Федеральное государственное бюджетное образовательное учреждение высшего образования
«Саратовский государственный технический университет имени Гагарина Ю.А.»

Институт прикладных информационных технологий и коммуникаций

Кафедра информационно-коммуникационных систем и программной инженерии

Направление 09.03.04 Программная инженерия

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

ВЕРИФИКАЦИЯ АВТОМАТНЫХ ПРОГРАММ НА ЯЗЫКЕ PROMELA

Выполнила студентка
Болдырева Юлия Юрьевна
курс 4
группа бПИНЖ41

Руководитель работы:
к.ф.-м.н., доцент кафедры ИКСП
Хворостухина Екатерина Владимировна

Допущен к защите
Протокол № _____ от _____ г.

Зав. кафедрой ИКСП д.т.н., профессор _____А.А. Сытник

Саратов 2019

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На настоящий момент, функционирование современных систем опирается на сложные модели, зависящие от таких параметров, как распределённость и параллельность. Для систем реального времени наиболее важными являются такие характеристики, как управление временем и структура (модель) самой системы.

Традиционный способ тестирования систем ориентирован на разработку тестовых наборов, которые представляют собой входные воздействия и ожидаемые реакции. Такая проверка систем, особенно для распределённых систем реального времени, становится всё более ненадежной из-за большой сложности воспроизведения ситуаций, которые приводят именно к конкретным ожидаемым реакциям. В связи с этим, для обеспечения полноты проверки современных распределённых систем, используется формальная верификация. Данный способ проверки систем позволяет обеспечить гарантию того, что системы будут соблюдать свои ключевые свойства для всех путей исполнения заданной модели.

За последнее десятилетие такой метод формальной верификации, как поверка модели, преимущественно используется в проверке свойств не только современных распределённых систем, но и систем высокой надёжности, таких как медицинские системы или системы в космической сфере.

Поскольку зачастую проведение формальной верификации вручную является чересчур трудозатратным, то для этого используются многообразные утилиты, которые применяют различные способы и подходы для проверки моделей систем.

Для проведения формальной верификации (и, соответственно, выполнения проверки моделей) важно не только правильно построить модель тестируемой системы, но и корректно описать требования к модели, так как это влияет на правильность и адекватность поведения системы, а, соответственно, на корректность работы программ, разработанных на основе её модели.

Поэтому целью данной работы является разработка тестового примера для верификации автоматных программ на основе метода проверки модели. Для достижения данной цели были поставлены следующие основные задачи:

- изучить теоретические сведения об автоматном подходе к программированию;
- изучить принципы моделирования систем;
- изучить теоретические сведения о формальной верификации;
- изучить основы темпоральной логики;
- выбрать средства для верификации системы (метод подхода к проверке модели, утилиты для верификации и графического интерфейса - для работы с верификатором);
- разработать модель системы «Кофейный автомат»;
- сформулировать требования к системе «Кофейный автомат»;
- реализовать систему и требования к ней на языке верификатора;
- провести верификацию модели системы «Кофейный автомат».

Данная выпускная квалификационная работа состоит из введения, четырёх глав, заключения и трёх приложений.

Во введении отражена актуальность выпускной квалификационной работы, её цели и задачи. В первом разделе приводятся теоретические сведения о методах и технологиях, использованных в данной работе. Во втором разделе производится анализ и сравнение аналогичных продуктов, производится выбор необходимых продуктов и технологий. В третьем разделе описано моделирование системы «Кофейный автомат» и составление требований к ней для последующей верификации. В четвёртом разделе описывается процесс верификации системы. Заключение включает основные выводы по работе и планируемые перспективы развития. В приложения вынесены листинг программы, фрагмент результата верификации и методические рекомендации.

1 Необходимые теоретические сведения

1.1 Конечный автомат и граф переходов

Для введения понятия «конечный автомат» рассмотрим понятие «абстрактный автомат».

Абстрактный автомат - это модель вычислений, основанная на гипотетической машине состояний, т.е. в один момент времени только одно состояние может быть активным [1]. Таким образом, для выполнения каких-либо действий машина должна менять свое состояние.

Формально, абстрактный автомат – это математическая модель дискретной системы, которая представляет собой упорядоченную шестерку:

$A = \{X, Y, S, \delta, \lambda, s_0\}$, где:

- $X = \{x_1, \dots, x_m\}$ – множество входных сигналов (входной алфавит) (представляет собой множество воздействий, влияющих на поведение автомата);
- $Y = \{y_1, \dots, y_p\}$ – множество выходных сигналов (выходной алфавит) (представляет собой множество реакций системы на внешние воздействия; характеризуют поведение системы);
- $S = \{s_1, \dots, s_n\}$ – множество состояний автомата (множество внутренних состояний);
- $\delta: X \times S \to S$ – функция переходов автомата из одного состояния в другое (определяет состояние $\delta(x,s)$, в которое переходит автомат в зависимости от текущего состояния x и поступившего входного сигнала s);
- $\lambda: X \times S \to Y$ - функция выходов определяет выходной сигнал системы $\lambda(x,s)$ в зависимости от входного сигнала s и текущего состояния x;

- $s_0 \in S$ – начальное состояние автомата (состояние, в котором система начинает функционировать).

Абстрактный автомат называется конечным, если множества X, S, Y – конечны.

Существуют следующие способы наглядного представления конечного автомата.

- Граф переходов (диаграмма состояний) - графическое представление конечного автомата. Граф переходов представляет собой ориентированный граф, вершины которого являются состояниями, а ребра - переходы между ними. Рёбра такого ориентированного графа помечаются соответствующими входными и выходными сигналами (метки дуг).
- Таблица переходов - табличное представление конечного автомата. В такой таблице каждой строке соответствует одно состояние, а столбцу - один допустимый входной сигнал. В ячейке на пересечении строки и столбца записывается состояние, в которое должен перейти автомат, если в данном состоянии он получил данный входной сигнал.

1.2 Применение автоматного подхода к программированию

Автоматное программирование - это один из стилей программирования, при котором программа представляется как модель конечного автомата. Граф или таблица переходов такого автомата преобразуется в код известными методами [2]. В качестве примеров использования автоматного программирования можно отметить не только построение компиляторов и реализацию протоколов, но и решение задач логического управления.

Автоматное программирование определено следующими особенностями [3].

- Время выполнения программы разбивается на шаги автомата; каждый из шагов автомата в такой программе - это выполнение определённой части кода с единственной точкой входа (состояния автомата).

- Передача информации между шагами автомата осуществляется только через явно обозначенное множество переменных (сигналы автомата).
- Между шагами автомата программа не может содержать неявных элементов состояния (таких как: значения локальных переменных, значение счётчика, адреса возврата из функций и пр.)

Основной задачей автоматного подхода к программированию является проектирование системы – так как уже на данном этапе обеспечивается качество написания автоматной программы, и кодирование системы начинается только после её проектирования

Автоматный подход к программированию включает в себя множество преимуществ, одним из которых следует отметить то, что продуманное применение конечных автоматов облегчает организацию и написание программ – причём, как логики самой системы, так и логики пользовательского интерфейса. Таким образом, представление алгоритма программы в виде конечного автомата позволяет отделить логику программы от ее реализации, что приводит к тому, что автоматные программы требуют минимальной отладки и сокращению времени их кодирования.

Как было сказано ранее, примером области применения автоматного подхода к программированию могут служить анализаторы – как синтаксические, так и лексические, а также - тестирование программного обеспечения на основе моделей. Причём, для проверки автоматных программ могут генерироваться, так называемые, отладочные протоколы, которые отображают поведение программ в терминах автоматов (состояний, переходов, значений входных и выходных сигналов).

Стоит отметить, что автоматные методы хорошо автоматизируются, обеспечивают достаточно полную проверку системы и позволяют находить сложные ошибки, но данные методы требуют квалификации специалиста, наличия точного и полного описания поведения системы.

1.3 Описание принципа моделирования системы

Поведение конечного автомата задаётся графами переходов (диаграммами состояний). На графах переходов входные и выходные воздействия для компактности обозначаются символами, помещёнными над стрелками соответствующих переходов или в кружках, обозначающих состояния (зависит от модели, в терминах которой описывается конечный автомат). Расшифровка данных символов обозначается на схеме связей. Задание поведения программ с помощью графов переходов позволяет проверять корректность их построения (например, полноту, непротиворечивость и пр.). Программа, построенная по графу переходов, может быть проверена сверкой её текста с данным графом.

Один из методов реализации графов переходов - с помощью конструкции switch (оператор ветвления) по соответствующим шаблонам («Switch-технология»).

Программы, создаваемые на основе switch-технологии, могут быть эффективно верифицированы методом model checking (проверка модели) [4], т.к. в таких программах управляющие состояния явно выделены, а их количество обозримо. Такой способ позволяет не только строить компактные модели даже для больших и сложных программ с множеством состояний, но и эффективно их верифицировать.

В конструкции switch, которая лежит в основе switch-технологии, используются символы входных и выходных воздействий. Функции же, которые реализуют данные символы, описываются отдельно. Такая декомпозиция позволяет отделить логику переключения от детализации происходящего, и, тем самым, упростить понимание логики программы за счет достижения компактности представления.

Для каждого автомата создаётся словесное описание алгоритма, граф переходов и реализация на языке программирования (в данном случае, на языке программирования, понятном верификатору).

Реализация выполняется таким образом, что в каждом цикле программы или при каждом запуске в кодированном на языке программирования автомате выполняется не больше чем один переход. Таким образом, номер каждого состояния конечного автомата становится доступен для его окружения, что позволяет не вводить новых переменных для того, чтобы обеспечить взаимодействие внутри автомата.

1.4Валидация систем и формальная верификация

Основной проблемой для систем (особенно систем, критичных в отношении безопасности) является то, что быстро возрастает их сложность и, как следствие, число возможных ошибок при их проектировании и реализации [5].

Стоимость исправления ошибок можно схематически изобразить (рис. 1), взяв за основу кривую Боэма [20].

Поэтому ввиду возрастания размеров и сложности систем, становится важным обеспечение процесса валидации (процесс проверки корректности спецификаций, дизайна и продукта) систем.

Рисунок 1 - Экспоненциальный рост исправления стоимости ошибки

Важнейшими методами валидации систем являются [6]:

- экспертный анализ - процедура получения оценки проблемы на основе мнения экспертов (специалистов в данной области) с целью последующего принятия решения;
- тестирование - процесс исследования программного продукта с целью проверки соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определенным образом;
- симуляция - имитация управления каким-либо процессом, аппаратом или транспортным средством с помощью различных устройств;
- формальная верификация - формальное доказательство соответствия или несоответствия формального предмета верификации его формальному описанию.

Однако, хоть и тестирование систем является достаточно эффективным методом для проверки того, что данная реализация согласуется с абстрактной спецификацией, но по определению, тестирование может быть использовано только после реализации прототипа системы. Формальная верификация же, в противовес тестированию, основана на математическом доказательстве корректности программ. В дополнение вышесказанного, тестирование не может гарантированно доказать, что система, алгоритм или программа не содержит ни единой ошибки или дефекта и удовлетворяет определённому свойству, в отличие от математических методов доказательств корректности систем.

Можно отметить, что в области тестирования возрастает интерес к разработке алгоритмов и программ для автоматической генерации тестов по формальной спецификации системы.

Основой формальной верификации являются программы для автоматического доказательства теорем и проверки доказательств, однако для их использования важно корректно построить модель верифицируемой системы.

1.5 Метод Model Checking

Так как формальная верификация представляет собой формальное доказательство на абстрактной математической модели системы, примерами часто используемых для неё математических объектов могут служить: формальная семантика языков программирования, конечные автоматы, сети Петри, временные автоматы, гибридные автоматы, исчисление процессов и пр.

