МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Факультет **Информационных технологий и программирования**

Направление **Прикладная математика и информатика**

Академическая степень **магистр прикладной математики и информатики**

Кафедра «**Компьютерные технологии**» Группа 6538

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему:

«Применение шаблонов требований для формальной спецификации и верификации автоматных программ»

Автор магистерской диссертации: А.А. Клебанов

Научный руководитель: О.Г. Степанов

Санкт-Петербург
2010

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Автоматное программирование [1] – это метод разработки программного обеспечения (ПО), основанный на расширенной модели конечного автомата. В рамках данного подхода программы представляются системой *автоматизированных объектов управления*, логика поведения которых задается системой взаимодействующих управляющих автоматов.

В ряде работ [2 – 5] показано, что к автоматным программам хорошо применима верификация на модели (*Model Checking*) [6, 7]. Ее суть состоит в проверке соответствия модели с конечным числом состояний (*структуры Крипке*) формальной спецификации, заданной в виде набора формул темпоральной логики.

При верификации преимуществом автоматного подхода перед традиционными подходами к разработке ПО является высокая степень автоматизации, так как в автоматных программах модель поведения с конечным числом состояний задается априори. Разработаны методы [8 – 10], позволяющие автоматически преобразовывать как управляющие автоматы в модель, пригодную для верификации, а при наличии ошибки выполнять по построенному верификатором контрпримеру автоматический переход к автоматной программе. Однако как при верификации автоматных программ, так и при верификации программ общего вида, *существует следующая проблема* – необходимость записи формальных требований в виде формул темпоральных логик, работа с которыми достаточно трудоемка и требует значительной математический подготовки.

Эта проблема распадается на две подпроблемы:

1. Трудоемкость работы с темпоральными формулами.
2. Сложность корректного перевода неформальных требований в формальные утверждения.

Целью настоящей работы является разработка единого подхода для решения описанных проблем.

В рамках автоматного подхода в работе [11] задача частично решается использованием контрактов [12]. Хотя контракты являются более простым формализмом (и, как следствие, ошибки в подобной формальной спецификации менее вероятны), рассматриваемый этот подход имеет несколько недостатков. Во-первых, контракты значительно уступают темпоральным логикам в выразительных возможностях. К записи требований они применим только в том случае, когда необходимо специфицировать свойства инвариантности, предусловия или постусловия. Во-вторых, специфицирование требований, распространяющихся на группу состояний, может быть достаточно трудоемким процессом, поскольку для каждого состояния из группы потребуется разработать свою спецификацию. Таким образом, рассматриваемую проблему нельзя считать решенной.

В настоящей работе предлагается подход к записи формальных требований для автоматных программ, который позволяет скрывать сложность темпоральных логик. При этом предлагается записывать требования на подмножестве естественного языка (*controlled natural language*), заданного приводимой в разд. 4.1 формальной грамматикой. Эта грамматика основывается на наборе шаблонов требований [13, 14] (в русскоязычной литературе краткий обзор шаблонов приведен в книге [7]) – обобщенном описании (на естественном и формальном языках) часто встречающихся ограничений на допустимые последовательности состояний в модели системы с конечным числом состояний. Таким образом, для каждого полученного требования существует эквивалентная формальная запись, позволяющая осуществить верификацию.

Актуальность применения шаблонов требований в контексте автоматного программирования отмечается в работе [3]: «… важным является вопрос о *шаблонах* (структуре) темпоральных свойств, наиболее применимых и адекватных для верификации автоматных программ. Наличие таких шаблонов позволяло бы говорить о *классах* темпоральных свойств автоматных моделей, что, несомненно, облегчало бы построение технологической схемы проверки

автоматных программ на корректность относительно спецификации». Однако в указанной работе выделяется только одно требование, которое является частным случаем существующего шаблона, и дальше этот вопрос никак не прорабатывается.

Настоящая работа имеет следующую структуру. В первой главе содержится общий обзор. Приводятся постановка задачи и схема предлагаемого решения, а также анализ существующих работ. Во второй главе описываются шаблоны требований. Ставится и решается задача адаптации существующей системы шаблонов требований для применения в рамках автоматного программирования. В третьей главе приводится анализ применимости шаблонов требований к формальной спецификации автоматных программ. В четвертой главе описывается методика записи верифицируемых требований на подмножестве естественного (русского и английского) языка. В заключении сделаны выводы по работе и рассмотрены направления дальнейших исследований.

Основные положения работы докладывались на следующих конференция и семинарах:

1. VII Межвузовская конференция молодых ученых. Научная школа «Технологии программирования и искусственный интеллект». Секция «Автоматное программирование». СПбГУ ИТМО, 20.04.2010 – 23.04.2010. Диплом за лучший доклад на секции. Подана статья в «Научно-технический вестник СПбГУ ИТМО».
2. Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE’10). Нижний Новгород, 1.06.2010 – 2.06.2010. Опубликован.
3. Workshop on Program Semantics, Specification and Verification: Theory and Applications (PSSV’10) в рамках Computer Science Symposium in Russia (CSR’10). Казань, 14.06.2010 – 15.06.2010. Опубликован.

ГЛАВА 1. ОБЗОР

В целом ряде работе [13 – 19] отмечается, что основным препятствием для широкого внедрения и использования на практике верификации программного обеспечения (ПО) является отсутствие достаточных знаний и опыта спецификации формальных требований на языке темпоральных логик. Таким образом, актуальной задачей является разработка подхода, облегчающего понимание и корректную запись верифицируемых требований.

В первом разделе настоящей главы приводится анализ существующих решений, как для традиционного, так и для автоматного подходов к разработке ПО. Рассматриваются работы и инструментальные средства, в которых используются различные подходы к записи формальных требований. Более детально описываются работы, связанные с шаблонами требований. Во втором разделе главы приводится общая схема предлагаемого решения поставленной задачи с разбиением на подзадачи.

1.1. Анализ существующих работ

1.1.1. Традиционный подход к разработке ПО

Существует ряд подходов, направленных на облегчение работы с формальными требованиями. Большинство из них использует некую графическую нотацию, которая либо отображается в формулы темпоральных логик или их представление (наиболее распространенный подход), либо напрямую поддерживается инструментальным средством. Рассмотрим некоторые из них:

1. Инструментальное средство *Charmy* [20] использует *UML*-подобную нотацию (диаграмму последовательности), называемую диаграммой последовательности свойств (*Property Sequence Charts)*, которая транслируется в автомат Бюхи (автоматное представление *LTL*-формулы) и вместе с моделью системы подается на вход верификатора *Spin*.

2. В работе [21] вводится графическая интервальная логика (*Graphical Interval Logic*) и описываются инструментальные средства для ее поддержки. Отметим, что существует запись шаблонов требований для этой логики.

Еще четыре аналогичных метода описываются в работе [22].

Следует отметить, что подобный подход решает только одну из описанных проблем – за счет интуитивно-понятного графического представления облегчается восприятие требований, однако при этом все еще остается открытой проблема корректной записи их формально.

Альтернативным решением является выделение верифицируемых требований из спецификаций, записанных на естественном языке. Отметим два основных направления [19] – применение технологий обработки естественного языка (*natural language processing*) и применение формальных грамматик, ограничивающих естественный язык. Работа с неограниченным естественным языком может быть достаточно трудоемкой. Поэтому более предпочтительным является второй подход. При этом необходимо подобрать некий «базис» для построения формальной грамматики на его основе.

В работе [15] отмечается, что для записи большинства требований достаточно лишь небольшого набора (темпоральных) конструкций определенного вида. Таким образом, искомым «базисом» и, как следствие, решением описанной проблемы может стать система шаблонов требований. Анализ одной из подобных систем приводится в работах [13, 14]. Классификация требований в группы может производиться исходя из нескольких критериев, таких как, например, структура темпоральной формулы или семантика требования. Последний подход наиболее удобен на практике. Он используется в настоящей работе, а также лежит в основе ряда инструментальных средств.

1.1.2. Инструментальные средства с поддержкой шаблонов требований

Рассмотрим инструментальные средства, в которых внедрена поддержка шаблонов требований.

1.1.2.1. Propel

Инструментальное средство *Propel* [16, 17] предоставляет среду для спецификации требований, основывающихся на шаблонах. Особое внимание уделяется тонким деталям спецификации поведения, возникающим при использовании шаблонов. Пользователю доступны несколько вариантов представления требований – конечный автомат и набор фраз на ограниченном естественном языке. Для облегчения выбора необходимого шаблона и ограничения используются деревья вопросов (*question trees*) [17].

1.1.2.2. Prospec

Инструментальное средство *Prospec* [18] реализует процесс спецификации требований к программам при помощи графического и текстового руководства действиями пользователя. Полученные в результате работы требования могут быть автоматически переведены в следующие формальные нотации: *Future Interval Logic* (*FIL*), *Linear Temporal Logic* (*LTL*), *Meta Event Definition Language* (*MEDL*).

1.1.2.3. Spider

Инструментальное средство *Spider* [19] предоставляет графическую среду для специфицирования и верификации требований к *UML*-моделям. Поддерживаются формализмы, на которые существуют отображения шаблонов требований, а также ряд логик реального времени (для них разработана отдельная система шаблонов требований): *Metric Temporal Logic* (*MTL*), *Timed Computational Tree Logic* (*TCTL*), *Real-time Graphical Interval Logic* (*RTGIL*).

1.1.2.4. Сравнительный анализ инструментальных средств

Все три инструментальных средства объединены идеей пошагового руководства действиями пользователя путем интерактивного «диалога» – предоставления набора вопросов, последовательные ответы на которые помогают выбрать необходимые шаблон и ограничение. Важным отличием между инструментальными средствами являются формализмы, в которые переводятся полученные требования, так как именно ими определяется круг инструментов (например, верификаторов), используемых в дальнейшем. Сводный анализ приводится в табл. 1.

Таблица 1. Сравнительный анализ инструментальных средств

Критерий сравнения / Инструментальное средство	Способ представления формального требования	Режим доступа
Propel	• Конечный автомат. • Набор фраз на ограниченном естественном (английском) языке.	По запросу
Prospec	• FIL. • LTL. • MEDL.	Открытый
Spider	• Набор фраз на ограниченном естественном (английском) языке. • Формализмы, на которые существуют отображения шаблонов	Нет доступа

	требований. • Логики реального времени: *MTL*, *TCTL*, *RTGIL*.	

Достоинством средства *Propel* является возможность записи требований на естественном языке, однако существующие методы отображения не позволяют подать результат работы на вход ни одному верификатору. Средство *Procpec* наоборот лишено этого недостатка, но при этом не обладает поддержкой естественного языка. В качестве недостатка в контексте автоматного программирования можно отметить отсутствие отображения в такой популярный формализм, как *Computation Tree Logic* (*CTL*) (поддержка *CTL* рассматривается, как направление дальнейших исследований). По результатам анализа средство *Spider* представляется наиболее предпочтительным, однако к нему отсутствует доступ. Таким образом, в настоящее время отсутствует инструментальное средство, которое позволило бы удобно и гарантиованно корректно осуществить формальную спецификацию автоматных программ.

1.1.3. Автоматный подход к разработке ПО

Требования к автоматной программе можно разделить на зависящие и независящие от конкретной модели [1]. Зависящие от модели требования проверяются верификаций, а независящие – валидацией. На основе описанной классификации проведем анализ существующих работ.

Независящая от модели проверка представляет собой проверку требований, которые должны выполняться для любого автомата. Они могут иметь различную суть – начиная от достаточно простых, синтаксических требований, заканчивая более сложными, проверяющими «типовую корректность» модели. В работах [3, 5] предлагается набор общих

темпоральных требований для любой иерархической системы взаимодействующих автоматов [23]. Рассмотрим некоторые из них:

- «Тупиковое состояние» – свойство, описывающее невозможность попадания автоматной программы в тупиковое состояние. Формальной записью на *LTL* является выражение `!<>(act == end)` или `[](act != end)`.
- «Забытый автомат» – свойство, выражающее отсутствие путей, при которых с некоторого момента вложенному автомату A_k никогда более не будет передано управление. Формальной записью на *LTL* является выражение `!<>([](auto != k)` или `[]<>(auto == k)`.

Важно отметить, что описанные выше требования по своей сути не являются шаблонами, а представляют собой всего лишь некий готовый набор. Поскольку эти требования не зависят от конкретной модели, то кажется целесообразным встроить их автоматическую проверку в инструментальное средство, используемое для разработки автоматных программ. Таким образом, помимо синтаксической корректности (проверка которой уже реализована в средстве *UniMod* [24]), можно будет гарантировать и некоторую «типовую корректность» модели.

Валидация не гарантирует семантической корректности модели. Для ее проверки необходима верификация. Анализ показывает, что шаблоны требований в контексте верификации автоматной модели упоминаются только в работе [3], однако в не более. Указанный шаблон `[](`p_1 `->` `(`p_2 `U` p_3`))`, где p_i – элементарное высказывание, является частным случаем одного из существующих шаблонов, рассматривающихся в следующей главе.

1.2. Постановка задачи и схема предлагаемого решения

В настоящей работе для записи требований предлагается использовать естественный язык (русский или английский). Как отмечается в работе [19], подобный подход является наиболее предпочтительным и, как следствие,

наиболее распространенным в индустрии. Однако из-за неоднозначности языковых конструкций формализация требований, записанных на неограниченном естественном языке, является трудоемкой задачей, решение которой не всегда может увенчаться успехом. В разд. 1.1.1 рассмотрено несколько вариантов возможных решений, среди которых можно выделить описание ограниченного естественного языка формальной грамматикой. При этом грамматика основывается на шаблонах требований и их распространенных вариантах. Отметим следующие достоинства предложенного в работе [19] подхода:

- Язык грамматики конечен и для любого требования, порожденного грамматикой, существует верифицируемая формальная запись. Эти свойства выполняется по построению грамматики – фактически в процессе порождения определяется все необходимые данные для записи формального требования при помощи одного из шаблонов.
- Простота адаптации под конкретную предметную область. Словарь одной предметной области может отличаться от словаря, используемого в другой. Для адаптации грамматики достаточно изменить лишь терминалы, относящиеся к шаблонам, что не требует математической подготовки, а только знаний предметной области.
- Наличие записи требования на любом из формализмов, запись на которых разработана для системы шаблонов.

