

Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,
МЕХАНИКИ И ОПТИКИ”

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

*РАЗРАБОТКА МЕТОДОВ ФОРМАЛЬНОЙ ВЕРИФИКАЦИИ
ПРОМЫШЛЕННЫХ КИБЕР-ФИЗИЧЕСКИХ СИСТЕМ В
ЗАМКНУТОМ ЦИКЛЕ*

Автор Овсянникова Полина Александровна _____

Направление подготовки 01.04.02 Прикладная математика и информатика _____

Квалификация _____ Магистр _____

Руководитель Ульянцев В. И., доцент, к.т.н. _____

К защите допустить

Зав. кафедрой КТ Васильев В.Н., проф., д.т.н. _____

“ ” _____ 2018 г.

Санкт-Петербург, 2018 г.

Студент Овсянникова П.А. Группа М4236с Кафедра КТ Факультет ИТиП

Направленность (профиль), специализация

Технологии проектирования и разработки программного обеспечения

Консультант(ы):

а) Вяткин В.В., проф., д.т.н.

ВКР принята « » 2018 г.

Оригинальность ВКР	%
100%	100%

ВКР выполнена с оценкой

Дата защиты « » июня 2018 г.

Секретарь ГЭК *Павлова О.Н.*

Листов хранения _____

Демонстрационных материалов/Чертежей хранения отсутствуют

Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,
МЕХАНИКИ И ОПТИКИ”

УТВЕРЖДАЮ

Зав. кафедрой *КТ* _____

проф. Васильев В. Н. _____

_____ 2018 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Студенту *Овсянниковой П.А.* _____ Группа *М423бс* Кафедра *КТ* Факультет *ИТиП*

Руководитель *Ульянцев В.И., к.т.н., Университет ИТМО, доцент каф. КТ* _____

1 Наименование темы: *Разработка методов формальной верификации промышленных кибер-физических систем в замкнутом цикле* _____

Направление подготовки (специальность) *01.04.02 Прикладная математика и информатика* _____

Квалификация _____ *Магистр* _____

2. Срок сдачи студентом законченной работы _____ *15 мая 2018 г.* _____

3. Техническое задание и исходные данные к диссертации

Требуется разработать метод формальной верификации кибер-физических систем, включающий в себя автоматическую генерацию формальной модели объекта управления в виде автомата с использованием подхода активного обучения. Должна быть предусмотрена возможность построения моделей объектов управления, обладающих вещественными переменными.

4 Содержание выпускной работы (перечень подлежащих разработке вопросов)

- а) Изучение предметной области.*
- б) Обоснование необходимости разработки нового метода.*
- в) Разработка и реализация метода построения формальной модели объекта управления.*
- г) Тестирование разработанного метода на примере кибер-физической системы «лифт».*

5 Перечень графического материала (с указанием обязательного материала)

Графические материалы и чертежи работой не предусмотрены

6 Исходные материалы и пособия

1. E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT press, 1999.
2. I. Buzhinsky and V. Vyatkin, "Automatic inference of finite-state plant models from traces and temporal properties," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, 2017.
3. D. Angluin, "Queries and concept learning," *Machine Learning*, vol. 2, no. 4, Apr 1988.

7 Дата выдачи задания «21» «ноября» 2016г.

Руководитель _____

Задание принял к исполнению _____

«21» «ноября» 2016г.

Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,
МЕХАНИКИ И ОПТИКИ”

АННОТАЦИЯ

ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Студента Овсянникова Полина Александровна

Наименование темы ВКР: Разработка методов формальной верификации промышленных кибер-физических систем в замкнутом цикле

Наименование организации, где выполнена ВКР Университет ИТМО

ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

1 Цель исследования разработка метода формальной верификации промышленных кибер-физических систем в замкнутом цикле на основе автоматического синтеза формальной модели объекта управления с помощью активного обучения

2 Задачи, решаемые в ВКР

а) Разработать алгоритм активного обучения бесконтекстных детерминированных объектов управления.

б) Обеспечить поддержку вещественных переменных.

в) Реализовать и протестировать разработанный алгоритм на симуляционной модели кибер-физической системы.

3 Число источников, использованных при составлении обзора 23

4 Полное число источников, использованных в работе 30

5 В том числе источников по годам

Отечественных			Иностранных		
Последние 5 лет	От 5 до 10 лет	Более 10 лет	Последние 5 лет	От 5 до 10 лет	Более 10 лет
0	0	0	7	10	10

6 Использование информационных ресурсов Internet да, 3

7 Использование современных пакетов компьютерных программ и технологий

Пакеты компьютерных программ и технологий	Параграф работы
<i>Java – для реализации алгоритма построения модели</i>	<i>Глава 3</i>
<i>NxtStudio – среды для симуляционного моделирования</i>	<i>Глава 3</i>
<i>NuSMV – символьный верификатор</i>	<i>Глава 3</i>

8 Краткая характеристика полученных результатов

Был разработан метод формальной верификации промышленных кибер-физических систем, включающий в себя автоматическую генерацию моделей объектов управления с помощью активного обучения. Полученный метод был успешно протестирован на симуляционной модели промышленной кибер-физической системы.

9 Полученные гранты, при выполнении работы

Грантов или других форм государственной поддержки и субсидирования в процессе работы не предусматривалось

10 Наличие публикаций и выступлений на конференциях по теме выпускной работы да

а) Овсянникова П. Разработка метода автоматической генерации формальных моделей кибер-физических систем на основе активного обучения // Сборник тезисов докладов конгресса молодых ученых. Электронное издание. — 2018.

б) Ovsiannikova P. et al. Closed-loop verification of a compensating group drive model using synthesized formal plant model // 22nd IEEE International Conference on Emerging Technologies and Factory Automation. — IEEE, 2017. — С. 1-4.

Студент Овсянникова П.А.

Руководитель Ульянцев В.И.

“ ” 2018 г.

ОГЛАВЛЕНИЕ

ОГЛАВЛЕНИЕ	4
ВВЕДЕНИЕ	6
ГЛАВА 1. КИБЕР-ФИЗИЧЕСКИЕ СИСТЕМЫ И ИХ ВЕРИФИКАЦИЯ	8
1.1 Кибер-физические системы.....	8
1.2 Проверка КФС на соответствие требованиям.....	9
1.2.1 Логика линейного времени и логика деревьев вычислений	9
1.2.2 Тестирование.....	10
1.2.3 Верификация	11
1.3 Построение формальной модели объекта управления.....	13
1.3.1 Объект управления	13
1.3.1.1 Зависимость поведения ОУ от текущего состояния	13
1.3.1.2 Симуляционная модель	14
1.3.1.3 Черный ящик.....	14
1.3.1.4 Формальное представление объекта управления	15
1.3.1.5 Дискретизация вещественных переменных	15
1.4 Автоматическая генерация формальных моделей ОУ.....	16
1.4.1 Генерация моделей ОУ на основе пассивного обучения	16
1.4.1.2 Метод, основанный на ограничениях и метод явных состояний.....	17
1.4.1.1 Проверка полноты сгенерированной формальной модели.....	19
1.4.3 Генерация конечно-автоматных моделей на основе активного обучения	19
1.4.3.1 Алгоритм L*	21
ВЫВОДЫ ПО ГЛАВЕ 1	25
ГЛАВА 2. ПРЕДЛАГАЕМЫЙ МЕТОД.....	26
2.1 Метод генерации формальной модели ОУ на основе поиска в ширину.....	26
2.1.1 Основная идея	26

2.1.2 Взаимодействие с симуляционной моделью ОУ	27
2.1.3 Обработка вещественных переменных	28
2.1.3.1 Проблема обратных петель	29
2.1.3.2 Проблема циклов.....	31
2.2 Реализация	34
2.3 Примечания	37
ВЫВОДЫ ПО ГЛАВЕ 2	38
ГЛАВА 3. ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ	39
3.1 Экспериментальная система	39
3.2 Методы, участвующие в сравнении	41
3.2.1 Алгоритм активного обучения L^*	41
3.2.2 Алгоритмы пассивного обучения	43
3.4 Результаты	44
3.5 Обсуждение результатов.....	49
ВЫВОДЫ ПО ГЛАВЕ 3	52
ЗАКЛЮЧЕНИЕ	53
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	55

ВВЕДЕНИЕ

В данной работе речь пойдет о кибер-физических системах (КФС) и способе проверки корректности их работы, что особенно критично в системах промышленной автоматизации. Тенденция развития и усложнения касается в том числе и промышленных КФС, в связи с чем, традиционные методы тестирования уже не всегда позволяют проверить все пространство состояний системы на соответствие требованиям. Также следует принимать во внимание, что количество и качество выпускаемого товара напрямую зависят от правильности работы автоматизирующих структур. Более того, при совмещении машинного и человеческого труда, вероятность выхода того или иного прибора из строя должна быть сведена к минимуму, так как любая неисправность может стоить не только денег, но и человеческих жизней. Следовательно, разработка методов формальной верификации КФС, позволяющих проверить все пространство состояний системы, является актуальной.

Целью данной работы является создание метода формальной верификации промышленных кибер-физических систем в замкнутом цикле, основанного на автоматической генерации формальной модели объекта управления (ОУ) с использованием подхода активного обучения. Автоматическая генерация позволит получить формальную модель, избегая ошибок, связанных с человеческим фактором при построении модели вручную, а верификация в замкнутом цикле сделает доступной проверку сложных систем. Предлагаемый метод состоит из трех этапов: (1) получение формальной модели ОУ, (2) получение формальной модели КФС путем соединения формальных моделей ОУ и контроллера, и (3) проверка соответствия системы требованиям, заданным формализмами линейной темпоральной логики (LTL), с помощью одного из формальных верификаторов, таких как SPIN [1] или NuSMV [2].

Работа структурирована следующим образом. В главе 1 определяется предметная область, рассматриваются основные понятия, относящиеся к КФС и их проверке. Также рассматриваются основные свойства, которыми должны обладать

и обычно обладают ОУ для возможности автоматической генерации моделей по ним указанными методами, также приводится обзор существующих подходов к автоматической генерации моделей ОУ, классифицированных по возможности взаимодействия с ОУ во время процесса построения модели. Глава 2 содержит подробное описание предлагаемого метода верификации формальной модели КФС, в которой ОУ соответствует свойствам, описанным в главе 1. Глава 3 описывает систему, для которой проводилось экспериментальное исследование предложенного метода. Также здесь приводятся результаты сравнения разработанного метода с существующими.

ГЛАВА 1. КИБЕР-ФИЗИЧЕСКИЕ СИСТЕМЫ И ИХ ВЕРИФИКАЦИЯ

Данная глава содержит основные понятия области верификации кибер-физических систем, описаны способы проверки КФС на соответствие заявленным при проектировании требованиям.

1.1 Кибер-физические системы

КФС – это система, которая сочетает в себе физический и программный компонент, где первый является объектом управления, а второй – управляющим устройством или контроллером, их взаимодействие показано на рисунке 1. Здесь видно, что выходные сигналы контроллера являются входными для ОУ, а выходные сигналы ОУ – входными для контроллера. Также может быть введен какой-то внешний модуль, сигналы которого также учитываются контроллером при генерации выходных сигналов.

Взаимодействие между контроллером и ОУ осуществляется циклически. За один цикл контроллер считывает выходные данные ОУ и какого-либо внешнего модуля, если он присутствует, и генерирует на их основе входные сигналы ОУ.

