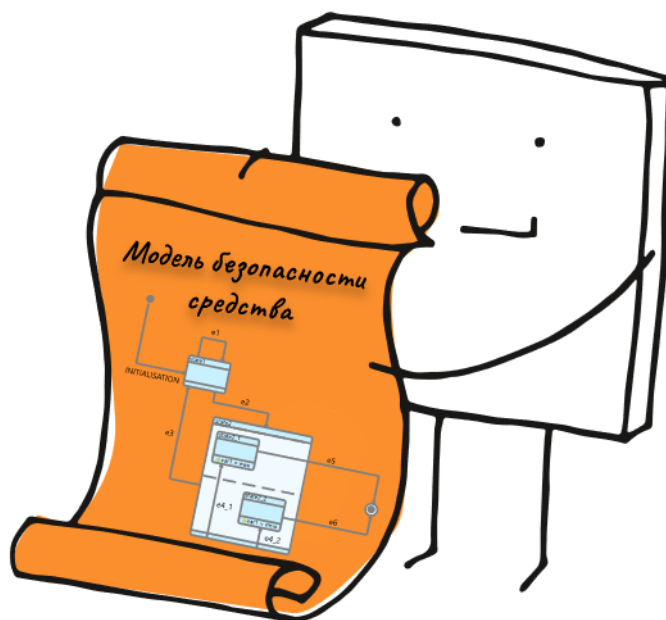


**А.Н. Бегаев, С.В. Кашин, Д.Д. Павлов,  
Н.А. Маркевич**

**МОДЕЛЬ БЕЗОПАСНОСТИ СРЕДСТВА,  
ИЛИ КАК ФОРМАЛЬНО ОПИСАТЬ  
ПОДСИСТЕМЫ ПРОГРАММНОГО  
ОБЕСПЕЧЕНИЯ**



**Санкт-Петербург  
2022**

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

**А.Н. Бегаев, С.В. Кашин, Д.Д. Павлов,  
Н.А. Маркевич**  
**МОДЕЛЬ БЕЗОПАСНОСТИ СРЕДСТВА,  
ИЛИ КАК ФОРМАЛЬНО ОПИСАТЬ  
ПОДСИСТЕМЫ ПРОГРАММНОГО  
ОБЕСПЕЧЕНИЯ**

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО  
по направлению подготовки 10.03.01 Информационная безопасность  
в качестве Учебно-методического пособия для реализации основных  
профессиональных образовательных программ высшего образования  
бакалавриата

 УНИВЕРСИТЕТ ИТМО

Санкт-Петербург  
2022

Бегаев А.Н., Кашин С.В., Маркевич Н.А., Павлов Д.Д., Модель безопасности средства, или Как формально описать подсистемы программного обеспечения– СПб: Университет ИТМО, 2022. – 44 с.

Рецензент(ы):

Заколдаев Данил Анатольевич, кандидат технических наук, доцент, декан факультета безопасности информационных технологий, Университета ИТМО.

Учебно-методическое пособие разработано в соответствии с программами дисциплин «Технология сертификации средств защиты информации» и «Сертификация продукции в различных системах сертификации (Минобороны, ФСТЭК, ФСБ)» и предназначено для студентов, обучающихся по программам направления подготовки 10.03.01 Информационная безопасность.

Учебно-методическое пособие содержит теоретический и практический материал, посвященный документации разработчика, необходимой для прохождения сертификации программного обеспечения в системе сертификации ФСТЭК России.



**Университет ИТМО** – национальный исследовательский университет, ведущий вуз России в области информационных, фотонных и биохимических технологий. Альма-матер победителей международных соревнований по программированию – ICPC (единственный в мире семикратный чемпион), Google Code Jam, Facebook Hacker Cup, Яндекс.Алгоритм, Russian Code Cup, Topcoder Open и др. Приоритетные направления: IT, фотоника, робототехника, квантовые коммуникации, трансляционная медицина, Life Sciences, Art&Science, Science Communication. Входит в ТОП-100 по направлению «Автоматизация и управление» Шанхайского предметного рейтинга (ARWU) и занимает 74 место в мире в британском предметном рейтинге QS по компьютерным наукам (Computer Science and Information Systems). С 2013 по 2020 гг. – лидер Проекта 5–100.

© Университет ИТМО, 2022

© Бегаев А.Н., Кашин С.В., Маркевич Н.А., Павлов Д.Д., 2022

## Содержание

Введение.....	4
1. Сертификация программного обеспечения .....	5
1.1. Общие положения .....	5
1.2. Документация разработчика.....	6
2. Модель безопасности.....	8
2.1. Требования руководящего документа .....	8
2.2. Реализация .....	9
3. Метод Event-B и набор инструментов iUML-B .....	10
3.1. Event-B .....	10
3.2. iUML-B .....	12
4. Средство формального описания Rodin .....	20
4.1. Общая информация и основные функции Rodin.....	20
4.2. Работа с Rodin .....	20
5. Практическая работа.....	38
5.1. Порядок выполнения работы .....	38
5.2. Дополнительное задание .....	38
6. Контрольные вопросы .....	40
Заключение .....	41
Список литературы.....	42

## **ВВЕДЕНИЕ**

Данное учебно-методическое пособие содержит материалы, которые позволят студентам получить теоретические и практические знания по формальному описанию политик безопасности, которое необходимо выполнять при подготовке к сертификации в системе сертификации ФСТЭК России в отношении программных средств, реализующих политики управления доступом или политики фильтрации информационных потоков.

Пособие познакомит студентов с нотацией Event-B и даст базовые навыки проектирования частей программного обеспечения с помощью iUML-B State Machine diagrams.

Структурно пособие состоит из шести разделов. Разделы содержат как теоретический, так и практический материал. Практический материал предназначен для подготовки к выполнению практической работы и получения знаний в части формального описания ПО и формирования документа «Модель безопасности». Пособие содержит практическую работу и контрольные вопросы, призванные помочь студентам в усвоении результатов обучения по дисциплинам «Технология сертификации средств защиты информации» и «Сертификация продукции в различных системах сертификации (Минобороны, ФСТЭК, ФСБ)».

Дополнительно приведен библиографический список (список литературы), который включает руководящие документы и литературу, рекомендуемую авторами для более глубокого освоения содержания дисциплины.

# 1. СЕРТИФИКАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## 1.1. Общие положения

В соответствии с Федеральным законом № 184 «О техническом регулировании» сертификация представляет собой форму осуществляемого органом по сертификации подтверждения соответствия объектов требованиям технических регламентов, документам по стандартизации или условиям договоров.

Сертификация может быть обязательной и добровольной. Обязательная сертификация осуществляется на основании законов и законодательных положений и проводится в системах обязательной сертификации, установленных Постановлением Правительства РФ от 26 июня 1995 г. № 608 «О сертификации средств защиты информации». Добровольная сертификация проводится только по инициативе заявителя (юридического или физического лица) в системах добровольной сертификации.

В соответствии с Постановлением Правительства № 608 обязательная сертификация проводится в рамках систем сертификации Федеральной службы по техническому и экспортному контролю (ФСТЭК России), Федеральной службы безопасности Российской Федерации (ФСБ России), Министерства обороны Российской Федерации (Минобороны России).

Система сертификации ФСТЭК России – это структура, состоящая из следующих участников сертификации:

- федерального органа по сертификации (ФСТЭК России);
- организаций, аккредитованных ФСТЭК России в качестве органов по сертификации (органы по сертификации);
- организаций, аккредитованных ФСТЭК России в качестве испытательных лабораторий (испытательные лаборатории);
- изготовители (производители) средств защиты информации.

Сертификация в системе ФСТЭК России проводится в соответствии с Положением о системе сертификации средств защиты информации, утвержденным приказом ФСТЭК России от 3 апреля 2018 года № 55.

Сертификация средств защиты информации осуществляется на соответствие нормативным правовым актам ФСТЭК России, техническим условиям (ГОСТ 2.114-2016), техническим заданиям (ГОСТ 19.201-78), заданиям по безопасности (ГОСТ Р ИСО/МЭК 15408).

Со 2 июня 2020 года сертификация средств защиты информации проводятся согласно приказу ФСТЭК России № 76 «Об утверждении Требований по безопасности информации, устанавливающих уровни доверия к средствам технической защиты информации и средствам обеспечения безопасности информационных технологий» (далее – Приказ №76).

В соответствии с Приказом № 76 для разграничения требований по безопасности информации к программным, программно-техническим средствам технической защиты информации, средствам обеспечения безопасности информационных технологий, включая защищенные средства обработки информации (далее – средства), устанавливается 6 уровней доверия (далее – УД).

Каждое средство, соответствующие определенному уровню, может применяться в пяти типах систем определенного класса (таблица 1-1).

Таблица 1-1 – Общие требования уровней доверия

	6 УД	5 УД	4 УД	3 УД	2 УД	1 УД
<b>КИИ</b> <sup>1</sup>	3 категория	2 категория	1 категория	Применяются для защиты сведений, составляющих государственную тайну		
<b>ГИС</b> <sup>2</sup>	3 класс	2 класс	1 класс			
<b>АСУ ТП</b> <sup>3</sup>	3 класс	2 класс	1 класс			
<b>ИСПДн</b> <sup>4</sup>	3, 4 уровень	2 уровень	1 уровень			
<b>ИСОП</b> <sup>5</sup>			II класс			

## 1.2. Документация разработчика

В ходе подготовки к сертификации разработчик должен подготовить комплект документации на средство в соответствии с требованиями, устанавливаемыми к уровню доверия Приказом № 76. Перечень необходимой документации содержит, но не ограничивается следующими документами:

1. Модель безопасности средства.
2. Архитектура безопасности. Документ включает обоснование безопасности процесса инициализации средства, обоснование обеспечения собственной защиты средства от несанкционированного доступа и обоснование невозможности обхода функций безопасности.
3. Функциональная спецификация. Данный документ включает в себя перечень всех функций средства, включая функции безопасности; описание

<sup>1</sup> КИИ (сокр. – критическая информационная инфраструктура). Категории устанавливаются в соответствии с Федеральным законом № 187-ФЗ от 26.07.2017 и Постановлением Правительства РФ № 127 от 08.02.2018.

<sup>2</sup> ГИС (сокр. – государственная информационная система). Классы устанавливаются в соответствии с Приказами ФСТЭК России № 17 от 11.02.2013 и № 27 от 15.02.2017.

<sup>3</sup> АСУ ТП (сокр. – автоматизированная система управления технологическим процессом). Классы устанавливаются в соответствии с Приказами ФСТЭК России № 31 от 14.03.2014 и № 138 от 09.08.2018.

<sup>4</sup> ИСПДн (сокр. – информационная система персональных данных). Уровни защищенности устанавливаются в соответствии с Постановлением Правительства № 1119 от 01.11.2012.

<sup>5</sup> ИСОП (сокр. – информационная система общего пользования). Классы устанавливаются в соответствии с Приказом ФСБ России и ФСТЭК России № 416/489 от 31.08.2010.

назначения и способов использования каждого интерфейса функций безопасности и иных функций средства.

4. Проектная документация (эскизный и технический проект). В проектной документации описывается программное обеспечение на уровне подсистем и модулей, а также информация о их составе и взаимодействии.