Таким образом, решение задачи формальной спецификации автоматных программ распадается на ряд последовательных подзадач:

1. Анализ применимости существующих шаблонов требований для формальной спецификации автоматных программ.
2. В случае положительного результата анализа – адаптация шаблонов требований для использования в рамках автоматного подхода. В случае отрицательного – разработка шаблонов требований, отражающих специфику автоматных программ.

3. Для вывода требований на подмножестве естественного языка создание формальной грамматики, основывающейся на шаблонах требований.
4. Создание методики записи требований.

Поставленные задачи решаются в следующих главах.

Выводы по главе 1

В настоящей главе проанализированы существующие подходы и инструментальные средства к записи верифицируемых требований. Большинство подходов предлагают использовать либо графическую нотацию, либо (ограниченный) естественный язык.

Для дальнейшего использования в настоящей работе выбран подход, основанный на ограниченном естественном языке, который задается формальной грамматикой, основанной на шаблонах требований.

Предложена общая схема для решения задачи формальной спецификации автоматных программ.

ГЛАВА 2. ШАБЛОНЫ ТРЕБОВАНИЙ

В настоящей главе описывается система шаблонов требований [13, 14]. Ставится и решается задача адаптации этой системы для использования в рамках автоматного подхода.

2.1. Описание системы шаблонов требований

Под системой шаблонов требований понимается набор шаблонов, организованных в иерархическую структуру на основе их семантики (рис. 1). Обозначения в скобках после имен шаблонов являются индексом и вводятся в настоящей работе для удобства обращения к конкретному шаблону в дальнейшем.

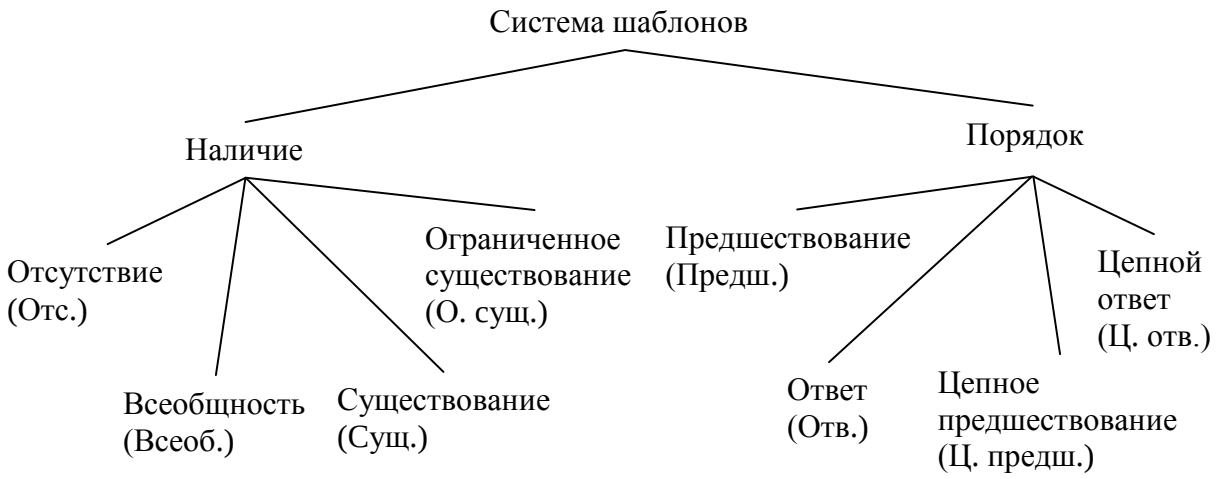

Рис. 1. Система шаблонов требований

Шаблон представляет собой обобщенное описание (формальное и на естественном языке) часто встречающегося требования на допустимую последовательность состояний или событий в автоматной модели системы с конечным числом состояний. Помимо описания некоего аспекта поведения системы приводится запись на языках различных формализмов (таких как *CTL*, *LTL* и т.п.). В настоящей работе используются только те понятия и формализмы, которые применимы в контексте автоматного программирования. Запись шаблонов организована согласно с хорошо известным в индустрии форматом, состоящим из имени шаблона (или списка имен), цели, записи на различных формализмах примера использования и связи с другими шаблонами.

Отметим, что в дальнейшем под наличием или отсутствие некоторого состояния, будем понимать наличие или отсутствие состояния, в котором выполняется заданное требование.

Каждое требование имеет ограничение (*scope*) – ту часть пути исполнения, на котором этот требование должно выполняться. Всего выделяется пять видов ограничений (рис. 2).

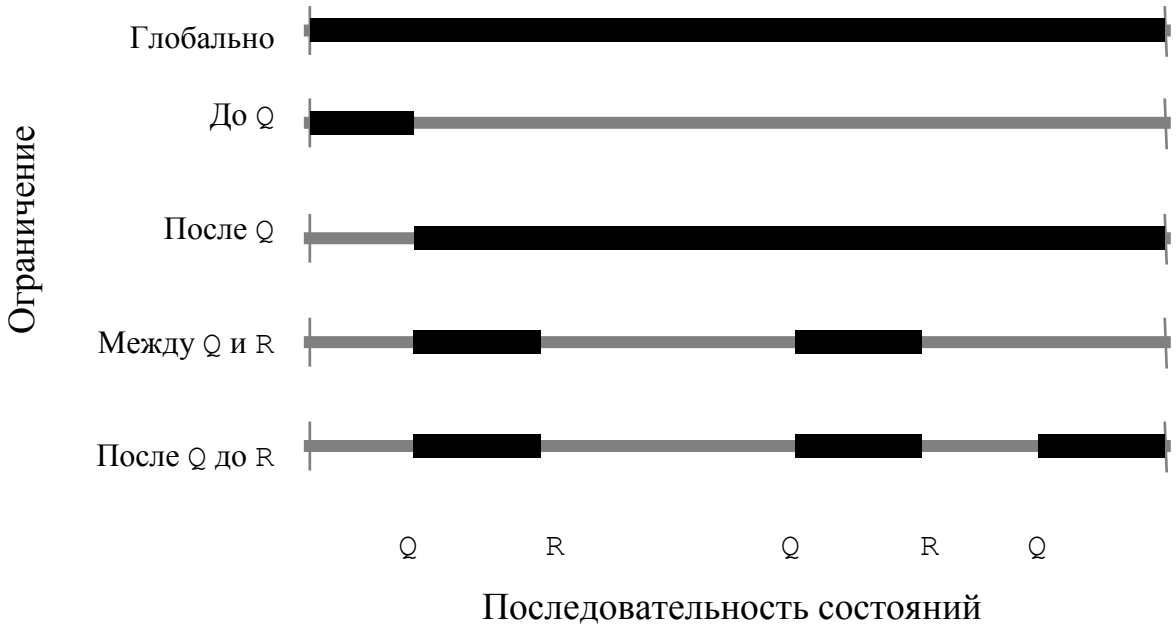

Рис. 2. Виды ограничений

Перечислим их:

- *Глобально* – на всем пути исполнения.
- *До* – на пути до заданного состояния.
- *После* – на пути после заданного состояния.
- *Между* – на пути между двумя заданными состояниями.
- *После-до* – на пути после первого заданного состояния, до наступления следующего. Требование выполняется, даже если второе состояние не наступило.

Стандартно для формализмов, ориентированных на состояние (темпоральные логики), интервал, на котором должно выполняться требование, замкнут на левом конце и открыт на правом. Однако существуют методы, позволяющие открыть левый и замкнуть правый конец интервала.

2.2. Задача адаптации шаблонов требований

Предварительный анализ системы шаблонов требований показал, что большинство примеров (секция «Пример использования») относится к сфере параллельного программирования. Подобные примеры малоприменимы в контексте автоматного программирования. Поэтому возникает задача адаптации системы шаблонов, таким образом, чтобы в качестве примеров использовались утверждение об автоматной модели.

2.3. Адаптированные шаблоны требований

Опишем *адаптированные* (разд. 2.2) шаблоны требований.

Первая группа шаблонов – «Наличие» (номера с первого по четвертый включительно) отвечает за наличие или отсутствие некоторого состояния на пути исполнения системы. Далее приводится описание каждого шаблона из этой группы.

1. Отсутствие

<table>
<tr><td colspan="2">Цель</td><td colspan="2">Используется для описания части пути исполнения системы, в которой нет заданного состояния. Также известен, как «Никогда».</td></tr>
<tr><td rowspan="11">Запись</td><td rowspan="5">LTL</td><td>Ограничение</td><td>Запись</td></tr>
<tr><td>Глобально</td><td>[](!P)</td></tr>
<tr><td>До R</td><td><>R -> (!P U R)</td></tr>
<tr><td>После Q</td><td>[](Q -> [](!P))</td></tr>
<tr><td>Между Q и R</td><td>[]((Q & !R & <>R) -> (!P U R))</td></tr>
<tr><td>После Q до R</td><td>[](Q & !R -> (!P W R))</td></tr>
<tr><td rowspan="5">CTL</td><td>Ограничение</td><td>Запись</td></tr>
<tr><td>Глобально</td><td>AG(!P)</td></tr>
<tr><td>До R</td><td>A[(!P | AG(!R)) W R]</td></tr>
<tr><td>После Q</td><td>AG(Q -> AG(!P))</td></tr>
<tr><td>Между Q и R</td><td>AG(Q & !R -> A[(!P | AG(!R)) W</td></tr>
</table>

			R])
		После Q до R	AG(Q & !R -> A[!P W R])
Пример использования		Этот шаблон может быть применен для описания общих свойств модели в целом или отдельной группы состояний. Для специфицирования свойства безопасности (*safety*) следует использовать это шаблон и ограничение «Глобально». Например, в том случае, когда необходимо выразить свойство вида: «Автомат A никогда не перейдет в состояние s», «В состоянии s никогда не будет обработано событие e» или свойство «Тупиковое состояние» из разд. 1.1.3.	
Связь с другими шаблонами		Этот шаблон двойственен шаблону «Существование». Действительно, во многих спецификациях отрицание существования используется для обозначения отсутствия. Для того чтобы исключить состояния по множественным критериям часто вместо формулы P используют дизъюнкцию соответствующих пропозициональных формул.	

2. Всеобщность

Цель		Используется для описания части пути исполнения системы, в которой содержаться только те состояния, в которых выполняется необходимое требование. Также известен, как «Впредь» и «Всегда».	
Запись	LTL	Ограничение	Запись
		Глобально	[](P)
		До R	<>R -> (P U R)
		После Q	[](Q -> [](P))
		Между Q и R	[]((Q & !R & <>R) -> (P U R))
		После Q до R	[](Q & !R -> (P W R))

	CTL	Ограничение	Запись
		Глобально	`AG(P)`
		До `R`	`A[(P | AG(!R)) W R]`
		После `Q`	`AG(Q -> AG(P))`
		Между `Q` и `R`	`AG(Q & !R -> A[(P | AG(!R)) W R])`
		После `Q` до `R`	`AG(Q & !R -> A[P W R])`
Пример использования		Этот шаблон может быть применен для описания общих свойств модели в целом или отдельной группы состояний. Например, в том случае, когда необходимо выразить свойство вида: «Если автомат находится в состоянии `s`, то верно свойство `S`». При использовании ограничения «Глобально» подстановка темпоральных выражений `<>f` или `AF(f)` в качестве параметра `P` позволяет выразить свойство справедливости (*fairness*). Например, свойство «Забытый автомат» из разд. 1.1.3.	
Связь с другими шаблонами		Этот шаблон тесно связан с шаблонами «Отсутствие» и «Существование». Наличие состояние может рассматриваться как отрицание его отсутствия.	

3. Существование

Цель		Используется для описания части пути исполнения системы, в которой содержатся определенные состояния. Также известен, как «Со временем».	
Запись	LTL	Ограничение	Запись
		Глобально	`<>(P)`
		До `R`	`!R W (P & !R)`
		После `Q`	`[](!Q) | <>(Q & <>P))`

		Между Q и R	[]((Q & !R -> (!R W (P & !R)))
		После Q до R	[]((Q & !R -> (!R U (P & !R)))
	CTL	Ограничение	Запись
		Глобально	AF(P)
		До R	A[!R W (P & !R)]
		После Q	A[!Q W (Q & AF(P))]
		Между Q и R	AG(Q & !R -> A[!R W (P & !R)])
		После Q до R	AG(Q & !R -> A[!R U (P & !R)])
Пример использования		Этот шаблон может быть применен для описания общих свойств модели в целом или отдельной группы состояний. Для специфицирования свойства живости (*liveness*) следует использовать это шаблон и ограничение «Глобально». Например, в том случае, когда необходимо выразить свойство вида: «В некоторый момент будет выполнено выходное воздействие z» или «Автомат A обязательно перейдет в состояние s».	
Связь с другими шаблонами		Этот шаблон двойственен шаблону «Отсутствие». Действительно, во многих спецификациях отрицание отсутствия используется для обозначения существования. Для специфицирования того, что некоторое состояние может присутствовать не более чем определенное число раз, используется шаблон «Ограниченно существование». Для того чтобы ограничить существование состояния по множественным критериям часто вместо формулы P используют дизъюнкцию соответствующих пропозициональных формул.	