Существуют несколько подходов к формальной верификации. Ниже приведены некоторые из них:

- формальная семантика языков программирования;
- написание программ, которые верны по построению;
- проверка моделей;
- логический вывод;
- символьное выполнение;
- абстрактная интерпретация.

Верификация взаимодействия в параллельных системах, несмотря на значительную вычислительную сложность, может быть выполнена на основе относительно новой на сегодняшний день техники верификации model checking (проверка модели). Именно эту технику использует верификатор SPIN, рассмотренный во второй главе данной работы.

Главный смысл метода model checking [7] состоит в запуске алгоритмов проверки корректности систем, которые выполняет компьютер. При использовании данного подхода пользователь вводит описание модели системы (возможное поведение, описанное при помощи конечного автомата) и описание спецификации требований (желаемое поведение), а верификацию проводит специальное инструментальное средство - верификатор. Если ошибка найдена, предоставляется контрпример, который показывает при каких обстоятельствах может быть обнаружена ошибка. Контрпример представляет собой сценарий, в

котором модель ведет себя нежелательным образом - т.е. свидетельствует о том, что модель неверна и должна быть исправлена.

Таким образом, проверка модели заключается в специфицировании свойств системы в виде логических формул [8]. Для определения свойств в виде формул используется язык темпоральной логики - специальный язык, который позволяет описывать поведение системы во времени.

У model-checking существует и недостаток, называемый «взрывом количества состояний». При моделировании времени как непрерывной сущности даже самая простая модель имеет бесконечное число состояний. Наиболее известным подходом к решению этой проблемы является упрощение проблемы с помощью факторизации множеств (формирования классов эквивалентности).

Основные ограничения метода проверки моделей состоят в следующем:

- подход применим в основном не к приложениям обработки данных, а к приложениям, в которых компоненты взаимодействуют между собой;
- при использовании проверки моделей верифицируется только модель системы, а не сама система: тот факт, что модель обладает определенными свойствами, не гарантирует, что финальная реализация будет обладать теми же свойствами;
- построение модели системы и описание требований к ней на языке темпоральной логики требуют соответствующей квалификации;

Тем не менее, методом model checking могут быть эффективно верифицированы программы, созданные на основе switch-технологии (способ программирования, при котором ветвления в программе создаются с помощью оператора switch), т.к. в таких программах управляющие состояния явно выделены, а их количество обозримо. Это позволяет строить компактные модели даже для больших программ.

1.6 Темпоральная логика

Темпоральная логика, или логика линейного времени, LTL (англ. «Linear Time Logic») - это логика, в высказываниях которой учитывается временной аспект. Её можно определить множеством базовых формул [9].

Множество формул, построенных в соответствии с этими правилами, называется формулами LTL (формулами логики линейного времени).

Синтаксис логики линейного времени включает в себя множество пропозициональных переменных, булевы связки (¬, ∨, ∧) и темпоральные операторы [10]:

- X (neXt) – «Xp» – в следующий момент будет выполнено p;
- U (Until) – «pUq» – существует состояние, в котором выполнено q и до него во всех предыдущих выполняется p.

Формулы в логике LTL определяются по индукции.

Пусть AP – множество атомарных предложений (высказываний), тогда

- p является формулой для всех $p \in AP$;
- если φ – формула, то ¬φ – формула;
- если φ и ψ – формулы, то φ ∨ ψ, φ ∧ ψ – формулы;
- если φ – формула, то X φ – формула;
- если φ и ψ – формулы, то φ U ψ – формула.

Логика линейного времени расширяет классическую логику, добавляя временные операторы, для того чтобы можно было судить об истинности высказываний в разные моменты времени. Пропозициональные переменные в каждый момент времени могут быть истинными или ложными. Для составления утверждений о времени событий применяются следующие дополнительные темпоральные операторы [10]:

- F (in the Future) – «Fp» – в некоторый момент в будущем будет выполнено p (альтернативное обозначение - ◊);
- G (Globally in the future) – «Gp» – всегда в будущем выполняется p (альтернативное обозначение - □);

- R (Release) – «pRq» – либо во всех состояниях выполняется q, либо существует состояние, в котором выполняется p, а во всех предыдущих выполнено q;
- W (Weak until) - более слабый вариант оператора Until, - unless. «φWψ» - φ выполняется непрерывно либо до момента, когда впервые будет выполняться ψ, либо в течение всей последовательности; определяется следующим образом (1):

$$\varphi W \psi = G\varphi \vee (\varphi U \psi) \quad (1)$$

Описанные выше темпоральные операторы G и F определяются следующим образом (2, 3):

$$\mathrm{F}\,\varphi = \mathrm{true}\,\mathrm{U}\,\varphi \quad (2)$$

$$G\,\varphi = \neg F \neg\varphi \quad (3)$$

Оператор U можно определить через F и W следующим образом (4):

$$\varphi \mathrm{U} \psi = \mathrm{F}\psi \wedge (\varphi \mathrm{W} \psi) \quad (4)$$

Логика LTL подразумевает, что некоторое утверждение будет выполняться для всех путей (т.е. в ней есть квантор ∀, но отсутствует квантор ∃). По этой причине можно строить доказательство от противного и проверять наличие пути, на котором будет выполняться отрицание данной формулы. В случае, если такой путь не будет обнаружен, то формула выполнима.

2 Выбор средств для верификации системы

2.1 Выбор метода подхода к проверке модели

Как было сказано ранее, формальная верификация системы - это проверка соответствия между требованиями к системе и свойствами работающей системы; в целом, при формальной верификации, существует большое количество подходов к анализу моделей.

Из описанных в первой главе данной работы способов проверки системы, было принято решение выбрать именно формальную верификацию, которая основана на математическом подходе к проверке систем. Формальные модели представляют особый интерес при верификации, поскольку они поддаются алгоритмизации, и, следовательно, процесс верификации можно автоматизировать с помощью компьютера.

Model checking - это автоматизированный метод, который для заданной модели поведения системы (с конечным числом состояний) и логического свойства (выраженного через язык логики), проверяет справедливость этого свойства в данной модели. Стоит отметить, что проверка модели поддерживает частичную верификацию: т.е. для системы может быть проверена частичная спецификация при рассмотрении только некоторого подмножества всех требований.

Таким образом, можно выделить следующие достоинства при использовании метода model-cheking:

- это общий подход с приложениями к верификации разнообразного класса систем;
- подход поддерживает частичную верификацию: проект может быть верифицирован по частичной спецификации, что обеспечивает высокую эффективность подхода, так как можно ограничиться проверкой наиболее важных свойств;

- встраивание проверки моделей в процесс проектирования приводит к уменьшению времени разработки, или, по крайней мере, использует такое же время, как проведение тестирования или симуляции;
- благодаря автоматизации, проверка моделей не требует высокой степени взаимодействия с пользователем;
- математическая точность при проверки моделей;
- надежность проверки программ - основу проверки моделей составляют стандартные и хорошо известные алгоритмы.

2.2 Выбор утилиты для верификации модели. Верификатор SPIN и язык Promela.

При анализе существующих программных продуктов (таблица 1) был выбран инструмент SPIN – утилита для автоматизированной верификации корректности программных моделей [11]. Основными преимуществами SPIN перед аналогичными инструментами является работа не только в режиме model-checking, но и в режиме симулятора, исполняя один из возможных путей работы системы и предоставляя программисту результаты этого исполнения.

Таблица 1 – Сравнительный анализ верификаторов

Верификатор	Режим	Язык	Основные понятия	Способ верификация	Графическая оболочка	ОС
SPIN	- Симуляция - Верификация - Система обоснованной аппроксимации	Promela	- Процессы - Буферизованные каналы	Проверка LTL-формул (верификация свойств состояний, недостижимого кода и т. д.) - проверяет, есть ли вычисление, при котором утверждение нарушается. Его исполнение не имеет побочных эффектов	Xspin и др.	Все версии Unix, Linux, Plan9, Inferno, Solaris, и Windows
SMV	Верификация	SMV	- Процессы	Проверка CTL-формул - использует BDD (двоичные разрешающие диаграммы) для представления и хранения множества состояний компактным образом с целью расширения возможностей проверки моделей. Отсутствие « голодания» - т.е. не может быть ситуации, когда процесс, который хочет войти в критическую секцию он в нее когда-нибудь попадет	NuSMV	Windows и Unix и им подобные
HyTech	Верификация	Hybrid automata	- Гибридный автомат (конечный автомат с конечным числом вещественных переменных, которые изменяются непрерывно, но не с постоянной скоростью, а со скоростью,	Проверка CTL-формул - графическую спецификацию и генерацию контрпримеров	HyTech	Windows и Unix и им подобные.

Продолжение таблицы 1

			выражающейся через дифференциальные уравнения и дифференциальные неравенства)			
PRISM	Верификация вероятностных систем	PRISMlanguage	- Процессы	Проверка CSL- и PCTL –формул. Поддерживает три модели – Марковские цепи дискретного времени (DTMCs), Марковские цепи непрерывного времени (CTMCs) и Марковские процессы регулирования (MDPs)	PRISM	Unix и подобные.
CBMC	Верификация (ограниченная проверка моделей)	ANSI-C C++.	- Функции	Ограниченная проверка моделей - позволяет верифицировать выход за границу массива (переполнение буфера), безопасность указателей, исключения и пользовательские утверждения	Плагин для Eclipce	Windows и Unix и им подобные

В отличие от многих программ для проверки моделей, SPIN не выполняет работу сам, а генерирует программу на языке C, которая решает конкретную задачу. За счет этого достигается экономия памяти и повышение производительности, и становится возможным использовать фрагменты кода на языке C непосредственно из модели. SPIN предоставляет множество опций для ускорения проверки моделей (сжатие состояний, хеширование битовых состояний и др.).

С целью уменьшения времени поиска в глубину при наличии контрпримеров, SPIN создаёт множества состояний и переходов «на лету» («on-the-fly») только для части построенной системы переходов; данная часть системы постепенно достраивается в ограниченной области памяти.

SPIN представляет собой мощный инструмент для верификации программ. Данный инструмент использовался во многих серьёзных проектах при проведении проверки моделей систем. Так, вскоре после создания верификатора SPIN, в Нидерландах была успешно проведена проверка алгоритмов управления противопаводковым шлюзом.

Ещё одним примером может служить проверка с помощью SPIN программного обеспечения в космической сфере. Некоторые алгоритмы для ряда космических миссий (миссия «Deep Space 1», миссия «Cassini», исследовательские марсоходы и др.) были проверены с помощью метода model checking в SPIN. Так, для миссии «Cassini» проверялась правильность работы алгоритмов передачи обслуживания для процессоров двойного управления, а для марсоходов была проверена правильность работы арбитража ресурсов, - процесса, который управляет использованием всех двигателей на автоматизированном исследовательском аппарате. Для космического аппарата «Deep Impact» с помощью данного верификатора были проверены аспекты модуля файловой системы флэш-памяти - это позволило выявить проблемы до встречи с кометой Темпеля 1.

Проверка модели в SPIN широко используется и в медицине. Так, SPIN использовался в течение десяти лет для проверки международных стандартов.

Эта работа началась с проекта «IQPS» (англ. «Improving the Quality of Protocol Standards» - улучшение качества стандартов протоколов). Стандарты, на которые повлияла работа по проверке с помощью SPIN, включают в себя IEEE 1394 и ISO / IEEE 11073-20601 для медицинских приборов.

Пакет SPIN позволяет:

- строить модели параллельных программ и широкого класса дискретных систем;
- выразить требуемые свойства поведения (т.н. «темпоральные свойства»);
- автоматически проверить выполнение темпоральных свойств параллельных систем на их моделях на основе формального подхода.

Для верификации проектирования с помощью SPIN необходимо построить формальную модель с использованием языка Promela.