5. Описание средств разработки. Содержит описание средств, применяемых для создания ПО, включая информацию об используемых опциях.

6. Документация по управлению конфигурацией средства. Описывает уникальную маркировку средства, список элементов конфигурации средства, включающий в том числе документацию и порядок управления изменениями средства и документации.

7. Документация по безопасной разработке. Содержит описание всех физических, процедурных, организационных и других мер безопасности, применяемых в среде разработки средства для защиты конфиденциальности и целостности проектной документации и реализации средства, а также применяемые меры безопасности, направленные на снижение вероятности возникновения в средстве уязвимостей и иных недостатков, и их обоснование.

8. Формуляр, оформленный согласно ГОСТ 19.501-78. Содержит контрольные суммы дистрибутива и исполняемых файлов программного обеспечения средства.

9. Руководство пользователя, содержащее сведения о режимах работы средства, принципах безопасной работы средства, функциях и интерфейсах функций средства, доступных каждой роли пользователей, параметрах (настройках) безопасности средства, доступных каждой роли пользователей, и их безопасных значениях, а также сведения о типах событий безопасности, связанных с доступными пользователю функциями средства и возможных действиях после сбоев и ошибок эксплуатации средства.

10. Руководство администратора. Документ содержит действия по приемке поставленного средства, действия по безопасной установке и настройке средства и действия по реализации функций безопасности среды функционирования средства.

11. Планы и результаты тестирования.

12. Процедура устранения недостатков.

13. Регламент обновления средства потребителем. Описывает порядок получения, установки и контроля установки обновления ПО средства.



## 2. МОДЕЛЬ БЕЗОПАСНОСТИ

При сертификации средства на соответствие 4 уровню доверия и выше разработчиком должен быть составлен документ «Модель безопасности», формально (математически) описывающий условия безопасности, выполнение которых указывает на реализацию политик. Такое описание позволяет провести независимую от разработчика модели проверку ее корректности.

Модель безопасности разрабатывается только если программное обеспечение реализует:

- политики управления доступом (для средств, в которых предусмотрена функция разграничения доступа и для которых функция разграничения доступа является основной). Примерами таких средств являются: средства защиты информации от несанкционированного доступа (Secret Net Studio, Dallas Lock и др.), системы управления базами данных (Postgres, MySQL, MSSQL и др.) и операционные системы (Windows, UNIX, MAC OS и др.);
- политики фильтрации информационных потоков (для средств, в которых предусмотрены функции фильтрации и контроля трафика и для которых эти функции являются основными функциями средства). Примеры: межсетевые экраны (Cisco ASA, Рубикон-С и др.), средства виртуализации (HyperV, VMWare, Xen и др.) и операционные системы.

### 2.1. Требования руководящего документа

Согласно требованиям Приказа № 76, документ «Модель безопасности», содержащий политики управления доступом и (или) политики фильтрации информационных потоков, должен включать:

- описание условий безопасности, выполнение которых указывает на реализацию политик;
- выполнение условий безопасности, доказанное формальным (математическим) способом;
- взаимосвязь условий безопасности с режимами функционирования средств, описанная неформально (нематематически);
- дополнительную информацию (для 3 УД и выше).

Цель разработки Модели безопасности – усиление доверия к ПО путем формального (математического) описания важных механизмов защиты средства. Доверие к модели поддерживается доказательством того, что в ней не содержится противоречий.

Модель безопасности должна формально устанавливать принципы безопасности на основании их характеристик с приведением доказательств, полученных математическими методами.

## 2.2. Реализация

Для формального описания разработчиком может использоваться любой язык спецификации. Примеры формальных языков:

- Язык спецификаций  $Z$  ( $Z$ -нотация<sup>6</sup>) – формальный язык спецификации, используемый для описания и моделирования программ и их формальной верификации.

- Coq – интерактивное программное средство доказательства теорем, использующее собственный язык функционального программирования (Gallina) с зависимыми типами.

- Event-B – формальный метод моделирования и анализа на системном уровне.

Последний вариант наиболее популярен среди разработчиков программных средств. Ключевые особенности Event-B: в качестве нотации моделирования используется теория множеств, для представления систем на разных уровнях абстракции используются уточнения, а для проверки согласованности между уровнями уточнения – математические доказательства.

---

<sup>6</sup> Нотация – система условных обозначений (словарь).

## 3. МЕТОД EVENT-B И НАБОР ИНСТРУМЕНТОВ IUML-B

### 3.1. Event-B

Формальный метод Event-B разработан J-R. Abrial на основе метода В. Event-B позволяет моделировать системы с дискретными переходами между состояниями (событийно-ориентированных систем). Спецификации в нотации Event-B описывают модели программ в виде автоматных моделей, с состояниями, выраженными с помощью переменных, правилами переходов между состояниями (событиями), и требованиями к состояниям (инвариантами).

Модели Event-B поделены на две части – статическую («context» – контекст) и динамическую («machine» – машина).

Ниже представлена общая структура контекста.

#### CONTEXT

<имя контекста>

#### EXTENDS

<список расширяемых контекстов>

#### SETS

<список множеств>

#### CONSTANTS

<список констант>

#### AXIOMS

<имя аксиомы>: <предикат>

Вначале контексту присваивается имя, которое будет являться его идентификатором. Затем приводится список контекстов, которые расширяются данным контекстом (модели в Event-B могут быть уточнены – это позволяет углубляться в модель, с каждым шагом детализируя ее абстрактное представление о реальной системе). В контексте перечисляются множества и константы, свойства которых задаются в виде аксиом. Каждая аксиома идентифицируется по имени и представляет собой предикат – определенное утверждение о множествах и константах.

На рисунке 3.1 представлен пример контекста.

```
CONTEXT
  m0_StateMachine_sm0
SETS
  - sm0_STATES
CONSTANTS
  - sm0_NULL
  - awaiting
  - processing
  - login_ok
  - login_failed
AXIOMS
  - typeof_sm0_NULL: sm0_NULL ∈ sm0_STATES not theorem
  - typeof_awaiting: awaiting ∈ sm0_STATES not theorem
  - typeof_processing: processing ∈ sm0_STATES not theorem
  - typeof_login_ok: login_ok ∈ sm0_STATES not theorem
  - typeof_login_failed: login_failed ∈ sm0_STATES not theorem
  - distinct_states_in_sm0_STATES: partition(sm0_STATES, {awaiting}, {processing}, {login_ok}, {login_failed}, {sm0_NULL}) not theorem
END
```

Рисунок 3.1 – Пример контекста на Event-B

Машина, как сказано выше, представляет собой динамическую часть модели.

**MACHINE**

<ИМЯ МАШИНЫ>

**REFINES**

<СПИСОК УТОЧНЯЕМЫХ МАШИН>

**SEES**

<СПИСОК РЕЛЕВАНТНЫХ КОНТЕКСТОВ>

**VARIABLES**

<СПИСОК ПЕРЕМЕННЫХ>

**INVARIANTS**

<ИМЯ ИНВАРИАНТА>: <ПРЕДИКАТ>

**VARIANTS**

<ВАРИАНТ>

**EVENTS**

<СПИСОК СОБЫТИЙ>

Описание машины содержит имя машины, список уточняемых машин (которые расширяются данной машиной), список контекстов, которые доступны машине, список переменных, список инвариантов (при помощи которых обычно описываются условия безопасности), варианты<sup>7</sup> и события.

**О событиях**

События машины определяют то, как система развивается – как изменяются переменные при срабатывании события.

События могут быть обычными (ordinary), конвергентными (convergent) и предполагаемыми (anticipated).

События могут содержать локальные переменные (выступают параметрами события) и сторожевые выражения, которые должны быть истинны, чтобы событие было инициировано.

Если сторожевое выражение истинно, событие запускает свое действие, которое в свою очередь назначает переменным новые значения.

Действия могут быть описаны в детерминированном (1) или недетерминированном (2) виде. При детерминированном подходе происходит явное присваивание нового значения переменной. При недетерминированном подходе `before_after_predicate` показывает взаимосвязь между переменной прямо перед и сразу после присваивания (это называется темпоральная логика действий). На примерах в формуле (2) переменная со

---

<sup>7</sup> Вариант – это некоторое числовое выражение (можно назвать счетчиком), которое уменьшается каждый раз при наступлении так называемых «конвергентных» событий. Если вариант становится отрицательным, то такое событие не сможет больше запускаться. Таким образом описывается защита от захвата системы определенными событиями.

штрихом обозначает состояние переменной на следующем шаге. Может показаться, что представленные примеры очень похожи ввиду того, что недетерминированное присваивание обобщает детерминированное.

$$\begin{aligned} < \text{variable\_identifier} > := < \text{expression} > \\ \text{пр. 1: } x := x + 1 \\ \text{пр. 2: } x, y := x + 1, y + 1 \end{aligned} \quad (1)$$

$$\begin{aligned} < \text{variable\_identifier\_list} > : | < \text{before\_after\_predicate} > \\ \text{пр. 1: } x : | x' = x + 1 \\ \text{пр. 2: } x, y : | x' = x + 1 \wedge y' = y + 1 \end{aligned} \quad (2)$$

На рисунке 3.2 представлен пример описания машины. В описании событий можно увидеть ключевые слова, такие как `where` (описывают сторожевые выражения) или `then` (описывают действия).

```

MACHINE
  m0 >
SEES
  ◦ m0_Statemachine_sm0
VARIABLES
  ◦ sm0 >
  ◦ conn >
  ◦ login >
INVARIANTS
  ◦ typeof_sm0: sm0 ∈ sm0_STATES not theorem >
  ◦ sm0_invariants1: (sm0 = processing) ⇒ (conn = TRUE) not theorem >
  ◦ sm0_invariants2: (sm0 = login_ok) ⇒ (login = TRUE) not theorem >
  ◦ sm0_invariants3: (sm0 = login_failed) ⇒ (login = FALSE) not theorem >
  ◦ typeof_login: login ∈ BOOL not theorem >
EVENTS
  ◦ INITIALISATION: not extended ordinary >
    THEN
      ◦ init_sm0: sm0 = awaiting >
      ◦ init_conn: conn = FALSE >
      ◦ init_login: login = FALSE >
    END
  ◦ response: not extended ordinary >
    WHERE
      ◦ isin_login_failed: sm0 = login_failed not theorem >
    THEN
      ◦ enter_awaiting: sm0 = awaiting >
      ◦ act1: conn = FALSE >
    END

```

Рисунок 3.2 – Пример машины на Event-B

### 3.2. iUML-B

iUML-B – это набор инструментов для создания и редактирования диаграмм для Event-B. Такие диаграммы используются для разработки моделей (машин и их контекстов).

iUML-B позволяет создавать State-machine diagrams (конечные автоматы<sup>8</sup>) и Class diagrams (диаграммы классов – не затронуты в этой методике). iUML-B снижает порог входа в Event-B и позволяет визуализировать модели. Кроме того, построение диаграмм быстрее, чем написание чистого Event-B кода (добавили один элемент – получили пару строк кода). По сути, iUML-B переводит полужформальное описание (диаграмму) в формальное (код на Event-B) при помощи трансляции.

В данной методике вы познакомитесь с iUML-B State-machine diagrams – редактором диаграмм конечных автоматов для описания систем, работу которых можно представить в виде последовательности событий.