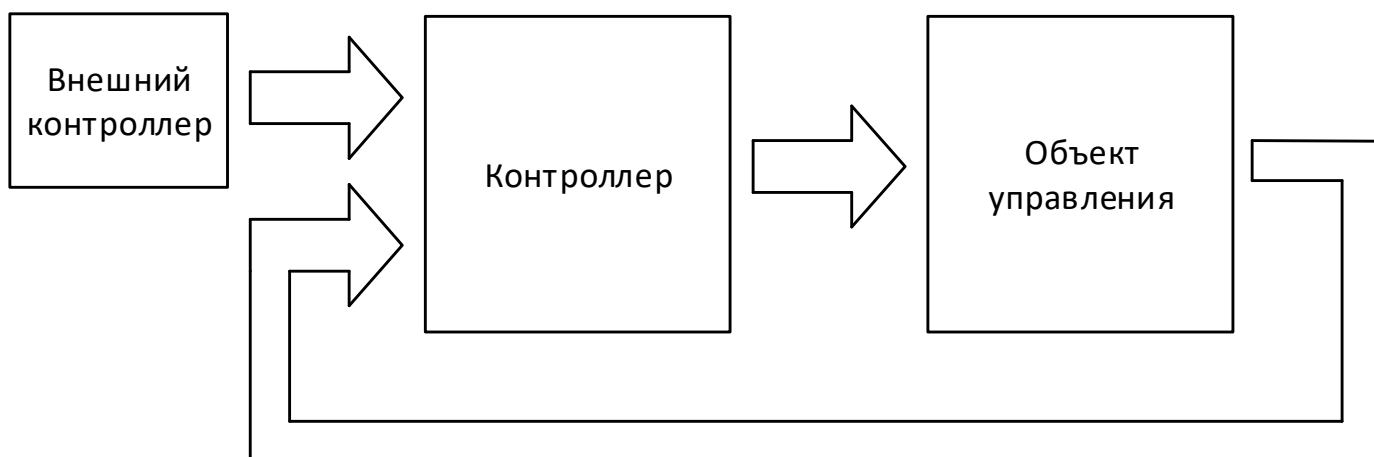


Рисунок 1 – Структура кибер-физической системы. Выходы ОУ замкнуты на входы контроллера

Примерами КФС являются: модули самолета, лифты, системы в составе ядерных реакторов, программируемые фрезерные станки, системы промышленной

автоматики, и многое другое. В данный момент с запуском проекта Industry 4.0 [4] по введению использования КФС в различные области жизни человека набирает популярность промышленная автоматизация, и на предприятиях, если это возможно, человеческий труд заменяется машинным. Существуют полностью и частично автоматизированные предприятия, производства же, на которых не задействована ни одна КФС, найти сложно.

1.2 Проверка КФС на соответствие требованиям

Несмотря на то, что речь идет о повсеместной автоматизации, проектированием большинства КФС все еще занимается человек, а значит, любая ответственная система еще на этапе проектирования должна быть подвергнута тщательному анализу и проверке. Для этих целей чаще всего применяются два подхода: тестирование и верификация, которые будут описаны в данном разделе. В то время как тестирование позволяет проверить только ограниченное число сценариев работы КФС, с помощью формальной верификации, а конкретно, метода проверки моделей [3], может быть рассмотрено все пространство состояний модели. Также в разделе 1.2.1 будет дано определение темпоральных логик линейного времени (LTL) и деревьев вычислений (CTL), формализмы которых удобно использовать для задания спецификаций системы при формальной верификации, и которые будут использоваться в данной работе.

1.2.1 Логика линейного времени и логика деревьев вычислений

В качестве описания требований к системе удобно использовать формализмы темпоральных логик LTL и CTL, высказывания которых в своем составе имеют темпоральные операторы, то есть учитывают временной аспект. LTL описывает свойства линейного времени системы и содержит следующие темпоральные (временные) операторы:

- $\varphi_1 \text{ U } \varphi_2$ – Until – φ_1 должно быть истинно до тех пор, пока истинно φ_2 , после чего значение φ_1 может быть любым;

- $G \varphi$ – **G**lobally – φ должно выполняться на всем пространстве путей системы, начиная с текущего состояния;
- $X \varphi$ – **N**ext – в следующем состоянии φ должно быть истинно;
- $\varphi_1 R \varphi_2$ – **R**elease – φ_2 должно быть истинным до тех пор, пока φ_1 не станет истинным;
- $F \varphi$ – **F**uture – когда-либо в будущем φ обязательно должно стать истинным.

В дополнение к вышеуказанным, формулы CTL также дополняются следующими кванторами:

- A – какое-либо свойство выполняется на всех путях;
- E – существует хотя бы один путь, на котором выполняется свойство.

Каждое высказывание логики CTL содержит какой-либо из вышеуказанных кванторов перед каждым временным оператором. Основное отличие LTL от CTL в том, что в CTL рассматривается каждый путь в отдельности, тогда как в LTL рассматриваются вообще все пути.

1.2.2 Тестирование

Одним из способов проверки корректности системы является тестирование. Так как разработка тестов вручную – долгий и дорогой процесс, то для сложных система чаще всего применяется автоматическая генерация тестов. Так, например, в работах [5] и [6] используется случайная генерация тестов. Первый метод [5] является самым простым в плане реализации и применим для моделей с закрытым исходным кодом. Второй метод [6] адаптивного случайного тестирования учитывает, что входные данные, приводящие к ошибке, обычно сгруппированы, а также решает проблему возможного неравномерного тестового покрытия первого метода. Недостаток обоих этих алгоритмов заключается в отсутствии автоматического определения того, является ли результат прохождения системой теста верным. Тем временем, это является важным моментом, и подход, называемый *тестированием на основе модели* [7], позволяет получить эту

информацию. Данный метод, в отличие от первых двух, использует стратегию тестирования «белого ящика», то есть предназначен для тестирования моделей, детали реализации которых доступны. Формальная модель также используется в работе [8], где и требование к системе и сама система представляются в виде конечных автоматов. В [9] разработана теория систем с помеченными переходами. Здесь также рассмотрены конечные автоматы, а также соотношение $ioco$ (соответствие входов и выходов) между реализацией и требованиями к системе. Таким образом, доступна генерация наборов тестов, с помощью которых можно проверить, соответствует ли система ее определению. В [10] используются системы символьных переходов, которые генерируются из спецификаций, представляемых в виде последовательных функциональных диаграмм.

Несмотря на разнообразие существующих методов тестирования, все они имеют общий недостаток: невозможно гарантировать проверку всего пространства состояний системы, что необходимо в случае применения к ответственным КФС.

1.2.3 Верификация

Широкое распространение в области проверки правильности поведения КФС получила верификация методом проверки моделей, так как, используя ту или иную формальную модель системы, можно охватить все возможные сценарии ее поведения. В работе [11] с помощью функционального анализа определяется частота появления максимальной задержки в распределенной системе. Далее, с помощью проверки моделей выявляются связи между проверяемой частью системы, способом организации входящих сообщений и частотой максимальных задержек. Работа [12] рассматривает проблему обнаружения неточностей в системах промышленной автоматизации, предлагаемое решение заключается в следующем. Во-первых, в формате AutomationML [13] строится топология системы. Далее выполняется автоматическая генерация компонентов (объектов управления) системы в виде вероятностных детерминированных временных автоматов и их встраивание в созданную топологию. Затем происходит

параллельный запуск симуляционной и формальной моделей системы с одинаковыми входными данными. Если в выходных данных при этом существуют различия, то обнаружена аномалия.

Также методы верификации могут быть классифицированы по замкнутости используемой формальной модели. Так, можно выделить *верификацию в замкнутом цикле* и *верификацию в открытом цикле*. В работе [14] эти два подхода сравниваются на примере КФС передвигающего детали манипулятора, делается вывод о взаимодополняемости данных подходов.

Вид верификации КФС, в котором рассматривается только модель контроллера, называется верификацией в открытом цикле. Так как в процессе верификации перебираются все возможные комбинации входных переменных, такой подход применим для несложных систем с небольшим числом входов и выходов из-за возможности столкнуться с проблемой взрыва числа состояний в противном случае. Также при применении такого подхода можно столкнуться с тем, что некоторые свойства ОУ не смогут быть проверены, в то время как другие свойства могут иметь ложный отрицательный результат верификации из-за невозможного в реальной системе контрпримера.

Другой подход называется верификацией в замкнутом цикле [15-17]. В этом случае формальная модель контроллера взаимодействует с формальной моделью объекта управления (ОУ), а значит, число возможных состояний системы сокращается, следовательно, становится доступной верификация более сложных систем. Однако такой сценарий подразумевает наличие также и формальной модели ОУ. Так как ОУ, как правило, представляет физический компонент и рассматривается как «черный ящик», создание его формальной модели нетривиально и практически никогда не осуществляется в процессе разработки. Также такой подход к верификации может занимать больше времени в связи с размером результирующей модели, однако позволяет проверять свойства, связанные не только с контроллером, но и с ОУ.

Подводя итоги, предоставляя возможность проверить систему более тщательно, формальная верификация все же менее популярна, чем тестирование.

Одним из ее недостатков является необходимость разработки формальной модели системы. Зная, что контроллер чаще всего представляет собой программу, получить его формальную модель относительно просто, существуют работы, описывающие этот процесс [18, 19]. Напротив, формальная модель ОУ из-за высоких затрат создается редко, и еще реже поддерживается в актуальном состоянии.

1.3 Построение формальной модели объекта управления

Автоматизированное создание формальной модели ОУ является первым этапом предлагаемого метода. Существуют способы ручной и автоматической генерации моделей ОУ, в данной работе делается акцент именно на автоматическую генерацию, так как в этом случае число ошибок, связанных с человеческим фактором, сводится к минимуму.

1.3.1 Объект управления

В данном разделе приводятся основные понятия, связанные с ОУ, определяются начальные условия для алгоритмов автоматической генерации формальных моделей, обосновывается выбор формального представления результирующей модели.

1.3.1.1 Зависимость поведения ОУ от текущего состояния

В КФС объектом управления является физический компонент, в идеальном случае его поведение *детерминировано*, и в каком бы состоянии ни находилась система, каждый переход будет заканчиваться всегда в одном и том же состоянии, независимо от того, каким образом система оказалась в начальном состоянии. Такой способ функционирования будем называть *бесконтекстным* (англ. *context-free*) и вместе с детерминированностью позволяет в процессе построения модели

не производить дополнительных проверок переходов, которые уже были обнаружены.

1.3.1.2 Симуляционная модель

Разработка формальной модели в процессе проектирования ОУ трудоемка, кроме того, при ручном построении существует риск появления ошибок, связанных с человеческим фактором. Традиционно при проектировании и разработке КФС создается *симуляционная* модель разрабатываемого ОУ, отражающая его свойства. Такие модели проектируются в специальных средах (например, Matlab/Simulink [20]), в которых доступно моделирование поведения объекта, связанного с его физическими характеристиками. В данной работе подразумевается, что существует такая симуляционная модель ОУ, по которой необходимо построить формальную модель, и в дальнейшем под ОУ будет подразумеваться его симуляционная модель.

1.3.1.3 Черный ящик

Чаще всего ОУ рассматривается как «черный ящик», то есть о его внутреннем устройстве ничего не известно. Единственная доступная информация – это его реакция (выходные сигналы) на входные управляющие сигналы (определенную



Рисунок 2 – Объект управления представлен в виде черного ящика, с известными входами и выходами. I и O – входные и выходные символы соответственно

комбинацию значений входных переменных). Назовем комбинацию значений всех входных переменных ОУ *входным символом* (I) и обозначим его как кортеж $(v(I_1), v(I_2), \dots, v(I_n))$, где $v(I_1) \dots v(I_n)$, – значения входных переменных $I_1 \dots I_n$. Аналогично, комбинация всех значений выходных переменных или выходных сигналов называется *выходным символом* и обозначается как кортеж $(v(O_1), v(O_2), \dots, v(O_n))$, где $v(O_1) \dots v(O_n)$ – значения выходных переменных. Соответственно, все возможные входные символы – это *входной алфавит* системы, выходные – *выходной алфавит*. Тогда будем строить формальную модель ОУ по *примерам поведения* или *трассировкам*, где пример поведения – пара (I, O) .

1.3.1.4 Формальное представление объекта управления

ОУ допускает все комбинации входных переменных и реагирует на них соответствующим его внутренней структуре образом. Это значит, что даже если некоторая последовательность входных символов (слово) приводит систему в аварийное состояние, ОУ все равно сгенерирует отражающий это определенный выходной символ, и слово будет допустимым. Состояние системы описывается значениями ее выходных переменных (выходным символом), так как больше ничего о ней не известно. Имея в виду все вышесказанное, в качестве формального представления объекта управления был выбран конечный автомат Мура, в котором выходной сигнал в каждом состоянии соответствует определенному выходному символу системы, и все состояния являются допускающими.