Язык Promela (Process Meta Language - язык метапроцессов) [19] - абстрактный язык спецификации алгоритмов. С помощью минимальных выразительных средств (параллельные процессы и коммуникация с помощью каналов, простейшие типы данных и управляющие структуры) позволяет строить абстрактные модели реальных систем с конечным числом состояний. Таким образом, можно подвести итог, что Promela является не языком реализации систем, а языком описания моделей.

2.3 Выбор графического интерфейса для работы с верификатором SPIN.

Работа с верификатором SPIN может проходить как в консольном режиме, так и с помощью графического интерфейса. Последний способ работы с верификатором наиболее удобен.

Существует несколько графических интерфейсов для Spin, как то: iSpin, jSpin, xSpin, Tau. Рассмотрим их подробнее.

2.3.1 GUI Tau

Tau – единственный графический интерфейс для Spin, обеспечивающий возможность непосредственного создания конечных автоматов. Приложение распространяется по LGPL лицензии (лицензия свободного программного обеспечения) как учебный инструмент для формальной верификации и конечных автоматов. Данный интерфейс написан на устаревшей версии Tcl/Tk, из чего следует использование не интуитивно понятных графических компонентов (рис. 2).

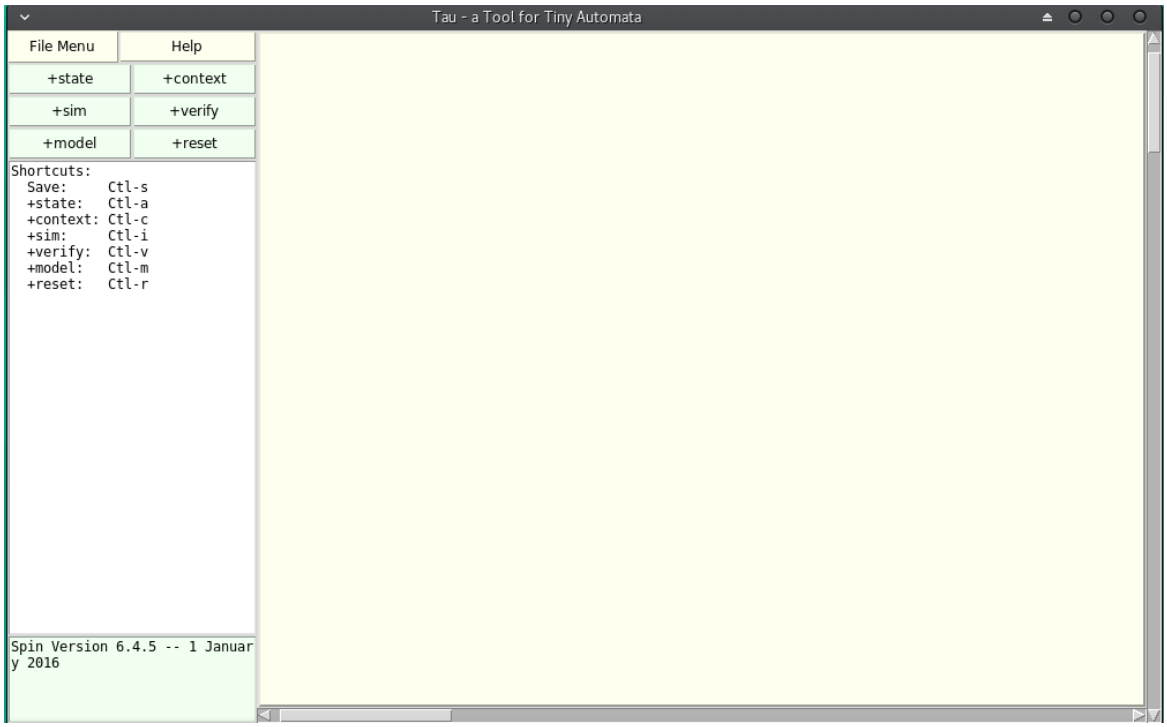

Рисунок 2 - Главное окно графического интерфейса Tau

Tau работает по следующему принципу: пользователь конструирует конечный автомат, он переводится в Promela-описание, которое впоследствии подвергается симуляции или верификации - в зависимости от выбора режима симуляции или верификации, Promela-описание подается на вход верификатора SPIN.

Несмотря на преимущества использования Tau, заключающееся, как было отмечено ранее, в конструировании графа переходов непосредственно в графическом интерфейсе, присутствуют и существенные недостатки:

- в процесс нельзя передавать параметры, отсутствует понятие контекста процесса для определения локальных переменных;
- флаги, используемые при верификации и симуляции нельзя изменить или редактировать, т.е. нельзя задать семя для симуляции, методы поиска, методы хранения, параметры безопасности и др.;
- не поддерживается построение диаграммы состояний;
- не поддерживается работа со Swarm (установка параметров и запуска большого количества небольших задач верификации в параллельном режиме);
- используются не интуитивно понятные графические компоненты без группировки по функциональности;
- горячие клавиши конфликтуют при работе с текстом, отсутствуют некоторые стандартные горячие клавиши, общепринятые в графических интерфейсах (например, «Delete»).

Все перечисленные выше недостатки значительно усложняют работу с интерфейсом.

2.3.2 GUI iSpin

iSpin – комплексное средство для верификации и симуляции описаний на языке Promela. iSpin был создан разработчиками Spin и распространяется по той же лицензии. Данный интерфейс (рис. 3) написан на устаревшей версии Tcl/Tk. iSpin может быть установлен как на Unix-подобных системах, так и в Cygwin на Windows при наличии предустановленного стандартного компилятора gcc и самого верификатора SPIN.

iSpin работает в нескольких режимах: редактирование, симуляция, верификация и Swarm.

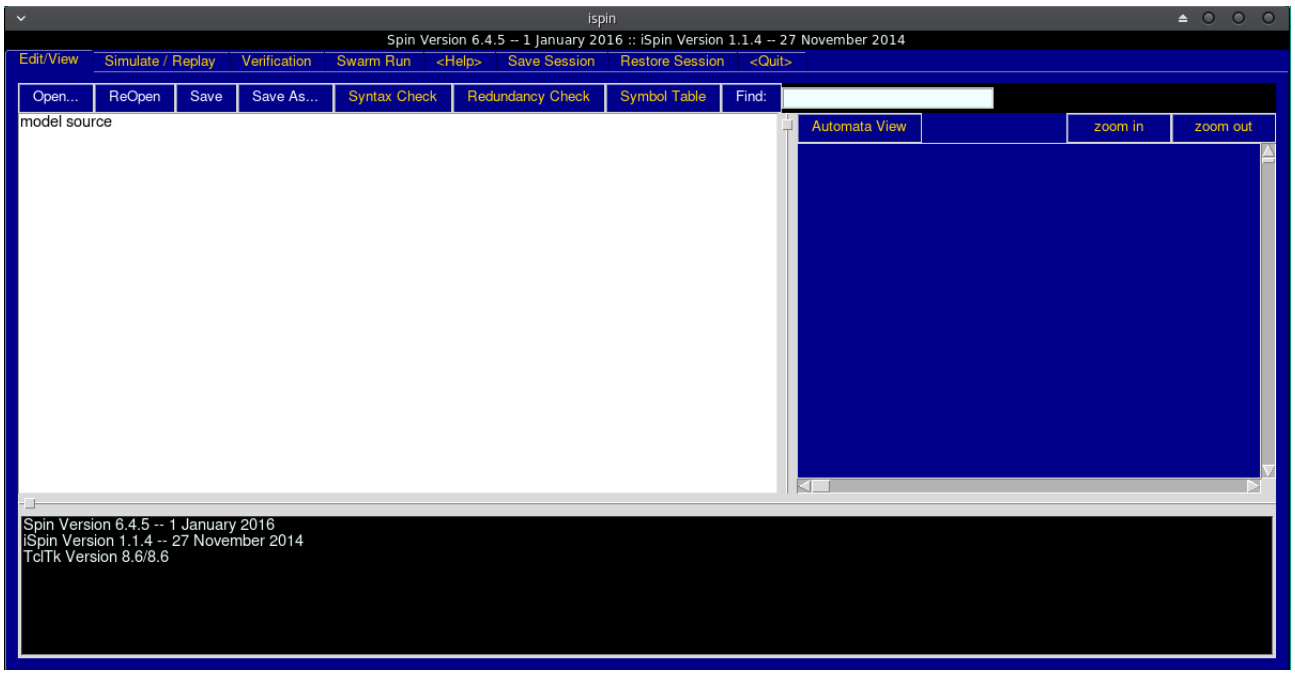

Рисунок 3 - Главное окно графического интерфейса iSpin

В режиме редактирования, помимо свободного редактирования программы на языке Promela, доступна её загрузка, сохранение, проверка синтаксиса, символьный поиск, проверка на избыточность кода, построение символьных таблиц для переменных и процессов, а также построение конечных автоматов по описанию на Promela.

iSpin предоставляет широкий набор инструментов для симуляции Promela-описаний. Помимо семени для случайной симуляции можно выбрать режим интерактивной симуляции (где каждый выбор предоставляется пользователю), а также заранее сгенерированный ход событий из *.trail файла. Также, имеется возможность отслеживать переменные на каждом шаге программы. Кроме того, можно задавать различные параметры для отображения симуляции: ширину текста, задержку на вывод, фильтрацию. Важным инструментом для симуляции является развертка диаграммы процессов (вывод на панели диаграммы взаимодействия MSC)

Для верификации в iSpin используется огромное число различных флагов. Важнейшими из них являются: флаги управления безопасностью, хранением и поиском. Также можно задать расширенные настройки, связанные с ограничениями памяти, оптимизациями, прерыванием на ошибках и пр.

Результаты верификации отображаются в терминале в нижней части экрана. Можно отметить, что iSpin имеет самые широкие функциональные возможности среди интерфейсов для Spin: проверки синтаксиса, использование огромного количества флагов, преобразование Promela-описания в конечные автоматы, развертка диаграммы процессов, интерактивная симуляция, Swarm. Из нефункциональных преимуществ интерфейса iSpin следует выделить удобную группировку компонентов по функциональным возможностям, реализованную с помощью вкладок.

Единственным существенным функциональным недостатком iSpin является отсутствие конструктора для конечных автоматов. Помимо этого, из-за того, что инструмент написан на устаревшей версии Tcl/Tk, в iSpin существует ряд недостатков, связанных с графическим интерфейсом: полное отсутствие контекстных меню и всплывающих подсказок, отсутствие поддержки горячих клавиш, текст из редактора копируется вместе с номерами линий; отсутствует подсветка синтаксиса языка Promela.

Тем не менее эти недостатки не сильно осложняют работу с интерфейсом, особенно учитывая обилие функциональных возможностей.