На рисунке 3.3 представлен простейший конечный автомат, состоящий из двух состояний (s1, s2) и одного перехода (e1).

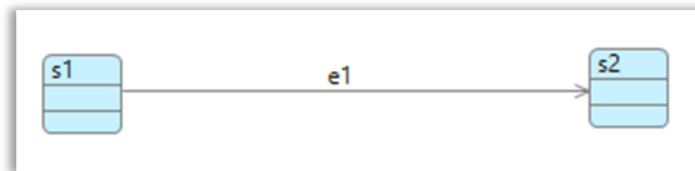


Рисунок 3.3 – Пример конечного автомата

Что мы можем увидеть? Переход e1 возможен только если активно состояние s1. Переход e1 изменяет состояние на s2. Описание на Event-B выглядит следующим образом:

**EVENTS**

```
e1 ≜ WHEN < in s1 > THEN < becomes s2 > END
```

, где <in s1> и <becomes s2> – информация о состояниях.

Конечно, необходимо знать, в каком состоянии находится система в каждый момент времени. Для этого состояния системы представляются в виде элементов множества (это называется Enumeration translation) либо в виде булевых переменных (Variables translation).

Enumeration translation представляет состояния конечного автомата как множество.

**CONTEXT**

...

**SETS**

```
sm_STATES = {s1, s2}
```

<sup>8</sup> Конечный автомат можно представить в виде ориентированного графа, где точки – это определенные состояния, а стрелки – направления переходов.

**MACHINE**

...

**VARIABLES** $sm \in sm\_STATES$ **EVENTS** $e1 \triangleq WHEN\ sm = s1\ THEN\ sm := s2\ END$ 

Variables translation представляет каждое состояние конечного автомата как отдельную переменную. Когда происходит переход из  $s1$  в  $s2$ , изменяются значения соответствующих переменных. В каждый момент времени только одна из переменных может быть TRUE.

**MACHINE**

...

**VARIABLES** $s1 \in BOOL$  $s2 \in BOOL$ **EVENTS**

$$e1 \triangleq WHEN\ s1 = TRUE$$

$$THEN\ s1 := FALSE$$

$$s2 := TRUE$$

$$END$$

Если взглянуть на рисунок 3.3, то возникает закономерный вопрос – как мы оказались в состоянии  $s1$ ? А все потому, что каждый автомат должен содержать начальное состояние и первоначальный (инициализирующий) переход.

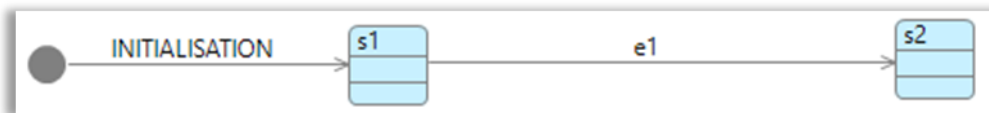


Рисунок 3.4 – Пример конечного автомата с начальным состоянием и инициализирующим переходом

Событие INITIALISATION создается автоматически при создании машины. С его помощью объявляются и инициализируются переменные. Так, для нашего примера на рисунке 3.4 при инициализации возможны два варианта развития событий (в зависимости от вида представления состояний):

1. Случай представления состояний в виде элементов множества (Enumeration translation). В этом случае происходит присвоение переменной  $sm$  состояния  $s1$ .

**INITIALISATION** $sm := s1$

2. Случай представления состояний в виде булевых переменных (Variables translation). В этом случае происходит присвоение переменной  $s1$  значения  $TRUE$ , а переменной  $s2$  значения  $FALSE$ .

**INITIALISATION**

$s1 := TRUE$   
 $s2 := FALSE$

Конечный автомат, представленный на рисунке 3.4, все еще очень прост – в нем не хватает одной из главных составляющих, которая описывает требования безопасности. Конечно, речь идет об инвариантах.

Инварианты привязаны к состояниям, и они должны быть строго выполнены, если система находится в таком состоянии (рисунок 3.5).



Рисунок 3.5 – Пример конечного автомата с инвариантом в состоянии  $s2$

На формальном языке инвариант описывается как импликация (в естественном языке импликация имеет смысл выражения «ЕСЛИ ..., ТО ...»), но его описание может различаться в зависимости от того, какое представление состояний мы выбрали.

1. Случай представления состояний в виде элементов множества. В этом случае в качестве «ЕСЛИ ...» выступает условие того, что переменной  $sm$  присвоено значение состояния  $s2$ , а в качестве «ТО ...» необходимое нам «условие безопасности».

**INVARIANTS**

$(sm = s2) \Rightarrow (safe = TRUE)$

2. Случай представления состояний в виде булевых переменных. В этом случае в качестве «ЕСЛИ ...» выступает условие того, что булевой переменной, обозначающей состояние  $s2$ , присвоено значение  $TRUE$ , а в качестве «ТО ...» необходимое нам «условие безопасности».

**INVARIANTS**

$(s2 = TRUE) \Rightarrow (safe = TRUE)$

Так как переходы, по сути, являются полужформальным представлением событий, удобно прямо на диаграмме задавать им параметры, назначать на них сторожевые выражения и действия (рисунок 3.6).



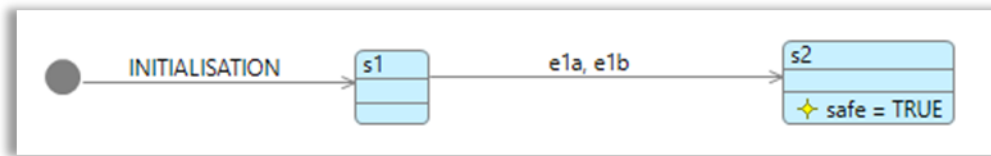


Рисунок 3.6 – Пример конечного автомата с двумя событиями (показаны в виде одной стрелки) и сторожевыми выражениями (на рисунке 3.7)

```

e1a ≐
STATUS
ordinary
WHEN
  isin_s1 : sm0 = s1
  grd1    : const ≥ 5
THEN
  enter_s2 : sm0 = s2
END

e1b ≐
STATUS
ordinary
WHEN
  isin_s1 : sm0 = s1
  grd1    : const < 5
THEN
  enter_s2 : sm0 = s2
END
  
```

Рисунок 3.7 – Описание событий e1a, e1b в нотации Event-B

Как вы уже знаете, модели в Event-B можно подвергать уточнению, что помогает проще осмыслить систему. Зачем формально описывать всю систему целиком, когда можно сделать это пошагово?

Уточнение конечных автоматов производится с использованием вложенности.

### Важно!

При уточнении на диаграмме можно создавать только вложенные состояния.

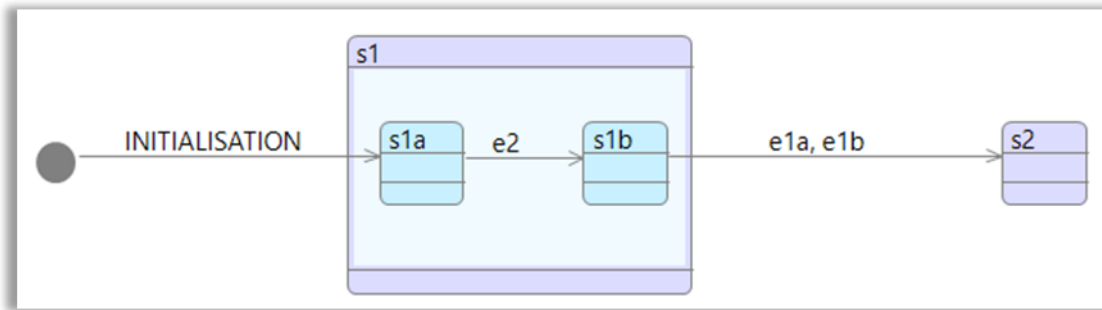


Рисунок 3.8 – Уточненная модель, построенная с использованием вложенности состояний

На рисунке 3.8 мы уточнили модель и добавили два вложенных состояния  $s1a$  и  $s1b$ , а также переход между ними –  $e2$ . При уточнении разрешается:

- устанавливать инвариант как на вложенные, так и на суперсостояния;
- привязывать переход как к вложенным, так и к суперсостояниям;
- добавлять параллельные конечные автоматы внутри суперсостояния (чтобы строить схемы, подобные изображенной на рисунке 3.9).

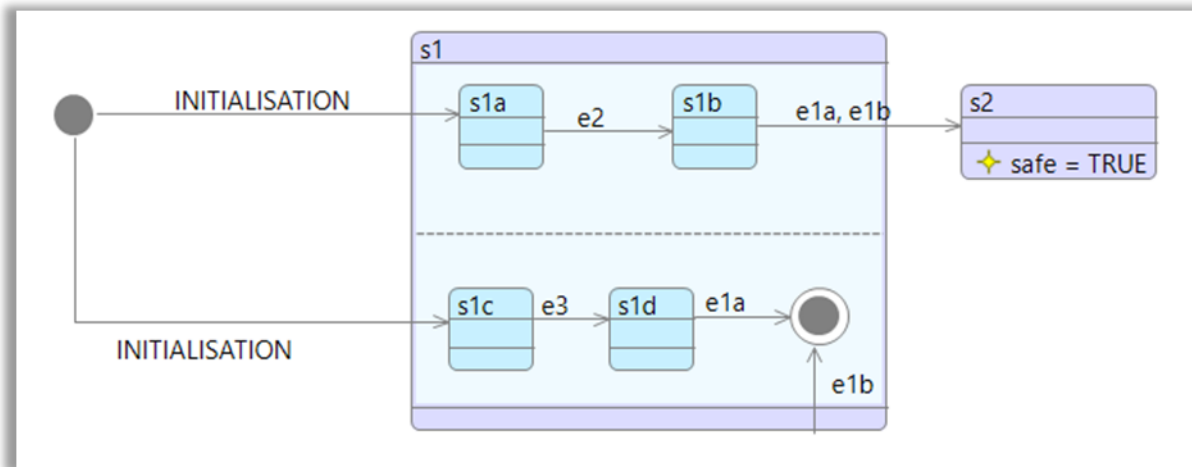


Рисунок 3.9 – Пример уточненного конечного автомата с параллельными состояниями

Конечно, никто не застрахован от ошибок, поэтому, когда создаете сложные схемы с множеством переходов, будьте внимательны и не создавайте невыполнимых переходов (пример на рисунке 3.10).

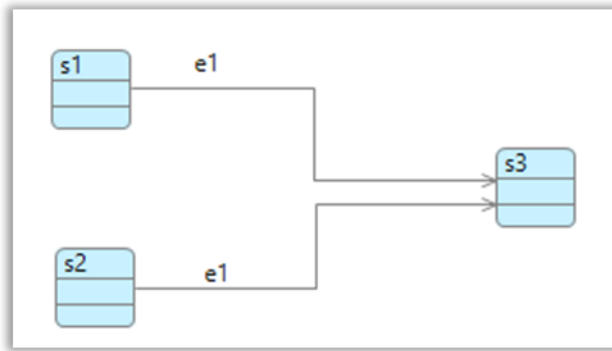


Рисунок 3.10 – Ошибочное представление

Разумеется, конечный автомат не может находиться в двух состояниях одновременно (исключение – параллельные состояния, которые будут рассмотрены далее). Для случая, показанного выше, необходимо использовать специальные соединительные узлы (Junction) (рисунки 3.11 и 3.12).