1.3.1.5 Дискретизация вещественных переменных

Для представления системы в виде автомата необходимо, чтобы она была *дискретной*, то есть переход системы из одного состояния в другое происходил бы только под действием входного сигнала. Также все переменные должны быть дискретными. Однако зачастую это свойство нарушается в силу того, что ОУ является физическим компонентом, что означает, как правило, наличие вещественных переменных. Такие переменные должны быть дискретизированы

посредством разделения их области значений на конечное множество смежных интервалов. Например, если вещественная переменная O принимает любое значения из интервала $[0; 15]$, она может быть дискретизирована на конечное множество интервалов $O' = \{[0; 5), [5; 10), [10; 15]\}$. После этого, при построении модели вместо конкретного значения переменной будет использоваться номер интервала, которому принадлежит это значение. Пример дискретизации показан на рисунке 3.

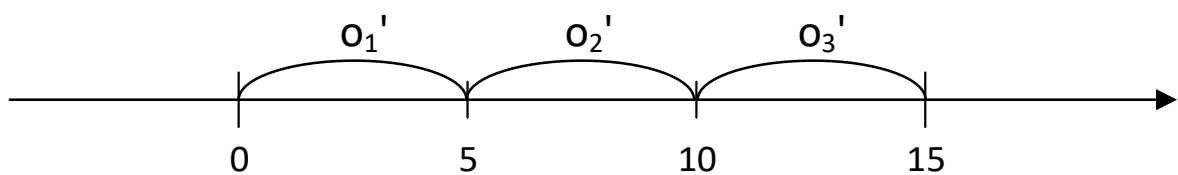


Рисунок 3 - дискретизация переменной $O \in [0; 15]$ на конечное множество интервалов $O' = \{[0; 5), [5; 10), [10; 15]\}$

1.4 Автоматическая генерация формальных моделей ОУ

В данной главе описаны существующие способы автоматической генерации формальных моделей ОУ, классифицированные по доступности взаимодействия с ОУ в процессе построения модели. Раздел 1.1 содержит основные понятия, которые будут использоваться с настоящего момента и до конца работы. В разделах 1.2 и 1.3 приведены обзоры, общая концепция и определение подходов *пассивной* и *активной генерации* формальных моделей соответственно. В разделе 1.3 особое внимание уделяется описанию классического алгоритма активного обучения автоматов L^* [21].

1.4.1 Генерация моделей ОУ на основе пассивного обучения

Суть пассивного обучения заключается в том, что построение модели происходит по заранее подготовленному набору примеров поведения, и это единственная информация, доступная пассивным алгоритмам во время работы.

Следовательно, использование такого подхода ведет к необходимости решения задачи создания таких сценариев сбора примеров поведения, при которых в общем



Рисунок 4 – Общая схема работы алгоритма пассивного обучения случае результирующий набор трассировок покрывает все возможные состояния системы, либо, в частных случаях, является необходимым и достаточным для построения полной модели определенным методом. Преимуществом таких методов является возможность построения формальных моделей объектов, симуляционные модели которых недоступны. Общая схема работы пассивных методов показана на рисунке 4. Генерация формальных моделей ОУ с помощью пассивных методов рассмотрена в работах [22 - 24].

1.4.1.2 Метод, основанный на ограничениях и метод явных состояний

Было использовано два метода пассивного обучения из работы [23]. Первый – метод явных состояний, второй – метод, основанный на ограничениях. Первый метод напоминает предлагаемый в работе подход. На основе полученных сценариев поведения генерируется недетерминированный автомат Мура, где состояния, так же, как и в описанном методе, являются определенными выходными символами, а переходы между ними выполняются по определенным входным символам. Однако так как описываемый метод является пассивным, некоторые переходы могут отсутствовать в сценариях поведения, приводя к тупиковым ситуациям. Для обеспечения возможности корректной формальной верификации таких моделей неподдерживаемые переходы должны быть добавлены, следуя логике, что неизвестный переход ведет в то же состояние, что и ближайший переход из этого же состояния. Результирующая модель содержит информацию о

числе неподдерживаемых переходов. Также вводятся *ограничения справедливости* для предотвращения вечного выполнения обратных петель, то есть той же ситуации, о которой упоминалось в разделе 2.1.3. В данном же методе наложение таких ограничений позволяет в процессе верификации не рассматривать пути, на которых обратная петля выполняется. В то же время, таким образом можно отбросить обратные петли, которые действительно существуют, и результат верификации будет ложным.

Метод, основанный на ограничениях, генерирует полностью символьную модель ОУ, которая является более сжатой, чем модель, построенная предыдущим методом. Здесь для каждого входа и выхода создаются свои переменные состояния, затем, основываясь на собранных сценариях поведения, в модель добавляются следующие ограничения:

- пара выходных переменных может принимать два определенных значения одновременно тогда и только тогда, когда в трассировках существует такой элемент, в котором данные переменные принимают эти значения;
- выходная переменная может принимать два определенных значения последовательно только при условии, что эти значения она принимает в паре последовательных элементов трассировок;
- выходная переменная может принимать определенное значение после того, как входная переменная принимает определенное значение только при условии наличия такого перехода в трассировках.

Сформулированные таким образом ограничения позволяют провести символьную верификацию, например, верификатором NuSMV. Модели такого типа также страдают от обратных петель, и применяется то же решение, что и для моделей, построенных методом явных состояний: ограничения справедливости, говорящие о том, что как минимум одна выходная переменная когда-нибудь изменится.

1.4.1.1 Проверка полноты сгенерированной формальной модели

Следующий важный момент, который затрагивает все автоматически синтезированные методом пассивного обучения формальные модели – это определение близости результирующей формальной модели к моделируемой системе. Исследования в этой области также проводятся. Так, например, в работе [25], был предложен метод определения точности автоматически сгенерированной модели при помощи разделения всего набора примеров поведения ОУ на обучающий и тестовый в соотношении примерно 90% к 10%. После обучения степень соответствия сгенерированной формальной модели и симуляционной модели определялась следующим образом. Сначала все примеры поведения, относящиеся к тестовой выборке, записываются в виде CTL свойств. Проверяется наличие таких путей и начального состояния, что из определенного в примерах поведения состояния существует определенный переход в определенное следующее состояние. Затем с помощью символьного верификатора (например, NuSMV) определяют корректность таких свойств, при этом, чем больший процент соответствия им формальной модели, тем более точной она является.

1.4.3 Генерация конечно-автоматных моделей на основе активного обучения

В отличие от пассивного обучения, методам, основанным на активном обучении [26], не требуются заранее сгенерированные обучающие данные. Вместо этого необходимые примеры поведения запрашиваются в ходе построения модели, и задача сбора трассировок решается самим алгоритмом. Работа большинства таких методов заключается в последовательном уточнении модели-гипотезы, получаемой на каждом шаге после выполнения определенных алгоритмом операций, и может быть отображена при помощи схемы, приведенной на рисунке 5.

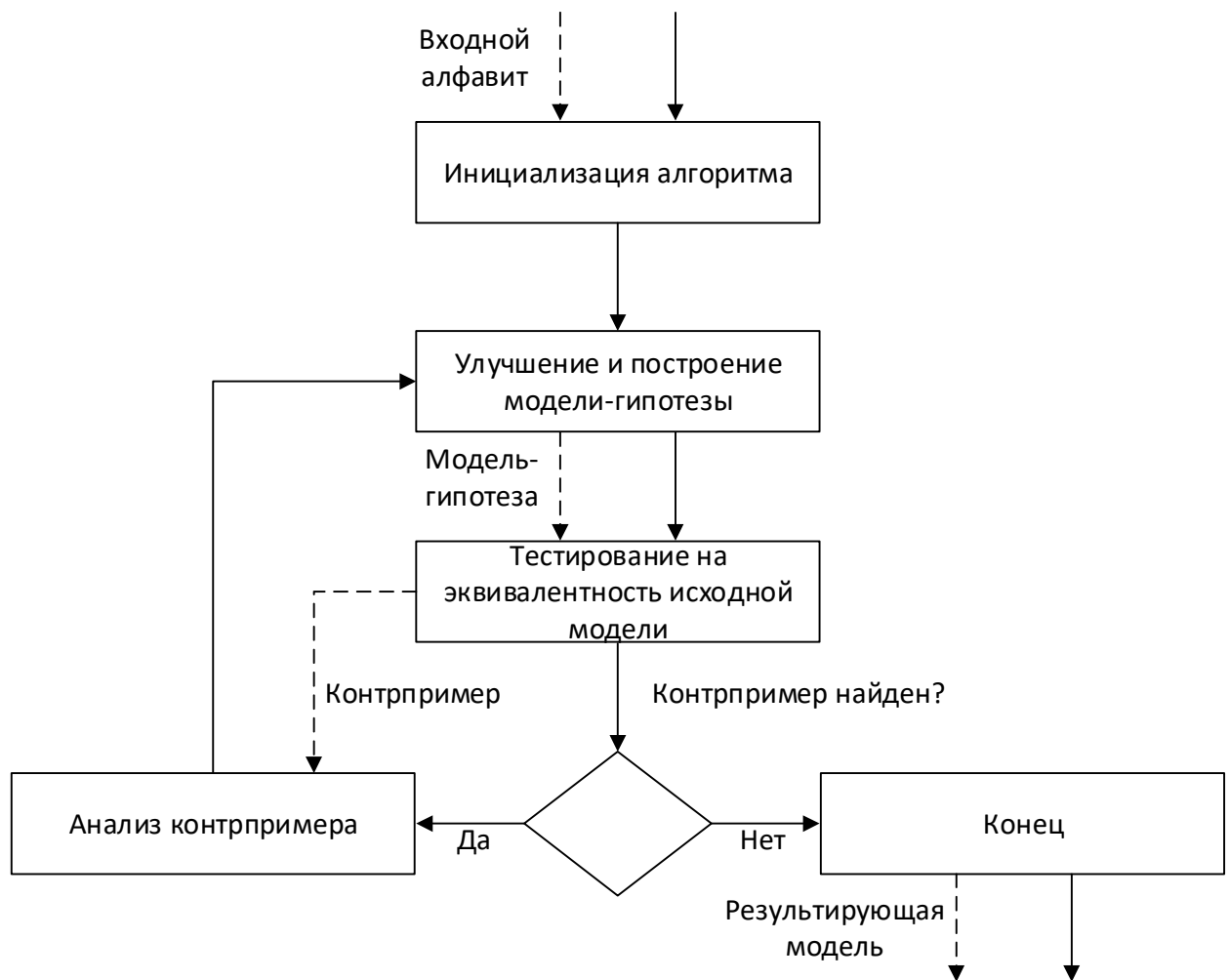


Рисунок 5 – Блок-схема работы стандартного алгоритма активного обучения

Преимущество таких подходов заключается в том, что собираются только необходимые и достаточные для построения точной модели примеры поведения. С другой стороны, стоит учитывать, что большинство методов активного обучения требуют наличия возможности установки симуляционной модели в начальное состояние. Более того, если на результат работы пассивных алгоритмов можно повлиять при помощи подачи на вход различных наборов трассировок, то результат работы алгоритмов активного обучения будет одинаковым для заданной конфигурации системы.

Толчок развитию методов активного обучения дал алгоритм генерации детерминированных конечных автоматов по заданному автомату L^* , разработанный Даной Англуин в 1988 году. Позднее появились его

модификации [27], а также он был распространен на случай КФС, как в работе [28]. Рассмотрим L^* подробнее.

1.4.3.1 Алгоритм L^*

Алгоритм L^* — это классический алгоритм активного обучения детерминированных конечных автоматов (ДКА), подразумевающий наличие оракула, способного ответить на вопрос, является ли какой-либо ДКА эквивалентным данному. С его помощью возможно синтезировать ДКА, принимающий тот же регулярный язык, что и исходный. В основе L^* лежит теорема Майхилла-Нероуда [29], которая гласит, что два слова u и u' , принадлежащие одному языку L , равны тогда и только тогда, когда для них нет *различающих суффиксов*. Различающий суффикс — это такой суффикс v , результат конкатенации которого с u и u' приводит к тому, что только одно из слов uv' и uv принадлежит L , а другое нет.

Алгоритм L^* по этапам своей работы следует схеме, приведенной на рисунке 5, а именно:

1. конструирование автомата-гипотезы;
2. проверка созданного автомата на соответствие исходному;
3. анализ контрпримеров при их наличии.

Допустим, что существует исходный ДКА A , принимающий язык L , и ДКА-гипотеза H , принимающий язык L' . На первом этапе автомат-гипотеза строится по *таблице наблюдений*, содержащей информацию о том, принадлежат ли некоторые последовательности символов алфавита Σ языку L . Для установления принадлежности используются *запросы на членство* или, в оригинале, *Membership queries (MQ)*. Исходному ДКА A отправляется запрос в виде цепочки символов, далее, в зависимости от того, допускает ли A полученное слово, генерируется ответ: «+», если да, и «-», если нет.