2.3.3 GUI jSpin

jSpin – интерфейс для верификатора Spin, написанный на Java. Для работы с jSpin требуется предустановленная Java SDK или JRE, а также сам верификатор Spin (рис. 4)

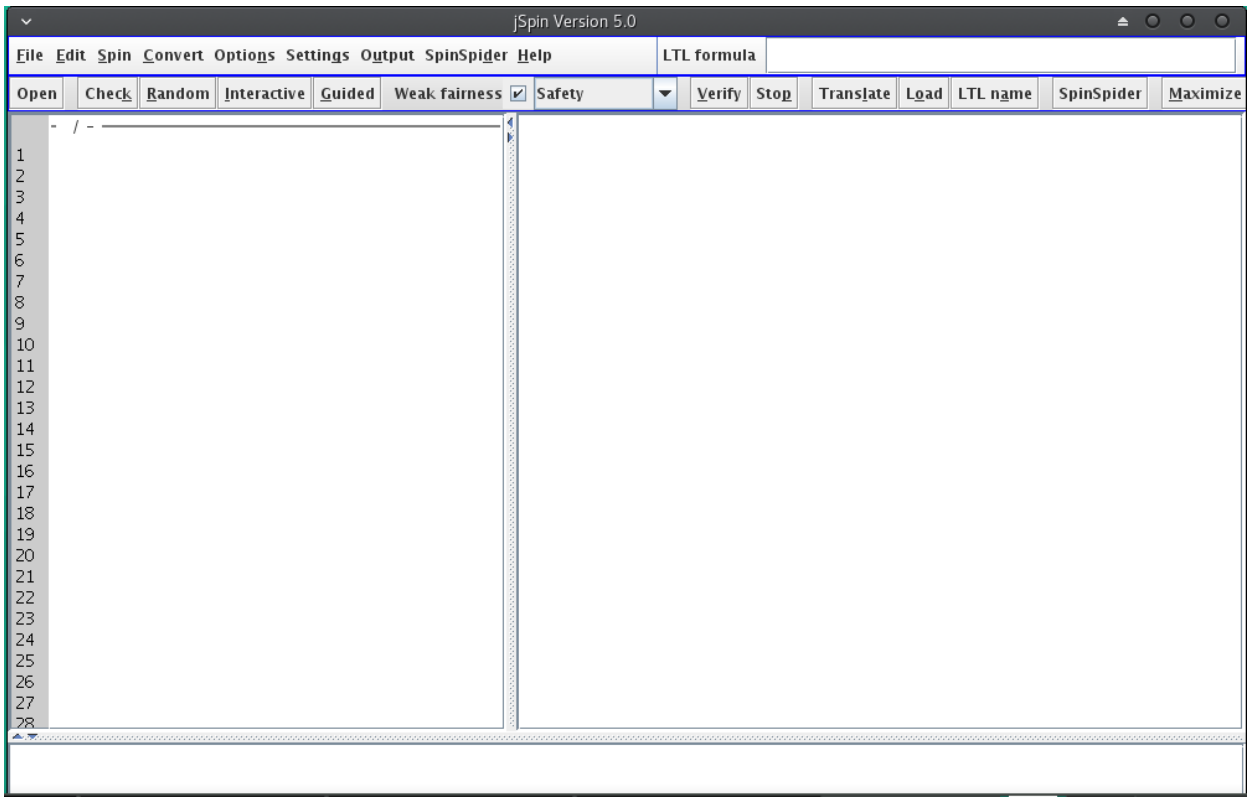

Рисунок 4 - Главное окно графического интерфейса jSpin

jSpin работает в нескольких режимах: симуляция, верификация и SpinSpider. Все настройки приложения определены в специальном конфигурационном файле (в том числе и флаги для симуляции и верификации). Поддерживается три различных метода симуляции Promela-описаний: случайный, интерактивный, предустановленный. Эти режимы симуляции аналогичны используемым в iSpin. SpinSpider позволяет преобразовать Promela-описание в изображение конечного автомата. SpinSpider поддерживает четыре типа форматов: *.png, *.dot, *.fsm, *.ps. Из достоинств данного интерфейса можно отметить, что доступ к некоторым функциям дублируется горячими клавишами и кнопками в панели инструментов.

В данном графическом интерфейсе существуют следующие недостатки:

- большинство флагов задается вручную;
- не поддерживается работа со Swarm;
- отсутствует конструктор для конечных автоматов;
- · полное отсутствие контекстных меню и всплывающих подсказок;
- отсутствует подсветка синтаксиса языка Promela.

2.3.4 GUI xSpin

xSpin – устаревший графический интерфейс для Spin (рис. 5), который в настоящее время не поддерживается разработчиками. Из уникальных особенностей данного интерфейса следует выделить редактор LTL формул, который позволяет интерактивно добавлять предикаты и операторы в формулу и Promela-описание одновременно, а также выводит результат верификации.

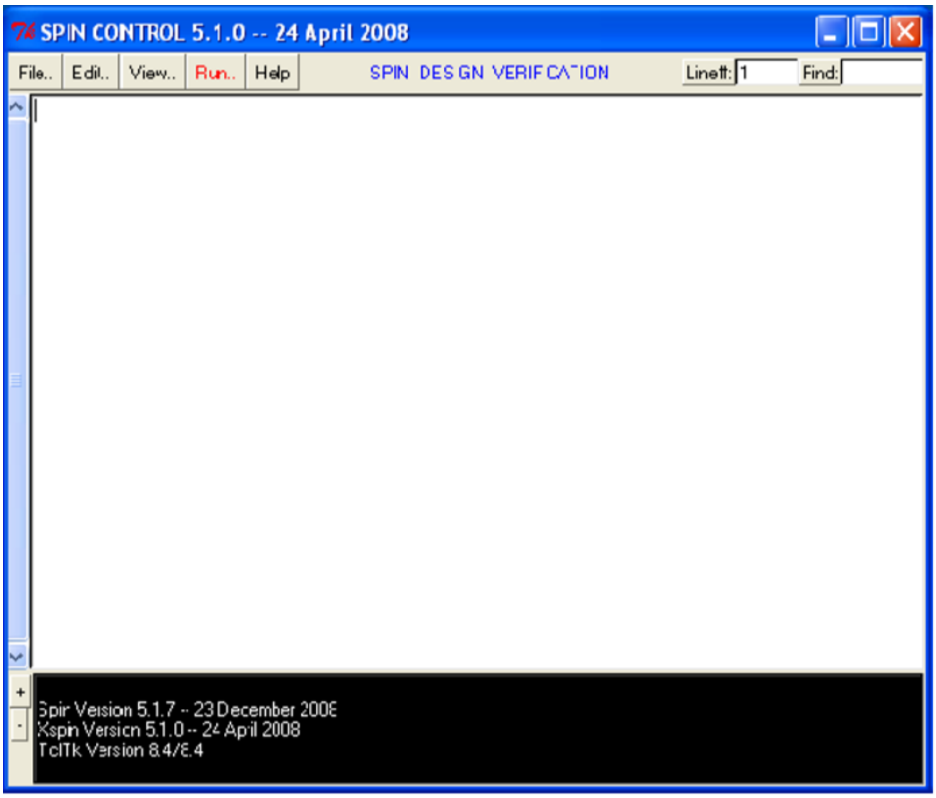

Рисунок 5 - Главное окно графического интерфейса xSpin

Ввиду вышеперичисленных особенностей каждой из утилит, был выбран графический интерфейс iSpin [12].

3 Система «Кофейный автомат»

3.1 Модель системы «Кофейный автомат»

В качестве системы для верификации была выбрана система «Кофейный автомат».

Кофейный автомат – это устройство, взаимодействующее с пользователем, которое должно выполнять следующие функции: прием денег, выбор напитка, регулирование количества сахара, выдача сдачи, приготовление и выдача напитка [13].

Начальное состояние данной системы предлагает ввести деньги для последующей покупки напитка. Затем можно отрегулировать количество сахара (некоторое количества сахара задано по умолчанию) или вернуть деньги (если принято решение отмены приготовления напитка). После этого следует выбрать напиток. После выбора напитка автомат выдаёт сдачу, если она есть, и начинает приготовление напитка. После того, как напиток готов, автомат выдаёт напиток, и возвращается в начальное состояние.

По данному описанию был построен граф переходов автомата (рис. 6) «Кофейный аппарат», спроектированный по модели автомата Мили.

Что касается данной модели, законы функционирования автомата Мили представлены следующим образом (5, 6)

$$s(t+1) = \delta(s(t), x(t)) \quad (5)$$

$$y(t) = \lambda(s(t), x(t)) \text{ , где:} \quad (6)$$

- t – текущий момент времени;
- t+1 – следующий момент времени;
- s(t+1) – состояние автомата в следующий момент времени;

- s(t), x(t), y(t) – элементы описания автомата в текущий момент времени (состояние автомата, входной сигнал, выходной сигнал, соответственно).

Таким образом, на рис. 6 изображен граф переходов автомата «Кофейный аппарат».

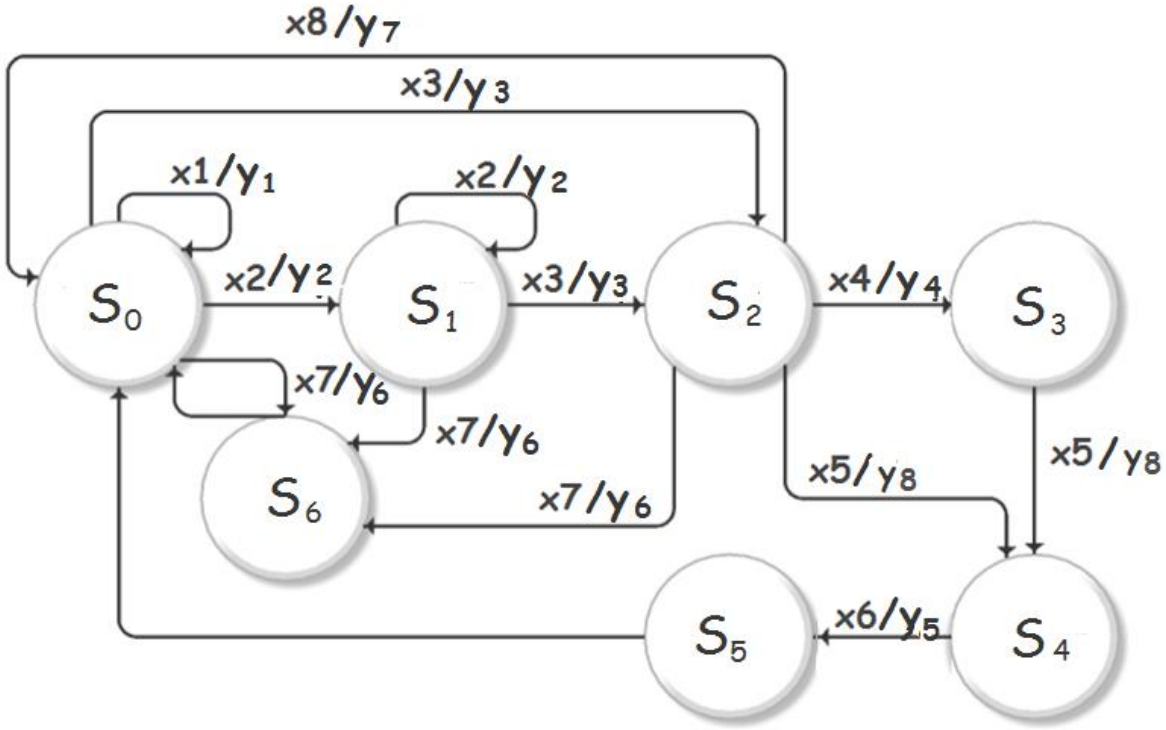

Рисунок 6 - Граф переходов автомата «Кофейный автомат»

Входные сигналы автомата:

- X1 – введена купюра;
- X2 – нажата кнопка «+» или «-» (регулировка сахара);
- X3 – нажата кнопка выбора напитка;
- X4 – выдать сдачу (сдача больше 0);
- X5 – сдача равна 0 (сдача выдана);
- X6 – напиток готов;
- X7 – нажата кнопка «Возврат денег»;
- X8 – денег на счету меньше, чем стоимость напитка

Выходные сигналы автомата:

- Y1 – вывести сообщение «Введите купюру»;
- Y2 – перейти к регулировке сахара
- Y3 – вывести название выбранного напитка;
- Y4 – вывести сообщение «Возьмите сдачу»; вернуть сдачу;
- Y5 – вывести сообщение «Возьмите напиток»; выдать напиток;
- Y6 – вернуть деньги;
- Y7 – вывести сообщение «Недостаточно денег»;
- Y8 – вывести сообщение: «Идет приготовление напитка»

Состояния автомата:

- S0 – начальный экран;
- S1 – регулировка количества сахара;
- S2 – выбор напитка;
- S3 – выдача сдачи;
- S4 – приготовление напитка;
- S5 – выдача напитка;
- S6 – выдача остатка денег

3.2 Требования к системе «Кофейный автомат». Спецификация LTL-формул

Рассмотрим множество атомарных предложений: AP = {$drink_i$, $sugar_i$, $cancel_i$, $cash_i$, $cost_i$, $s_i \in S$, $y_i \in Y$, $i < j \mid 1 \leq i, j \leq N$}, где $drink_i$ означает, что выбран i-ый напиток, $sugar_i$ - количество сахара в i-ом напитке, $cancel_i$ – состояние отмены приготовления i-ого напитка, $cash_i$ – количество денег, введённое покупателем для покупки i-ого напитка, cost – цена i-ого напитка, $s_i \in S$ – i-ое состояние автомата, $y_i \in Y$ – i-ый выходной сигнал автомата (информация, показываемая на экране), а $i < j$ – что идентификатор i меньше, чем идентификатор j (в линейном порядке идентификаторов); i и j используются в качестве идентификаторов процессов, N - число процессов.