4. Ограниченное существование

Цель		Используется для описания части пути исполнения системы, в которой переход между состояниями происходит не более наперед заданного числа раз.	
Запись	LTL	Ограничение	Запись (переход в P-состояния происходит не более двух раз)
		Глобально	`(!P W (P W (!P W (P W []!P))))`
		До R	`<>R -> ((!P & !R) U (R | ((P & !R) U (R | ((!P & !R) U (R | ((P & !R) U (R | (!P U R)))))))))`
		После Q	`<>Q -> (!Q U (Q & (!P W (P W (!P W (P W []!P))))))`
		Между Q и R	`[]((Q & <>R) -> ((!P & !R) U (R | ((P & !R) U (R | ((!P & !R) U (R | ((P & !R) U (R | (!P U R)))))))))))`
		После Q до R	`[](Q -> ((!P & !R) U (R | ((P & !R) U (R | ((!P & !R) U (R | ((P & !R) U (R | (!P W R) | []P)))))))))`
	CTL	Ограничение	Запись (переход в P-состояния происходит не более двух раз)
		Глобально	`!EF(!P & EX(P & EF(!P & EX(P & EF(!P & EX(P)))))))`
		До R	`!E[!R U (!P & !R & EX(P & E[!R U (!P & !R & EX(P & E[!R U (!P & !R & EX(P & !R))])))])]`

		После Q	!E[!Q U (Q & EF(!P & EX(P & EF(!P & EX(P & EF(!P & EX(P)))))))]
		Между Q и R	AG(Q -> !E[!R U (!P & !R & EX(P & E[!R U (!P & !R & EX(P & E[!R U (!P & !R & EX(P & !R & EF(R)))]))]))])
		После Q до R	AG(Q -> !E[!R U (!P & !R & EX(P & E[!R U (!P & !R & EX(P & E[!R U (!P & !R & EX(P & !R))]))]))])
Пример использования	Это шаблон может использоваться для специфицирования требования вида: «Если автомат находится в состоянии s, то выходное воздействие z может быть вызван не более двух раз». Замечание. Шаблон не использовался при спецификации автоматных программ.		
Связь с другими шаблонами	Важно отметить, что этот шаблон не требует существования какого-либо числа заданных состояний. Для этого стоит использовать шаблон «Существование» или его вариации.		

Вторая группа шаблонов – «Порядок» специфицирует порядок состояний во время исполнения системы. Описание приводится ниже.

5. Предшествование

<table>
<tr><td colspan="2">Цель</td><td colspan="2">Используется для описания отношения между парой состояний, где появление одного является обязательным предусловием появления второго. Говорится, что появление второго обусловлено появлением первого.</td></tr>
<tr><td rowspan="11">Запись</td><td rowspan="6">LTL</td><td>Ограничение</td><td>Запись (S предшествует P)</td></tr>
<tr><td>Глобально</td><td>!P W S</td></tr>
<tr><td>До R</td><td><>R -> (!P U (S | R))</td></tr>
<tr><td>После Q</td><td>[]!Q | <>(Q & (!P W S))</td></tr>
<tr><td>Между Q и R</td><td>[]((Q & !R & <>R) -> (!P U (S | R)))</td></tr>
<tr><td>После Q до R</td><td>[](Q & !R -> (!P W (S | R)))</td></tr>
<tr><td rowspan="5">CTL</td><td>Ограничение</td><td>Запись (S предшествует P)</td></tr>
<tr><td>Глобально</td><td>A[!P W S]</td></tr>
<tr><td>До R</td><td>A[(!P | AG(!R)) W (S | R)]</td></tr>
<tr><td>После Q</td><td>A[!Q W (Q & A[!P W S])]</td></tr>
<tr><td>Между Q и R</td><td>AG(Q & !R -> A[(!P | AG(!R)) W (S | R)])</td></tr>
<tr><td>После Q до R</td><td>AG(Q & !R -> A[!P W (S | R)])</td></tr>
<tr><td colspan="2">Пример использования</td><td colspan="2">Требование предшествования часто встречаются в спецификациях автоматных программ. Одним из наиболее частых примеров является описания требования вида: «Если автомат перешел в состояние p, то до этого обязательно произошло S». Другими словами: «Не допускается ситуация, когда автомат перешел в состояние p, но до этого не произошло S».</td></tr>
<tr><td colspan="2">Связь с другими шаблонами</td><td colspan="2">Важно отразить связь между «Предшествование» и «Ответом». «Предшествование» утверждает, что некоторая причина предшествует каждому следствию, а «Ответ» – что</td></tr>
</table>

	результат следует за причиной. Эти требования не эквивалентны – «Ответ» позволяет следствиям происходить без причин (аналогично «Предшествование» позволяет причинам происходить без последующих следствий).

6. Ответ

Цель		Используется для описания причинно-следственной связи между парой состояний. Появление первого (причины) обязательно должно привести к появлению второго (следствия). Так же известен, как «Следует» или «Приводит».	
Запись	LTL	Ограничение	Запись (`S` отвечает `P`)
		Глобально	`[](P -> <>S)`
		До `R`	`<>R -> (P -> (!R U (S & !R))) U R`
		После `Q`	`[](Q -> [](P -> <>S))`
		Между `Q` и `R`	`[]((Q & !R & <>R) -> (P -> (!R U (S & !R))) U R)`
		После `Q` до `R`	`[](Q & !R -> ((P -> (!R U (S & !R))) W R)`
	CTL	Ограничение	Запись (`S` отвечает `P`)
		Глобально	`AG(P -> AF(S))`
		До `R`	`A[((P -> A[!R U (S & !R)]) | AG(!R)) W R]`
		После `Q`	`A[!Q W (Q & AG(P -> AF(S))]`
		Между `Q` и `R`	`AG(Q & !R -> A[((P -> A[!R U (S & !R)]) | AG(!R)) W R])`
		После `Q` до `R`	`AG(Q & !R -> A[(P -> A[!R U (S & !R)]) W R])`
Пример		Требование причинно-следственной связи часто встречаются	

использования	в спецификациях автоматных программ. Одним из наиболее частых примеров является описания требования вида: «Если автомат перешел в состояние `p`, то обязательно произойдет `S`». Существуют варианты шаблона, ограничивающие возможности, когда должен произойти ответ – в будущем (классический вариант) или на следующем шаге (вместо темпорального оператора `F` следует использовать темпоральный оператор `X`).
Связь с другими шаблонами	Аналогично шаблону «Предшествование».

7. Цепное предшествование

Цель		Этот шаблон является масштабируемым. Рассмотрим версию, в которой существует одна причина и два следствия. Состояния, соответствующие следствию, должны различаться, однако одно из состояний может совпадать с тем, в котором выполняется причина.	
Запись	LTL	Ограничение	Запись (`P` предшествует `S,T`)
		Глобально	`(<>(S & o<>T)) -> ((!S) U P))`
		До `R`	`<>R -> ((!(S & (!R) & o(!R U (T & !R)))) U (R | P))`
		После `Q`	`([]!Q) | ((!Q) U (Q & ((<>(S & o<>T)) -> ((!S) U P)))`
		Между `Q` и `R`	`[]((Q & <>R) -> ((!(S & (!R) & o(!R U (T & !R)))) U (R | P)))`
		После `Q` до `R`	`[](Q -> (!(S & (!R) & o(!R U (T & !R))) U (R | P) | [](!(S &`

			o<>T))))
		Ограничение	Запись (S, T предшествует P)
		Глобально	<>P -> (!P U (S & !P & o(!P U T)))
		До R	<>R -> (!P U (R | (S & !P & o(!P U T))))
		После Q	([]!Q) | (!Q U (Q & <>P -> (!P U (S & !P & o(!P U T))))
		Между Q и R	[](Q & <>R) -> (!P U (R | (S & !P & o(!P U T)))))
		После Q до R	[](Q -> (<>P -> (!P U (R | (S & !P & o(!P U T))))))
	CTL	Ограничение	Запись (P предшествует S, T)
		Глобально	!E[!P U (S & !P & EX(EF(T)))]
		До R	!E[(!P & !R) U (S & !P & !R & EX(E[!R U (T & !R)]))]
		После Q	!E[!Q U (Q & E[!P U (S & !P & EX(EF(T)))])]
		Между Q и R	AG(Q -> !E[(!P & !R) U (S & !P & !R & EX(E[!R U (T & !R & EF(R))]))])
		После Q до R	AG(Q -> !E[(!P & !R) U (S & !P & !R & EX(E[!R U (T & !R)]))])
		Ограничение	Запись (S, T предшествует P)
		Глобально	!E[!S U P] & !E[!P U (S & !P & EX(E[!T U (P & !T)]))]

		До R	!E[(!S & !R) U (P & !R)] & !E[(!P & !R) U (S & !P & !R & EX(E[(!T & !R) U (P & !T & !R)]))]
		После Q	!E[!Q U (Q & E[!S U P] & E[!P U (S & !P & EX(E[!T U (P & !T)]))])]
		Между Q и R	AG(Q -> !E[(!S & !R) U (P & !R & EF(R))] & !E[(!P & !R) U (S & !P & !R & EX(E[(!T & !R) U (P & !T & !R & EF(R))]))])
		После Q до R	AG(Q -> !E[(!S & !R) U (P & !R)] & !E[(!P & !R) U (S & !P & !R & EX(E[(!T & !R) U (P & !T & !R)]))])
Пример использования	Это шаблон может использоваться для специфицирования требований вида: «Если было появилось событие e, то оно будет обработано и автомат перейдет в состоние s». Замечание. Этот шаблон еще не использовался при спецификации автоматных программ.		
Связь с другими шаблонами	Этот шаблон является обобщением шаблона «Предшествование».		

8. Цепной ответ

Цель	Этот шаблон является масштабируемым. Рассмотрим версию, в которой существует одна причина и два следствия. Используется для описания отношения между состоянием P и последовательностью состояний (S, T), в которой

		появлению S с последующим появлением T предшествует появление P.	
Запись	LTL	Ограничение	Запись (S, T отвечают на P)
		Глобально	[] (P -> <>(S & o<>T))
		До R	<>R -> (P -> (!R U (S & !R & o(!R U T)))) U R
		После Q	[] (Q -> [] (P -> (S & o<> T)))
		Между Q и R	[] ((Q & <>R) -> (P -> (!R U (S & !R & o(!R U T)))) U R)
		После Q до R	[] (Q -> (P -> (!R U (S & !R & o(!R U T)))) U (R | [] (P -> (S & o<> T))))
		Ограничение	Запись (P отвечает на S, T:)
		Глобально	[] (S & o<> T -> o(<>(T & <> P)))
		До R	<>R -> (S & o(!R U T) -> o(!R U (T & <> P))) U R
		После Q	[] (Q -> [] (S & o<> T -> o(!T U (T & <> P))))
		Между Q и R	[] ((Q & <>R) -> (S & o(!R U T) -> o(!R U (T & <> P))) U R)
		После Q до R	[] (Q -> (S & o(!R U T) -> o(!R U (T & <> P))) U (R | [] (S & o(!R U T) -> o(!R U (T & <> P)))))
	CTL	Ограничение	Запись (S, T отвечают на P)
		Глобально	AG(P -> AF(S & AX(AF(T))))
		До R	!E[!R U (P & !R & (E[!S U R] |

			E[!R U (S & !R & EX(E[!T U R]))]))]
		После Q	!E[!Q U (Q & EF(P & (EG(!S) | EF(S & EX(EG(!T))))))]
		Между Q и R	AG(Q -> !E[!R U (P & !R & (E[!S U R] | E[!R U (S & !R & EX(E[!T U R]))]))])
		После Q до R	AG(Q -> !E[!R U (P & !R & (E[!S U R] | EG(!S & !R) | E[!R U (S & !R & EX(E[!T U R] | EG(!T & !R)))]))])
		Ограничение	Запись (P отвечает на S, T:)
		Глобально	!EF(S & EX(EF(T & EG(!P))))
		До R	!E[!R U (S & !R & EX(E[!R U (T & !R & E[!P U R])]))]
		После Q	!E[!Q U (Q & EF(S & EX(EF(T & EG(!P)))))]
		Между Q и R	AG(Q -> !E[!R U (S & !R & EX(E[!R U (T & !R & E[!P U R])]))])
		После Q до R	AG(Q -> !E[!R U (S & !R & EX(E[!R U (T & !R & (E[!P U R] | EG(!P & !R)))]))])
Пример использования		Это шаблон может использоваться для спецификации требований вида: «Если было обработано событие e, то автомат перейдет в состояние s, и будет верно P». Замечание. Этот шаблон не использовался при спецификации автоматных программ.	

Связь с другими шаблонами	Важно отметить, что это шаблон не гарантирует, что каждое вхождение результирующей последовательности означает наличие соответствующего условия.

Выводы по главе 2

В настоящей главе описана система шаблонов требований. Проведена ее адаптация для использования в рамках автоматного подхода.

Шаблоны требований упрощают процесс формального специфицирования требований и уменьшают число потенциальных ошибок в полученной спецификации. Адаптированные шаблоны переносят эти достоинства на автоматный подход, позволяя быстро и корректно записывать требования к автоматным программам.

ГЛАВА 3. ПРИМЕНИМОСТЬ ШАБЛОНОВ ТРЕБОВАНИЙ К ФОРМАЛЬНОЙ СПЕЦИФИКАЦИИ АВТОМАТНЫХ ПРОГРАММ

В настоящей главе рассматривается вопрос применимости шаблонов требований к формальной спецификации автоматных программ. Поскольку система шаблонов была получена на основе требований к программам общего вида, то необходимо проанализировать возможность ее применение для формальной спецификации автоматных программам.

3.1. Критерии соответствия шаблона требованию

На основе работы [14] опишем несколько критериев соответствия шаблона требованию, которые используются при дальнейшем анализе:

- Требование соответствует шаблону – исходное формальное требование получается подстоновкой соответсвующих параметров в шаблон.
- Спецификация требования является формальным эквивалентом одного из шаблонов. В этом случае приводится доказательство эквивалентности.
- Спецификация требования может быть получена из шаблона подстановкой темпоральной формулы вместо параметра (см. пример использования шаблона «Всеобщность»).
- Спецификация требования в источнике ошибочна. При этом корректная спецификация соответствует одному из шаблонов.