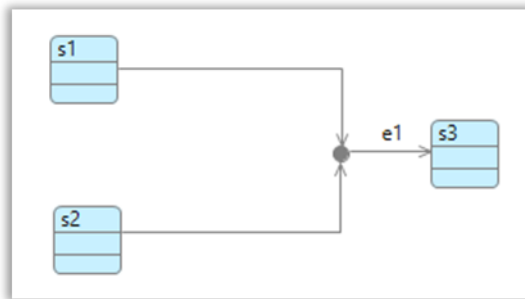


Рисунок 3.11 – Пример использования Junction

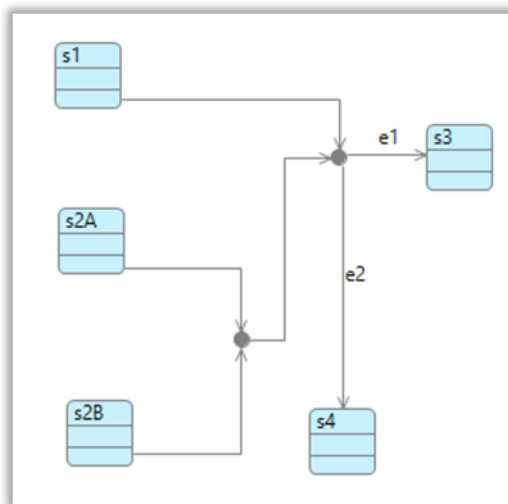


Рисунок 3.12 – Комбинированное использование соединительных узлов

### Обратите внимание!

Событие привязывается только к исходящему из последнего узла переходу. При этом для входящих в любой узел переходов можно задать сторожевые выражения.

Если вернуться к рисунку 3.9, то можно увидеть очень интересный переход e1b. Такие переходы (привязанные к суперсостоянию) позволяют переходить из любого вложенного состояния к желаемому.

Если вложенности нет, то для перехода из любого состояния можно воспользоваться узлом ANY (символ звезды на рисунке 3.13).

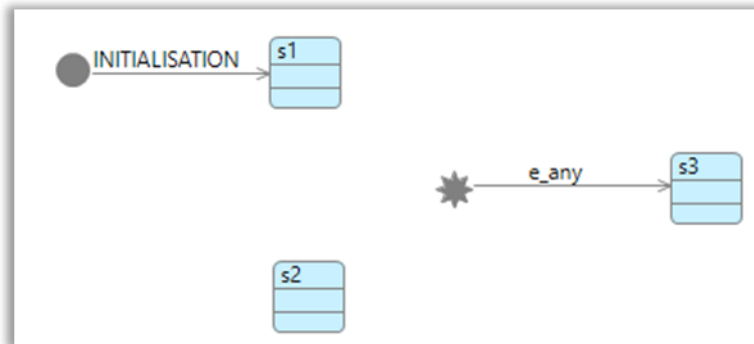


Рисунок 3.13 – Узел ANY позволяет переходить из любого состояния в желаемое

## 4. СРЕДСТВО ФОРМАЛЬНОГО ОПИСАНИЯ RODIN

### 4.1. Общая информация и основные функции Rodin

Свободно распространяемая платформа Rodin предоставляет собой средства разработки, анализа и верификации моделей на Event-B. Платформа содержит текстовый редактор моделей, набор систем автоматического доказательства и поддержку интерактивного доказательства. Функционал Rodin может быть расширен за счет использования плагинов.

Доступные плагины позволяют писать собственные расширения формального метода Event-B, дополнительные системы автоматического доказательства, инструменты для анимации и model checking, и т. д.

### 4.2. Работа с Rodin

Для начала работы с платформой Rodin убедитесь, что у вас установлена Java версии 8. Для этого откройте терминал и введите:

```
java -version
```

В терминале должно отобразиться:

```
java version "1.8.0_xyz" //1.8.0 - означает 8 версию, а _xyz - № обновления
```

Если вы получили ошибку «java не является внутренней или внешней командой...», то скорее всего, на вашем рабочем компьютере отсутствует Java. Скачайте и установите 64-разрядную версию Java, после чего перезагрузите ПК.

Далее необходимо установить Rodin с официального сайта (актуальная версия платформы на момент написания методики – 3.6.0). Для запуска платформы Rodin кликните на исполняемый файл rodin.exe, дождитесь загрузки мастера выбора рабочей директории (workspace). Рабочая директория необходима для хранения информации о ваших проектах (рисунок 4.1). Если в этот момент вы получили ошибку Java was started but returned exit code=13, то у вас установлена 32-разрядная версия Java.

После старта Rodin вы увидите окно приветствия, содержащее обзор платформы и иную полезную информацию (рисунок 4.2). Закройте окно приветствия, щелкнув по крестику у вкладки Welcome.

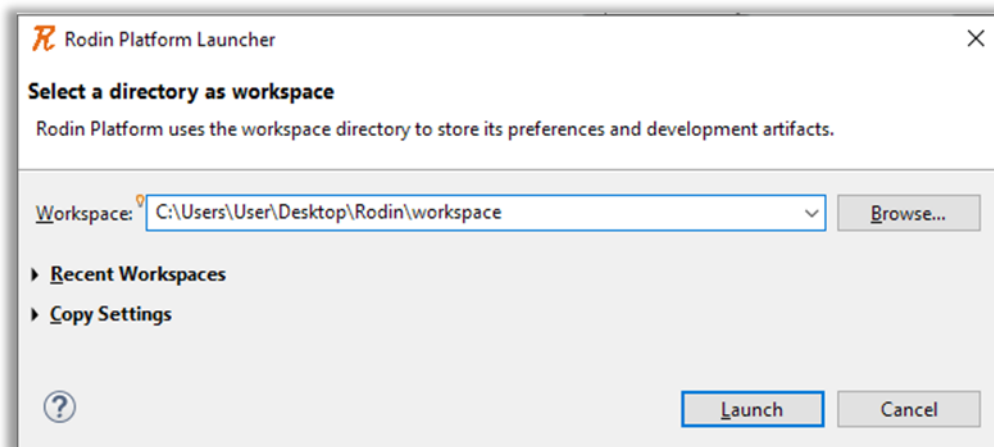


Рисунок 4.1 – Выбор рабочей директории




Рисунок 4.2 – Окно приветствия Rodin

Далее необходимо установить обязательные плагины, которые понадобятся для выполнения работ по методике. Для этого перейдите в раздел Help – Install new software в поле Work with выберите All available sites и установите следующие плагины:

- iUMLB State-Machine;
- iUMLB State-Machine Animation;
- ProB for Rodin3x.

### Создание проекта

Чтобы создать проект, нажмите кнопку меню File – New – Event-B Project или вызовите контекстное меню создания элементов, нажав

сочетание клавиш (alt+shift+N), и во всплывающем меню выберите Event-B Project или нажмите на значок  обозревателя Event-B (левая панель – Event-B Explorer). В открывшемся мастере создания проекта введите имя вашего проекта и нажмите Finish.

### Создание модели

Как вы помните, модель состоит из двух частей – контекста и машины. Создать контекст для вашей модели можно несколькими путями:

- вызвать контекстное меню проекта, кликнув по нему правой кнопкой мыши, и нажать New – Event-B Component;
- вызвать контекстное меню создание элементов и нажать Event-B Component;
- нажать на кнопку меню File – New – Event-B Component.

В открывшемся мастере выберите тип создаваемого компонента и введите его имя. Для вашей первой машины можно выбрать имя m0, а для контекста c0, или же вы можете выбрать имя по своему усмотрению – главное, чтобы имя сразу давало вам понять, уточненный это компонент или корневой.

### Создание диаграммы

Для создания конечного автомата вызовите контекстное меню машины, кликнув по ней правой кнопкой, и нажмите Add Statemachine. Введите понятное имя (например, sm0) и нажмите ОК.

После всех проделанных манипуляций окно обозревателя Event-B должно выглядеть так, как показано на рисунке 4.3.

Кликните два раза на свою Statemachine, чтобы открыть окно редактирования диаграммы.

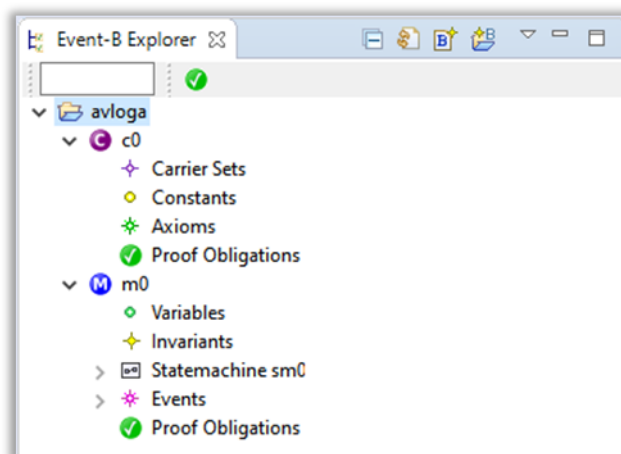


Рисунок 4.3 – Окно приветствия Rodin

## Проектирование подсистемы ПО на примере

Разберем пример на подсистеме идентификации и аутентификации. Начнем с формулирования требований на неформальном языке к абстрактному ПО, имеющему в своем составе функции безопасности:

1. После запуска ПО должно отображать приглашение ко входу в систему.
2. Ни один пользователь не может получить доступ к активам ПО (перейти в рабочий режим), пока не пройдет процедуру идентификации и аутентификации.
3. Пользователи, обладающие ролью «Администратор», дополнительно должны быть идентифицированы по IP-адресу машины, с которой осуществляется подключение.
4. У всех пользователей есть три попытки на прохождение идентификации и аутентификации. Если все попытки исчерпаны, ПО отображает окно входа в систему и дает возможность залогиниться еще (блокировки пользователя нет).

Из неформального описания сразу можно вывести несколько состояний:

1. Инициализация (про нее не стоит забывать).
2. Ожидание (приглашение ко входу в систему).
3. Идентификация и аутентификация.
4. Авторизация (здесь можно проверить роль пользователя и на основании этого принять решение о дополнительной идентификации).
5. Дополнительная идентификация по IP-адресу.
6. Рабочий режим (финальное состояние).
7. Генерация сообщения об ошибке (если идентификация или аутентификация не удалась).

Расставив состояния и переходы на диаграмме, получим схему, представленную на рисунке 4.4.