Таким образом, были выделены следующие требования к модели «Кофейный автомат».

- «Начальный экран» (7). В момент времени t_0 (начало цикла работы кофейного автомата) показан начальный экран. Начальный экран показывается сколь угодно долго до того, как нажата кнопка, переводящая автомат в состояние приготовления напитка (регулировка сахара или выбор напитка)

$$s_0 U (s_1 \vee s_2) \tag{7}$$

- В любой момент времени на экране кофейного автомата показывается только одно (текущее) состояние (8)

$$G[\exists i: y_i \wedge (\forall j \neq i: \neg y_j)] \tag{8}$$

Примечание: строго говоря, кванторы не является частью LTL. Квантификация в данном случае рассматривается как сокращение

- «Регулировка количества сахара» (9). Количество сахара не может быть меньше 0 и больше максимальной нормы на порцию напитка.

$$G[0 \leq sugar \leq sugar_{max}] \tag{9}$$

- «Выбор напитка» (10). В один цикл работы автомата можно выбрать только один напиток

$$G[\exists i: drink_i \wedge (\forall j \neq i: \neg drink_j)] \tag{10}$$

- После выбора и оплаты напитка пользователь не может отменить покупку (11)

$$G[s_3 \rightarrow X(cancel \coloneqq false)] \tag{11}$$

- После выдачи остатка денег пользователю, автомат переходит в начальное состояние («Начальный экран») (12)

$$G[s_6 \rightarrow X(s_0)] \quad (12)$$

- Автомат не может приготовить выбранный напиток, если на счету нет достаточного количества денег для оплаты (13)

$$G[s_2 \wedge (cash < cost) \rightarrow X(s_0)] \quad (13)$$

Данные требования [14] и будут проверяться с помощью верификатора SPIN.

4 Реализация и верификация системы

4.1 Реализация системы и требований к ней на языке Promela

4.1.1 Синтаксис языка Promela

Программную реализацию автомата можно выполнить на любом языке программирования различными способами [15].

В данной работе алгоритм программы реализован в соответствии с топологией графа переходов конечного автомата. Этот способ является универсальным и состоит в следующем: с каждым состоянием связывается операция NEXT, выполняющая функцию ожидания и считывания следующего символа, а также последующий анализ этого символа. Если прочитан ожидаемый символ, то выдается соответствующий результат и читается следующий символ. При этом если при поступлении данного символа автомат не меняет состояния, то организуется цикл, иначе проверка символа проходит операцией ЕСЛИ и происходит переход в другое состояние.

Поскольку верификация модели «Кофейный автомат» будет продолжена в SPIN, то данная модель была реализована на языке Promela [16].

Верификатор SPIN позволяет описать проверяемые свойства модели формулами линейной темпоральной логики, используя темпоральные операторы F, G и U, атомарные предикаты и обычные булевы операторы И, ИЛИ, НЕ, определённые в языке Promela. Однако при построении LTL формул в системе SPIN не может быть использован темпоральный оператор X (neXt), поскольку понятия «следующее состояние» в системе параллельных процессов нет: в общем случае каждый из параллельных процессов может выполнить независимый шаг.

Таким образом, помимо верификации базовых свойств, в верификаторе SPIN пользователь может сам определить специфичные требования к модели, выразив их в виде LTL формул.

Обозначение логических операторов в языке Promela приведено в табл. 2.

Таблица 2 - Обозначение темпоральных операторов в языке Promela.

Тип оператора / операнда	Обозначение	Расшифровка
Операнды	true, false, определённые пользователем имена переменных, выражения в фигурных скобках	-
Унарные операторы	[]	Оператор G, всегда
	<>	Оператор F, хотя бы один раз в будущем
	!	Отрицание
Бинарные операторы	U	Оператор U, до тех пор пока
	W	Оператор W, только когда верна формула
	V	(p V q) означает !(!p U !q)
	&&	Логическое И
	||	Логическое ИЛИ
	/\	Альтернативная форма записи оператора &&
	\/	Альтернативная форма записи оператора ||
	->	Импликация, следование
	<->	Эквивалентность

Синтаксис LTL-формул в программном коде, написанном на Promela, выглядит следующим образом: *ltl название_свойства { LTL-формула }*

В качестве примера можно привести вышеописанное свойство «Регулировка количества сахара» (количество сахара не может быть меньше 0 и больше максимальной нормы на порцию напитка). Установив, что

максимальная порция сахара в напитке – 6 единиц, получим следующий код на языке Promela: *ltl f3 { [](sugar >= 0 && sugar <= 6)}*

Полный листинг программы с LTL-формулами, описанными на языке верификатора SPIN, приведён в Приложении А.

В целом, программа, написанная на Promela, состоит из трех основных типов объектов: процессы (задают поведение), каналы сообщений и переменные (определяют окружение, в котором выполняются процессы).

Синтаксис Promela представляет собой синтаксис, схожий с языком C. Ниже приведены основные элементы и конструкции языка Promela.

- Типы процессов. Всегда объявляются глобально и описываются при помощи объявления proctype. Для разумного использования модели должно быть хотя бы одно объявление типа процесса и хотя бы один экземпляр процесса какого-нибудь типа. Процесс init - базовый процесс, который всегда активируется в начальном состоянии модели. Для создания нового экземпляра процесса используется оператор run. Количество процессов, которое может быть создано в Spin - 255.
- Разделители. Разделителем является точка с запятой, но не является указателем конца оператора. В Promela допускается пустой оператор. Разделители стрелка «->» и точка с запятой «;». эквивалентны. Стрелка иногда используется как неформальный способ указания на причинно-следственное отношение между двумя операторами.
- Типы данных. Типы данных Promela близки к данным языка C: bit (0, 1), bool (false, true), byte (0..255), chan (тип каналов, их количество в диапазоне 1..255), pid (идентификатор процесса; количество процессов в диапазоне 0..255), mtype (перечисляемый тип, в диапазоне 1..255), short (-215 .. 215 - 1), int (-231 .. 231 - 1), unsigned (0 .. 232 - 1).
- Массивы. Поддерживаются только одномерные массивы (например, int arr[N] объявляет массив с именем arr из N элементов). Обращение к элементу массива происходит по индексу (любое выражение, которое вычисляет целочисленное значение).

- Синтаксис оператора printf подобен аналогичному оператору в C. В строке могут задаваться форматы выводимых переменных: d – целое, i – беззнаковое целое, c – один символ, e – константа типа mtype

- Каналы. Используются для моделирования передачи данных от одного процесса к другому (например, chan msgs = [10] of { short } - канал msgs, который может хранить до 10 сообщений типа short) Для работы с каналами существуют операторы посылки («!») и получения («?») сообщений. Каналы передают сообщения в порядке FIFO (англ. «first in – first out» - «первым вошел – первым вышел»).

- Операторы. Схожи с операторами языка C: скобки, ! (отрицание), ++ (инкремент), -- (декремент), * (умножение), / (деление), % (модуль), + (сложение), - (вычитание), << и >> (сдвиги), операторы отношений (<, <=, >, >=), == (равенство), != (неравенство), & (побитовое И), ^ (побитовое исключающее ИЛИ), | (побитовое ИЛИ), && (логическое И), || (логическое ИЛИ), = (присваивание). Использование инкремента и декремента в Promela отличается от их использования в C: они могут быть только постфиксными и использоваться только в выражении, но не в операторе присваивания.

- Оператор if. Несколько отличен от обычных условных операторов современных языков программирования. Структура if содержит, как минимум, две последовательности операторов, каждая предваряется двойным двоеточием. Выполняется только одна последовательность из списка возможных (выполнимых). Если выполнимыми оказываются несколько операторов, недетерминировано выбирается какой-либо один. При этом порядок перечисления альтернатив ни на что не влияет.

- Цикл. В структуре do содержатся последовательности операторов, которые предваряются двойным двоеточием. Для текущего выполнения может быть выбрана только одна опция. Обычный способ выхода из цикла – с помощью оператора break.

Следует отметить, что размер модели не влияет на режим симуляции, но отражается на времени верификации.

4.1.2 Реализация модели системы «Кофейный автомат» на языке Promela

Для функционирования системы «Кофейный автомат» были объявлены переменные, представленные на рис. 7.

```
int credit = 0, cash = 100, change = 0;
int sugar = 3, sug_plus = 1, sug_minus = -1;
int coffe_price = 30, cocoa_price = 20, tea_price = 10;
bool costenaugh = false;
bool cancel = false;
mtype = {tea, cocoa, coffee};
mtype whatdrink = 0;
int s = 0;
int sug = 1;
chan ch = [6] of {int};
```

Рисунок 7 - Объявление переменных

Дадим интерпретацию этих переменных:

- credit – сумма денег на счету у пользователя;
- cash – введённые пользователем купюры;
- change – сдача - рассчитывается как разность между переменной credit и стоимостью напитка;
- sugar - количество добавляемого сахара;
- sug_plus, sug_minus – добавление сахара путем нажатия пользователем кнопки регулировки сахара («+» и «-» соответственно);
- coffe_price, cocoa_price, tea_price – переменные, хранящие значение о цене напитков;
- cancel – флаг, хранящий данные о нажатии кнопки «Отмена»;
- costenaugh – флаг, хранящий данные о том, достаточно ли денег для покупки напитка;
- mtype = {tea,cocoa,coffee} – перечисление допустимых типов напитка в автомате;
- whatdrink – напиток из множества mtype, выбранный пользователем.
- s – порядковый номер состояния автомата;

- sug – отвечает за нажатие пользователем кнопок регулировки сахара;
- ch – канал для передачи номера состояний между процедурами.

Процессы S1(), S2(), … S6() являются состояниями автомата S_1, S_2, … S_6 соответственно. Процесс init запускает процесс S0(), соответствующий начальному состоянию автомата S_0.