Таблица наблюдений состоит из двух частей, в обеих имена столбцов — *суффиксно-замкнутое* конечное множество строк E (суффиксов). Что касается

имен строк, то в первой части это *префиксно-замкнутое* конечное множество строк S (префиксов), вторую (дополнительную) часть образует множество $S\Sigma$ – конкатенация всех слов из S и символов из алфавита Σ . В ячейках таблицы находятся значения MQ от конкатенации соответствующих префиксов и суффиксов. *Префиксно-замкнутое* множество строк – это такое множество, в котором все префиксы входящих в него строк также являются его членами. Так, например, множество строк $X = \{\emptyset, "a", "ab", "abc", "d", "de", "def"\}$ является префиксно-замкнутым. Аналогично определяется *суффиксно-замкнутое* множество строк относительно суффиксов. Пример таблицы наблюдений представлен на рисунке 6.

Введем $row(s) = \{MQ(s, e_1), \dots, MQ(s, e_n)\}$, где s принадлежит множеству $S \cup S\Sigma$, e_i – суффикс, n – число суффиксов (колонок в таблице).

Для того, чтобы по таблице наблюдений можно было построить H , нужно, чтобы она была *замкнута* и *целостна*. Таблица *замкнута*, если для всех t из $S\Sigma$ найдется такое s из S , что $row(t) = row(s)$. Если для какого-то t это условие не выполняется, то t перемещается в S , а в $S\Sigma$ добавляются результаты конкатенации t со всеми символами из Σ . Таблица *целостна*, если для всех s из S выполняется условие, что если $row(s_1) = row(s_2)$, то при любом a из Σ выполняется $row(s_1 \cdot a) = row(s_2 \cdot a)$, где $str_1 \cdot str_2$ – конкатенация строк str_1 и str_2 , в противном случае a является различающим суффиксом и добавляется в множество суффиксов E .

По таблице, отвечающей указанным свойствам, следующим образом строится автомат-гипотеза H . Множество состояний автомата – все различные элементы множества $\{row(s) : s \in S\}$. За начальное состояние принимается $row(\emptyset)$. Допускающими состояниями являются такие состояния, в которых $\{row(s), s \in S, MQ(s) = 1\}$. Переходы формируются с помощью дополнительной части таблицы: следующим состоянием при переходе из $row(s)$ по a из Σ будет $row(s \cdot a)$.

	s_1	s_2	...	s_n
p_1	+	-		-
p_2	-	-		+
...				
p_n	-	+		+
$p_1 \cdot a_1$	-	-		-
$p_2 \cdot a_1$	-	+		+
$p_n \cdot a_m$	+	+		-

Рисунок 6 – Таблица наблюдений – центральная сущность алгоритма L^* .
 Строки и столбцы – префиксы и суффиксы соответственно, на пересечениях – результаты запросов на принадлежность. Верхняя часть таблицы содержит строки, которые впоследствии будут представлять состояния автомата-гипотезы (множество S), нижняя часть таблицы – результат конкатенации каждого префикса с символами входного алфавита (множество $S\Sigma$)

На втором этапе полученный автомат-гипотеза H сравнивается с исходным при помощи *запросов на эквивалентность* или *Equivalence queries (EQ)*. Данные запросы направляются оракулу, который отвечает «ОК» в случае, если H эквивалентен A , либо, в обратном случае, генерирует *контрпример* – слово, которое допускается одним автоматом, но не допускается другим.

На третьем этапе, если был получен сигнал «ОК», то H считается результирующим автоматом, иначе, все возможные префиксы контрпримера добавляются в множество S . После этого алгоритм переходит к шагу 1.

Рассмотрим сложность описанного алгоритма. Пусть n – число состояний результирующего автомата, m – максимальная длина контрпримера, k – мощность входного алфавита. Изначально S и E содержат по одному элементу (пустое слово). Каждый раз, когда оказывается, что таблица наблюдений не замкнута, в S добавляется один элемент, каждый раз, когда оказывается, что она не целостна, элемент добавляется в E . При обработке каждого контрпримера в S добавляется максимум m элементов. Число элементов в E не превышает n . Максимальная длина строки в E не превышает $n - 1$. Число элементов в S не превышает $n + m(n - 1)$, так как таблица может быть незамкнута максимум $n - 1$ раз, может быть сгенерирован $n - 1$ контрпример, каждый из которых добавит в S максимум m строк. С каждой добавленной строкой максимальная длина строки в S увеличивается на единицу, следовательно, максимальная длина строки будет равна $m + n - 1$. Из всего вышесказанного следует, что максимальная мощность множества $(S \cup S\Sigma)E$ будет равна $(k + 1)(n + m(n - 1))n = O(mn^2)$, а максимальная длина строки в этом множестве составляет $m + 2n - 1 = O(m + n)$. Таким образом, размер всей таблицы можно выразить как $O(m^2n^2 + mn^3)$.

ВЫВОДЫ ПО ГЛАВЕ 1

1. Кибер-физические системы – системы, которые сочетают в себе программный и физический компонент. Существует два основных способа проверки корректности их работы: тестирование и верификация. В то время как тестирование позволяет проверить лишь ограниченный набор тестовых сценариев, верификация позволяет проверить все пространство состояний формальной модели КФС на соответствие требованиям.
2. Объект управления обычно рассматривается как черный ящик, о котором известны только его реакции на определенные входные данные. Комбинация значений всех входных переменных называется входным символом, последовательность входных символов – входным словом. Аналогично определяются выходной символ и выходное слово.
3. Моделируемая система должна быть дискретной, детерминированной системой без контекста, области значений вещественных переменных должны быть разбиты на конечное множество смежных интервалов.
4. Результирующая формальная модель будет строиться по примерам поведения или трассировкам, представляющим собой последовательность пар (I, O) , в виде конечного автомата Мура.
5. Методы автоматической генерации формальных моделей ОУ по наличию взаимодействия с ОУ во время процесса построения могут быть разделены на активные и пассивные. Алгоритмы пассивного обучения получают примеры поведения системы однажды в качестве входных данных, в то время как алгоритмы активного обучения итеративно улучшают конструируемую модель, запрашивая необходимые данные у ОУ во время работы.

ГЛАВА 2. ПРЕДЛАГАЕМЫЙ МЕТОД

В данной главе описан предлагаемый в настоящей работе метод автоматической генерации формальных моделей ОУ на основе поиска в ширину с использованием активного обучения. В разделе 2.1 метод представляется с теоретической точки зрения, раздел 2.2 содержит описание алгоритма.

2.1 Метод генерации формальной модели ОУ на основе поиска в ширину

Данный раздел описывает разработанный алгоритм построения формальной модели ОУ. Вначале описываются основная идея алгоритма и возможные способы активного взаимодействия с ОУ. Далее в разделе 2.1.3 приводится расширение базового алгоритма, необходимое для работы с ОУ, описанными в главе 1.

2.1.1 Основная идея

Рассмотрим ОУ, обладающий свойствами, описанными в главе 1, то есть представляемый дискретной детерминированной системой без контекста, в которой отсутствуют внутренние переменные и нет зависимости от времени. В настоящей работе предполагается, что дискретизированные интервалы вещественных переменных, если такие имеются в системе, предоставляются пользователем, так как зависят от свойств рассматриваемого ОУ и от требований, соответствие с которыми необходимо проверить.

Знание о дискретности и детерминированности ОУ дает возможность сделать вывод о том, что, если система находится в каком-то состоянии s и все переходы из данного состояния были проверены, то проверять их снова не требуется, так как, учитывая то, что система не имеет контекста, состояния, в которые будут сделаны эти переходы, не изменятся. Это значит, что, посетив каждое возможное состояние системы, то есть попав в каждый возможный выходной символ и совершив из него переходы по всем входным символам, можно построить точную модель такого ОУ в соответствии с заданной дискретизацией. Имея в виду вышеописанные

наблюдения, можно сделать вывод, что модель ОУ может быть сгенерирована посредством алгоритма активного обучения, в основе которого лежит алгоритм обхода графа в ширину.

Основная идея такого подхода заключается в последовательном выполнении переходов по всем входным символам (по всему входному алфавиту ОУ) из каждого состояния, достижимого из начального. Состояние, из которого были совершены все переходы, будем называть *обработанным*. В течение процесса обучения обработанные состояния добавляются в отдельный список, и, если какое-либо следующее новое состояние содержится в этом списке, то оно не добавляется в очередь тех, которые будут обработаны на следующем шаге. Таким образом, ни одно состояние не обрабатывается дважды. Алгоритм заканчивает работу, когда очередь новых состояний пуста, и, соответственно, все достижимые состояния были обработаны. С помощью такого алгоритма может быть сгенерирован автомат Мура, изображенный на рисунке 7.

2.1.2 Взаимодействие с симуляционной моделью ОУ

Логика алгоритма, описанная в разделе 2.1.1, сочетается с активным взаимодействием с системой: каждый новый переход порождается запросом, который отправляется ОУ только тогда, когда ответ на этот запрос необходим при построении модели. Такие запросы могут осуществляться двумя способами.

Первый – перед каждым новым запросом система устанавливается в начальное состояние, которое определяется как переход по пустому слову. Далее, каждый запрос представляется в виде последовательности символов входного слова ОУ. Затем симулируется выполнение таких входных слов: один за другим системе отправляются составляющие слово символы и считывается реакция симуляционной модели, то есть выходное состояние. Описанная модель взаимодействия подходит для тех случаев, когда невозможно установить симуляционную модель ОУ ни в одно состояние, кроме начального.

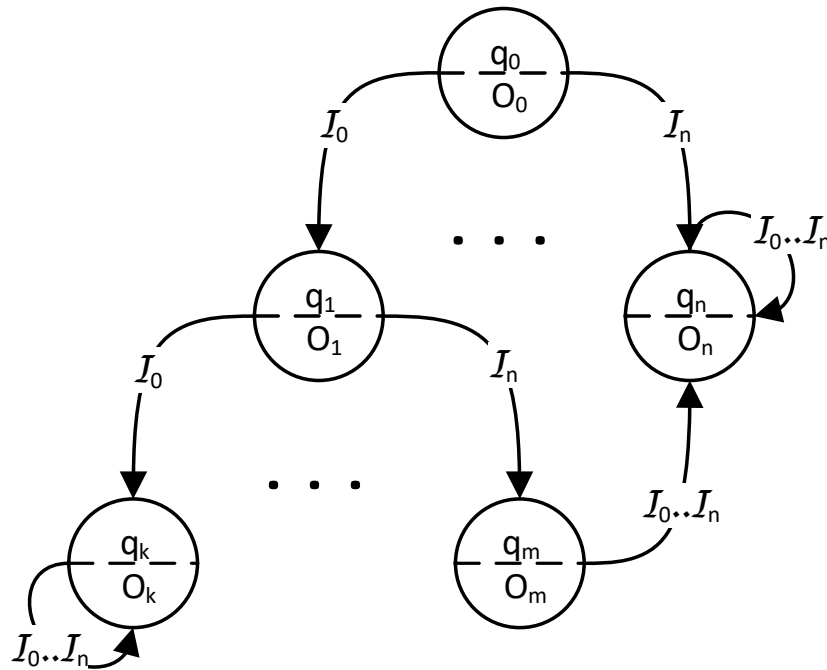


Рисунок 7 – Пример формального представления ОУ, который может быть сгенерирован с помощью базовой версии предлагаемого метода

Второй способ, значительно сокращающий время работы алгоритма, становится доступен, когда по команде симуляционную модель ОУ можно установить в любое возможное состояние. Тогда, находясь в каком-то состоянии s , и зная, что оно определяется некоторой комбинацией значений выходных переменных, можно передать комбинацию значений переменных данного состояния на выходы ОУ, а затем выполнить запрос, который порождается, в данном случае, только одним символом – искомым неизвестным переходом. Важно понимать, что выставление определенных выходов ОУ должно означать действительный переход внутренней структуры системы в состояние, которое соответствует данным выходным переменным.