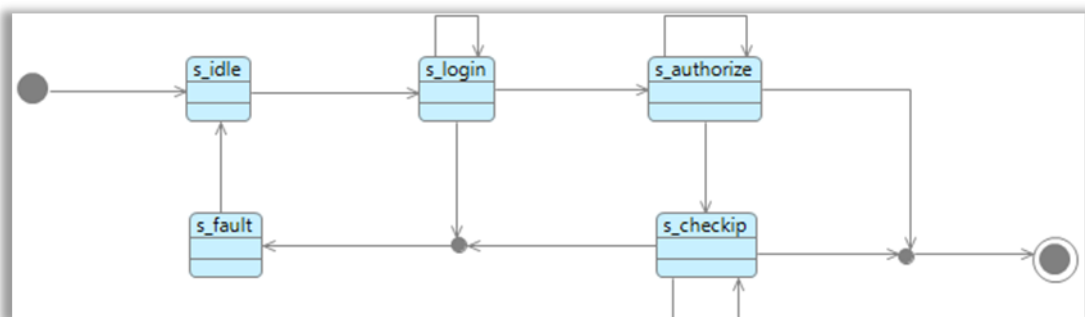



Рисунок 4.4 – Схема состояний с переходами

На панели инструментов вы можете увидеть группу из четырех кнопок . Первая кнопка позволяет проверить вашу диаграмму на соответствие определенным правилам перед ее трансляцией в Event-B



(валидация диаграммы). Вторая кнопка запускает трансляцию диаграммы. Третья кнопка анимирует диаграмму, что позволяет пошагово пройти по диаграмме и найти ошибки и неточности. Четвертая кнопка останавливает анимацию.

Сохраните диаграмму при помощи сочетания клавиш `ctrl+s` и попробуйте транслировать представление диаграммы в формальное описание. Event-B сообщит вам о том, что нарушено одно из ограничений – трансляция не будет осуществлена, пока с каждым их переходов не будет связано событие (Transition should elaborate an event) (рисунок 4.5).

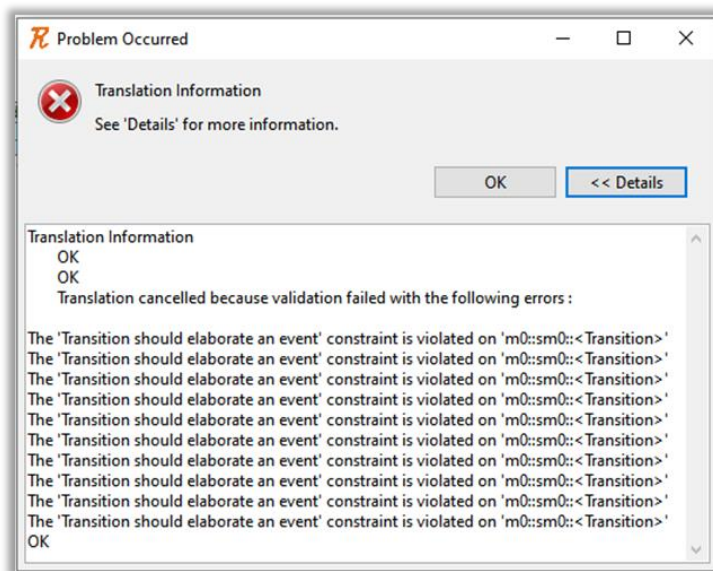


Рисунок 4.5 – Ошибки

Чтобы трансляция завершилась без ошибок, свяжите каждый переход с событием. Тем самым вы получите заготовку модели на Event-B.

Для того, чтобы связать переход с событием, щелкните на переход и в открывшемся меню в нижней части экрана выберите Properties – Overview (рисунок 4.6). Нажмите на кнопку Create & Link, введите имя события и нажмите ОК.

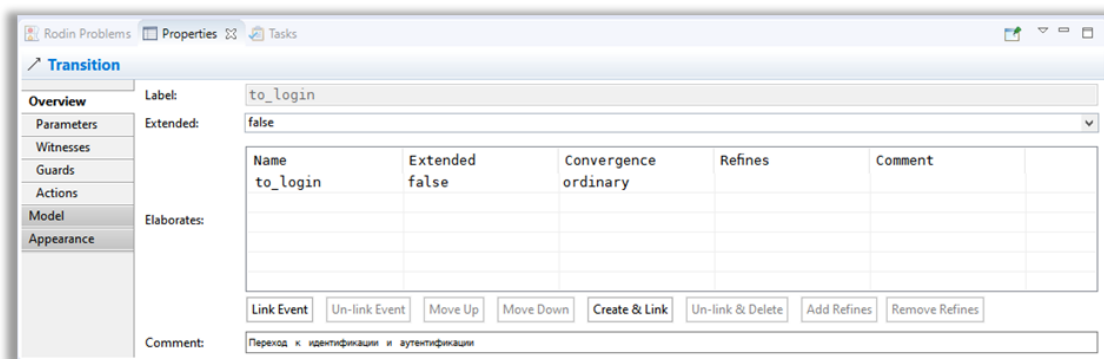


Рисунок 4.6 – Основные свойства перехода

## Как определить имя перехода?

Имя перехода можно задавать как «to\_имя\_состояния%», если переход изменяет состояние, и «do\_имя\_состояния%», если переход выполняет какое-то действие над состоянием.

## Важно!

Не забудьте про событие INITIALISATION. Оно создается вместе с машиной, а значит, создавать руками его не нужно. Кликните на самый первый переход, затем в окне свойств нажмите Link Event и выберите из списка INITIALISATION. С первым переходом всегда нужно связывать только событие INITIALISATION.

После завершения назначения событий на переходы сохраните диаграмму и попробуйте транслировать ее в Event-B. Откройте машину, щелкнув по ней двойным щелчком, и убедитесь, что все, что вы отразили на диаграмме, транслировалось в Event-B (рисунок 4.7).

```
MACHINE
  m0 >
VARIABLES
  ◦ s_idle >
  ◦ s_login >
  ◦ s_authorize >
  ◦ s_checkip >
  ◦ s_fault >
INVARIANTS
  ◦ typeof_s_idle: s_idle ∈ BOOL not theorem >
  ◦ typeof_s_login: s_login ∈ BOOL not theorem >
  ◦ typeof_s_authorize: s_authorize ∈ BOOL not theorem >
  ◦ typeof_s_checkip: s_checkip ∈ BOOL not theorem >
  ◦ typeof_s_fault: s_fault ∈ BOOL not theorem >
  ◦ distinct_states_in_sm0: TRUE ∈ {s_idle, s_login, s_authorize, s_checkip, s_fault} ⇒ partition({TRUE},
EVENTS
  ◦ INITIALISATION: not extended ordinary >
    THEN
      ◦ init_s_idle: s_idle = TRUE >
      ◦ init_s_login: s_login = FALSE >
      ◦ init_s_authorize: s_authorize = FALSE >
      ◦ init_s_checkip: s_checkip = FALSE >
      ◦ init_s_fault: s_fault = FALSE >
  END
```

Рисунок 4.7 – Представление на Event-B

## Переменные, множества и константы

Перейдем к определению необходимых для описания модели переменных, множеств и констант. Для этого обратимся к неформальному описанию требований к подсистеме:

1. После запуска ПО должно отображать приглашение ко входу в систему.

Здесь мы не будем вводить никаких дополнительных переменных, множеств и констант.

2. Ни один пользователь не может получить доступ к активам ПО (перейти в рабочий режим), пока не пройдет процедуру идентификации и аутентификации.

Исходя из описания требования, становится понятно, что в модели должна быть переменная, отражающая результат идентификации и аутентификации.

Достаточно булевой переменной, принимающей TRUE, если идентификация и аутентификация пройдены успешно, и FALSE в противном случае. Пусть такая переменная называется `has_access`.

3. Пользователи, обладающие ролью «Администратор», дополнительно должны быть идентифицированы по IP-адресу машины, с которой осуществляется подключение.

Если до этого момента речь шла только о пользователях в целом, то в этом требовании у нас появляется новая сущность – «Администратор». Значит, в нашем ПО есть как минимум два типа пользователей – обычные пользователи и администраторы. Чтобы отделить юзеров от админов, необходимо определить множество ролей `roles`, а также константы, из которых состоит множество: `user`, `admin`, и ввести переменную, которая отражает роль текущего пользователя – `role`.

Кроме того, необходимо определить множество IP-адресов `ips`, константы, отражающие адрес, с которого разрешен и запрещен доступ администратору: `ip_allowed`, `ip_disallowed`, а также IP-адрес, с которого осуществляется подключение – `ip`.

4. У всех пользователей есть три попытки на прохождение идентификации и аутентификации. Если все попытки исчерпаны, ПО отображает окно входа в систему.

Для реализации данного требования можно ввести целочисленную переменную `login_counter` и присвоить ей значение при инициализации, равное 3.

Подытожим, что мы должны определить:

- переменные: `has_access`, `role`, `ip`, `login_counter`;
- множества: `roles`, `ips`;
- константы: `role_NULL`, `user`, `admin`, `ip_NULL`, `ip_allowed`, `ip_disallowed`.

Введение NULL-констант обусловлено необходимостью инициализации переменных `role` и `ip` в условиях отсутствия на стадии инициализации ПО информации о том, какой пользователь с какого IP-адреса осуществляет подключение.

Добавить такие специфичные параметры из диаграммы нельзя, поэтому придется вписывать их вручную. Щелкните правой кнопкой мыши по вашей машине и в контекстном меню выберите `Open With - Event-V Machine Editor`. Должно открыться окно редактора (рисунок 4.8).

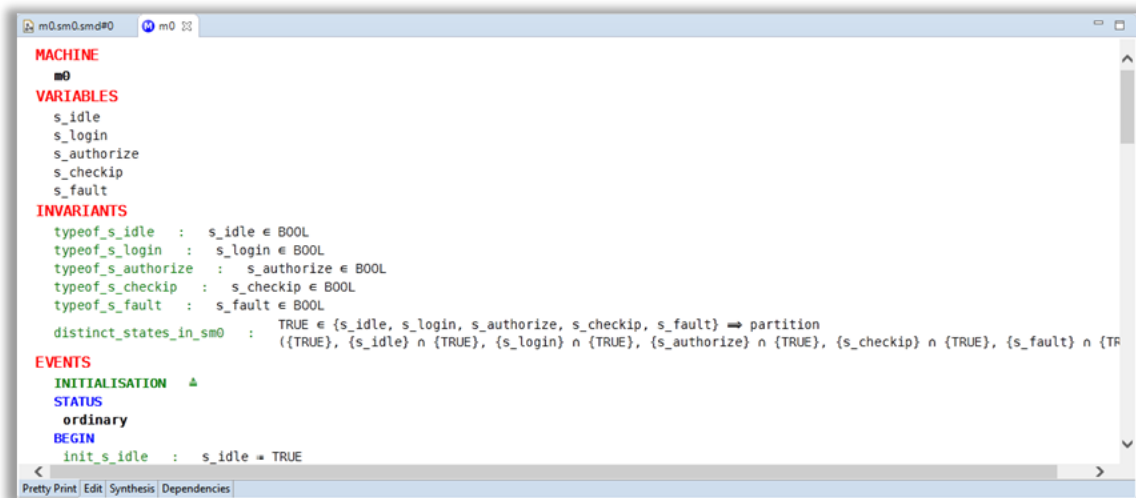


Рисунок 4.8 – Окно редактора Event-B машины

Щелкните по закладке Edit (в левом нижнем углу окна), чтобы перейти в режим редактирования машины. У списка VARIABLES кликните на кнопку «Добавить» (+ в ●), чтобы добавить новую переменную (рисунок 4.9).