Процессы вызывают друг друга согласно графу переходов автомата модели «Кофейный автомат», описанного ранее. Например, процесс, описывающий состояние S0 выглядит следующим образом (рисунок 8).

```
21  proctype S0()
22  {
23
24      s = 0;
25      ch ! s;
26
27      printf("\nS0. Начальный экран."); //MSC:
28
29      if
30      :: (true) -> cancel = true; run S6();
31      :: (true) -> cancel = false;
32          printf ("MSC: \nВведите купюру! (ваш кредит: %d)", credit);
33      s0:
34          if
35          :: (true) -> cash = 1; credit = credit + cash; printf ("\n(ваш кредит: %d)", credit);
36          :: (true) -> cash = 2; credit = credit + cash; printf ("\n(ваш кредит: %d)", credit);
37          :: (true) -> cash = 5; credit = credit + cash; printf ("\n(ваш кредит: %d)", credit);
38          :: (true) -> cash = 10; credit = credit + cash; printf ("\n(ваш кредит: %d)", credit);
39          :: (true) -> cash = 50; credit = credit + cash; printf ("\n(ваш кредит: %d)", credit);
40          :: (true) -> cash = 100; credit = credit + cash; printf ("\n(ваш кредит: %d)", credit);
41          fi
42          if
43          :: (credit >= tea_price || credit >= coffe_price || credit >= cocoa_price) -> costenaugh = true;
44           printf("\nCOSTENAUGH == %d, CREDIT = %d\n", costenaugh, credit);
45              if
46              :: (true) -> run S1();
47              :: (true) -> run S2();
48              fi
49          :: (credit < tea_price && credit < tea_price && credit < tea_price) -> costenaugh = false;
50           printf("\nCOSTENAUGH == %d, CREDIT = %d\n", costenaugh, credit);goto s0;
51          fi
52      fi
53  }
```

Рисунок 8 - Процесс состояния S0

Данный процесс отражает состояние «Начальный экран», в котором можно нажать кнопку «Отмена» (произойдёт возврат денег и переход в состояние S_6), вставить купюру или монету, определённые в настройках кофейного автомата (при этом происходит прибавление суммы к общему кредиту); если денег на счету достаточно для покупки напитка, возможен переход к состоянию регулировки сахара или выбора напитка (по нажатию соответствующей кнопки на кофейном автомате); если на счету недостаточно

денег для покупки напитка, то произойдёт переход к метке s0 и повторный запрос пользователю на ввод наличных.

4.2 Верификация требований в SPIN. Работа с GUI iSpin

4.2.1 Работа с GUI iSpin. Режим симуляции

Формальная верификация - это математическое (строгое) доказательство корректности системы, дополняющий метод к симуляции и тестированию. Основной смысл верификации - построить формальную (математическую) модель, которая определяет возможное поведение системы. При этом требования корректности записываются в виде формальной спецификации требований, отражающих желаемое поведение системы. На основе описанных системы и требований к ней, можно формальным доказательством проверить, действительно ли возможное поведение согласуется с желаемым.

Для проведения формальной верификации требуются:

- модель системы, включающая состояния, которые хранят информацию о значениях переменных, программных счетчиках и пр., и отношение переходов между данными состояниями;
- метод спецификации, необходимый для выражения требований в формальном виде;
- множество правил доказательства, позволяющих определить, удовлетворяет ли модель сформулированным требованиям.

Верификация для выполнения поставленных задач, а именно – для верификации требований к системе «Кофейный автомат» - будет проводиться в выбранном ранее графическом интерфейсе iSpin на компьютере с операционной системой Manjaro (ветка Arch Linux, ядро версии 4.4.41)

Окно утилиты iSpin состоит из нескольких вкладок (сгруппированных по выполняемым функциям):

- «Simulate/Replay» - выбор режима, настройки, запуска, остановки и выполнение по шагам, просмотра результата процесса моделирования;
- «Verification» - задание верифицируемых свойств модели, настройка параметров и запуска/остановки процесса верификации модели, просмотр и сохранение результатов верификации;
- «Help» - получение справки по программе;
- «Swarm Run» - установка параметров и запуск большого количества небольших задач верификации в параллельном режиме;
- кнопки «Save Session»/«Restore Session» - сохранение текущей и восстановление сохраненных ранее сессий работы с программой;
- кнопка «Quit» - выход из программы.

Для проверки модели «Кофейный автомат» были использованы три вкладки, которые являются основными: «Edit/view» («Правка/Просмотр»), «Simulate/Replay» («Симуляция/Повторная симуляция»), «Verification» («Верификация»)

Для загрузки программы в iSpin используется режим редактирования (вкладка «Edit/view»). В режиме редактирования доступна загрузка Promela-описания модели, сохранение, проверка синтаксиса, символьный поиск, проверка на избыточность кода, построение символьных таблиц для переменных и процессов, а также построение конечных автоматов по описанной модели.

Главное окно с загруженной программой выглядит, как представлено на рис. 9.

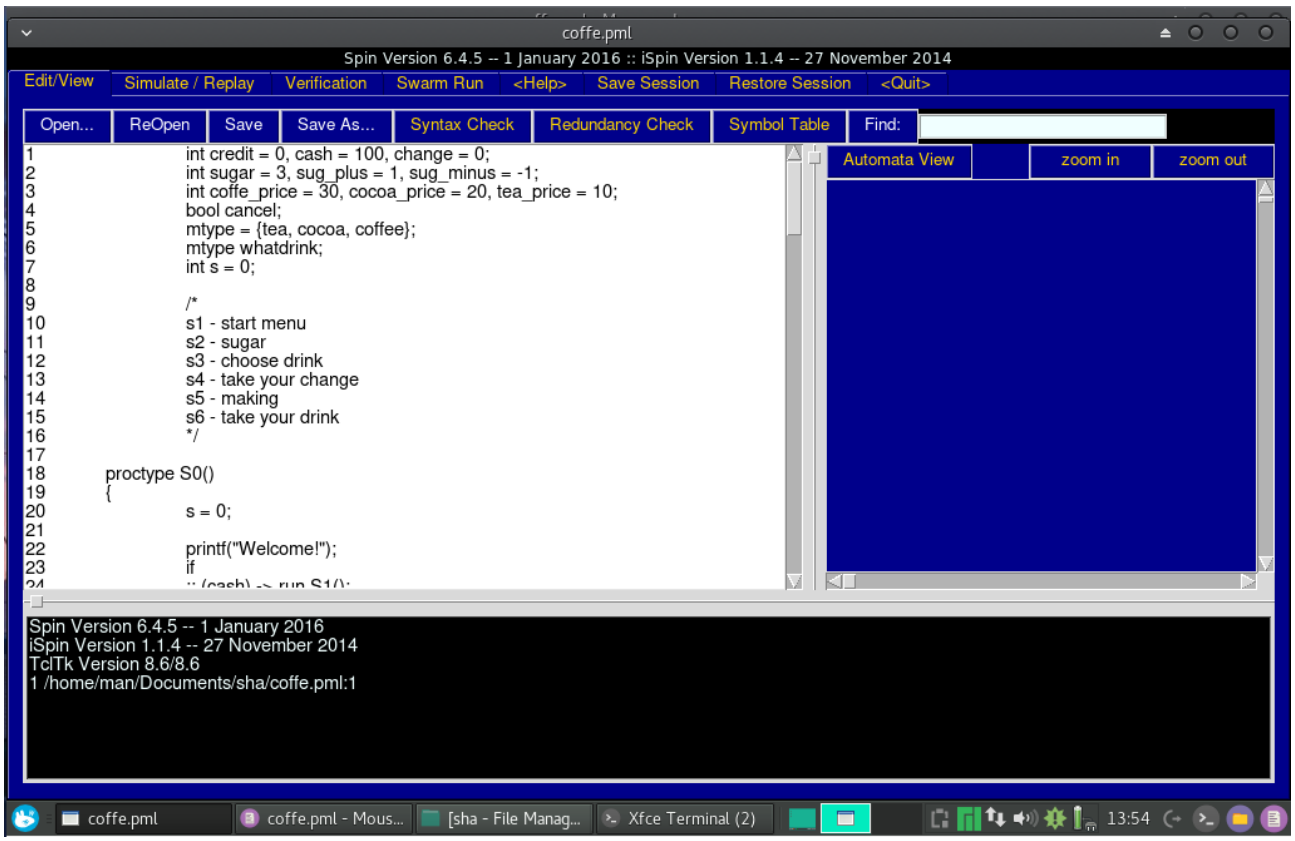

Рисунок 9 - Окно GUI iSpin

В левой части вкладки «Edit/View» находится окно редактирования исходного кода модели на языке Promela, в правой части - окно графического (в виде конечного автомата) представления процессов системы, описанных в коде с помощью ключевого слова proctype, начального процесса init, а также специальных процессов, получаемых из утверждений о невозможности (never claims), которые используются для проверки свойств модели. В нижней части вкладки располагается окно системного лога, в котором отражаются все действия пользователя в программе и результаты выполнения команд.

После загрузки программы и проверки синтаксиса проверим построенную модель в режиме симуляции. Режим симуляции позволяет исполнить один из возможных путей работы системы. Данный режим доступен в iSpin во вкладке моделирования «Sumulate/Replay».

Во вкладке «Sumulate/Replay» при выполнении симуляции (моделирования) можно выбрать или изменить следующие параметры:

- выбрать режим моделирования и задания ряда настроек моделирования; предусмотрено три режима моделирования: случайное моделирование (с параметром целочисленного значения - «зерна» моделирования; выполнение следующего шага выбирается случайным образом), интерактивное моделирование (выполнение следующего шага в системе выбирается пользователем), моделирование контрпримера (за основу берется контрпример, который может быть сгенерирован в результате процесса верификации, завершившегося ошибкой, и записан в файл с расширением *.trail);
- задать дополнительные параметры: количество пропускаемых от начала шагов моделирования, максимальное количество шагов моделирования, отслеживание значений переменных в процессе моделирование;
- задать настройки поведения системы в случае заполнения каналов передачи сообщений: если очередь сообщений заполнена, то система может либо блокировать отправителя до момента появления свободного места для сообщения, либо терять сообщения (loses new messages);
- задать настройки фильтрации результатов моделирования (например, задать нужные идентификаторы процессов или очередей сообщений в виде регулярных выражений).

Во вкладке «Simulate/Replay» также присутствует окно (в верхней правой части) отображения консольной команды SPIN с фактическими аргументами, которая выполняется при запуске процесса моделирования.

Запустить симуляцию с выбранными параметрами можно, используя кнопки управления процессом моделирования: «(Re)Run» (запуск/перезапуск моделирования), «Stop» (остановка моделирования), «Rewind» (переход на первый шаг моделирования без его перезапуска), «Step Forward»/«Step Backward» (моделирование по шагам - на один шаг вперед/назад, соответственно);

При соответствующей отметке флага «MSC+stmnt» в правой части вкладки отобразится графическое представление последовательности

пересылаемых между процессами системы сообщений (англ. «MSC» - Message Sequence Chart). Для того, чтобы это было возможно, строка вывода в функции printf() должна начинаться с символов: «MSC: ».

Результаты симуляции отображаются в нижней части вкладки.

Для симуляции модели кофейного аппарата, были выбраны параметры, показанные на рис.10.