2.1.3 Обработка вещественных переменных

Описанный выше простой алгоритм эффективен только когда входные и выходные переменные изначально имеют дискретные области значений. На самом же деле, так как ОУ – физический компонент, он почти всегда имеет какую-либо непрерывную составляющую, изменение которой отражается изменением одной

или нескольких непрерывных переменных, что осложняет формальное моделирование. Конечно, дискретизация вещественных переменных может сделать систему дискретной, но, как будет видно из следующего примера, приведенный выше алгоритм не сгенерирует корректную модель. Возникает две проблемы, которые могут помешать достижению результата: *проблема обратных петель* и *проблема циклов*. В следующих подразделах подробно описаны причины возникновения каждой из них, а также предлагаемые решения.

2.1.3.1 Проблема обратных петель

Рассмотрим систему с одной логической входной переменной I_1 и одной вещественной выходной переменной $O_1 \in [0; 15)$, область значений которой разделена на дискретные интервалы $O_1' \in \{[0; 5), [5; 15)\}$. Тогда результирующий автомат будет состоять из двух состояний, каждое из которых будет соответствовать одному из интервалов O_1' . В начальном состоянии $O_1 = 0$, то есть $O_1 \in [0; 5)$; после перехода по $I_1 = \text{TRUE}$, значение O_1 увеличивается на 1. Следуя вышеописанному алгоритму, в ходе построения модели сначала будут проверены два перехода: по $I_1 = \text{FALSE}$ и по $I_1 = \text{TRUE}$. После первого перехода O_1 также будет равна нулю, а после второго перехода значение изменится ($O_1 = 1$), но останется прежним дискретный интервал $O_1' = [0; 5)$. В итоге, оба запроса приведут к образованию в модели обратных петель, и ни одно новое состояние не будет сгенерировано, а значит, алгоритм закончит работу, так и не достигнув состояния, в котором $O_1 \in [5; 15)$. Таким образом, можно заметить, что изменяющееся конкретное значение вещественной переменной не всегда ведет к изменению ее дискретного интервала, а значит, и к изменению состояния. Отсюда следует, что модель получается неверной из-за перехода, образующего обратную петлю из-за неучтенной возможности вещественной переменной изменять свой дискретный интервал более чем за один цикл взаимодействия между контроллером и ОУ. Тогда такую обратную петлю, в которой вещественная переменная изменяет свое значение, будем называть *ложной обратной петлей*.

Для выяснения того, является ли обратная петля ложной, заметив, что после запроса I_1 конкретное значение переменной O_1 изменилось, но не вышло за границы текущего дискретного интервала, можно предположить, что следующий запрос I_1 изменит это значение таким же образом и проверить это предположение, отправив ОУ на вход еще один запрос I_1 . Затем, если предположение оказалось верным, то определенное число запросов I_1 (в случае рассматриваемой системы – пять) может привести систему к состоянию, в котором значение переменной O_1 находится в следующем дискретном интервале.

Рассматривая общий случай, назовем ситуацию, в которой после какого-то перехода непрерывная переменная изменяет свое конкретное значение, но остается в том же дискретном интервале, *движением внутри интервала* – значение непрерывной переменной монотонно возрастает или убывает. Однако не все изменения вещественных переменных монотонны, так как, имея дело с физическим компонентом, можно наблюдать шум, короткие скачки в значениях переменных, не приводящие при этом к изменению состояния, флуктуации. Следовательно, необходим метод, позволяющий определить, что с течением времени значение переменной либо монотонно убывает, либо возрастает. Проблему обнаружения монотонности поведения вещественной переменной внутри интервала назовем *проблемой обратных петель*.

Для решения описанной задачи была разработана следующая стратегия распознавания движения внутри интервала. Во-первых, запрос, вызвавший изменение непрерывной переменной повторяется C раз, и каждый ответ системы сохраняется вместе с номером запроса. Таким образом формируется упорядоченное по времени множество значений такой переменной, являющееся входными данными для алгоритма определения характера ее изменения. Этот алгоритм работает следующим образом. Во-первых, рассчитывается коэффициент корреляции Пирсона между значениями непрерывной переменной на каждом шаге и номером запроса. Далее, при значении коэффициента корреляции больше, чем 0,5, принимается решение о том, что значение переменной изменяется почти монотонно, следовательно, чтобы выйти из данного интервала в следующий нужно

повторять тот же самый запрос. Если же коэффициент корреляции меньше 0,5, то необходимость повторять тот же самый запрос отсутствует, так как это не приведет систему в состояние в следующем интервале, а переход по символу запроса является обратной петлей. Если же система перешла в следующее состояние менее чем за C запросов, то расчет корреляции не требуется, и алгоритм построения формальной модели ОУ продолжает работу.

2.1.3.2 Проблема циклов

Теперь рассмотрим ситуацию, когда в системе есть и дискретные, и дискретизированные выходные переменные. В этом случае, даже используя логику для обработки обратных петель, в результате можно получить неверную модель. Данная проблема проиллюстрирована на экспериментальной системе, работа которой отображена на рисунке 9. В этой системе присутствуют две логические входные переменные I_1 и I_2 и две выходные переменные: одна логическая O_1 , другая вещественная $O_2 \in [0, 15]$ с дискретными интервалами $O_1' \in \{[0, 5); [5, 10); [10, 15]\}$. Система работает следующим образом. Если значение переменной O_2 меньше пятнадцати, то при входном сигнале $I_2 = \text{TRUE}$, значение переменной O_2

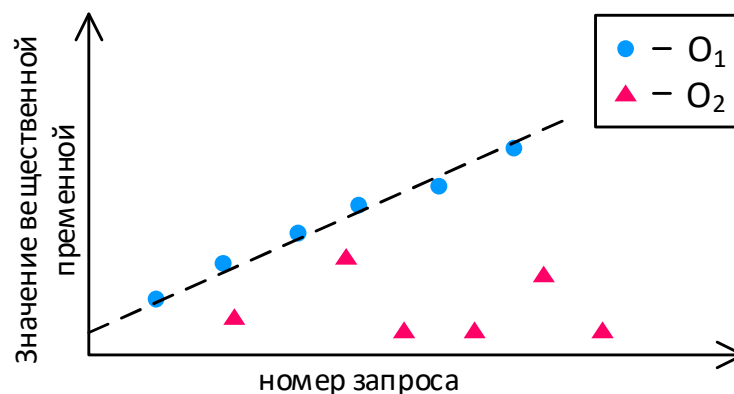


Рисунок 8 – Определение монотонности поведения переменных. Переменная O_1 монотонно возрастает, коэффициент корреляции Пирсона больше 0,5.

Переменная O_2 изменяется немонотонно, коэффициент корреляции меньше 0,5

увеличивается на единицу, если больше, то остается прежним, в то время как при каждом запросе O_1 принимает значение I_1 .

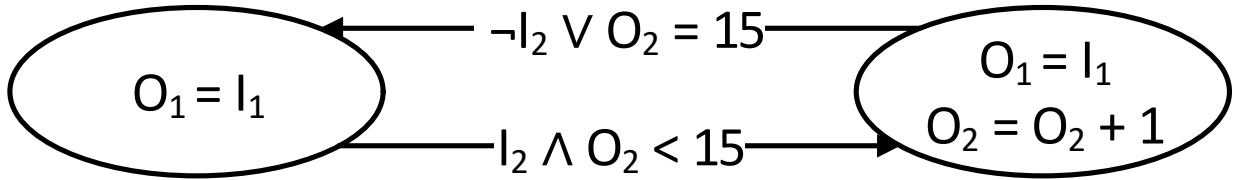


Рисунок 9 – Пример системы с двумя логическими входами I_1 и I_2 и двумя выходами: логическим O_1 и вещественным O_2

Используя базовый алгоритм в сочетании с решением проблемы обратных петель, можно получить автомат Мура, часть которого изображена на рисунке 10 (для большей иллюстративности показаны только переходы для $I_2 = \text{TRUE}$). Здесь проблема заключается в том, что при переходе из состояния q_0 по комбинации символов $I_2 \wedge I_1$ изменяется не только вещественная переменная O_2 , но и дискретная переменная O_1 , а значит переход не является обратной петлей. Вследствие чего создается новое состояние q_1 в котором O_2 находится в том же дискретном интервале $[0; 5)$, несмотря на то, что при переходе по $I_2 \wedge I_1$ конкретное ее значение изменилось. Аналогичная ситуация происходит и с переходом из состояния q_1 по $\neg I_2 \wedge I_1$ в q_0 , а также во втором интервале $O_2 \in [5; 10)$.

Такие переходы действительно существуют в системе, однако, построив автомат, содержащий их, можно заметить циклы между q_0 и q_1 , а также между q_2 и q_3 . Теперь, проверяя на такой модели свойство «если значение O_2 находится в интервале $[0; 5)$ и I_2 всегда равно TRUE, то когда-нибудь обязательно система придет в состояние, где значение O_2 принадлежит интервалу $[5; 10)$ » можно получить ложные негативные результаты верификации, так как контрпримером будет вечно выполняющийся цикл включающий состояния q_0 и q_1 , которого, на самом деле не существует, так как при повторяющихся входных символах, содержащих в себе $I_2 = \text{TRUE}$, O_2 когда-нибудь перейдет в следующий интервал. Можно заметить, что в данном случае все также наблюдается движение

непрерывной переменной внутри интервала, в дополнение к которому изменяются значения дискретных переменных.

Таким образом, следующей задачей является расширение логики обработки ложных обратных петель до обнаружения таких ненужных циклов, дающих неверные результаты при формальной верификации.

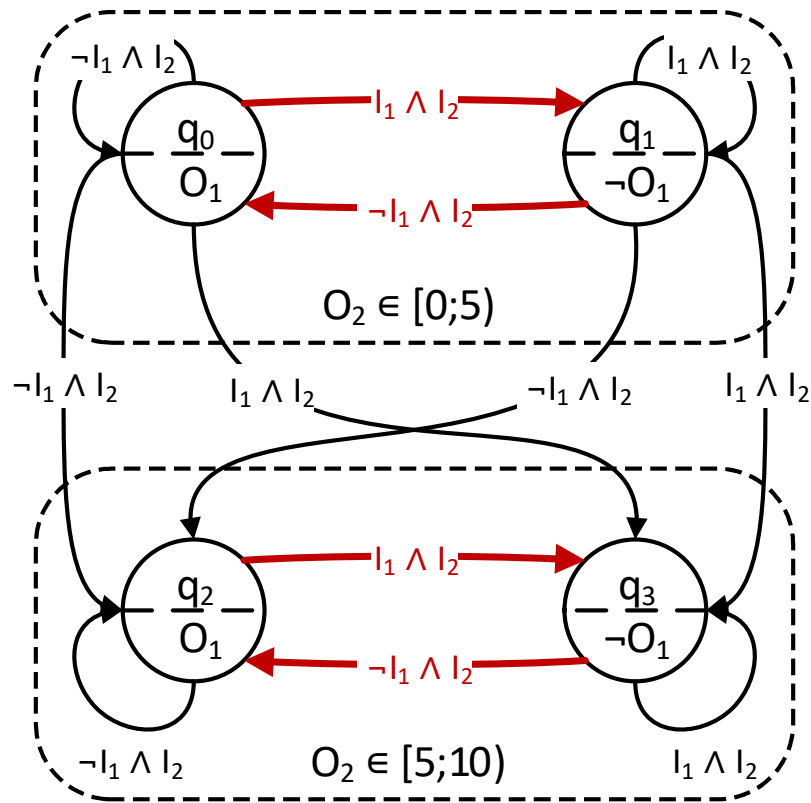


Рисунок 10 – Часть модели, представленной автоматом Мура, которая содержит переходы, образующие циклы (показаны жирными стрелками) в первом и втором дискретных интервалах переменной O_2

Для решения этой проблемы предыдущий механизм обнаружения ложных обратных петель был доработан следующим образом. Если в процессе перехода из состояния q_n в состояние q_k (при этом q_k может быть равно q_n), вызванного каким-либо запросом R , обнаруживается движение внутри интервала для какой-либо непрерывной переменной, состояние q_k сохраняется как временное. Далее, во-первых, делается проверка, является ли переход по R из состояния q_k обратной петлей, так, как это описано выше. Если да, то больше никаких запросов, кроме R ,

из q_k не проверяется, и R повторяется до достижения вещественной переменной ее следующего интервала, а затем q_k регистрируется, как новое состояние. Иначе, q_k отмечается как состояние, в котором вещественные переменные будут сравниваться не по их дискретным интервалам, а по конкретным значениям. Далее, если любое состояние является отмеченным таким образом, то все состояния, которые будут сгенерированы после любого перехода из данного, будут также отмечены для сравнения по конкретным значениям, если они принадлежат тому же интервалу вещественной переменной.