Задайте понятное имя для переменной, например, как уже обсуждалось выше, has\_access (рисунок 4.10). Но мало только определить имя переменной, необходимо также определить ее тип. Для этого кликните на «Добавить» рядом с INVARIANTS, задайте понятное имя (лучше всего брать пример с того, как уже заданы иные переменные – typeof\_has\_access) и в качестве предиката (справа от двоеточия) введите has\_access ∈ BOOL (рисунок 4.11). Тем самым мы определили булеву переменную, то есть она сможет принимать значения TRUE или FALSE. Нажмите ctrl+s для сохранения машины.

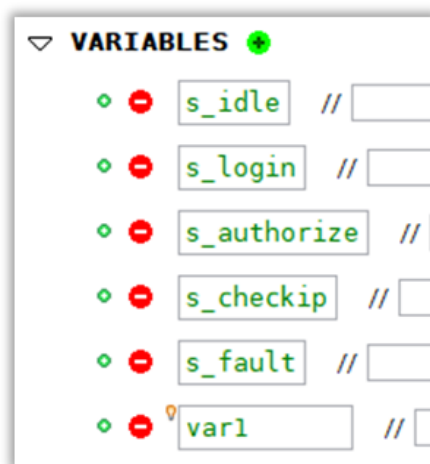


Рисунок 4.9 – блок VARIABLES

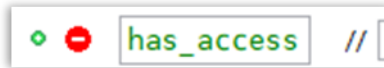


Рисунок 4.10 – Название переменной в блоке VARIABLES

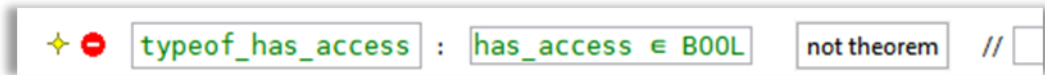


Рисунок 4.11 – Определение типа в блоке INVARIANTS

### Об использовании символов

Математические символы можно вставлять в выражения, используя таблицу в правом нижнем углу (Symbols) или вводя их с клавиатуры (так, двоеточие преобразуется в  $\in$ ). Наведите мышью на символ в таблице символов, чтобы узнать больше.

Если вдруг в вашей модели имеются какие-либо ошибки (здесь речь идет про синтаксические ошибки, а не про логические), то в обозревателе Event-B вы увидите предупреждающие символы (рисунки 4.12 и 4.13).

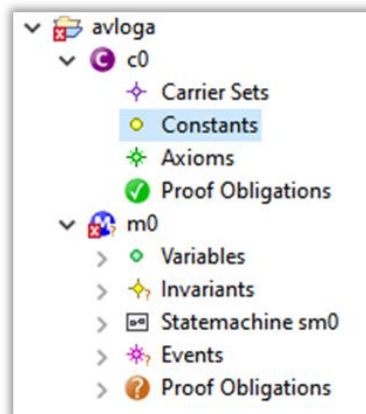


Рисунок 4.12 – Ошибки

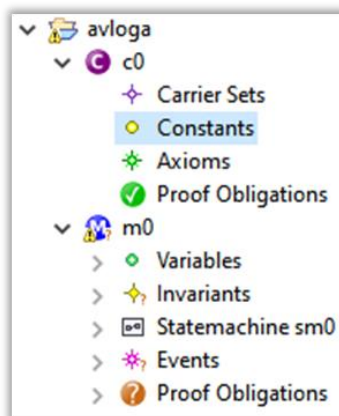


Рисунок 4.13 – Предупреждения

✘ означает, что есть критические ошибки, например, если задано только имя переменной без указания ее типа.

⚠ означает, что есть небольшие ошибки. Некоторые из них Rodin может автоматически исправить, например, если переменная была определена корректно, но не проинициализирована, Rodin присвоит ей значение по умолчанию – такое предупреждение вы как раз и должны наблюдать. Мы определили имя переменной, определили ее тип, но не присвоили ей значение.

Присвоение значений всем переменным происходит в событии INITIALISATION. Если вы откроете машину в режиме редактирования и раскроете список событий, то рядом с событием INITIALISATION можно увидеть предупреждение (рисунок 4.14).

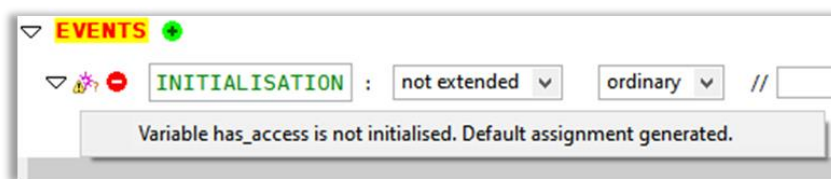


Рисунок 4.14 – Предупреждение о неинициализированной переменной

Чтобы присвоить переменной значение, раскройте событие INITIALISATION и нажмите на «Добавить» у THEN.

Введите понятное имя, например, `init_has_access` и предикат, выставляющий значение переменной в FALSE (`has_access:=FALSE`) (рисунок 4.15). После сохранения машины предупреждение должно пропасть.

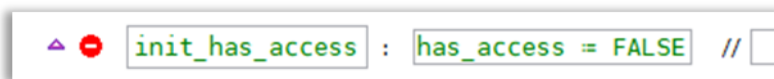


Рисунок 4.15 – Инициализация переменной

### Об инициализации

Инициализируйте переменные корректно:

- 1) Определите имя переменной в VARIABLES.
- 2) Определите тип переменной в INVARIANTS.
- 3) Присвойте значение переменной в событии INITIALISATION.

По аналогии вы можете попробовать определить и проинициализировать переменную `login_counter` самостоятельно (или читайте далее, как это сделать).

### Решение

- 1) Определяем имя переменной: `login_counter`
- 2) Определяем ее тип: `typeof_login_counter: login_counter ∈ Z`
- 3) Присваиваем ей значение: `init_login_counter: login_counter := 3`

Давайте вычеркнем из списка то, что мы уже определили:

- переменные: `has_access`, `role`, `ip`, `login_counter`;
- множества: `roles`, `ips`;
- константы: `role_NULL`, `user`, `admin`, `ip_NULL`, `ip_allowed`, `ip_disallowed`.

Для того чтобы определить оставшиеся переменные, необходимо определить для них новые типы данных при помощи множеств и констант.

Как уже говорилось ранее, множества задаются в контексте модели.

Вызовите контекстное меню вашего контекста, щелкнув по нему правой кнопкой мыши, и нажмите `Open With - Event-B Context Editor`.

Перейдите во вкладку `Edit` и нажмите на кнопку «Добавить» рядом с `SETS`. Введите имя множества `roles`.

Нажмите «Добавить» рядом с `CONSTANTS`. Введите имя `role_NULL`. Добавьте аналогично `user` и `admin`.

Нажмите «Добавить» рядом с `AXIOMS`. Здесь мы зададим утверждение, что множество `roles` состоит из `role_NULL`, `user`, `admin`. Мы зададим его как разбиение – `partition(S, {x}, {y}...)`<sup>9</sup> (рисунок 4.16).

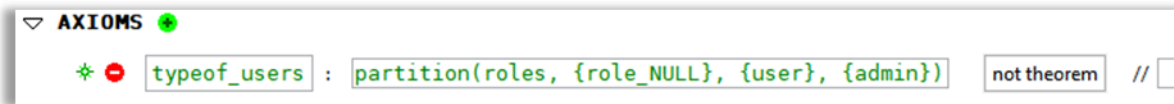


Рисунок 4.16 – Блок AXIOMS

По аналогии задайте множество и константы для IP-адресов и в конце сверьтесь с рисунком 4.17.

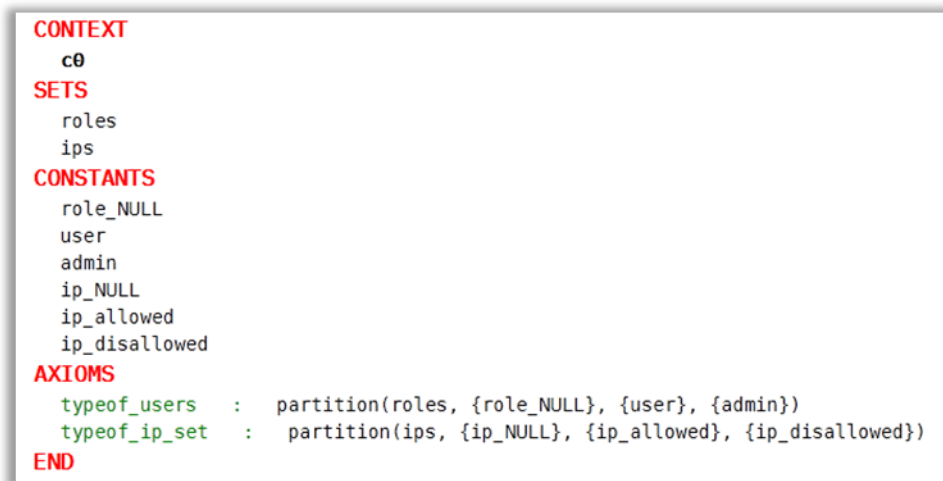


Рисунок 4.17 – Описание контекста

<sup>9</sup> `partition(S, {x}, {y})` – разбиение множества `S`. Задаёт его в виде объединения попарно непересекающихся непустых подмножеств (enumerated set).

Давайте снова вернемся к списку и вычеркнем то, что мы уже определили:

- переменные: `has_access`, `role`, `ip`, `login_counter`;
- множества: `roles`, `ips`;
- константы: `role_NULL`, `user`, `admin`, `ip_NULL`, `ip_allowed`, `ip_disallowed`.

А теперь вернемся к определению переменных. Попробуйте сделать это самостоятельно. Определите в `VARIABLES` имена переменных, определите их типы в `INVARIANTS` и наконец присвойте им значения в `INITIALISATION EVENT`. Если Rodin сообщает вам об ошибке, сравните ваши значения с указанными на рисунках 4.18 – 4.20.

```
VARIABLES  
s_idle  
s_login  
s_authorize  
s_checkip  
s_fault  
has_access  
login_counter  
role  
ip
```

Рисунок 4.18 – Переменные

```
INVARIANTS  
typeof_s_idle : s_idle ∈ BOOL  
typeof_s_login : s_login ∈ BOOL  
typeof_s_authorize : s_authorize ∈ BOOL  
typeof_s_checkip : s_checkip ∈ BOOL  
typeof_s_fault : s_fault ∈ BOOL  
distinct_states_in_sm0 : TRUE ∈ {s_idle, s_login, s_authorize, s_checkip, s_fault}  
typeof_has_access : has_access ∈ BOOL  
typeof_login_counter : login_counter ∈ ℤ  
typeof_role : role ∈ roles  
typeof_ip : ip ∈ ips
```

Рисунок 4.19 – Инварианты



```

EVENTS
INITIALISATION ≙
STATUS
ordinary
BEGIN
init_s_idle : s_idle = TRUE
init_s_login : s_login = FALSE
init_s_authorize : s_authorize = FALSE
init_s_checkip : s_checkip = FALSE
init_s_fault : s_fault = FALSE
init_has_access : has_access = FALSE
init_role : role = role_NULL
init_login_counter : login_counter = 3
init_ip : ip = ip_NULL
END

```

Рисунок 4.20 – Событие инициализации

### События, инварианты и сторожевые выражения

Вернемся к диаграмме. После того, как вы добавили на диаграмму события, она должна выглядеть примерно так, как на рисунке 4.21.