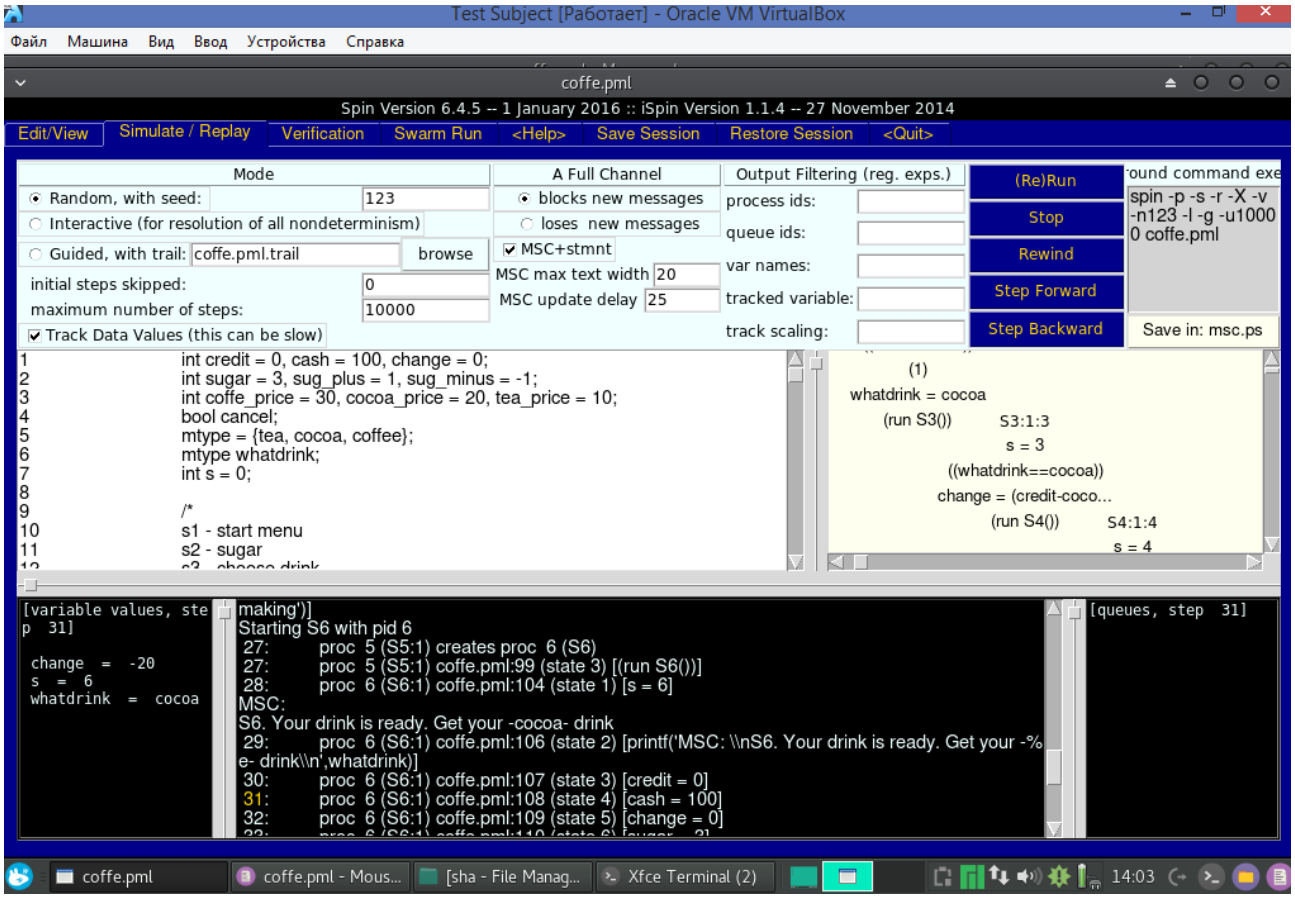

Рисунок 10 - Параметры для симуляции

Запуск программы в режиме симуляции показал, что построенная модель в одном из своих путей исполнения работает корректно. Но для точного понимания, что система построена корректно в целом, недостаточно провести симуляцию один, сто или тысячу раз, - необходимо провести верификацию требований к ней.

Говоря о преимуществах iSpin, следует выделить построение автомата по описанной на Promela модели. Такой автомат можно увидеть на вкладке «Edit/view» в правой части окна. При этом вывод автомата разбивается на состояния, которые необходимо выбрать из предложенного списка. Таким

образом, автомат, построенный верификатором на основе программы на языке Promela содержит состояния, которые могут содержать внутри себя в свою очередь другие автоматы.

Также, стоит отметить, что автомат, построенный верификатором является моделью Крипке.

Модель Крипке – это граф переходов (S, T, s_0, L, F), где:

- S – конечное множество состояний;
- $T \subseteq S \times S$ – множество переходов;
- s0 – начальное состояние;
- $L: S \rightarrow 2^{AP}$, где AP – множество атомарных высказываний;
- $F \subseteq S$ – множество допускающих состояний.

Путь в этом графе $\pi = s_0, s_1, s_2, \ldots s_n, \ldots$ - последовательность вычислений системы.

Модель Крипке преобразуется из автомата Мили следующим образом:

- установка состояний на событиях и выходных воздействиях (переменных);
- создание полного графа переходов;
- редукция полного графа переходов с внесением тесных отрицаний внутрь атомарной формулы.

Результат преобразования представлен на рис. 11 и 12. Здесь показаны все переходы в состояниях S1 («Начальный экран») и S3 («Выбор напитка»), соответственно.

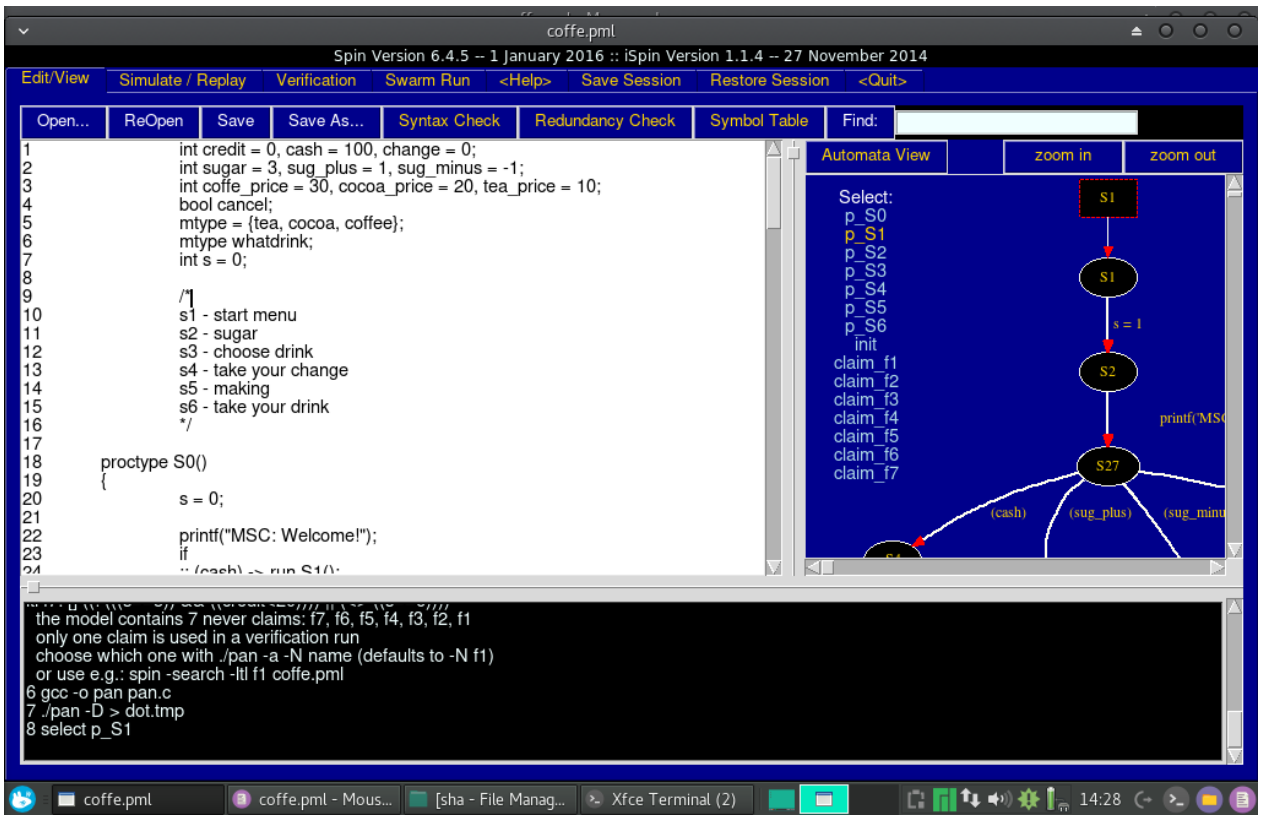

Рисунок 11 - Переходы в состоянии S1

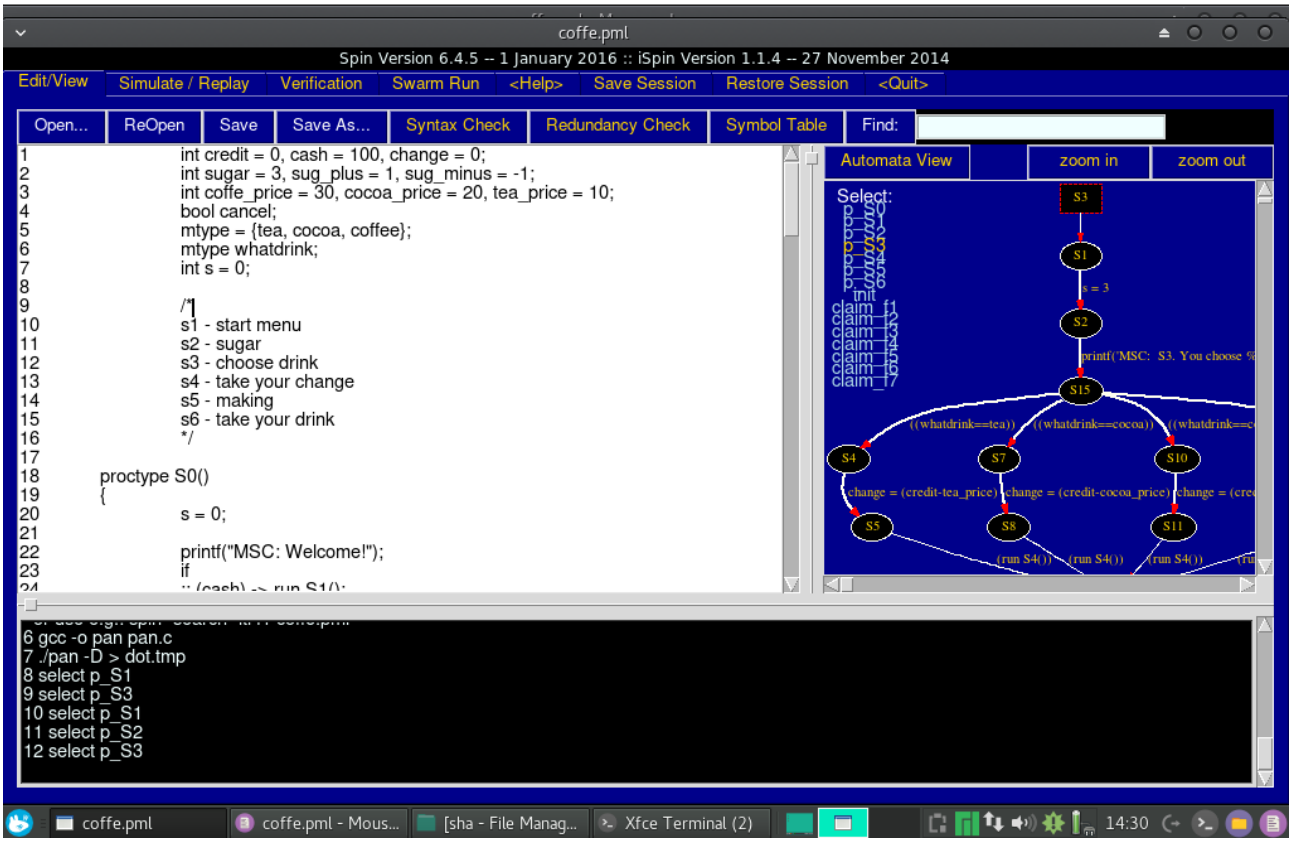

Рисунок 12 - Переходы в состоянии S3

Автомат можно сохранить в отдельный файл, выполнив соответствующие команды (рис. 13).