Отметим, что в ряде случаев в формальной спецификации отсутсвуют темпоральные операторы «Глобально» (`AG` в *CTL* и `[]` в *LTL*), если тело оператора соответствует шаблону, то засчитывается совпадение.

3.2. Анализ применимости

Рассмотрим вопрос применимости описанных выше шаблонов требований к формальной спецификации автоматных программ. Для этого проанализируем требования к различным программам, разработанным в

СПбГУ ИТМО, Ярославском государственном университете им. П. Г. Демидова и в ОАО Концерн «НПО «Аврора». Большинство проанализированных требований доступно на сайте [25]. Результат анализа приводится в приложении. Пример промежуточной организации результатов – в табл. 2.

Таблица имеет следующую структуру:

- Столбец «№» – порядковый номер требования.
- Столбец «Требование» – исходное требование на естественном языке. В некоторых случаях может отсутствовать.
- Столбец «Исходная запись» – формальная запись требования на *LTL* или *CTL*. В некоторых случаях может быть неединственной.
- Столбец «Шаблон, Ограничение» – шаблон (краткое имя шаблона, введенное на рис. 1) и ограничение, позволяющее выразить требование. Приводятся значения параметров шаблона для подстановки, в некоторых случаях и формальное доказательство эквивалентности;
- Столбец «Источник» – номер источника, из которого было взято требования, в списке источников настоящей работы.

Таблица 2. Пример промежуточной организации результатов

№	Требование	Исходная запись	Шаблон, ограничение	Источник
84	Автомат `A2` никогда не попадет в состояние `s3`.	`G !(A2 in state s3)`	Отсутствие, Глобально `[](!P)`, `P: A2 in state s3`	36

Всего было рассмотрено 118 требований и их вариантов из порядка 20-и источников. Сводные результаты анализа приводятся в табл. 3.

Таблица 3. Сводные результаты анализа

Ограничение / Шаблон	Глобально	Между Q и R	После Q	∑ (по шаблонам)
Отсутствие	13	5	7	25
Всеобщность	37			37
Существование	6	1	1	8
Предшествование	9		6	15
Ответ	19		1	20
∑ (по ограничениям)	84	6	15	105

Отметим, что около 85% требований покрывается шаблонами. Процентное соотношение между использованными шаблонами приведено на рис. 3.

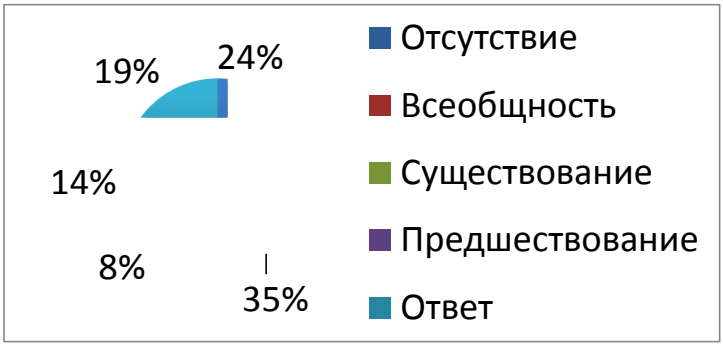

Рис. 3. Процентное соотношение между использованными шаблонами

Процентное соотношение между использованными ограничениями представлено на рис. 4.

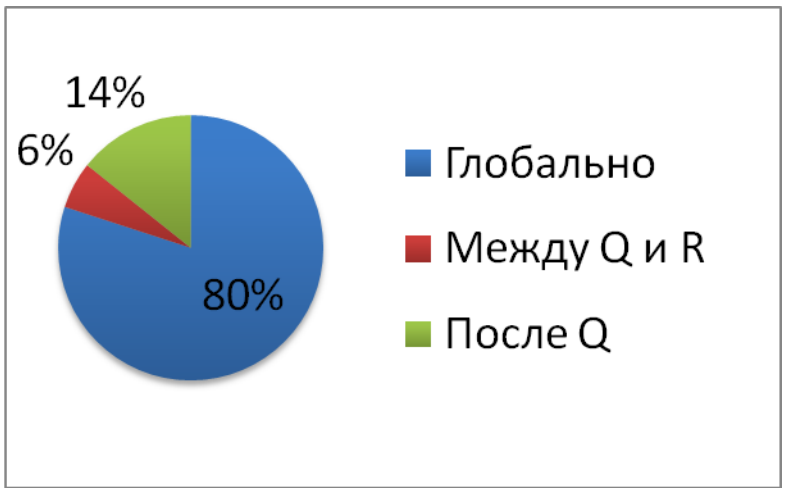

Рис. 4. Процентное соотношение между использованными ограничениями в шаблонах

Большой процент требований, покрываемых шаблонами, доказывает утверждение из работы [26]: «…при проектировании системы обычно создаются те же естественные абстракции, что используются в спецификации. Поэтому процесс переноса словесных утверждений в утверждения автоматной модели обычно не составляют труда». Таким образом, при правильном проектировании формальная спецификация автоматных программ не отличается от спецификации программ общего вида, но, возможно, является более легким, за счет четко выделенных на этапе проектирования сущностей. Однако в случае плохо спроектированной модели процесс спецификации значительно усложняется, как это показано в следующем разделе.

3.3. Анализ требований, которые не удалось выразить при помощи шаблонов

Предварительный анализ позволяет предположить, что существует несколько причин, которые не позволяют выразить требования при помощи шаблонов или вынуждают использовать не совсем подходящий шаблон. К этим причинам относятся:

- Ограниченность системы шаблонов требований. Авторы оригинальной системы шаблонов требований не ставили цели создания полной системы (системы, гарантировано позволяющей выразить любое требование). Поэтому данная причина возможна в некоторых особых случаях.

- Особенности конкретной автоматной модели приложения. Невозможность подобора параметров для шаблона может быть косвенным признаком плохо спроектированной модели. В этом случае проблема по возможности должна решаться перепроектированием системы или ее отдельной части.
- Особенности алгоритма преобразования автоматной модели в структуру Крипке. Во многих алгоритмах в результате преобразования добавляются искуственные состояния, которые отсутствуют в исходной модели, что вынуждает учитывать это при спецификации. Подобная проблема отмечается и в работе [27].

В качестве примера второй причины рассмотрим требование № 30 (табл. 4), а третьей – требование № 63 (табл. 4):

Таблица 4. Примеры требований, невыразимых шаблонами

№	Требование	Исходная запись	Шаблон, ограничение	Источник
30	Если ресурс заняли, то он несвободен, пока его не освободят.	$o_1.x_1$ W $o_1.z_1$ & G ($o_1.z_2$ -> ($o_1.x_1$ W $o_1.z_1$) & $o_1.z_1$ -> (!o1.x1 W o1.z2))	-	27
63	Непосредственно после открытия двери загорается лампочка.	e_1 U z_1	Всеобщность, Глобально [](P), P: e_1 U z_1	11

Для первого требования по словесному описанию подходит шаблон «Отсутствие» с ограничением «Между Q и R» (если считать, что ресурс когда-либо освободят) или «После Q до R» – в противном случае. Формальной записью является выражение: []((Q & !R & <>R) -> (!P U R)), где

- Q: Ресурс заняли.

- `P`: Ресурс свободен.
- `R`: Ресур с освободили.

Однако не удается подобрать такие значения параметров `Q`, `P` и `R`, для того чтобы доказать формальную эквивалентность.

Для второго требования подходит вариант шаблона «Ответ» (на следующем шаге) с ограничением «Глобально» (`G(`e_1 `->` `X(`z_1`))`). Однако эту формулу нельзя будет использовать после преобразования автоматной модели в модель, пригодную для верификации. Между смежными состояниями исходной модели (левая часть рис. 5), где истинны e_1 и z_1 соответственно добавятся новые состояния, где z_1 ложно (правая часть рис. 5).

Рис. 5. Преобразование автоматной модели в структуру Крипке

Выводы по главе 3

По результатам анализа можно сделать вывод, что шаблоны требований применимы в контексте автоматного программирования. Детальный анализ требований, которые не выражаются шаблонами остается важной открытой проблемой.

Использование относительно простых шаблонов и ограничений можно объяснить тем, что источником требований в большинстве случаев являлись работы студентов и аспирантов, в которых описывались новые разработки, для апробации которых и выбирались наиболее простые требования.

Стоит отметить, что отсутствие общих конструкций в высказываниях об автоматных моделях не позволило обнаружить какие-либо шаблоны на более близком к автоматной модели уровне. Подобные шаблоны позволили бы создать набор высказываний об автоматной модели, что вместе с существующим набором шаблонов требований значительно снизило бы число ошибок в процессе спецификации.

ГЛАВА 4. ЗАПИСЬ ВЕРИФИЦИРУЕМЫХ ТРЕБОВАНИЙ НА ОГРАНИЧЕННОМ ЕСТЕСТВЕННОМ ЯЗЫКЕ

В настоящей главе рассматривается методика записи верифицируемых требований на подмножестве естественного языка. Описываются две грамматики для записи требований на подмножестве русского и английского языков, приводится неформальный алгоритм записи. Глава завершается примерами, иллюстрирующими предложенный подход.

4.1. Формальные грамматики

Грамматики, задающие правила вывода требований на русском и английском языках, приводятся в табл. 5, 6 соответственно. Стартовым нетерминальным символом первой грамматики является `<требование>`, второй – `<requirement>`. Нетерминальные символы грамматик соответствуют системе шаблонов требований (включены вариации шаблона «Ответ»). Они используются для выбора одного из возможных ограничений и шаблонов. Терминальные символы позволяют получить конечное представление требования на подмножестве естественного языка. Подобная конструкция грамматики гарантирует, что для каждого требования будет существовать формальная запись (поскольку однозначно определены и шаблон, и ограничение). `Моноширинным шрифтом` выделены указатели места заполнения шаблона реальными требованиями.

Таблица 5. Грамматика для вывода верифицируемых требований на ограниченном русском языке

<требование>	::= <ограничение> <шаблон>
<ограничение>	::= «Для любого состояния верно, что»
	| «До состояния, в котором `Q`, верно что»
	| «После состояния, в котором `Q`, верно что»
	| «Между состоянием, в котором `Q`, до состояния, в котором `R`, верно что»
	| «После состояния, в котором `Q`, до состояния, в

	котором R, верно что»
<шаблон>	::= <отсутствие> | <всеобщность> | <существование> | <предшествование> | <ответ> | <ответ на следующем шаге>
<отсутствие>	::= «никогда не выполняется P».
<всеобщность>	::= «всегда выполняется P».
<существование>	::= «когда-нибудь выполнится P».
<предшествование>	::= «если выполнено P, то до этого было выполнено S».
<ответ>	::= «если выполнено P, то когда-нибудь выполнится S».
<ответ на следующем шаге>	::= «если выполнится P, то в следующем состоянии выполнится S».

В рамках проведенных исследований была также предложена аналогичная грамматика и для английского языка (табл. 6).

Таблица 6. Грамматика для вывода верифицируемых требований на ограниченном английском языке

<requirement>	::= <scope> <pattern>
<scope>	::= «For all the states holds that» | «Before the state where Q, holds that» | «After the state where Q, holds that» | «Between the states where Q and R, holds that» | «After the state where Q, before the state where R, holds that»
<pattern>	::= <absence> | <universality> | <existence> | <precedence> | <response> | <next step response>
<absence>	::= «never P.»
<universality>	::= «always P.»
<existence>	::= «eventually P.»
<precedence>	::= «always if P, then previously S.»

<response>	::= «always if P, then eventually S.»
<next step response>	::= «always if P, then on the next step S.»

4.2. Методика записи требований

Для получения формального требования на *LTL* или *CTL* с использованием шаблонов требований необходимо определить следующие данные:

1. Элементарные формальные требования для шаблона и ограничения. В случае ограничения это некоторые высказывание о модели, в случае шаблона – помимо высказывания о модели возможна и простая темпоральная формула.
2. Шаблон.
3. Ограничение.

Таким образом, структура шаблона определяет неформальный алгоритм для записи требований:

1. Выделить элементарные формальные требования из словесного описания.
2. Выбрать шаблон и ограничение на основе словесного описания.
3. Выполнить порождение, подставить требование, полученное на шаге1 вместо параметра.
4. Используя данные шагов 1 и 2, получить формальную запись для верификации.

Отметим, что шаги 1 и 2 выполняются пользователем, при этом для реализации шага 2 может быть разработана интерактивная (диалоговая) система поддержки, аналогичная описанным в разд. 1.1.2. Шаги 3 и 4 могут быть легко автоматизированы.

4.3. Примеры записи требований

В качестве примера рассмотрим *реальные* требования к различным системам управления, реализованным при помощи автоматного подхода, и выполним по шагам алгоритм, описанный в разд. 4.2. Формальные

доказательства эквивалентности полученных в примерах и оригинальных требований приводятся в приложении. Отметим, что полученные формальные требования в дальнейшем используются при верификации на модели.

4.3.1. Шаблон «Отсутствие»

Рассмотрим требование к системе управления кофеваркой, которое приводится в работе [5]: «Система управления кофеваркой никогда не попадет в такое состояние, в котором она не реагирует ни на события системного таймера, ни на нажатие кнопок «ОК» и «С».

Шаг 1. В автоматной модели кофеварки требованию «никак не реагирует ни на события системного таймера, ни на нажатие кнопок «ОК» и «С» соответствует предикат `act = end`.

Шаг 2. Наречие «никогда» подсказывает, что должен быть использован шаблон «Отсутствие» с ограничением «Глобально».

Шаг 3. Выполним порождение:

<требование> → <ограничение> <шаблон> → Для любого состояния верно, что <шаблон> → Для любого состояния верно, что <отсутствие> → Для любого состояния верно, что никогда не выполняется `P`.

Шаг 4. Подставив вместо `P` реальное требование, получим искомое формальное требования на естественном языке: «Для любого состояния верно, что никогда не выполняется `act = end`». Этому требованию на языках *CTL* и *LTL* соответствуют выражения `AG!(act = end)` и `[]!(act = end)`.