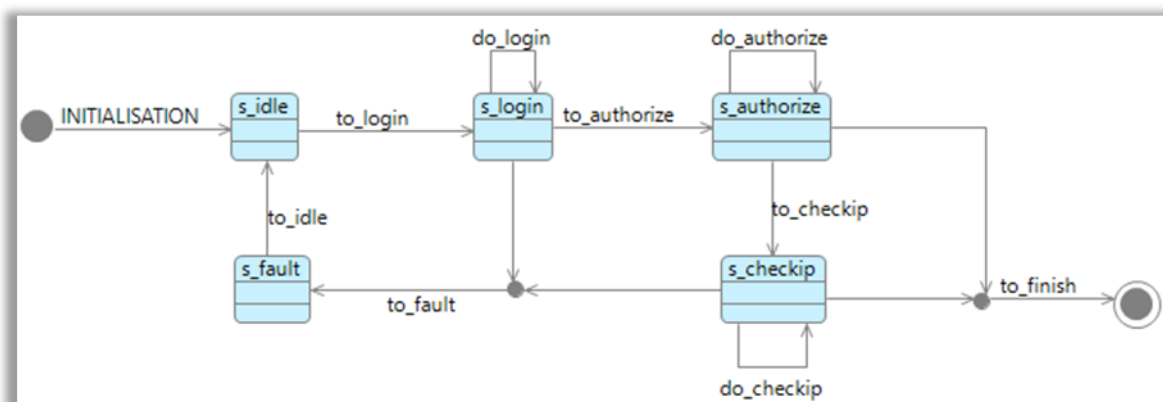



Рисунок 4.21 – Диаграмма после добавления событий

Нажмите на кнопку Run animation  на панели инструментов Rodin. Внешний вид Rodin должен измениться на ProB<sup>10</sup> – слева должна появиться панель Events, а справа State и History (рисунки 4.22, 4.23).

<sup>10</sup> ProB – это расширение для Event-B, позволяющее анимировать модель и занимающееся ее проверкой при помощи Model Checking.

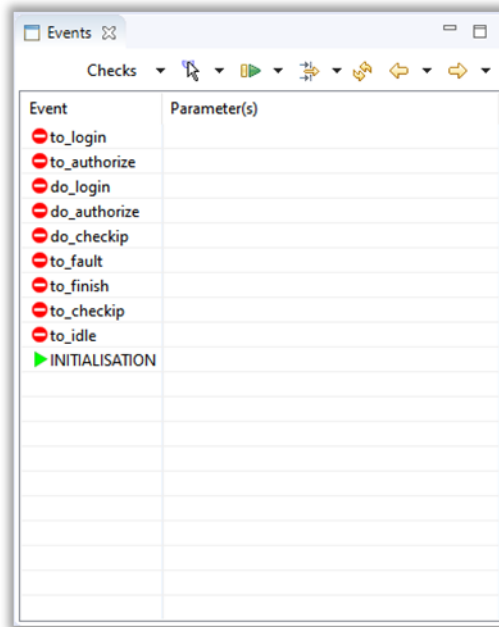


Рисунок 4.22 – Панель управления событиями

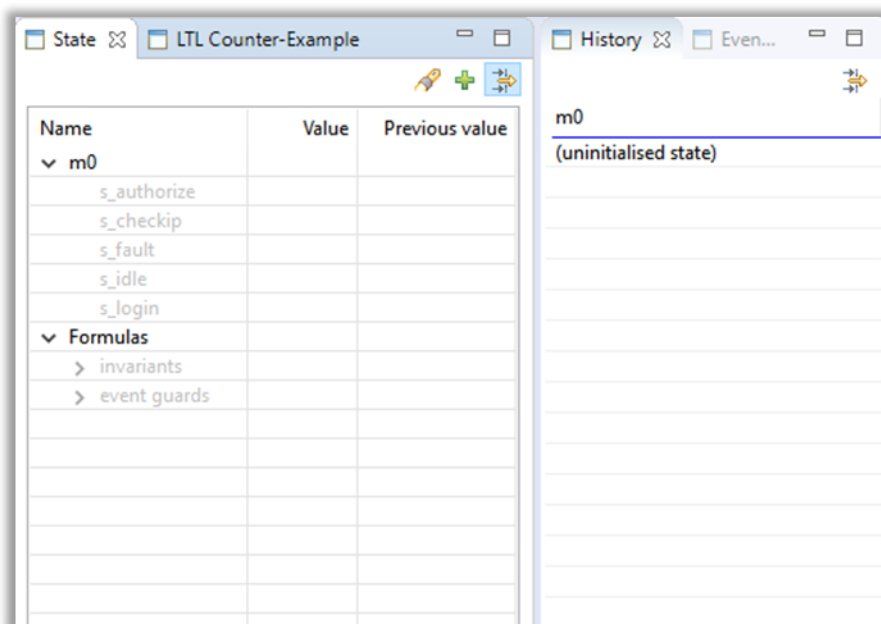


Рисунок 4.23 – Панель состояния модели и истории изменений

### Как изменять вид?

Старайтесь использовать кнопки на панели инструментов для запуска/отключения анимации. Это позволит корректно сменять вид ProB в стандартный вид Event-B и обратно. Если вид не сменяется автоматически, то через меню Windows – Show View.

Щелкните два раза на **INITIALISATION** на панели событий, чтобы запустить событие инициализации и совершить переход в состояние `s_idle`. Изменение состояния должно отобразиться и на диаграмме (рисунок 4.24)

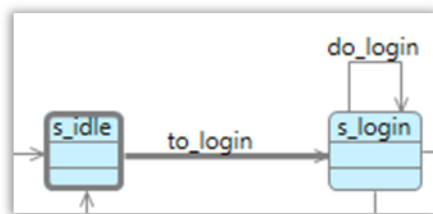


Рисунок 4.24 – Событие `s_idle` и переход `to_login`

Вы также можете запускать события, щелкая по переходам на диаграмме. Щелкните по `to_login` и перейдите в состояние `s_login`. Как вы можете заметить, из состояния `s_login` сразу же возможен переход `to_authenticate`, то есть модель построена так, что пользователь может перейти к авторизации, не пройдя процедуру идентификации и аутентификации (рисунок 4.25).

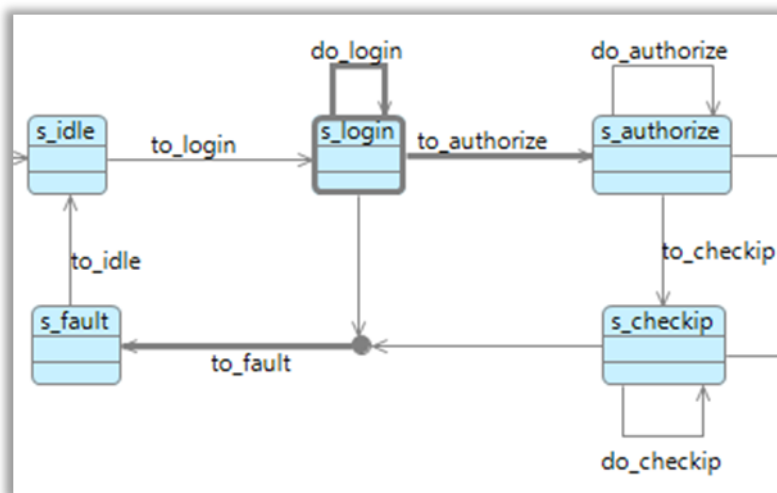


Рисунок 4.25 – Состояние `s_login` и связанные переходы

Чтобы это исправить, необходимо:

- присвоить параметр;
- установить инварианты;
- установить сторожевые выражения.

Выйдите из режима анимации, щелкнув по кнопке `Stop Animation`



Разберем состояние `s_login` и связанные с ним переходы. Первый переход `do_login` играет роль «функции» идентификации и аутентификации – с его помощью определяется, имеет пользователь доступ или нет. То есть `do_login` должен выставлять значение `has_access` в `TRUE` или `FALSE`. Чтобы иметь такую

возможность, щелкните по переходу `do_login` на диаграмме, затем в открывшемся окне свойств перехода выберите `Parameters` и нажмите `Add Parameter`. Задайте имя параметра `p_has_access` и тип `BOOL`. Чтобы в результате события переменной `has_access` присваивалось значение параметра, выберите `Actions` в окне свойств перехода и нажмите `Add Action`. Укажите имя `set_has_access` и действие `has_access:=p_has_access`.

Вы можете запустить анимацию, выполнить событие `do_login` и убедиться в том, что `has_access` присваивается то значение, которое вы выбрали – для этого, находясь в состоянии `s_login`, щелкните на переход `do_login` и выберите `do_login [TRUE]` (рисунок 4.26).

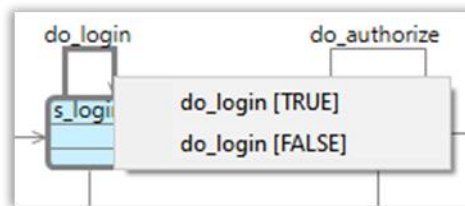


Рисунок 4.26 – Выбор параметра

Убедитесь, что в окне состояния модели переменная `has_access` изменила свое значение на `TRUE` (рисунок 4.27). Не забудьте выйти из режима анимации.

Name	Value	Previous value
m0		
has_access	TRUE	FALSE

Рисунок 4.27 – Окно состояния модели

Следующий шаг – установка инварианта на состояние `s_authorize`. Инвариант для `s_authorize` – это требование, согласно которому пользователь должен быть идентифицирован и аутентифицирован прежде, чем авторизован.

Выделите состояние `s_authorize` на диаграмме, в меню свойств щелкните на `Invariants` и нажмите кнопку `Add Invariant`. Введите понятное имя, например `has_access_inv`, и предикат `has_access=TRUE`. Нажмите `ctrl+s`, чтобы сохранить диаграмму. Транспируйте ее в `Event-B` и запустите анимацию.

Чтобы не щелкать по состояниям самому и не искать нарушения в модели вручную, можно поручить `Rodin` провести проверку модели. Для этого щелкните по кнопке `Checks` в окне событий, убедитесь, что выставлен флажок `Find Invariant Violations`, и нажмите `Start Model Checking`. Практически моментально вы получите уведомление о нарушении инварианта и чекер остановится на состоянии `s_authorize`. Взглянем на панель состояния модели и

увидим, что нарушен только что добавленный инвариант, поскольку переход в `s_authorize` был совершен со значением переменных `has_access=FALSE` (рисунок 4.28).

Formulas			
>	sets		
▼	★ invariants	↓	T
>	TRUE ∈ {s_idle,s_login,s_authorize,s_checkip,:	T	T
▼	★ s_authorize = TRUE => has_access = TRUE	↓	T
>	★ s_authorize = TRUE	T	↓
>	has_access = TRUE	↓	↓
>	login_counter ∈ N	T	T
>	event guards		

Рисунок 4.28 – Нарушение инварианта `s_authorize =TRUE=>has_access=TRUE`

Чтобы нарушения инварианта не происходило, необходимо добавить сторожевое выражение на переход `to_authorize`. Для этого выделите переход, в окне свойств щелкните по `Guards` и нажмите кнопку `Add Guard`. Введите понятное имя, например `has_access_grd`, и предикат `has_access=TRUE`. Сохраните диаграмму, транслируйте ее, запустите анимацию и проверку модели. В этот раз проверка модели должна завершиться успешно.