Используя описанную выше стратегию часть модели с рисунка 10 может быть перерисована так, как показано на рисунке 11.

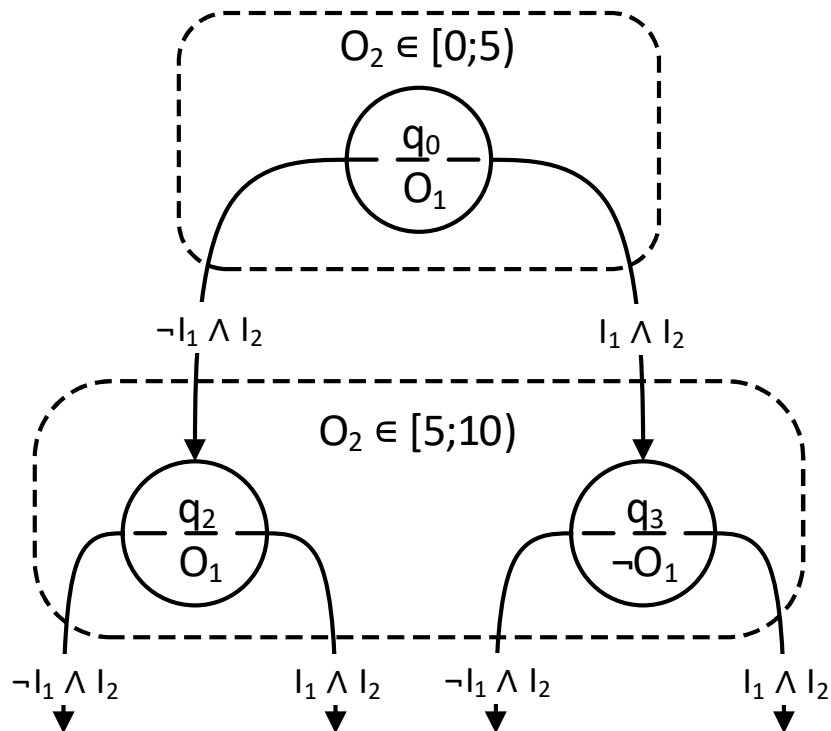


Рисунок 11 – Часть модели, представленной автоматом Мура, где убраны циклы из первого и второго дискретного интервала переменной O_2

2.2 Реализация

Псевдокод предложенного метода показан на рисунке 12. Предполагается, что доступна функция установки ОУ в произвольное состояние.

Функция `sendMQ(состояние, входной символ)` устанавливает систему в указанное состояние, совершает переход по входному символу и возвращает результат в виде перехода $T = \langle q_{start}, q_{end}, transition_symbol \rangle$. Либо, в случае `sendMQ(∅)` совершается переход по пустому символу для обнаружения начального состояния. Функция `correlation(начальное состояние, конечное состояние, символ)` возвращает значение коэффициента корреляции между вещественной переменной и количеством запросов. Функция `enableConcreteComparison(состояние)` делает возможным сравнение выходных переменных, образующих данное состояние, с другими переменными в других состояниях по их конкретным значениям, а не по дискретным интервалам. Все состояния, получающиеся посредством переходов из состояний, обладающих этим свойством, будут также обладать им до тех пор, пока вещественная переменная не перейдет в свой следующий интервал.

Реализация предложенного метода доступна по ссылке [30] и представляет собой платформу активного обучения автоматов по симуляционной модели объекта управления, выполненной в NxtStudio [31]. Взаимодействие с другими средами разработки симуляционных моделей возможно при реализации соответствующих интерфейсов. Возможность сообщения между программой и симуляционной моделью ОУ в NxtStudio была обеспечена при помощи TCP соединения. На данный момент поддерживаются логические входные и выходные переменные, а также вещественные выходные переменные ОУ. При необходимости возможно расширение для целочисленных переменных. Система определяется конфигурационным файлом, пример такого файла находится в этом же репозитории.

Исходные данные: *inputs* – множество входных переменных,
outputs – множество выходных переменных

Результат: *T* – множество переходов

```

1  Qprocessed ← ∅;
2  Qnext ← sendMQ(∅);
3  T ← ∅;
4  I ← ∏ inputsi;
5  O ← ∏ outputsi;
6  while Qnext ≠ ∅ do
7      T ← ∅;
8      for q ∈ Qnext do
9          for input ∈ I do
10             (qstart, qend, s) ← sendMQ(q, input);
11             T ← T ∪ {(qstart, qend, s)};
12         End
13     End
14     TinInterval ← ∅;
15     for (qstart, qend, s) ∈ T do
16         if movingInsideInterval(qstart, qend) then
17             TinInterval ← TinInterval ∪ {(qstart, qend, s)};
18         end
19     end
20     for (qstart, qend, s) ∈ TinInterval do
21         q'end ← sendMQ(qend, s).qend;
22         c ← correlation(qstart, qend, s);
23         if q'end = qend ∧ c > 0,5 then
24             while q'end = qend do
25                 qend ← q'end;
26                 q'end ← sendMQ(qend, s).qend;
27             end
28             qend ← q'end;
29         else if q'end = qend ∧ c < 0,5 then
30             qend ← q'end;
31         else
32             enableConcreteComparison(qend);
33         end
34     end
35     T ← T ∪ T;
36     Qprocessed ← Qprocessed ∪ {qstart | (qstart, qend, s) ∈ T};
37     Qnext ← {qend | (qstart, qend, s) ∈ T, qend ∉ Qprocessed};
38 end
39 return T

```

Рисунок 12 – Общий алгоритм синтеза формальной модели ОУ, основанный на
 обходе в ширину

2.3 Примечания

Во-первых, если ОУ обладает только дискретными выходными переменными, часть, относящаяся к обнаружению циклов и ложных обратных петель, можно опустить. Во-вторых, если при наличии вещественных переменных есть какой-либо оракул, который может гарантировать, что при обработке циклов функция `enableConcreteComparison` никогда не будет вызвана, то число возможных состояний результирующей системы можно ограничить сверху числом возможных комбинаций дискретных значений выходных переменных ОУ.

Также стоит отметить, что данный метод позволяет обнаружить только те состояния, которые достижимы из начального, полученного путем перехода по пустому символу, а значит существует риск построить только подпространство состояний системы. Если ОУ содержит несколько возможных точек входа, то существует два варианта развития событий. Первый – пользователь сам задает возможные начальные состояния системы. Второй – все возможные состояния ОУ считаются начальными и из каждого из них запускается описанный алгоритм.

ВЫВОДЫ ПО ГЛАВЕ 2

1. Был разработан алгоритм построения формальной модели детерминированного ОУ без контекста, представленного в виде черного ящика выходные переменные которого могут быть непрерывными, основанный на алгоритме обхода графа в ширину.
2. Предложены методы решения проблемы, связанных с обработкой выходных вещественных переменных ОУ, а именно: обработка ложных обратных петель и устранение ненужных циклов.
3. Была предложена стратегия построения формальных моделей ОУ, имеющих несколько начальных состояний, либо тех, чье пространство состояний состоит из нескольких непересекающихся подпространств.
4. Алгоритм был реализован на языке Java в виде платформы для построения формальных моделей ОУ, построенных в симуляционной среде NxtStudio.

ГЛАВА 3. ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ

Данная глава содержит описание экспериментальной системы и результаты верификации моделей, полученных с помощью методов активного и пассивного обучения. В разделе 3.1 описана экспериментальная система. Алгоритмы, с которыми производится сравнение предложенного метода, представлены в разделе 3.2. Раздел 3.3 содержит детали реализации рассматриваемого метода. Сравнение производительности алгоритмов и результаты верификации формальной модели экспериментальной системы содержатся в разделе 3.4. Раздел 3.5 заканчивает главу 3 обсуждением полученных результатов.

3.1 Экспериментальная система

В качестве экспериментальной системы была выбрана симуляционная модель трехэтажного лифта (рисунок 13), выполненная в NxtStudio. В состав модели входят взаимодействующие друг с другом контроллер и ОУ. Логические входные переменные ОУ, которые соответствуют выходным переменным контроллера и определены следующим образом:

- `moveUp` – команда лифту «ехать вверх»;
- `moveDown` – команда лифту «ехать вниз»;
- `openDoor0..2` – открыть двери лифта на этаже 0, 1 или 2 соответственно.

Также контроллер имеет собственные входные переменные, не замкнутые на модель ОУ – это три логических переменных `buttonPressed0..2`, значение которых «истина», если лифт вызван на этаже 0, 1 или 2 соответственно.

Выходные переменные ОУ, кроме `carPos`, соответствуют входным переменным контроллера:

- `carAtFloor0..2` – значение переменной «истина», если лифт находится на этаже 0, 1 или 2 соответственно;

- `doorClosed0..2` – значения переменной «истина», если двери лифта открыты на этаже 0, 1 или 2 соответственно;
- `carPos` $\in [30; 419,5)$ – позиция кабины лифта в текущий момент. Область значений данной переменной была разбита на следующие дискретные интервалы: $[30; 30,5)$, $[30,5; 224,5)$, $[224,5; 226,5)$, $[226,5; 418,5)$, $[418,5; 419,5)$, где маленький промежуток соответствует позиции лифта на этаже, большой – между этажами.

Данная модель идентична рассмотренной в работе [25], за исключением небольших изменений в логике контроллера: ранее переменные `buttonPressed0..2` не были внешними для системы, их значение генерировалось внутри ОУ. Теперь инициатором изменения является внешняя среда, ОУ лишь сбрасывает их значения тогда, когда кабина лифта достигает указанного этажа.




	Этаж	CarPos
	2	4
		3
	1	2
		1
	0	0

Рисунок 13 – Графический интерфейс симуляционной модели лифта.

Числами от 0 до 4 в столбце `CarPos` обозначены дискретные интервалы $[30; 30,5)$, $[30,5; 224,5)$, $[224,5; 226,5)$, $[226,5; 418,5)$, $[418,5; 419,5)$ соответственно

Также для обеспечения более удобного взаимодействия с моделью в нее была добавлена возможность возвращения в любое допустимое состояние. Для проведения экспериментов формальная модель описанной системы генерировалась предложенным в данной работе методом, двумя методами пассивного обучения из работы [23], а также при помощи алгоритма L^* . Далее полученные модели совмещались с символьной моделью контроллера, выполненной в формате NuSMV и подвергались формальной верификации заранее определенных LTL свойств. Целью экспериментов было сравнение результатов верификации, времени, потраченного на построение модели, и точности полученной модели.

3.2 Методы, участвующие в сравнении

В сравнении участвовал предложенный метод, два алгоритма пассивного обучения и описанный в главе 1 алгоритм активного обучения автоматов L^* . Раздел 3.2.1 представляет изменения в алгоритме L^* , которые необходимо реализовать для его применения к конструированию формальных моделей киберфизических систем. В разделе 3.2.2 названы использованные алгоритмы пассивного обучения.

3.2.1 Алгоритм активного обучения L^*

В классическом виде алгоритм L^* применим для генерации ДКА, принимающего тот же язык, что и заданный исходный ДКА, однако для синтеза автомата по ОУ в алгоритм нужно внести некоторые изменения. Во-первых, как было показано в разделе 1.3.1.4, ОУ представляется в виде автомата Мура. Это значит, что ячейки таблицы наблюдений должны содержать не отметки того, допускается ли данная последовательность автоматом (потому что в случае ОУ все последовательности допустимы), а выходной символ, сгенерированный ОУ в качестве реакции на запрашиваемую последовательность, как в работе [28]. Во-вторых, в качестве контрпримера теперь выступает не слово, которое допускается одним автоматом и не допускается другим, а последовательность, реакция на

которую у автомата-гипотезы и ОУ различна, данное расширение также описано в работе [28].