4.3.2. Шаблон «Всеобщность»

Рассмотрим требование к системе управления кофеваркой, которое приводится в работе [5]: «Если автомат A_0 перешел в начальное состояние 0 «Готов к работе», следовательно, все вложенные автоматы уже находятся в своих начальных состояниях».

Шаг 1. В автоматной модели кофеварки заданному требованию соответствует предикат $y_0 = 0 \rightarrow y_1 = 0 \;\&\; y_2 = 0 \;\&\; y_{31} = 0 \;\&\; y_{32} = 0$.

Шаг 2. Отсутствие каких-либо ограничений подсказывает, что должен быть использован шаблон «Всеобщность» с ограничением «Глобально».

Шаг 3. Выполним порождение:

<требование> → <ограничение> <шаблон> → Для любого состояния верно, что <шаблон> → Для любого состояния верно, что <всеобщность> → Для любого состояния верно, что всегда выполняется `P`.

Шаг 4. Подставив вместо `P` реальное требование, получим искомое формальное требования на естественном языке: «Для любого состояния верно, что всегда выполняется y_0 = 0 -> y_1 = 0 & y_2 = 0 & y_{31} = 0 & y_{32} = 0». Этому требованию на языках *CTL* и *LTL* соответствуют выражения AG(y_0 = 0 -> y_1 = 0 & y_2 = 0 & y_{31} = 0 & y_{32} = 0) и [](y_0 = 0 -> y_1 = 0 & y_2 = 0 & y_{31} = 0 & y_{32} = 0).

4.3.3. Шаблон «Существование»

Рассмотрим требование к системе управления банкоматом, которое приводится в работе [28]: «Банкомат когда-нибудь выдаст деньги».

Шаг 1. Согласно автоматной модели банкомата деньги выдаются в состоянии автомата, описывающего клиентскую часть, под номером 11 – `stateAClient == 11`.

Шаг 2. Наречие «когда-нибудь» подсказывает, что должен быть использован шаблон «Существование» с ограничением «Глобально».

Шаг 3. Выполним порождение:

<требование> → <ограничение> <шаблон> → Для любого состояния верно, что <шаблон> → Для любого состояния верно, что <существование> → Для любого состояния верно, что когда-нибудь выполнится `P`.

Шаг 4. Подставив вместо `P` реальное требование, получим искомое формальное требования на естественном языке: «Для любого состояния верно, что когда-нибудь выполнится `stateAClient == 11`». Этому требованию на языках *CTL* и *LTL* соответствуют выражения `AF(stateAClient == 11)` и `<>(stateAClient == 11)`.

4.3.4. Шаблон «Предшествование»

Рассмотрим требование к системе управления банкоматом, которое приводится в работе [28]: «Банкомат выдает деньги только после авторизации. Проверяется отсутствие последовательности действий, при которой пользователь получает деньги до ввода правильного кода».

Шаг 1. Согласно автоматной модели банкомата выдача денег соответствует выходному воздействию P: $o_1.z_{10}$, а авторизация (ввод правильного *PIN*-кода) – S: e_{10}.

Шаг 2. Семантика требования подсказывает, что должен быть использован шаблон «Предшествование» с ограничением «Глобально».

Шаг 3. Выполним порождение:

<требование> → <ограничение> <шаблон> → Для любого состояния верно, что <шаблон> → Для любого состояния верно, что <предшествование> → Для любого состояния верно, что если выполнено P, то до этого было выполнено S.

Шаг 4. Подставив вместо P и S реальные требования, получим искомое формальное требования на естественном языке: «Для любого состояния верно, что если выполнено $o_1.z_{10}$, то до этого было выполнено e_{10}». Этому требованию на языках CTL и LTL соответствуют выражения A (!$o_1.z_{10}$ W e_{10}) и !$o_1.z_{10}$ W e_{10}.

4.3.5. Шаблон «Ответ»

Рассмотрим требование к системе управления банкоматом, которое приводится в работе [28]: «Банкомат не захватывает карту. Банкомат всегда возвращает карту после выдачи денег или баланса».

Шаг 1. Согласно автоматной модели банкомата выдаче денег или баланса соответствует предикат P: $A_1 = s_1$ | Action = z_5, а возврату карты – S: $A_2 = s_2$.

Шаг 2. Семантика требования подсказывает, что должен быть использован шаблон «Ответ» с ограничением «Глобально».

Шаг 3. Выполним порождение:

<требование> → <ограничение> <шаблон> → Для любого состояния верно, что <шаблон> → Для любого состояния верно, что <ответ> → Для любого состояния верно, что если выполнено P, то когда-нибудь выполнится S.

Шаг 4. Подставив вместо P и S реальные требования, получим искомое формальное требования на естественном языке: «Для любого состояния верно, что если выполнено $A_1 = s_1 \mid Action = z_5$, то когда-нибудь выполнится $A_2 = s_2$». Этому требованию на языках *CTL* и *LTL* соответствуют выражения $AG((A_1 = s_1 \mid Action = z_5) \rightarrow AF(A_2 = s_2))$ и $[]((A_1 = s_1 \mid Action = z_5) \rightarrow <>(A_2 = s_2))$.

Выводы по главе 4

В настоящей главе были описаны формальные грамматики и методика для записи верифицируемых требований на подмножестве естественного языка (русского и английского).

Для каждого из пяти шаблонов, наиболее применимых в контексте автоматного программирования (разд. 3.2), рассмотрен пример использования предложенного подхода для реального требования.

ЗАКЛЮЧЕНИЕ

Запись требований в виде формул темпоральной логики является неотъемлемой частью верификации на модели. В настоящей работе в рамках автоматного программирования предложен подход, который, во-первых, значительно упрощает этот процесс, а, во-вторых, минимизирует число потенциальных ошибок в самой спецификации.

Решены следующие задачи:

- исследован вопрос применимости шаблонов требований для формальной спецификации автоматных программ;
- существующая система шаблонов адаптирована для использования в рамках автоматного подхода;
- разработаны формальные грамматики для вывода требований на подмножестве русского и английского языков;
- разработана методика записи верифицируемых требований на подмножестве естественного языка.

Остается ряд открытых вопросов, которые можно выделить в качестве направлений дальнейших исследований.

С теоретической стороны, это, во-первых, детальное исследование тех требований, которые не удалось записать при помощи шаблонов. Подобное исследование отсутствует и в работе [14]. Поэтому оно может дать результаты как в рамках автоматного, так и традиционного подходов. Во-вторых, может быть исследована и обратная задача – получение записи на естественном языке для заданной темпоральной формулы. Наконец, может оказаться целесообразным исследовать вопрос применимости интервальных темпоральных логик (на которых существует запись шаблонов) в контексте автоматного программирования.

С практической стороны, стоит задача инструментальной поддержки и интеграции с существующими средствами разработки предложенного подхода.

Как показано в разд. 1.1.2, ни одно из существующих инструментальных средств полностью не решает задачи спецификации автоматных программ.

СПИСОК ИСТОЧНИКОВ

1. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. – СПб.: Питер, 2009. – 176 с. http://is.ifmo.ru/books/_book.pdf
2. *Виноградов Р. А., Кузьмин Е. В., Соколов В. А.* Верификация автоматных программ средствами CPN/Tools // Моделирование и анализ информационных систем. Т. 13. 2006. № 2, с. 4–15.
3. *Васильева К. А., Кузьмин Е. В.* Верификация автоматных программ с использованием LTL // Моделирование и анализ информационных систем. Т. 14. 2007. № 1, с. 3–14.
4. *Кузьмин Е. В., Соколов В. А.* О некоторых подходах к верификации автоматных программ / Сборник докладов семинара Go4IT – шаг к новым технологиям Интернета. М.: Институт системного программирования, 2007, с. 43–48.
5. *Кузьмин Е. В., Соколов В. А.* Моделирование, спецификация и верификация «автоматных» программ // Программирование. 2008. № 1, с. 38–60.
6. *Кларк Э., Грамберг О., Пелед Д.* Верификация моделей программ. Model Checking. М.: Изд-во МЦНМО, 2002. – 416 с.
7. *Карпов Ю. Г.* Model Checking. Верификация параллельных и распределенных программных систем. СПб.: БХВ, 2009. – 551 с.
8. *Гуров В. С., Яминов Б. Р.* Технология верификации автоматных моделей программ без их трансляции во входной язык верификатора / Тезисы научно-технической конференции «Научное программное обеспечение в образовании и научных исследованиях». СПбГУ ПУ. 2008, с. 36–40. http://is.ifmo.ru/download/2008-02-24_jaminov_verifikazija.pdf
9. *Лукин М. А., Шалыто А. А.* Автоматизация верификации визуальных автоматных программ / Материалы XV Международной научно-методической конференции «Высокие интеллектуальные технологии и

инновации в образовании и науке». СПбГПУ. 2008, с. 296, 297. http://is.ifmo.ru/download/2008-02-25_politech_tezis.pdf

10. *Kurbatsky E.* Verification of Automata-Based Programs / Proceedings of the Second Spring Young Researchers Colloquium on Software Engineering. 2008. V. 2, pp. 15–17. http://is.ifmo.ru/verification/_kurbatsky_syrcse.pdf
11. *Степанов О. Г.* Методы реализации автоматных объектно-ориентированных программ. Диссертация на соискание ученой степени кандидата технических наук. СПбГУ ИТМО, 2009. http://is.ifmo.ru/disser/stepanov_disser.pdf
12. *Мейер Б.* Объектно-ориентированное конструирование программных систем. М.: Русская Редакция, 2005. – 1204 с.
13. *Dwyer M. B., Avrunin G. S., Corbett J. C.* Property Specification Patterns for Finite-state Verification / Proceedings of the 2nd Workshop on Formal Methods in Software Practice. 1998.
14. *Dwyer M. B., Avrunin G. S., Corbett J. C.* Patterns in Property Specifications for Finite-state Verification / Proceedings of the 21st International Conference on Software Engineering. 1999.
15. *Manna Z., Pnueli A.* Tools and rules for the practicing verifier. Technical Report STAN-CS-90-1321, Stanford University, July 1990. Appeared in Carnegie Mellon Computer Science: A 25 year Commemorative, ACM Press, 1990.
16. *Smith R. L., Avrunin G. S., Clarke L. A., Osterweil L. J.* Propel: An approach supporting property elucidation / Proc. of the 24th Int. Conf. on Software Engineering. ACM Press. 2002, p.11–21.
17. *Cobleigh R. L., Avrunin G. S., Clarke L. A.* User guidance for creating precise and accessible property specifications / Proceedings of the Fourteenth ACM SIGSOFT Symposium on the Foundations of Software Engineering. NY: Association of Computing Machinery. 2006, p. 208–218.

18. *Mondragon O., Gates A., Roach, S.* Prospec: Support for Elicitation and Formal Specification of Software Properties // Proc. of Runtime Verification Workshop. ENTCS. 89(2), 2004.
19. *Konrad S., Cheng B. H. C.* Facilitating the Construction of Specification Pattern-based Properties / Proceedings of the IEEE International Requirements Engineering Conference (RE'05). Paris: 2005.
20. *Inverardi P., Muccini H., Pelliccione P.* CHARMY: An Extensible Tool for Architectural Analysis / ESEC-FSE'05. The fifth joint meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering. Research Tool Demos. Portugal, 2005.
21. *Dillon L. K., Kutty G., Moser L. E., Melliar-Smith P. M., Ramakrishna Y. S.* A graphical interval logic for specifying concurrent systems // ACM Transactions on Software Engineering and Methodology. 3(2). April 1994, pp.131–165.
22. Oh Y. Visual Approaches for System Property Specification. 2003. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.132.3067
23. *Кузьмин Е. В.* Иерархическая модель автоматных программ // Моделирование и анализ информационных систем. Т. 13. 2006. №1, с. 27–34.
24. *Гуров В. С., Мазин М.А., Нарвский А. С., Шалыто А. А.* UML. SWITCH-технология. Eclipse // Информационно-управляющие системы. 2004. № 6, с. 12–17. http://is.ifmo.ru/works/uml-switcheclipse/
25. Веб-сайт кафедры «Технологии программирования». http://is.ifmo.ru/
26. Разработка технологии верификации управляющих программ со сложным поведением, построенных на основе автоматного подхода. Выбор направления исследований и базовых компонентов. СПбГУ ИТМО, 2007. http://is.ifmo.ru/verification/_2007_01_report-verification.pdf
27. Разработка технологии верификации управляющих программ со сложным поведением, построенных на основе автоматного подхода. Теоретические исследования поставленных перед НИР задач. СПбГУ ИТМО, 2007. http://is.ifmo.ru/verification/_2007_02_report-verification.pdf

28. Разработка технологии верификации управляющих программ со сложным поведением, построенных на основе автоматного подхода. Экспериментальные исследования поставленных перед НИР задач. СПбГУ ИТМО, 2008. – Режим доступа: http://is.ifmo.ru/verification/_2007_03_report-verification.pdf
29. Разработка технологии верификации управляющих программ со сложным поведением, построенных на основе автоматного подхода. Обобщение и оценка результатов исследований. СПбГУ ИТМО, 2008. http://is.ifmo.ru/verification/_2007_04_report-verification.pdf
30. *Вельдер С. Э., Шалыто А. А.* О верификации простых автоматных программ на основе метода Model Checking //Информационно-управляющие системы. 2007. № 3, с. 27−38. http://is.ifmo.ru/download/27-38.pdf
31. *Степанов О. Г.* Метод автоматической динамической верификации автоматных программ // Научно-технический вестник СПбГУ ИТМО. № 53, с. 221–229. http://books.ifmo.ru/ntv/ntv/53/ntv_53.pdf
32. *Егоров К. В., Шалыто А. А.* Методика верификации автоматных программ // Информационно-управляющие системы. 2008. № 5, с.15–21. http://is.ifmo.ru/works/_egorov.pdf
33. *Степанов О. Г.* Методы реализации автоматных объектно-ориентированных программ. Диссертация на соискание ученой степени кандидата технических наук. СПбГУ ИТМО, 2009. http://is.ifmo.ru/disser/stepanov_disser.pdf
34. *Егоров К. В., Царев Ф. Н.* Совместное применение генетического программирования и верификации моделей для построения автоматов управления системами со сложным поведением / Сборник докладов конференции молодых ученых и специалистов «Информационные технологии и системы» (ИТиС’09). М.: ИППИ РАН. 2009, с. 77 – 82. http://is.ifmo.ru/genalg/_2010_01_14_egorov_tsarev.pdf