### Обратите внимание!

Вы можете наблюдать ошибку `Deadlock found` в случае, если включена проверка `Find Deadlocks` (состояний, из которых невозможно выйти). Это нормально, потому что ProB считает финальное состояние «дедлоком».

Следующий шаг построения модели – добавление трех попыток входа в систему. Для этого необходимо:

- установить декремент для `login_counter` – для этого создать Action для `do_login` с именем `decr_login_counter` и действием `login_counter:=login_counter-1;`
- создать сторожевое выражение для `do_login` с именем `do_login_grd1` и предикатом `has_access=FALSE∧login_counter>0`, которое позволяет событию запускаться в случае, если доступ еще не получен, и при этом счетчик попыток входа больше 0.

Попробуйте самостоятельно добавить сторожевое выражение на первую часть перехода от `s_login` к `s_fault`, чтобы сделать переход в `s_fault` доступным только для случаев, когда все попытки исчерпаны, а доступ так и не был получен.

### Решение

Имя – `to_fault_grd1`. Предикат – `has_access=FALSE∧login_counter=0`.

Запустите анимацию и убедитесь, что переход возможен только в случае трех неудачных попыток входа подряд.

Попробуйте добавить параметры для события `do_authorize` самостоятельно или обратитесь к решению ниже.

### Решение

Выделите `do_authorize` на диаграмме, в окне свойств щелкните по `Parameters` и нажмите `Add Parameter`. Задайте имя параметру `p_role` и тип `roles\{role_NULL}` – это позволит присваивать переменной только значения `user` или `admin`. Перейдите в `Actions` и добавьте действие с именем `set_role` и действием `role:=p_role`.

Запустите анимацию и попробуйте присвоить параметры переменной `role`. Попробуйте самостоятельно добавить следующие сторожевые выражения на:

- переход от `s_authorize` к финальному состоянию (только для пользователей);
- переход `to_checkip` (только для администраторов);
- переход `do_authorize` (чтобы его можно было запустить только один раз).

### Решение

- 1) Имя – `to_finish_grd1`. Предикат – `role=user`.
- 2) Имя – `to_checkip_grd1`. Предикат – `role=admin`.
- 3) Имя – `do_authorize_grd1`. Предикат – `role≠role_NULL`.

Аналогично выставьте параметры, инварианты и сторожевые выражения для оставшихся состояний и переходов (рисунок 4.29).

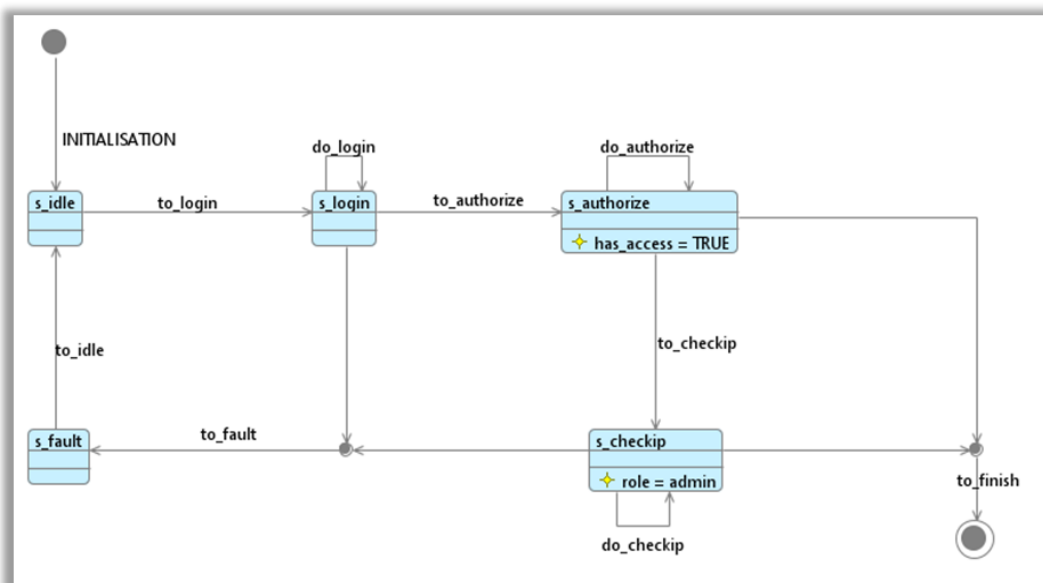


Рисунок 4.29 – Готовая диаграмма

## 5. ПРАКТИЧЕСКАЯ РАБОТА

Практическая работа нацелена на закрепление знаний по написанию документа Модель безопасности и описанию политик фильтрации информационных поток абстрактного ПО на языке Event-B.

### 5.1. Порядок выполнения работы

1. Установить средство разработки Rodin с официального сайта<sup>11</sup>.
2. Установить следующие плагины с помощью функционала `install new software`:
  - `iUMLB State-Machine`;
  - `iUMLB State-Machine Animation`;
  - `ProbB for Rodin3x`.
3. Описать на неформальном языке техническое задание по реализации политики фильтрации информационных потоков средства:
  - вариант 1 – межсетевого экранирования, выполняющего фильтрацию информационных потоков на основе правил;
  - вариант 2 – маршрутизации, выполняющего преобразование сетевых адресов (NAT).
4. Выделить необходимые переменные, множества и константы для реализации модели.
5. Составить графическую модель с использованием конечных автоматов.
6. Описать состояния и переходы в модели.
7. Продемонстрировать работу в режиме анимации, сопоставляя каждое действие с техническим заданием.
8. Подготовить отчет по практической работе с титульным листом, оформленные согласно [8].

Максимальная оценка за практическую работу – 30 баллов:

10 баллов – максимальная оценка за корректное оформление отчета;

10 баллов – максимальная оценка за полноту, точность и достоверность содержания отчета;

10 баллов – максимальная оценка за ответы на вопросы, касающиеся содержания учебно-методического пособия и выполнения практической работы.

### 5.2. Дополнительное задание

Подготовить презентацию на 5-10 слайдов, демонстрирующую работу подсистемы фильтрации информационных потоков выбранного средства.

Презентация должна включать:

---

<sup>11</sup> <http://www.event-b.org/install.html>

1. Титульный лист.
2. Указание средства, определенного вариантом практической работы, с описанием составленного технического задания.
3. Слайды, отражающие ход практической работы (решение задачи составления формальной модели).
4. Слайды, демонстрирующие работу модели.
5. Заключительный слайд.

В качестве шаблона презентации использовать официальный шаблон Университета ИТМО<sup>12</sup>.

За выполнение задания 5.2 можно дополнительно получить 10 баллов, которые можно засчитать как при защите данной практической работы (в том случае, если она не защищена на максимальное количество баллов), так и за другие работы, выполняемые в рамках дисциплин «Технология сертификации средств защиты информации» и «Сертификация продукции в различных системах сертификации (Минобороны, ФСТЭК, ФСБ)».

---

<sup>12</sup> <https://student.itmo.ru/files/1166> (актуальность ссылки проверена в ноябре 2021 года)



## 6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое сертификация?
2. Назовите основные руководящие документы, относящиеся к сертификации средств защиты информации.
3. Какие документы необходимы разработчику ПО для успешного прохождения сертификации? Где можно найти требования к документации?
4. Для чего необходим документ «Модель безопасности»? В каких случаях разработчик может не подготавливать данный документ?
5. Дайте определение понятию «формальное описание».
6. Что представляет из себя «конечный автомат»?
7. Что такое Event-B, модель на Event-B и ее содержимое?
8. Какие плагины использовались при выполнении практической работы и для чего они предназначены?

## **ЗАКЛЮЧЕНИЕ**

Данное учебно-методическое пособие призвано помочь студентам, изучающим в рамках своей образовательной программы дисциплины «Технология сертификации средств защиты информации» и «Сертификация продукции в различных системах сертификации (Минобороны, ФСТЭК, ФСБ)», эффективнее освоить изучаемый материал – перечень и состав документации разработчика, необходимой для успешного прохождения сертификации средства в системе сертификации ФСТЭК России.

Практическая работа, включенная в состав данного учебно-методического пособия, призвана помочь понять, как на практике формально описывать подсистемы программного обеспечения для разработки одного из обязательных документов – Модели безопасности.

## СПИСОК ЛИТЕРАТУРЫ

1. О техническом регулировании. - Текст: электронный // федеральный закон Российской Федерации от 27.12.2002 № 184-ФЗ, ред. От 02.07.2021. - URL: [http://www.consultant.ru/document/cons\\_doc\\_LAW\\_40241](http://www.consultant.ru/document/cons_doc_LAW_40241) (дата обращения: 13.09.2021).
2. Моделирование и верификация политик безопасности управления доступом в операционных системах / П. Н. Девянин, Д.В. Ефремов, В.В. Кулямин и др. – М.: Горячая линия – Телеком, 2019. – 214 с.: ил. ISBN 978-5-9912-0787-4.
3. Проект ГОСТ Р Защита информации. Формальное моделирование политики безопасности. Часть 1. Формальная модель управления доступом. - Текст: электронный. - URL: <http://docs.cntd.ru/document/560851933> (дата обращения: 13.09.2021).
4. Проект ГОСТ Р Защита информации. Формальное моделирование политики безопасности. Часть 2. Верификация формальной модели управления доступом. - Текст: электронный. - URL: <http://docs.cntd.ru/document/560537651> (дата обращения: 13.09.2021).
5. Ken Robinson. System Modelling & Design Using Event-B. - Текст: электронный. - URL: <https://wiki.event-b.org/images/SM&D-KAR.pdf> (дата обращения: 13.09.2021).
6. Michael Butler. RodinUser's Handbook. - Текст: электронный // edited by Michael Jastram. - URL: <https://www3.hhu.de/stups/handbook/rodin/current/pdf/rodin-doc.pdf> (дата обращения: 13.09.2021).
7. Безопасность операционной системы специального назначения Astra Linux Special Edition. Учебное пособие для вузов / П.В. Буренин, П.Н. Девянин, Е.В. Лебедеико и др.; под редакцией доктора техн. наук П.Н. Девянина. – 2-е издание, стереотипное. – М.: Горячая линия – Телеком, 2018. – 312 с.: ил. ISBN 978-5-9912-0623-5.
8. ГОСТ 7.32-2017 Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления. - Текст: электронный // Электронный фонд нормативно-технической и нормативно-правовой информации Консорциума «Кодекс»: [официальный сайт]. - URL: <https://docs.cntd.ru/document/1200157208> (дата обращения: 22.11.2021).

Тираж  
Отпечатано на ризографе

Кашин Семен Владимирович  
Бегаев Алексей Николаевич  
Маркевич Никита Алексеевич  
Павлов Денис Дмитриевич

**Модель безопасности средства, или Как формально описать  
подсистемы программного обеспечения**

**Учебно-методическое пособие**

В авторской редакции  
Редакционно-издательский отдел Университета ИТМО  
Зав. РИО Н.Ф. Гусарова  
Подписано к печати  
Заказ №

**Редакционно-издательский отдел**  
**Университета ИТМО**  
197101, Санкт-Петербург, Кронверкский пр., 49, литер А