В-третьих, ОУ часто содержит вещественные переменные, а значит их обработка также должна быть предусмотрена алгоритмом. Классический L^* в данном случае страдает от тех же проблем, что и классический поиск в ширину, поэтому техники обработки вещественных переменных, предложенные в разделе 2.1.3 для предлагаемого метода подходят также и для L^* . Нужно заметить, что в стандартном виде каждая следующая цепочка симулируется от начала и до конца, что ведет к большим временным затратам при применении вышеописанных техник. В том случае, когда система дискретна и детерминирована, может быть использована следующая оптимизация. В процессе обработки каждой последовательности сохраняются состояния системы после переходов по каждому символу данной последовательности. Затем, если какая-либо из следующих проверяемых цепочек хотя бы частично начинается так же, как одна из уже обработанных, то система сбрасывается в то состояние, на котором совпадение закончилось. Тем самым, можно симулировать не всю новую цепочку, а только ее необработанную часть.

В-четвертых, ОУ рассматривается как «черный ящик», а значит не существует такого оракула, который мог бы определить, является ли автомат-гипотеза эквивалентным ОУ. Из этого следует, что EQ не может быть применен в классическом виде и должен быть либо каким-то образом аппроксимирован с помощью MQ, либо должны быть применены какие-то другие методы тестирования результирующей модели.

Однако так как ОУ не имеет контекста (раздел 1.3.1.1), некоторые шаги алгоритма L^* не требуются для построения автомата. Один из них – это поиск различающих суффиксов: при генерации автомата без контекста таблица наблюдений всегда будет целостной. При равенстве двух строк, обозначающих состояния, различающие суффиксы появляются только тогда, когда по одному и тому же символу возможен переход в два различных состояния, а именно это и называется контекстом.

Далее, по той же причине, проверка на эквивалентность автомата-гипотезы исходному также будет излишней. В процессе замыкания таблицы происходит перенос из дополнительной части таблицы в основную строк, которые не имеют эквивалентных в основной части таблицы. После чего проверяется и заносится в дополнительную часть таблицы их конкатенация со всеми символами входного алфавита системы. В отсутствии контекста, который является в данном случае источником недетерминизма, таким образом достигается полное покрытие состояний исходного автомата. Это означает, что за один проход алгоритма L^* после последовательного замыкания таблицы будет построен автомат, эквивалентный исходному.

3.2.2 Алгоритмы пассивного обучения

Для сравнения были выбраны два алгоритма пассивного обучения из работы [23], представленные в разделе 1.4.1.2: метод явных состояний и метод, основанный на ограничениях. Так как данные методы являются пассивными, первый этап – это сбор сценариев исполнения или трассировок.

Для генерации сценариев исполнения использовался алгоритм *«полуслучайный» контроллер*, описанный в работе [25], так как он показал наилучшие результаты. Данный подход подразумевает отправку случайной комбинации входных переменных на вход ОУ, а затем повторение этой комбинации S раз, после чего комбинация генерируется случайным образом заново, и все повторяется. Трассировки записываются в файл и используются для построения формальной модели. Важно, чтобы за время сбора примеров поведения система посетила все возможные состояния. Однако так как время записи сценариев поведения включается в общее время работы алгоритма (алгоритмы активного обучения запрашивают необходимые последовательности во время построения моделей), максимальный промежуток времени сбора трассировок был равен максимальному времени работы алгоритма активного обучения.

3.4 Результаты

Эксперименты проводились на персональном компьютере с использованием процессора *Intel Core I7-7600U* с тактовой частотой 2,8 ГГц и оперативной памятью 8 Гб. Ход эксперимента был таков. Сначала производилась генерация формальной модели ОУ при помощи четырех упомянутых методов. Затем, построенная модель ОУ соединялась с формальной моделью контроллера, образуя, таким образом, замкнутую формальную модель КФС. После чего производилась верификация свойств: как всей системы в целом, так и свойств, касающихся только ОУ.

Сравнение метрик алгоритмов, участвовавших в сравнении, приведено в таблице 1. Нужно отметить, что время генерации трассировок для пассивных методов было ограничено сверху самым большим временем выполнения алгоритма активного обучения.

Таблица 1 – Сравнение алгоритмов автоматического синтеза формальной модели объекта управления.

Алгоритм	Время построения	Число состояний
Методы пассивного обучения		
Основанный на ограничениях	3ч сбор трассировок + 120с построение	220 ограничений
Основанный на явных состояниях	3ч сбор трассировок + 367с построение	20
Методы активного обучения		
Предлагаемый метод	135с	40
L^*	3ч	40

В таблице 2 отражены результаты верификации построенных различными способами моделей.

Таблица 2 – Результаты верификации формальных моделей КФС с моделями объекта управления, построенными разными методами

№	Темпоральное свойство	Комментарий	Корректное значение	Метод построения модели			
				ограничения	Явные состояния	Предлагаемый	L*
Свойства объекта управления							
φ ₁	$\mathbf{G} (\text{carAtFloor1} \wedge$ $\mathbf{G} \neg \text{motorUp} \wedge$ $\mathbf{G} (\text{motorDown} \vee$ $\text{carAtFloor0}) \rightarrow$ $\mathbf{G} \neg \text{carAtFloor2})$	Если кабина лифта находится на этаже 1 и никогда не посылается команда «ехать наверх» и всегда посылается команда «ехать вниз», либо кабина всегда на этаже 0, она никогда не окажется на этаже 2	+	+	+	+	+
φ ₂	$\mathbf{G} (\mathbf{G} \neg \text{motorUp} \wedge$ $\mathbf{G} (\text{motorDown} \vee$ $\text{carAtFloor0}) \rightarrow$ $\mathbf{F} \text{ carAtFloor0})$	Если кабина лифта находится на этаже 0 или всегда посылается команда «ехать вниз» и никогда не посылается команда «ехать наверх», кабина окажется на этаже 0	+	-	-	+	+

Продолжение таблицы 2

№	Темпоральное свойство	Комментарий	Корректное значение	Метод построения модели			
				ограничения	Явные состояния	Предлагаемый	L*
φ ₃	$\mathbf{G} (\mathbf{G} \neg \text{motorDown} \wedge \mathbf{G} (\text{motorUp} \vee \text{carAtFloor2}) \rightarrow \mathbf{F} \text{carAtFloor2})$	То же, что φ ₂ . Если кабина лифта едет всегда вверх, то она окажется на этаже 2	+	–	–	+	+
φ ₄	$\mathbf{GF} \neg \text{motorDown}$	Контроллер не может всегда посылать команду «ехать вниз»	–	–	–	–	–
φ ₅	$\mathbf{G} (\text{carAtFloor1} \wedge \mathbf{G} \neg \text{motorUp} \wedge \mathbf{G} \text{motorDown} \rightarrow \mathbf{F} \text{carAtFloor2})$	То же, что φ ₁ , но здесь не проверяется условие, что кабина лифта может быть на этаже 0	–	–	–	–	–
φ ₆	$\mathbf{G} \neg (\text{motorDown} \wedge \text{motorUp})$	Команды «ехать вниз» и «ехать вверх» никогда не встречаются одновременно	–	–	–	–	–
φ ₇	$\mathbf{G} (\text{carPos} = 4 \wedge \text{motorDown} \wedge \neg \text{motorUp} \rightarrow \mathbf{X} \text{carPos} = 3)$	Если кабина лифта находится на этаже и получает команду «ехать вниз», она окажется между этажами 1 и 2	+	–	–	+	+

Продолжение таблицы 2

№	Темпоральное свойство	Комментарий	Корректное значение	Метод построения модели			
				ограничения	Явные состояния	Предлагаемый	L*
φ_8	$\mathbf{G} (\text{carPos} = 2 \wedge \text{motorDown} \wedge \neg \text{motorUp} \rightarrow \mathbf{X} \text{carPos} = 1)$	То же, что φ_7 , но с этажом 1	+	–	+	+	+
φ_9	$\mathbf{G} (\text{carPos} = 2 \wedge \neg \text{motorDown} \wedge \text{motorUp} \rightarrow \mathbf{X} \text{carPos} = 3)$	Если кабина находится на этаже 1 и получена команда «ехать вверх», кабина окажется между этажами 1 и 2	+	–	+	+	+
φ_{10}	$\mathbf{G} (\text{carPos} = 0 \wedge \neg \text{motorDown} \wedge \text{motorUp} \rightarrow \mathbf{X} \text{carPos} = 1)$	То же, что φ_9 , только с этажом 0	+	–	+	+	+
Свойства системы в целом							
φ_{11}	$\forall k \in [0..2]$ $\mathbf{G} (\text{buttonPressed}k \rightarrow \mathbf{F} \text{carAtFloor}k)$	Если лифт вызван на этаж k, и он не застрял на каком-либо этаже, он прибует на этаж k	–	–	–	–	–

Продолжение таблицы 2

№	Темпоральное свойство	Комментарий	Корректное значение	Метод построения модели			
				ограничения	Явные состояния	Предлагаемый	L*
φ_{12}	$\mathbf{G} (\text{buttonPressed2} \wedge$ (не постоянно на каком-то этаже) $\rightarrow \mathbf{F} \text{ carAtFloor2})$	Если лифт вызван на этаж 1, и он не застрял на каком-либо этаже, он прибудет на этаж 1	+	–	–	+	+
φ_{13}	$\mathbf{G} (\text{buttonPressed1} \wedge$ (не постоянно на каком-то этаже) $\rightarrow \mathbf{F} \text{ carAtFloor1})$	То же, что φ_{12} для этажа 1	+	–	–	+	+
φ_{14}	$\mathbf{G} (\text{buttonPressed0} \wedge$ (не постоянно на каком-то этаже) $\rightarrow \mathbf{F} \text{ carAtFloor0})$	То же, что для этажа 0, однако между этажом 0 и этажом 2 на этаже 1 контроллер выберет этаж 2	–	–	–	–	–
φ_{15}	$\mathbf{G} (\text{carPos} \in \{1,3\} \rightarrow$ $\text{doorClosed0} \wedge$ doorClosed1 $\wedge \text{doorClosed2})$	Двери лифта всегда закрыты, если он находится между этажами	+	–	+	+	+

Также была проанализирована зависимость времени, затраченного на построение формальной модели, от числа дискретных интервалов вещественной переменной `carPos`. Число дискретных интервалов изменялось от 5 до 43 с шагом 2. График на рисунке 14 показывает, что такая зависимость линейна, и,

действительно, при увеличении числа дискретных интервалов, увеличивается число состояний генерируемого автомата, а значит ко времени работы алгоритма добавляется время обработки новых состояний. Таким образом, если в новом дискретном интервале m состояний, а n – размер входного алфавита объекта управления, то при добавлении одного такого дискретного интервала количество переходов увеличится на mn .

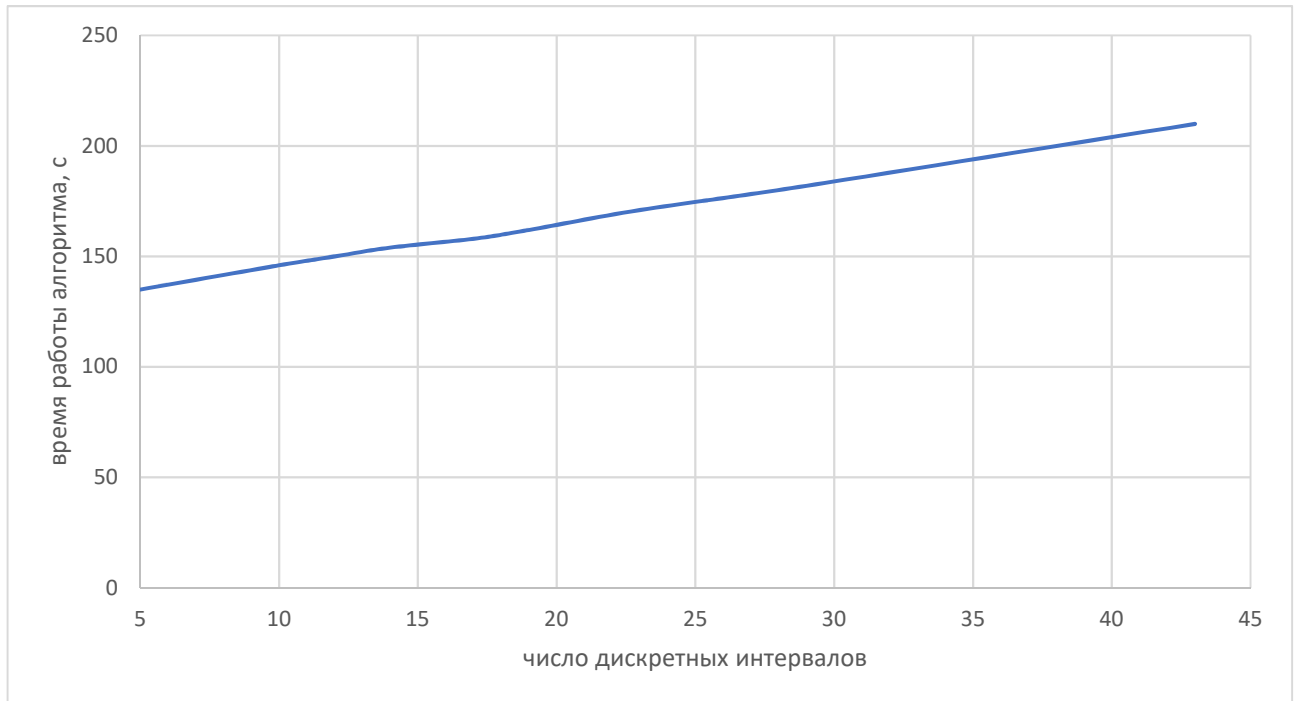


Рисунок 14 – Зависимость времени работы алгоритма от числа дискретных интервалов вещественной переменной

3.5 Обсуждение результатов

В ходе эксперимента была проведена автоматическая генерация формальной модели ОУ в виде детерминированного автомата Мура и её конвертация в формат верификатора NuSMV при помощи методов активного обучения: адаптированного L^* и предлагаемого, – а также при помощи методов пассивного обучения, описанных в работе [23].