35. *Вельдер С. Э., Шалыто А. А.* Верификация автоматных моделей методом редуцированного графа переходов // Научно-технический вестник СПбГУ ИТМО. №64, с. 66 – 77. http://is.ifmo.ru/works/_2010_01_29_velder.pdf
36. *Яминов Б. Р.* Сравнение методов верификации UniMod-моделей. Магистерская диссертация. СПбГУ ИТМО, 2009.
37. *Гиндин С. И.* Верификация автоматной модели мобильного банковского приложения. Бакалаврская работа. СПбГУ ИТМО, 2009. http://is.ifmo.ru/papers/_gindin_bachelor.pdf
38. *Егоров К. В.* Совместное применение генетического программирования и верификации моделей для построения автоматов управления системами со сложным поведением. Магистерская диссертация. СПбГУ ИТМО, 2010.
39. *Ремизов А. О., Шалыто А. А.* Верификация автоматных программ / Сборник докладов научно-технической конференции «Состояние, проблемы и перспективы создания корабельных информационно-управляющих комплексов. ОАО «Концерн «Моринформсистема «Агат». М.: 2010, с. 90 – 98. http://is.ifmo.ru/works/_2010_05_25_verific.pdf

Приложение. Анализ применимости шаблонов требований

№	Требование	Исходная запись	Шаблон, Ограничение	Источник
1	Если не сломаны клапаны и нагреватель и кофеварка варит кофе в обычном режиме, то после нажатия кнопки «С» основной автомат управления кофеваркой A_0 переходит в состояние «Прервано пользователем», а все вложенные автоматы обязательно должны вернуться в свои начальные состояния до того, как кофеварка будет запущена вновь.	AG(y_{31} != 4 & y_{32} != 4 & y_2 != 4 & y_0 == 2 -> EX act = e_{02} & AX (act = e_{02} -> AY_0 (y_0 == 4 & AG (y_0 == 0 & act = e_{01} -> y_{31} == 0 & y_{32} == 0 & y_2 == 0 & y_1 == 0))	-	5
2	Если произошла поломка нагревателя или одного из клапанов, то	AG((y_{31} = 4 | y_{32} = 4 | y_2 = 4) & y_0 = 2 -> A(y_0 = 2 U y_0 = 5)))	Ответ (огр.), Глобально AG(P -> A(S)),	5

	кофеварка (основной автомат A_0) обязательно перейдет в состояние 5 «Неисправность».		P: (y_{31} = 4 | y_{32} = 4 | y_2 = 4) & y_0 = 2, S: y_0 = 2, U y_0 = 5	
3	Если основной автомат управления кофеваркой A_0 перешел в состояние 5 «Неисправность», на дисплее обязательно высвечивается сообщение о типе неисправности (не допускается ситуация, при которой неисправность произошла, но пользователя кофеварка не информирует о типе неисправности).	!E((gen_act != z_{35} & gen_act != z_{25}) U y_0 = 5)	Предшествование, Глобально A(!P W S) ≡ !E(!S U (P & !S)), P: y_0 = 5, S: gen_act = z_{35} | gen_act = z_{25}	5
4	Если автомат A_0 находится в состоянии	AG(y_0 = 5 & act = e_{02} -> AY$_0$ (y_0 = 0 & AX(act =	-	5

	«Неисправность», то после нажатия кнопки «С» все автоматы, включая сам A_0, сбрасываются в начальное состояние, до того как кофеварка будет запущена вновь.	e_{01} -> y_1 = 0 & y_2 = 0 & y_{31} = 0 & y_{32} = 0)))		
5	Если автомат A_0 после неисправности находится в состоянии 0 «Готов к работе», то на дисплее кофеварки (включая дисплеи бойлера, нагревателя и клапанов) никакие сообщения о поломке не отображаются.	!EF((gen_act = z_{35} & E(gen_act != z_{36} U y0 = 0)) | (gen_act = z_{25} & E(gen_act != z_{26} U y0 = 0)) | (gen_act = z_{08} & E(gen_act != z_{03} U y_0 = 0)))	-	5
6	Кофеварка никогда не будет варить кофе без воды (соответственно автомат управления бойлером A_1 никогда	!EF(y_1 == 2 & E(act != x_{13} U y_1 = 4))	-	5

	не перейдет из состояния «Готов к варке следующей порции» в состояние «Кипятит», если воды недостаточно).			
7	Когда нагреватель достигает максимальной температуры, он всегда выключается – не существует такой ситуации (такого бесконечного процесса), при которой нагревательный элемент будет нагреваться бесконечно долго после преодоления максимального температурного порога.	!EF (y_2 == 2 & auto = 0 & y_0 == y_0 & AX (act != e_0 & act != end)) & !EF EG(y_2 == 2 & (auto = 2 -> AX (act!= e_0 & act != e_{22})))	-	5
8	Если автомат A_0 перешел в начальное состояние 0 «Готов к работе»,	AG(y_0 = 0 -> y_1 = 0 & y_2 = 0 & y_{31} = 0 & y_{32} = 0)	Всеобщное, Глобально AG(P),	5

	следовательно, все вложенные автоматы уже находятся в своих начальных состояниях.		P : y_0 = 0 -> y_1 =0 & y_2 = 0 & y_{31} = 0 & y_{32} = 0	
9	Система управления кофеваркой никогда не попадет в такое состояние, в котором она никак не реагирует ни на события системного таймера, ни на нажатие кнопок «ОК» и «С».	!(EF act = end)	Отсутствие, Глобально AG(!P) ≡ !EF(P), P: act = end	5
10	Существует путь структуры Крипке, в котором встречается переход, помеченный выходным воздействием z_2, и после этого перехода на протяжении оставшейся части пути до состояния y_0 = 0 переход с меткой z_2 больше не встречается.	EF(act = z_2 & EX E(act != z_2 U y_0 = 0))	-	

11	Если нажата кнопка «Сброс» до перехода банкомата в состояние снятия денег, то в любом случае до того, как банкомат начнёт новый цикл работы (т. е. попросит вставить карту) все автоматы управления вернуться в свои начальные состояния.	[] (y_0 != 9 & y_0 != 10 & y_0 != 11 & act = e_{03} -> (act != z_{01} U act = z_{01} & st_1 = 0 & st_2 = 0))	Предшествование, После Q []!Q | <>(Q & (!P W S)), Q: y_0 != 9 & y_0 != 10 & y_0 != 11 & act = e_{03}, S: st_1 = 0 & st_2 = 0, P: act = z_{01}	3
12	Если банкомат перешёл в состояние снятия денег, то он обязательно вернётся в начальное состояние, и до того, как он запросит вставить карту, автомат выдачи денег и автомат захвата карты	[] (y_0 = 9 -> (act != z_{01} U act = z_{01} & st_1 = 0 & st_2 = 0))	Предшествование, После Q []!Q | <>(Q & (!P W S)), Q: y_0 = 9, S: st_1 = 0	3

	вернуться в свои начальные состояния.		& st_2 = 0, P: act = z_{01}	
13	После перехода банкомата в состояние «9. Снятие денег» вплоть до начального состояния «0. Ожидание карты» нельзя произвести операцию сброса (т. е. завершить транзакцию нажатием кнопки «Сброс»).	[] (st_0 = 9 -> (act != e_{03} & st_0 != 0 U st_0 = 0))	Отс., Между Q и R [](Q & !R & <>R) -> (!P U R)), Q: st_0 = 9, P: act = e_{03}, R: st_0 = 0	3
14	После того, как банкомат вышел из состояния «5. Выбор суммы» без применения кнопки «Сброс», поле суммы обязательно очищается прежде, чем банкомат опять вернётся в это же	!<>(y_0 = 5 & act = ev_0 & act != e_{03} & (act != z_{040} & st_0 != 5 U st_0 = 5)) [] (y_0 != 5 | act != ev_0 | act = e_{03} | ((act = z_{040} | st_0 = 5) V st_0 != 5))	Предшествование, После Q []!Q | <>(Q & (!P W S)), Q: y_0 = 5 & act = ev_0 & act	3

	состояние.		!= e_{03}, S: act = z_{040}, P: st_0 = 5	
15	Если карта была вставлена, то рано или поздно она будет возвращена (т. е. автомат карты A_2 обязательно перейдёт в состояние «2. Карта на выдаче»), если в процессе работы использовались кнопки «Ввод» и «Сброс».	! <> (st_2 = 1 & [] st_2 != 2 & []<> act = e_{03} & []<> act = e_{05})	Ответ, После Q [](Q -> [](P -> <>S)) Q: st_2 = 1, P: <> act = e_{03} & <> act = e_{05}, S: st_2 = 2	3
16	Если произошло изъятие карты или денег, все автоматы вернуться в свои начальные состояния до того, как будет выведено приглашение	[]((act == z_{25} || act == z_{24}) -> (act != z_{01} U act == z_{01} && st_1 == 0 && st_2 == 0))	Предшество вание, После Q []!Q | <>(Q & (!P W S)),	3

	вставить карту.		Q: act == z_{25} || act == z_{24}, S: st_1 == 0 && st_2 == 0, P: act == z_{01}	
17	Если была нажата кнопка «открыть», то двери откроются.	(p1 -> F p2)	Ответ, Глобально [](P -> <>S), P: p1, S: p2	26
18	Если при открытых дверях была нажата кнопка открытия/закрытия двери, то двери закроются.	(s2 & e1) -> F s1	Ответ, Глобально [](P -> <>S), P: s2 & e1, S: s1	26
19	Было вызвано воздействие z_2, а	z_2 & (!z_1 U e_2)	Отсутствие, Между Q и	26

	затем произошло событие e_2, причем между ними не было вызовов к воздействию z_1.		R []((Q & !R & <>R) -> (!P U R)) Q: z_2, P: z_1, R: e_2	
20	Когда произойдет первое событие e_2, автомат окажется в состоянии s_1. Здесь важно поставить ограничение на проверку только первого возникновения события e_2 (первого при условии левой части — первого после вызова z_2), поскольку e_2 может возникать и при последующем открытии двери.	$(e_2 \rightarrow s_1)\ U\ (e_2\ \&\ s_1)$	-	26
21	Когда было вызвано	$z_1 \rightarrow !s_1\ W\ (z_2\ \&\ (!z_1\ U\ e_2))$	Предшество	26

	воздействие z_1 (двери стали открываться), автомат не попадет в состояние s_1 («Закрыты») до тех пор, пока не будет выполнено условие закрытия дверей.		вание, После Q []!Q | <>(Q & (!P W S)), Q: z_1, S: z_2 & (!z_1 U e_2), P: s_1	
22	Лифт движется только при закрытых дверях.	$A_1.s_2$ -> $A_2.s_1$	Всеобщность, Глобально [](P), P: $A_1.s_2$ -> $A_2.s_1$	26
23	В состояние 1 нельзя попасть, минуя состояние 6.	!E[!(Y=6)U(Y=1)]	Предшествование, Глобально A(!P W S)≡!E(!S U (P & !S)), P: Y=1,	26

			S: Y=6	
24	Когда лифт находится в движении, его двери закрыты.	AG((State = s_2 & Operation = 0) -> doors = s_1)	Всеобщность, Глобально AG(P), P: (State = s2 & Operation = 0) -> doors = s_1	27
25	Автомат всегда может перейти в состояние s_2.	AG(AF(State = s_2))	Всеобщность, Глобально AG(P), P: AF(State = s_2)	27
26	В состояние «5. Игрок выиграл» можно попасть только из состояния «4. Ход компьютера».	!((stateA_1 != 4) U (stateA_1 == 5))	Предшествование, Глобально !P W S, P: stateA_1 == 5, S: stateA_1 == 4	27
27	Автомат никогда не	G !isInState(A,	Отсутствие,	27

	попадет
	в состояние «Error».
		`Error)`	Глобально
			
			`[](!P),`
			
			`P: isInState( A, Error)`	

28	После действия `z1` всегда должно выполняться действие `z2`.	`G(z1 -> X z2)`	Отввет (следующий шаг),
			Глобально
			
			`[](P -> X S),`
			
			`P: z1,`
			`S: z2`
				27

29	Всегда при возникновении события `e1`, если условие `!o1.x1` выполнено, то автомат `A1` окажется в состоянии `s2` (после обработки события).	`G((wasEvent(e1) & wasTrue(!o1.x1)) -> isInState(A1, s2))`	Всеобщность,
			Глобально
			
			`[](P),`
			
			`P: ((wasEvent (e1) & wasTrue(!o 1.x1)) -> isInState( A1, s2)`
				27

30	Если ресурс заняли, то он несвободен, пока его не освободят.	$o_1.x_1$ W $o_1.z_1$ & G ($o_1.z_2$ -> ($o_1.x_1$ W $o_1.z_1$) & $o_1.z_1$ -> (!o1.x1 W o1.z2))	-	27
31	Если в состоянии «Opening» произошло событие e_4, то следующее состояние будет «Error».	G (wasInState("/A", "Opening") & wasEvent("e_4") -> isInState("/A", "Error"))	Всеобщность, Глобально [](P), P: wasInState("/A", "Opening") & wasEvent("e4") -> isInState("/A", "Error")	27
32	Если не происходит ошибки, то для любой истории автомат попадает в состояние «Opened» бесконечно часто.	G !wasEvent("e_4") -> GF isInState("/A", "Opened")	Отсутствие, Глобально [](!P), P: !wasEvent("e_4") Всеобщность, Глобально	27

