напрямую. Перечислим их:

- Изменяющееся поле. Если агент вернулся в клетку, в которой уже побывал, поле могло измениться, так как информация о поле была сгенерирована заново.
- Бесконечно большие препятствия. Такие препятствия запрещены условием задачи (алгоритмы семейства *BUG* не работают в случае бесконечно больших препятствий). В этом случае агент может работать бесконечно долго.
- Бесконечно далёкая цель. В модели невозможно хранить расстояние до цели, так как оно может быть сколь угодно большим.
- «Блуждающая цель». Так как в некоторых случаях информация о направлении на цель перечитывается недетерминированно, то цель в построенной модели может изменять свое местоположение.
- «Блуждающая сохраненная клетка». Аналогично вышеизложенному, координаты сохраненной агентом клетки не хранятся в модели, и она тоже может менять свое местоположение.

Была составлена 31 LTL-формула[5], учитывающая недостатки модели. Эти формулы полностью «покрыли» сформулированную гипотезу.

Шаг 3. Доказательство. Было доказано, что каждая автоматная программа, удовлетворяющая спецификации, всегда работает конечное время и решает задачу.

[5] Формулы и доказательство можно посмотреть по ссылкам: http://goo.gl/KQc7lH и http://goo.gl/JZ4Nlf

Шаг 4. По результатам верификации из 800 автоматов 231 удовлетворяет спецификации. Таким образом, для 231 автомата удалось доказать корректность работы. Вопрос о корректности остальных автоматов остается открытым. Возможно, для доказательства их корректности требуется строить другие гипотезы.

Статья[6], описывающая вышеизложенный подход, была принята на конференцию *IX IBM Haifa Verification Conference* 2014. При этом отметим, что автору не удалось обнаружить работы, посвященные верификации программ для случая, когда внешняя среда является бесконечным клетчатым полем.

Заключение

В работе получены следующие научные результаты:

- обоснован выбор математических моделей управляющих автоматов;
- разработаны алгоритмы построения *Spin*-моделей для одного автомата, системы вложенных автоматов (иерархического автомата) и системы параллельно работающих иерархических автоматов;
- разработан метод верификации автоматных программ: построение *Spin*-модели, преобразование *расширенных* LTL-формул в *Spin*-формулы, преобразование контрпримера в путь в системе автоматов;
- разработан подход для верификации программ в случаях, когда модель внешней среды нельзя представить в виде автомата, например, для клетчатого поля неограниченных размеров.

Также автором разработаны инструментальные средства для верификации автоматных программ разных типов. Результаты работы внедрены в практику разработки ПО и в учебный процесс на кафедре «Компьютерные технологии» Университета ИТМО.

Статьи в журналах из перечня ВАК

1. *Лукин М. А., Шалыто А. А.* Верификация автоматных программ с использованием верификатора Spin // Научно-технический вестник СПбГУ ИТМО. 2008. Вып. 53. Автоматное программирование, С. 145–162. 1,05/0,55 п. л.
2. *Лукин М. А., Шалыто А. А.* Разработка и автоматическая верификация распределенных автоматных программ // Информационно-управляющие системы. 2013. № 5 (66). С. 43 – 50. 0,5/0,25 п. л.
3. *Лукин М. А.* Верификация параллельных автоматных программ // Научно-технический вестник СПбГУ ИТМО. 2014. № 1 (89). С. 145–162. 0,375 п. л.

[6] *Lukin M. A, Buzdalov M. V, Shalyto A. A.* Formal Verification of 800 Genetically Constructed Automata Programs: A Case Study / Proceedings of the 10th IBM Haifa Verification Conference, 2014.

Монографии

4. *Вельдер С.Э., Лукин М.А., Шалыто А.А., Яминов Б.Р.* Верификация автоматных программ. СПб.: Наука, 2011. 244 с. 15,25/3,8125 п. л.

Учебные пособия

5. *Вельдер С.Э., Лукин М.А., Шалыто А.А., Яминов Б.Р.* Верификация автоматных программ. Учебное пособие. СПб.: НИУ ИТМО. 2011. 242 с. 15,125/3,7825 п. л.
6. *Lukin M., Velder S., Shalyto A., Yaminov B*. Verification of automata-based programs. Монография на английском (рукопись). СПб.: НИУ ИТМО. 2013. 104 с. 6,5/1,625 п. л.

Сборники трудов

7. *Лукин М. А., Шалыто А. А.* Автоматизация верификации визуальных автоматных программ / Сборник трудов XV Международной научно-методической конференции «Высокие интеллектуальные технологии и инновации в образовании и науке». СПбГПУ. 2008. С. 296—297. 0,125/0,0625 п. л.
8. *Лукин М. А.* Верификация визуальных автоматных программ с использованием инструментального средства Spin / Сборник трудов Всероссийской межвузовской конференции молодых ученых 2009. Секция 6, подсекция А. СПб.: НИУ ИТМО. 2009. 0,125/0,0625 п. л.
9. *Лукин М. А.* Разработка и верификация многопоточных автоматных программ / Сборник трудов конференции СПИСОК-2011. СПбГУ. 0,125 п. л.
10. *Лукин М. А., Шалыто А. А.* Верификация распределенных автоматных программ с использованием инструментального средства Spin / Сборник трудов международной конференции «Tools and Methods of Program Analysis» (TMPA). 2013. С. 78—93. 0,875/0,4375 п. л.
11. *Lukin M. A, Buzdalov M. V, Shalyto A. A.* Formal Verification of 800 Genetically Constructed Automata Programs: A Case Study / Proceedings of the 10th IBM Haifa Verification Conference. 2014. LNCS 8855. Pp. 165—170. 0,375/0,25 п. л.

Личный вклад. В работах, выполненных в соавторстве, личный вклад автора в равных долях с соавторами.

Тиражирование и брошюровка выполнены в учреждении
«Университетские телекоммуникации»
197101, Санкт-Петербург, Саблинская ул., 14
Тел. (812) 233 46 69
Объем 1,0 у.п.л. Тираж 100 экз.