Первый вывод, который можно сделать – при наличии возможностей установки системы в любое состояние и внешнего с ней взаимодействия, алгоритмы активного обучения строят более точную модель за меньшее время, чем

пассивные алгоритмы. В то время как у пассивных методов существует проблема генерации необходимых трассировок, покрывающих все пространство состояний системы, активные алгоритмы ей не подвержены, так как запрашивают те и только те последовательности входных символов, реакция на которые им нужна в процессе построения модели. Соответственно, во время работы пассивных алгоритмов также включается и время сбора трассировок. Но даже предполагая, что оно не учитывается, за то же время, что и пассивный метод, основанный на ограничениях, алгоритм, предложенный в данной работе, сгенерирует модель, которая пройдет все проверки, в отличие от первой.

Что касается результатов верификации, приведенных в таблице 2, можно заметить, что на формальных моделях КФС, в которых модель ОУ была сгенерирована при помощи пассивных методов, результаты верификации многих свойств часто являются неверными. Это можно объяснить тем, что время сбора трассировок было ограничено максимальным временем работы алгоритма активного обучения, а значит, примеров поведения было собрано слишком мало для построения точной модели. Следовательно, если в симуляционной модели ОУ доступна функция установки в произвольное состояние, алгоритмы активного обучения построят более полную модель ОУ быстрее, чем пассивные методы.

Далее, сравнивая алгоритмы активного обучения, предложенный и L^* , можно заметить, что результирующие модели абсолютно идентичны. Действительно, следуя описанию из раздела 1.4.3.1, можно сделать вывод, что производятся операции, которые по конечному смыслу идентичны тем, что выполняются в предложенном алгоритме, то есть происходят переходы по всем символам входного алфавита из всех обработанных состояний. Разница заключается в том, что в L^* строится промежуточная структура данных – таблица, которая достигает больших размеров за счет оформления каждого перехода как отдельной строки, что является причиной высоких затрат оперативной памяти. Затем, при каждой проверке на замкнутость совершается обход всех строк, что ведет к повышенным временным затратам. Таким образом, алгоритм L^* не стоит выбирать для синтеза детерминированных систем без контекста, так как для таких

систем предлагаемый метод позволяет получить результат гораздо быстрее и с меньшими затратами оперативной памяти.

ВЫВОДЫ ПО ГЛАВЕ 3

1. Алгоритм активного обучения L^* был адаптирован к генерации формальной модели физического компонента КФС: была добавлена поддержка вещественных переменных, а также, так как ОУ не имеет контекста, были убран этап нахождения различающих суффиксов и сравнения автомата-гипотезы с оригинальной моделью.
2. Разработанный метод был опробован на примере генерации формальной модели ОУ экспериментальной системы «лифт», реализованной в NxtStudio. Было проведено сравнение с тремя другими методами автоматической генерации моделей ОУ: пассивным методом, основанным на ограничениях, пассивным методом, основанном на явных состояниях, а также адаптированным алгоритмом активного обучения автоматов L^* .
3. Разработанный метод показал наилучшие результаты по времени генерации формальной модели, при этом модель, полученная таким образом прошла проверку всех темпоральных свойств корректно.

ЗАКЛЮЧЕНИЕ

Был разработан метод формальной верификации промышленных кибер-физических систем в замкнутом цикле, состоящий из трех этапов.

1. Автоматическая генерация формальной модели ОУ при помощи активного обучения в виде детерминированного автомата Мура и ее конвертация в формат верификатора NuSMV.
2. Получение формальной модели КФС путем соединения синтезированной на шаге 1 модели ОУ с формальной моделью контроллера.
3. Верификация требований, записанных в виде LTL свойств, для результирующей модели КФС при помощи верификатора NuSMV.

Основной упор был сделан на шаг 1 – построение формальной модели ОУ. Был использован подход активного обучения, при котором алгоритм итеративно уточняет конструируемую модель, взаимодействуя с симуляционной моделью ОУ. Основой разработанного алгоритма стал обход графа в ширину, адаптированный для работы с физическим компонентом КФС, то есть с системой, содержащей вещественные переменные, дискретизированные с помощью конечного множества смежных интервалов. Для обеспечения построения модели ОУ, содержащего непрерывные переменные были разработаны стратегии решения проблемы обратных петель и проблемы циклов, которые также были применены при адаптации алгоритма L^* . Результаты экспериментов сравнения пассивных и активных методов, показали, что методы активного обучения строят более точную модель, причем предлагаемый метод делает это быстрее всех участвовавших в сравнении алгоритмов.

По результатам работы был сделан доклад «Разработка метода автоматической генерации формальных моделей кибер-физических систем на основе активного обучения» на VII Конгрессе Молодых Ученых (Университет ИТМО, 17 – 20 апреля 2018 г.), а также была принята статья *Ovsiannikova P., Chivilikhin D., Ulyantsev V., Stankevich A., Vyatkin V., Shalyto A. Active learning of formal plant models for cyber-physical systems* на международную конференцию

IEEE 16th International Conference on Industrial Informatics (18 – 20 июля 2018 г., Порту, Португалия). Также по результатам промежуточных исследований была опубликована статья: *Ovsiannikova P., Chivilikhin D., Ulyantsev V. and Shalyto A. Closed-loop verification of a compensating group drive model using synthesized formal plant model* / Proceedings of the 22nd IEEE International Conference on Emerging Technologies and Factory Automation, 2017.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Holzmann, G.J.* The Spin model checker // IEEE Trans. Softw. Eng. — 1997. — Vol. 23, no. 5. — P. 279–295.
2. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking / *A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella.* // Proc. International Conference on Computer-Aided Verification (CAV 2002), ser. LNCS, vol. 2404. Copenhagen, Denmark: Springer, 2002.
3. *Clarke E. M., Grumberg O., Peled D.* Model checking. // MIT press. — 1999. — 330 p.
4. How virtualization, decentralization and network building change the manufacturing landscape: An Industry 4.0 Perspective / *Brettel M., Friederichsen N., Keller M., Rosenberg M.* // International Journal of Mechanical, Industrial Science and Engineering. — 2014. — Vol. 8., no. 1. — P. 37–44.
5. *White L. J.* Software testing and verification // Advances in computers. — 1987. — Vol. 26. — P. 335–391.
6. *Chen T. Y., Leung H., Mak I.* Adaptive random testing // Advances in Computer Science-ASIAN 2004. Higher-Level Decision Making. — Springer, 2004. — P. 320–329.
7. *Apfelbaum L., Doyle J.* Model based testing // Software Quality Week Conference. — 1997. — P. 296–300.
8. *Simao A., Petrenko A., Yevtushenko N.* Generating reduced tests for FSMs with extra states // Testing of Software and Communication Systems. — Springer, 2009. — P. 129–145.
9. *Tretmans J.* Model based testing with labelled transition systems // Formal methods and testing. — Springer, 2008. — P. 1–38.
10. *S. von Styp and L. Yu.* Symbolic model-based testing for industrial automation software // Hardware and Software: Verification and Testing. — Springer, 2013. — P. 78–94.

11. A hybrid approach to cyber-physical systems verification / *Kumar P., Goswami D., Chakraborty S., Annaswamy A., Lampka K., Thiele L.* // Proceedings of the 49th Annual Design Automation Conference. — ACM. 2012. — P. 688–696.
12. Anomaly detection in production plants using timed automata / *Maier A., Vodencarevic A., Niggemann O., Just R., Jaeger M.* // 8th International Conference on Informatics in Control, Automation and Robotics. — 2011. — P. 363–369.
13. AutomationML-the glue for seamless automation engineering / *Drath R., Luder A., Peschke J., Hundt L.* // IEEE International Conference on Emerging Technologies and Factory. — IEEE, 2008. — P. 616–623.
14. *Machado J., Denis B., Lesage J.-J.* Formal Verification of Industrial Controllers: with or without a Plant model? // 7th Portuguese Conference on Automatic Control. — 2006. — P. 341–346.
15. *Preuße S.* Technologies for Engineering Manufacturing Systems Control in Closed Loop. — Logos Verlag Berlin GmbH, 2013. — Vol. 10. — 145 p.
16. Closed-loop modeling in future automation system engineering and validation / *Vyatkin V., Hanisch H.-M., Pang C., Yang C.-H.* // IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews. — 2009. — Vol. 39, no. 1. — P. 17–28.
17. *Preuße S., Lapp H., Hanisch H.* Closed-loop system modeling, validation, and verification // 17th IEEE Conference on Emerging Technologies and Factory Automation. — IEEE, 2012. — P. 1–8.
18. *Pang C., Vyatkin V.* Automatic model generation of IEC 61499 function block using net condition/event systems // 6th IEEE International Conference on Industrial Informatics. — IEEE, 2008. — P. 1133–1138.
19. Modification of the method of generation of control finite-state machines with continuous actions based on training examples / *Buzhinsky I., Kazakov S., Ulyantsev V., Tsarev F., Shalyto A.* // Journal of Computer and Systems Sciences International. — 2015. — Vol. 54, no. 6. — P. 853–865.
20. Matlab. – URL: <https://www.mathworks.com/>

21. *Angluin D.* Queries and concept learning // Machine Learning. — 1988. — Vol. 2, no. 4. — P. 319–342.
22. *Maier A.* Online passive learning of timed automata for cyber-physical production systems // 12th IEEE International Conference on Industrial Informatics. — IEEE, 2014. — P. 60–66.
23. *Buzhinsky I., Vyatkin V.* Automatic inference of finite-state plant models from traces and temporal properties // IEEE Transactions on Industrial Informatics. — IEEE, 2017. — Vol. 13, no. 4. — P. 1521–1530.
24. *Giantamidis G., Tripakis S.* Learning moore machines from input-output traces // FM 2016: Formal Methods. — Cham: Springer International Publishing, 2016. — P. 291–309.
25. Plant trace generation for formal plant model inference: Methods and case study. / *Avdyukhin D., Chivilikhin D., Korneev G., Ulyantsev V., Shalyto A.* // IEEE 15th International Conference on Industrial Informatics. — IEEE, 2017. — P. 746–752.
26. *C. de la Higuera.* Grammatical Inference. Learning automata and grammars // Cambridge University Press, 2010. — P. 215–388.
27. *Rivest R., Schapire R.* Inference of finite automata using homing sequences // Information and Computation. — 1993. — Vol. 103., no. 2. — P. 299–347.
28. *Steffen B., Howar F., Merten M.* Introduction to automata learning from a practical perspective // Formal Methods for Eternal Networked Software Systems. — 2011. — P. 256–296.
29. *Nerode A.* Linear automaton transformations // Proceedings of the American Mathematical Society. — 1958. — Vol. 9, no. 4. — P. 541–544.
30. Реализация предложенного метода. —
URL: <https://github.com/ShakeAnApple/active-learning>
31. NxtControl. — URL: <http://www.nxtcontrol.com>