			`[](P), P: F isInState( "/A", "Opened")`	

33	При условии отсутствия ошибки, автомат бесконечно часто проходит через состояние «Closed».	`G !wasEvent("e4") -> GF isInState("/A", "Closed")`	Отсутствие, Глобально
			
			`[](!P), P: !wasEvent( "e4")`
			
			Всеобщность, Глобально
			`[](P), P: F isInState( "/A", "Closed")`
				27

34	Автомат A_3 должен быть вызван хотя бы один раз.	`(<>{stateA3 == 0}) U {stateA1 == 3}`	-	27
35	Банкомат выдает деньги только после авторизации. Проверяется отсутствие последовательности действий, при	`!E[!e10 U o1.z10]`	Предшествование, Глобально `A(!P W S) ≡ !E(!S U (P & !S)),`	28

	которой пользователь получает деньги до ввода правильного кода.		P: $o_1.z_{10}$, S: e_{10}	
36	Банкомат выдает деньги только при вставленной карте. Проверяется отсутствие последовательности действии, при которой пользователь получает деньги до вставки карты в банкомат или после ее изъятия.	$!E[!e_6\ U\ o_1.z_{10}]$	Предшествование, Глобально A(!P W S)≡!E(!S U (P & !S)), P: $o_1.z_{10}$, S: e_6	28
37	Банкомат выдает деньги только при указании суммы. Проверяется отсутствие последовательности действии, при которой пользователь получает деньги до указания	$!E[!y_8\ U\ o_1.z_{10}]$	Предшествование, Глобально A(!P W S)≡!E(!S U (P & !S)), P: $o_1.z_{10}$, S: y_8	28

	снимаемой суммы.			
38	Баланс выдается только после авторизации. Проверяется отсутствие последовательности действии, при которой пользователю выдается информация о балансе до ввода правильного кода.	$!E[!e_{10}\ U\ o_1.z_7\ ]$	Предшествование, Глобально $A(!P\ W\ S) \equiv !E(!S\ U\ (P\ \&\ !S))$, P: $o_1.z_7$, S: e_{10}	28
39	Выдаваемая сумма не превышает остаток на счете. Проверяется отсутствие последовательности действий, при которой пользователь получает сумму денег большую, чем имеющаяся на счете, но до тех пор, пока не	$G(e_{14} \rightarrow (!o_1.z_{10}\ U\ e_{13}))$	Отсутствие, Между Q и R $[]((Q\ \&\ !R\ \&\ <>R) \rightarrow (!P\ U\ R))$, Q: e_{14}, P: $o_1.z_{10}$, R: e_{13}	28

	будет запрошена сумма денег, возможная к снятию.			
40	Выдача достаточной суммы денег. Проверяется, что если пользователь вставил карту, набрал верный код и указал сумму денег, имеющуюся в наличие, то она будет выдана.	$o_2.z_9$ -> AF(!e_{14} -> $o_1.z_{10}$)	Ответ, Глобально AG(P -> AF(S)), P: $o_2.z_9$, S: !e_{14} -> $o_1.z_{10}$	28
41	Банкомат не захватывает карту. Банкомат всегда возвращает карту после выдачи денег или баланса.	AG((A_1 = s_1 | Action = z_5) -> AF(A_2 = s_2))	Ответ, Глобально AG(P -> AF(S)), P: A_1 = s_1 | Action = z_5, S: A_2 = s_2	28
42	Банкомат выдает карту по требованию. Проверяется, что пользователь в любой	EX E[!(AClient && InState) U y_{13}]	-	28

	момент может изъять вставленную карту из банкомата. Всегда возможно, сделав только один переход между состояниями в автомате AClient, попасть в состояние №13.			
43	Изъятие карты по таймеру. Проверяется, что банкомат может изъять карту только, когда будет получен сигнал от таймера изъятия карты.	!((Event!= e_{82}) EU (A_2 != s_3)	Предшествование, Глобально A(!P W S) ≡ !E(!S U (P & !S)), P: A_2 = s_3, S: Event= e_{82}	28
44	При любой последовательности действий пользователю будут выданы деньги.	AG(AF(A_1 = s_1))	Всеобщность, Глобально AG(P), P: AF(A_1 = s_1	28

		AF e_9	Существование, Глобально AF(P), P: e_9	
45	Повторный запрос карты. Проверяется, что банкомат может запросить карту, если она уже вставлена.	E[!e_7 U y_1]	-	28
46	Снятие денег после запроса баланса. Проверяется, что банкомат может разрешить снять деньги после запроса баланса без повторной авторизации.	AG ((A_0 = s_{11}) -> (A_1 != s_1 EU A_0 = s_0 && Action = z_{81})) y_6 -> E[!y_{13} U y_8]	-	28
47	Возвращение карты без ошибки. Проверяется, что при любом поведении банкомата пользователь сможет получить свою карту	A[!e_{15} U y_{13}]	Предшествование, Глобально A[!P W S], P: e_{15}, S: y_{13}:	28

	назад до того, как произойдет ошибка на сервере.			
48	Выход в главное меню. В течение всей работы автомата в любой момент есть возможность попасть в главное меню.	AG EF y_{14}	Всеобщность, Глобально AG(P), P: EF y_{14}	28
49	После извлечения карты её можно будет вставить снова.	AG((A_2 = s_2) -> AF(A_0 = s_0))	Ответ, Глобально AG(P -> AF(S)), P: A_2 = s_2, S: A_0 = s_0	28
50	Транзакция будет завершена в случае ошибки.	AG((A_0 != s_0) & (A_0 != s_2) & (Event = e_2) -> AF(A_0 = s_2))	Ответ, Глобально AG(P -> AF(S)), P: ((A_0 != s_0) & (A_0 != s_2) &	28

			(Event = e_2, S: A_0 = s_2)	
51	Если происходит ошибка, то карта в любом случае будет возвращена.	G e_{15} → F $o_1.z_{13}$ В спецификации ошибка, корректная расстановка скобок: G(e_{15} → F $o_1.z_{13}$)	Ответ, Глобально [](P -> <>S), P: e_{15}, S: $o_1.z_{13}$	28
52	Существует путь, не содержащий ошибок (событий e15), но не любой путь удовлетворяет этому условию	(EG!e15) && (!AG!e15)	- Отсутствие, Глобально AG(!P), P: e15	28
53	Банкомат когда-нибудь выдаст деньги.	<>({stateAClient == 11})	Существование, Глобально <>P, P: {stateAClient == 11}	28
54	Деньги не выдаются, пока пользователь не сделает	(G ($o_2.z_3$ -> X e_{23})) -> (!$o_1.z_{10}$ U e_{23})	-	28

	соответствующий запрос.			
55	Существует способ провести инициализацию устройства, не нажимая кнопку Reset.	E[(InEvent -> !x_0) U z_{00}]	-	30
56	Через некоторое время после вызова у лифта открываются двери.	$e_{1,1}$ -> F $s_{1,3}$	Ответ, Глобально	
57	После получения события, при котором значение входной переменной x_1 – истина, всегда выполняется выходное воздействие z_1.	Gx_1 -> Fz_1 В спецификации ошибка, корректная расстоновка скобок: G(x_1 → F z_1)	Ответ, Глобально [](P -> <>S), P: x_1, S: z_1	31
58	При получении любого события либо значение переменной x_1 – истина, либо в некоторый момент времени после него будут	G x_1 || (F z_1 && F z_2)	Всеобщность, Глобально [](P), P: x_1 Существование, Глобально <>(P),	31

	выполнены выходные воздействия z_1 и z_2		P: z_i, i = 1,2	
59	Если автомат A_2 находится в состоянии s_1, то следующим состоянием будет s_2.	G(isInState(A_2.s_1) -> (isInState(A_2.s_2)))	Всеобщность, Глобально [](P), P: isInState(A_2.s_1) -> (isInState(A_2.s_2))	32
60	После возникновения аппаратной ошибки система отменит последнюю операцию. Требование может быть переформулировано в высказывание об автомате: «После события p_1.e_{10}, рано или поздно, будет вызвано действие o_1.z_{10}», где p_1.e_{10} – событие,	-	Ответ, Глобально [](P -> <>S), P: p_1.e_{10}, S: o_1.z_{10}	32

	посылаемое при аппаратной ошибке, а o1.z10 – откат последней операции.			
61	При выходе показаний датчика напряжения за допустимый диапазон управляющее реле выключит питание.	x_2 -> F x_4	Ответ, Глобально [](P -> <>S), P: x_2, S: x_4	11
62	В момент отключения охлаждающего элемента показания термостата не должны превышать допустимые.	z_4 -> !x_3	Всеобщность, Глобально [](P), P: z_4 -> !x_3	11
63	Непосредственно после открытия двери загорается лампочка.	e_1 U z_1	Всеобщность, Глобально [](P), P: e_1 U z_1	11
64	Во время движения двери лифта закрыты.	[(z_1 | z_2) -> !z_4 U (z_3 | z_4)] & [z_4 -> !(z_1 | z_2) U (z_5 | !(z_1 | z_2))]	Отсутствие, После Q до R [](Q & !R -> (!P W R)),	11

			Q: z_1 | z_2, R: z_3 | z_4, P: z_4 Q: z_4, R: z_5 | !(z_1 | z_2), P: z_1 | z_2 Доказательство см. требование № 65	
65	Между двумя событиями u и v не происходит событие p.	u -> !p U (v | !p)	Отсутствие, После Q до R [](Q & !R -> (!P W R)), Q: u, R: v, P: p [](u & !v -> (!p W v)) ≡	11

			[](u & !v -> (!p U (v || [] !p)))	
66	Пока двери лифта открыты, свет в кабине включен.	(s_2 & x_4) & (s_4 & x_4)	Всеобщность, Глобально [](P), P: (s_2 & x_4) & (s_4 & x_4)	11
67	Во время движения двери не открываются.	!(s_1 & z_4)	Отсутствие, Глобально [](!P), P: s_1 & z_4	11
68	Движение начинается только после состояния, при выходе из которого двери закрыты.	!((s2 | s4) -> (s2 | s4 & !(s3 | s5)) U s1)	-	11
69	Через некоторое время после вызова у лифта открываются двери.	$e_{1,1}$ -> F $s_{1,3}$	Ответ, Глобально [](P -> <>S), P: $e_{1,1}$, S: $s_{1,3}$	11

70	Если было событие e11, то было вызвано действие z1. В терминах модели утверждение можно записать как при обработке события «открыть двери» обязательно будет начато открывание дверей.	G(wasEvent(ep.e11) => wasAction(co.z1))	Всеобщность, Глобально [](P), P: wasEvent(ep.e11) => wasAction(co.z1)	34, 38
71	Событие e4 обрабатывается тогда и только тогда, когда вызывается z3. В терминах модели: звонок в аварийную службу будет произведен тогда и только тогда, когда лифт сломается.	G(wasEvent(ep.e4) <=> wasAction(co.z3))	Всеобщность, Глобально [](P), P: wasEvent(ep.e4) <=> wasAction(co.z3)	34, 38
72	Если было событие e3, то было вызвано действие z1. Если препятствие мешает закрыть двери, то дверь начнет открываться.	G(wasEvent(ep.e3) => wasAction(co.z1))	Всеобщность, Глобально [](P), P: wasEvent(ep.e3) => wasAction(	34, 38

			co.z1)	
73	Если было событие e11, то следующим обработанным событием будет e4 или e2. В терминах модели: если была нажата кнопка «открыть двери», то следующее событие будет либо успешное открывание дверей, либо дверь сломается.	G(wasEvent(ep.e11) => X(wasEvent(ep.e4) or wasEvent(ep.e2)))	Ответ (следующий шаг), Глобально [](P -> X(S)), P: wasEvent(ep.e11, S: wasEvent(ep.e4) or wasEvent(ep.e2)	34, 38
74	Если было вызвано действие z1, то оно не будет больше вызвано, пока не будет вызвано z2 или событие e4. Если дверь начала открываться, то она не сможет повторно открыться до закрытия или поломки.	G(wasAction(co.z1) => X(!wasAction(co.z1) U(wasAction(co.z2) or wasEvent(ep.e4))))	Отсутствие, Между Q и R []((Q & !R & <>R) -> (!P U R)), Q: wasAction(co.z1), P: wasAction(co.z1),	34, 38

			R: wasAction(co.z2) or wasEvent(ep.e4)	
75	Событие e12 обрабатывается тогда и только тогда, когда вызывается z2. В терминах модели: при нажатии кнопки закрывания дверей будет начато закрывание, и закрывание дверей может начаться только при нажатии кнопки закрыть двери.	G(wasEvent(e12) <=> wasAction(z2))	Всеобщность, Глобально [](P), P: wasEvent(e12) <=> wasAction(z2)	38
76	Если было событие e2, то следующим обработанным событием будет e11 или e12. В терминах модели: если дверь успешно открылась или закрылась, то	G(wasEvent(e2) => X[wasEvent(e11) || wasEvent(e12)])	Ответ (следующий шаг), Глобально [](P -> X(S)), P: wasEvent(e2),	38

	следующее обработанное событие может быть только «открыть двери» или «закрыть двери»		S: wasEvent(e11) || wasEvent(e12)	
77	Если было событие e12, то следующим обработанным событием будет e2 или e3 или e4. В терминах модели: если нажали кнопку «закрыть двери», то либо двери закроются, либо препятствие помешает закрыть двери, либо лифт сломается.	G(wasEvent(e12) => X[wasEvent(e2) or wasEvent(e3) or wasEvent(ep.e4)])	Ответ (следующий шаг), Глобально [](P -> X(S)), P: wasEvent(e12), S: wasEvent(e2) or wasEvent(e3) or wasEvent(ep.e4)	38
78	Если было вызвано действие z1, то следующим обработанным событием будет e2 или e4. В терминах модели: если дверь	G(wasAction(z1) => X[wasEvent(e2) or wasEvent(e4)])	Ответ (следующий шаг), Глобально [](P -> X(S)),	38

	начала закрываться, то либо она успешно откроется, либо сломается.		P: wasAction(z1), S: wasEvent(e2) or wasEvent(e4)	
79	Если было вызвано действие z2, то оно не будет больше вызвано, пока не будет вызвано z1 или событие z4. В терминах модели: если дверь начала закрываться, то она не будет снова закрываться до тех пор, пока она не будет отрываться или не сломается.	G(wasAction(z2) => X[U(!wasAction(z2), wasAction(z1) or wasEvent(e4))])	Отсутствие, Между Q и R []((Q & !R & <>R) -> (!P U R)), Q: wasAction(z2), P: wasAction(z2), R: wasAction(z1) or wasEvent(e4)	38
80	Не верно, что в будущем будет	!F(wasEvent(e4) and X(	Отсутствие, После Q	38

	после события e4, когда-либо вызвано e11, e12, e2 или e3. В терминах модели: не верно, что после поломки лифта будут обработаны события «открыть двери», «закрыть двери», успешное открывание или закрывание дверей, препятствие мешает закрыть двери. Или, что то же самое, лифт не может быть починен.	F(wasEvent(e11) || wasEvent(e12) || wasEvent(e2) || wasEvent(e3)))	[](Q -> [](!P)) ≡ !<>(!(Q -> []!P)) ≡ !<>(!(Q -> !<>P)) ≡ !<>(Q & <>P), Q: wasEvent(e4), P: wasEvent(e11) || wasEvent(e12) || wasEvent(e2) || wasEvent(e3)	
81	Если произошло событие e_{14}, то невозможно попасть в состояние 10, минуя позицию $o_3.z_0$.	e_{14} -> !E[!$o_3.z_0$ U y_{10}]	Предшествование, После Q A[!Q W (Q & A[!P W S])],	35

			Q: e_{14}, P: y_{10}, S $o_3.z_0$:	
82	Когда-нибудь пользователь введет правильный *PIN*-код	F wasEvent(“e_{10}”)	Существование, Глобально <> (P), P: wasEvent(“e_{10}”)	36
83	-	AG ($A_0.s_1$ -> AF $A_1.s_1$)	Ответ, Глобально AG(P -> AF(S)), P: $A_0.s_1$, S: $A_1.s_1$	36
84	-	!(<>{lastEvent == e1})	Отсутствие, Глобально [](!P), P: lastEvent == e1 !(<>(P)) ≡ [](!(P))	36
85	Автомат A2 никогда	G !(A2 in state	Отсутствие,	36

	не попадет в состояние s3.	s3)	Глобально [](!P), P: A2 in state s3	
86	Если сигнализацию включили, то при сильных ударах она будет включать сирену, по крайней мере, до тех пор, пока ее не выключат.	G (e_{11} -> ((G (e_{22} -> A_1 in “3. Тревога”)) W e_{12}))	Существование, После Q до R [](Q & !R -> (P W R)) Q: e_{11}, P: e_{22} -> A_1 in “3. Тревога”), R: e_{12}	36
87	Когда-нибудь произойдет событие e1, генерируемое источником событий p	F(wasEvent(p.e1))	Существование, Глобально <> (P), P: wasEvent(“p.e1”)	36
88	*PIN* всегда должен проверяться перед операцией. Это означает, что	G(main_menu -> ((G(!oper)) || (F(pin_entry) U (oper))))	-	37

	пользователь после начала сеанса (`main_menu`) сеанса имеет две возможности: никогда не выполнить ни одной операции или в какой-то момент ввести *PIN* перед выполнением операции			

89	Пользователь после начала сеанса (`main_menu`) должен иметь возможность завершить сеанс (`close_session`)	`!F(main_menu && G(!close_session))`	Существование,
			После `Q`
			
			`[](Q -> [](P)) ≡ !<>(!(Q -> []P)) ≡ !<>(Q & ![]P),`
			
			`Q: main_menu,`
			
			`P: close_session`
				37

90	Обработка события	`G(wasEvent(T)`	Всеобщность,	38

	T равносильна тому, что первым было вызвано действие z5. В терминах модели утверждение можно записать как: срабатывание таймера приводит к прибавлению минуты к текущему времени, и прибавление минуты может быть только по срабатыванию таймера.	<=> wasFirstAction(z5))	Глобально [](P), P: wasEvent(T) <=> wasFirstAction(z5)	
91	Обработка события H равносильна вызову действия z1 или z3. В терминах модели: нажатие кнопки «часы» приводит к увеличению текущего времени на один час или времени будильника на один час, и увеличение времени на один час может	G(wasEvent(H) <=> [wasAction(z1) or wasAction(z3)])	Всеобщность, Глобально [](P), P: wasEvent(H) <=> [wasAction(z1) or wasAction(z3)]	38

	быть только по нажатию соответствующие кнопки.			
92	Неверно, что в будущем z1 и z3 могут быть одновременно. В терминах модели: нельзя одновременно увеличить время часов и время будильника на час.	!F(wasAction(z1) and wasAction(z3))	Отсутствие, Глобально [](!P), P: wasAction(z1) and wasAction(z3) !(<>(P)) ≡ [](!(P))	38
93	Обработка события M равносильна вызову действия z2 или z4. В терминах модели: нажатие кнопки «минуты» приводит к увеличению на одну минуту времени часов или времени будильника, и увеличение времени на минуту может быть только по	G(wasEvent(M) <=> [wasAction(z2) or wasAction(co.z4)])	Всеобщность, Глобально [](P), P: wasEvent(M) <=> [wasAction(z2) or wasAction(co.z4)]	38

	нажатию соответствующей кнопки.			
94	Неверно, что в будущем z2 и z4 могут быть одновременно. В терминах модели: нельзя одновременно увеличить время часов и время будильника на минуту.	!F(wasAction(z2) and wasAction(z4))	Отсутствие, Глобально [](!P), P: wasAction(z2) and wasAction(z4) !(<>(P)) ≡ [](!(P))	38
95	Если было обработано событие A и вызвано действие z_7, то не могут быть вызваны действия z_3 и z_4, пока не произойдет событие A. В терминах модели: если будильник был отключен, то нельзя выставлять время будильника, не нажав кнопу A.	G([wasEvent(A) and wasAction(z7)] => X(R[wasEvent(A), !wasAction(z3) and !wasAction(z4)]))	Отсутствие, После Q до R [](Q & !R -> (!P W R)) Q: wasEvent(A) and wasAction(z7), P: !wasAction	38

			(z3) and !wasAction (z4), R: wasEvent(A)	
96	Если было вызвано действие z3 или z4, то не будут вызваны действия ни z6, ни z7 до тех пор, пока не будет обработано событие A. В терминах модели: если мы выставляем время будильника, то звонок не будет включен и не будет выключен, пока не будет нажата кнопка A.	G([wasAction(z3) or wasAction(z4)] => X(R[wasEvent(A), !wasAction(z6) and !wasAction(z7)]))	Отсутствие, После Q до R [](Q & !R -> (!P W R)) Q: wasAction(z3) or wasAction(z4), P: !wasAction(z6) and !wasAction(z7), R: wasEvent(A)	38
97	Если было вызвано	G(	Отсутствие,	38

	действие z1 или z2, то нельзя вызвать z3 и z4 до события A. В терминах модели: если мы выставляем время часов, то нельзя выставлять время будильника, не нажав кнопку A.	[wasAction(z1) or wasAction(z2)] => X(R[wasEvent(A), !wasAction(z3) and !wasAction(z4)]))	После Q до R [](Q & !R -> (!P W R)) Q: wasEvent(z1) or wasAction(z2), P: !wasAction(z3) and !wasAction(z4), R: wasEvent(A)	
98	Если было вызвано действие z3 или z4, то нельзя вызывать z1 и z2 до события A. В терминах модели: если мы выставляем время будильника, то, не	G([wasAction(z3) or wasAction(z4)] => X(R[wasEvent(A), !wasAction(z1) and !wasAction(z2)]	Отсутствие, После Q до R [](Q & !R -> (!P W R)) Q: wasEvent(z	

	нажав A, нельзя выставлять время часов.	))	3) or wasAction(z4), P: !wasAction (z1) and !wasAction (z2), R: wasEvent(A)	
99	Не могут быть одновременно вызваны действия z6 и z7. В терминах модели: нельзя одновременно включить и отключить сигнал будильника.	!F(wasAction(z6) and wasAction(z7))	Отсутствие, Глобально [](!P), P: wasAction(z6) and wasAction(z7) !(<>(P)) ≡ [](!(P))	38
100	В будущем не верно, что произойдет A и не будет вызвано z7. В терминах: кнопка A не может	F(!(wasEvent(A) and !wasAction(z7)))	Существова ние, Глобально <>(P),	38

	нажиматься бесконечно, не отключив сигнал будильника, или что нажатие кнопки A рано или поздно отключит будильник.		P: !(wasEvent(A) and !wasAction(z7))	(часы)
101	-	G(!wasEvent(ep.IC) or wasAction(co.z1))	Всеобщность, Глобально [](P), P: !wasEvent(ep.IC) or wasAction(co.z1)	38
102	-	G((!wasEvent(ep.EC) or wasAction(co.z2)) and (!wasAction(co.z2) or wasEvent(ep.EC)))	Всеобщность, Глобально [](P), P: (!wasEvent(ep.EC) or wasAction(co.z2)) and (!wasAction (co.z2) or wasEvent(e p.EC))	38

103	-	G((!wasEvent(ep.A) or wasAction(co.z3)) and (!wasAction(co.z3) or wasEvent(ep.A)))	Всеобщность, Глобально [](P), P: (!wasEvent(ep.A) or wasAction(co.z3)) and (!wasAction (co.z3) or wasEvent(e p.A))	38
104	-	G(!wasEvent(ep.AE) or wasAction(co.z1))	Всеобщность, Глобально [](P), P: !wasEvent(ep.AE) or wasAction(co.z1)	38
105	-	G(!wasEvent(ep.AS) or wasAction(co.z4))	Всеобщность, Глобально [](P), P: !wasEvent(ep.AS) or	38

			wasAction(co.z4)	
106	-	G(!wasEvent(ep.CNL) or wasAction(co.z4))	Всеобщность, Глобально [](P), P: !wasEvent(ep.CNL) or wasAction(co.z4)	38
107	-	G(!wasEvent(ep.M) or wasAction(co.z8))	Всеобщность, Глобально [](P), P: !wasEvent(ep.M) or wasAction(co.z8)	38
108	-	G((!wasEvent(ep.MR) or wasAction(co.z9)) and (!wasAction(co.z9) or wasEvent(ep.MR)))	Всеобщность, Глобально [](P), P: (!wasEvent(ep.MR) or wasAction(co.z9)) and (	38

			!wasAction (co.z9) or wasEvent(e p.MR))	
109	-	G(!wasEvent(ep.ME) or wasAction(co.z8))	Всеобщность, Глобально [](P), P: !wasEvent(ep.ME) or wasAction(co.z8)	38
110	-	G((!wasEvent(ep.MS) or wasAction(co.z10)) and (!wasAction(co.z 10) or wasEvent(ep.MS)))	Всеобщность, Глобально [](P), P: (!wasEvent (ep.MS) or wasAction(co.z10)) and (!wasActio n(co.z10) or wasEvent(e p.MS))	38
111	-	G ((!wasEvent(ep.MP	Всеобщность, Глобально	38

		) or (wasAction(co.z4) and wasAction(co.z11) and wasAction(co.z12) and wasAction(co.z13))) and (!wasAction(co.z 13) or wasEvent(ep.MP)))	[](P), P: (!wasEvent (ep.MP) or (wasAction(co.z4) and wasAction(co.z11) and wasAction(co.z12) and wasAction(co.z13))) and (!wasActio n(co.z13) or wasEvent(e p.MP))	
112	-	G((!wasEvent(ep.C) or wasAction(co.z5)) and (!wasAction(co.z5) or wasEvent(ep.C))	Всеобщность, Глобально [](P), P: !wasEvent(ep.C) or wasAction(	38

		)	co.z5)) and (!wasAction (co.z5) or wasEvent(e p.C))	
113	-	G((!wasEvent(ep.CR) or wasAction(co.z6)) and (!wasAction(co.z6) or wasEvent(ep.CR)))	Всеобщность, Глобально [](P), P: (!wasEvent(ep.CR) or wasAction(co.z6)) and (!wasAction (co.z6) or wasEvent(e p.CR))	38
114	-	G((!wasEvent(ep.CP) or (wasAction(co.z7) and wasAction(co.z4))) and (!wasAction(co.z7) or wasEvent(ep.CP)	Всеобщность, Глобально [](P), P: (!wasEvent(ep.CP) or (wasAction (co.z7) and	38

		))	wasAction(co.z4))) and (!wasAction (co.z7) or wasEvent(e p.CP))	
115	-	G((!wasAction(co.z 11) or wasAction(co.z12)) and (!wasAction(co.z 12) or wasAction(co.z11)))	Всеобщность, Глобально [](P), P: (!wasActio n(co.z11) or wasAction(co.z12)) and (!wasActio n(co.z12) or wasAction(co.z11))	38
116	Любое другое воздействие, кроме x10 активно до тех пор, пока автомат не перейдет в состояние 1.	<>(({lastEvent > 10}) && (({lastEvent > 10}) U ({stateA0 == 1})))	-	39
117	Состояние 5	!(<>({stateA0 ==	Отсутствие,	39

	автомата никогда не будет достигнуто.	5}))	Глобально [](!P), !(<>(P) ≡ [](!{P}) P: stateA0 == 5	
118	Никогда не будет получено входного воздействия x16 в состоянии 3.	!(<>(({stateA0 == 3}) && ({lastEvent == 16})))	Отсутствие, Глобально [](!P), !(<>(P) ≡ [](!P) P: ({stateA0 == 3}) && ({lastEvent == 16})	39