В данном случае, схема автомата сохранится в файле с расширением *.ps (pan.ps)

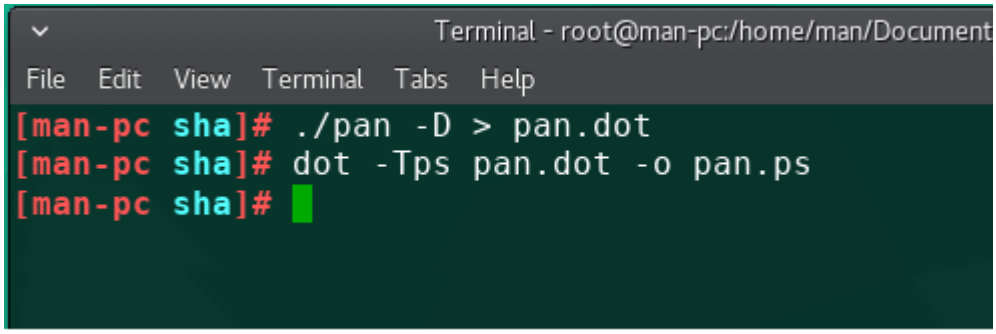

Рисунок 13 - Конвертация в формат *.ps

Состояния графов SPIN размечает самостоятельно в автоматическом режиме.

4.2.2 Работа с GUI iSpin. Режим верификации

Проблема верификации (то есть формального анализа) системы заключается в построении формального доказательства того, что данная система соответствует своей спецификации. Для верификации модели в iSpin, как было сказано выше, необходимо:

- построить математическую модель системы;
- формализовать понятие соответствия системы ее спецификации;
- разработать в рамках данной модели формальные процедуры проверки соответствия системы своей спецификации.

Построение модели и формализация требований к модели были рассмотрены выше. Процедуры же проверки указываются в самом верификаторе SPIN.

Вкладка «Verification» программы iSpin представлена на рис. 14.

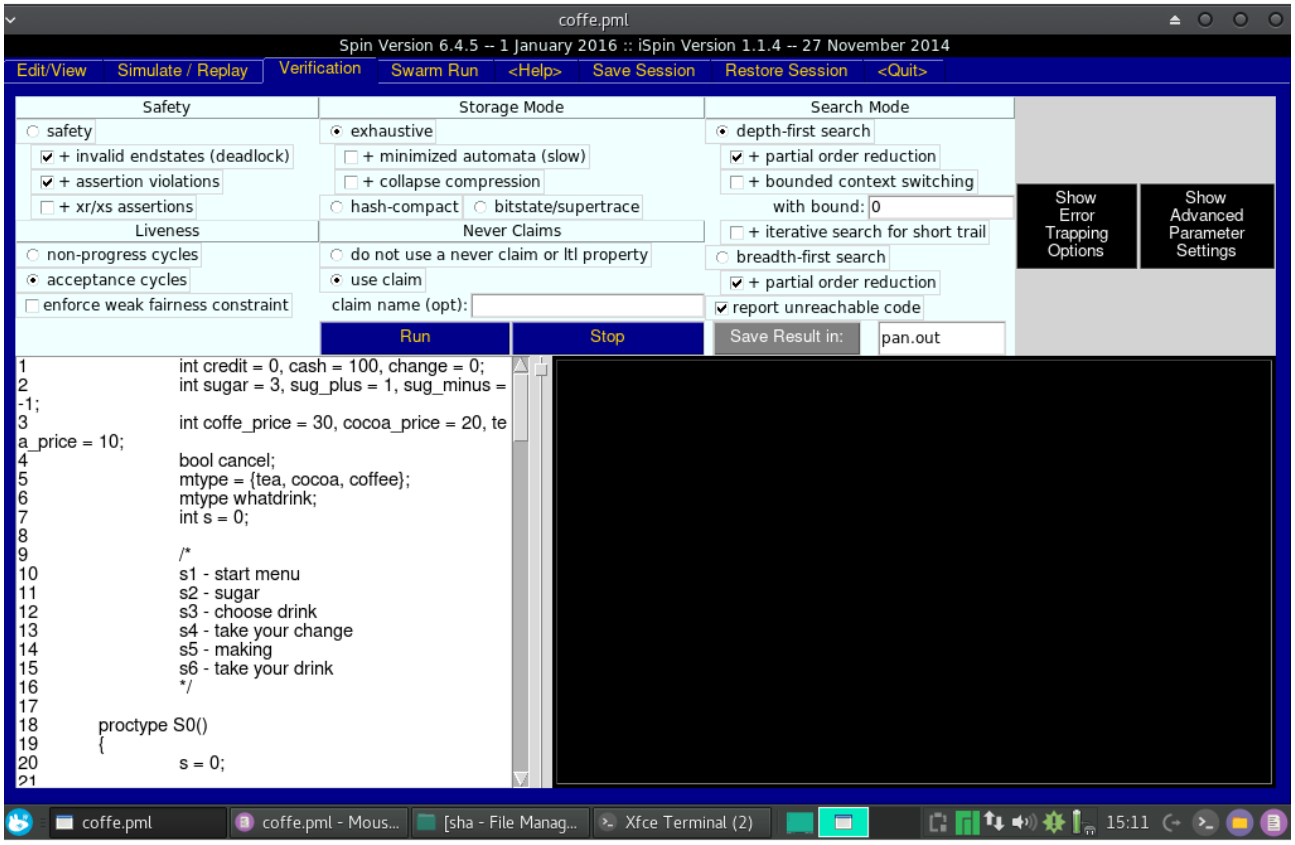

Рисунок 14 - Вкладка «Verification»

Вверху вкладки есть область для задания настроек проверяемых у модели свойств безопасности («Safety») и живости («Liveness») [16], настройки режима использования памяти, настройка утверждений о невозможности (для проверки конкретных утверждений), настройка режима обработки автомата (выбор использования поиска в глубину или поиска в ширину). Кроме того, с целью оптимизации процесса верификации для каждого из режимов можно включить использование алгоритма уменьшения пространства состояний модели и добавить в результирующий вывод информацию о всех недостижимых состояниях модели.

Во вкладке «Verification» в правой верхней части находятся две кнопки, в следствии нажатия которых отображаются скрытые поля с элементами для настройки дополнительных параметров процесса верификации: параметры обработки ошибок («Show Error Trapping Options») и параметры для задания дополнительных опций верификатора SPIN («Show Advanced Parameter Settings»). Среди данных расширенных параметров есть опция выбора как

поступить при переполнении канала сообщениями (блокировать или имитировать потерю), остановка при обнаружении определённого количества ошибок, опция построения таблиц состояний (по описанному в программе на Promela графу переходов), а также функции объяснения многих параметров – например, таких как доступная память или временные директивы для компиляции.

Для запуска или остановки процесса верификации необходимо нажать на кнопку запуска или останова соответственно. Результат проверки отображается в окне вывода результатов верификации, расположенном в правой части вкладки.

Подход к верификации на основе метода model checking заключается в том, что для модели системы формально проверяется выполнение логической формулы, выражающего свойство ее «правильного» поведения – т.е. одного из некоторого множества свойств, которые делают систему полезной для использования. Данные свойства должны быть проверены поочерёдно, одно за другим [17]. Традиционно свойства разбиваются на следующие классы:

- свойства достижимости (reachability) - определяют, что некоторые характерные состояния системы могут быть достигнуты; выражаются LTL формулой Fϕ;
- свойства безопасности (safety) - определяют, что нечто «плохое», - т.е. нежелательное, - никогда не произойдет с системой; выражается формулой G¬ϕ;
- свойства живости (liveness) - определяют, что при определённых обстоятельствах нечто «хорошее» в конце концов произойдет при любом развитии событий; выражается формулой GFϕ;
- свойства справедливости (fairness) - определяют, что нечто будет выполняться неопределенно часто; нужны для того, чтобы «отсеять» нереалистичные («несправидливые»), нереализуемые траектории, возникшие из-за ограниченных выразительных средств модели.

При разработке параллельных систем, кроме требований, которым должно удовлетворять поведение данной конкретной системы, для системы следует проверять еще и общие свойства, гарантирующие отсутствие некорректностей, типичных для параллельных систем. Исходя из этого, SPIN поддерживает верификацию некоторого набора общих свойств, называемых в SPIN базовыми.

Таким образом, начинать процесс верификации следует с определения режима верификации - проверки свойств безопасности или живости и настройки параметров выбранного режима. В случае выбора режима проверки свойств безопасности доступны следующие дополнительные опции: «invalid endstates (deadlock)» - проверка состояний модели на неправильные конечные состояния (процессы, которые не достигли конечного состояния), «assertion violations» и «xr/xs assertions» - проверка состояний модели на нарушение пользовательских ассертов и ассертов для эксклюзивного использования канала передачи сообщений.

Поясняя вышесказанное, оператор assert позволяет задавать локальные инварианты - т.е. такие свойства, которые должны выполняться в определенных точках программы. Если выражение, стоящее под оператором assert истинно, то не производится никакого эффекта. Если выражение будет ложно при симуляции, то SPIN выдаст сообщение «Error: assertion violated». При выполнении базовой верификации свойств безопасности нарушение операторов assert проверяется на всех конечных вычислениях.

В случае выбора режима проверки свойств живости, доступны следующие основные опции: «non-progress cycles» - проверка на бесконечные циклы, не помеченные специальной меткой с префиксом progress, «acceptance cycles» - проверка модели на циклы, в которых бесконечно часто посещаются состояния помеченные специальной меткой с префиксом accept, «enforce weak fairness constraint» - включение требования слабой справедливости к вычислениям проверяемой модели (без данной опции в Spin не делается никакого предположения о справедливости вычислений).

При проверке определённого свойства, описанного LTL-формулой, в iSpin необходимо явно указывать, какую формулу (требование) следует проверить. Это делается при помощи секции «Never Claims», используя переключатель «use claim (opt.)». В поле рядом с переключателем прописывается название свойства, описанного в тексте программы (которая, помимо описания модели, содержит и свойства системы в виде LTL-формул на языке Promela).

Таким образом, чтобы проверить 7 требуемых свойств, необходимо провести верификацию 7 раз – по одному на соответствующее свойство. Это же сообщает и iSpin при проверке синтаксиса (рис. 15).

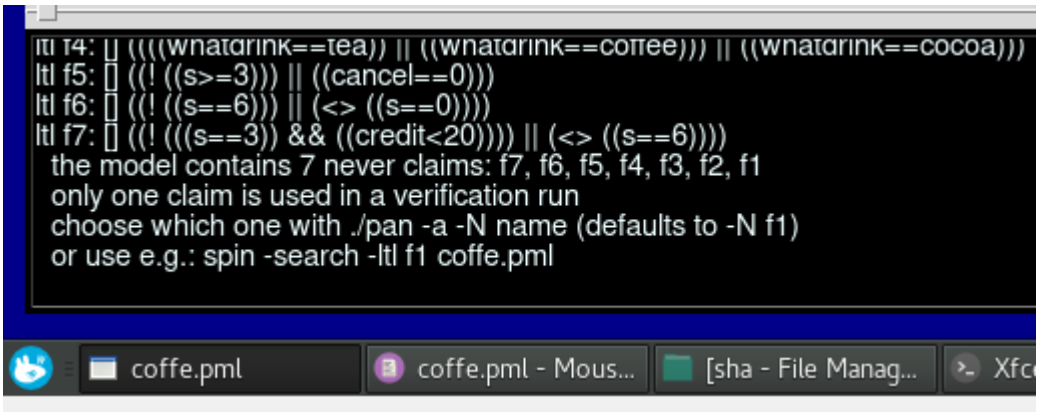

Рисунок 15 - Сообщение о проверке формул

Так как процесс верификации для каждой из семи описанных формул происходит схожим образом, приведём пример верификации одной из них – формулы f7 (рис. 16).

Напомним формулировку требования к модели системы, обозначенного в LTL-формуле как f7: «Автомат не может приготовить выбранный напиток, если на счету нет достаточного количества денег для оплаты». Это одно из важных свойств модели «Кофейный автомат».

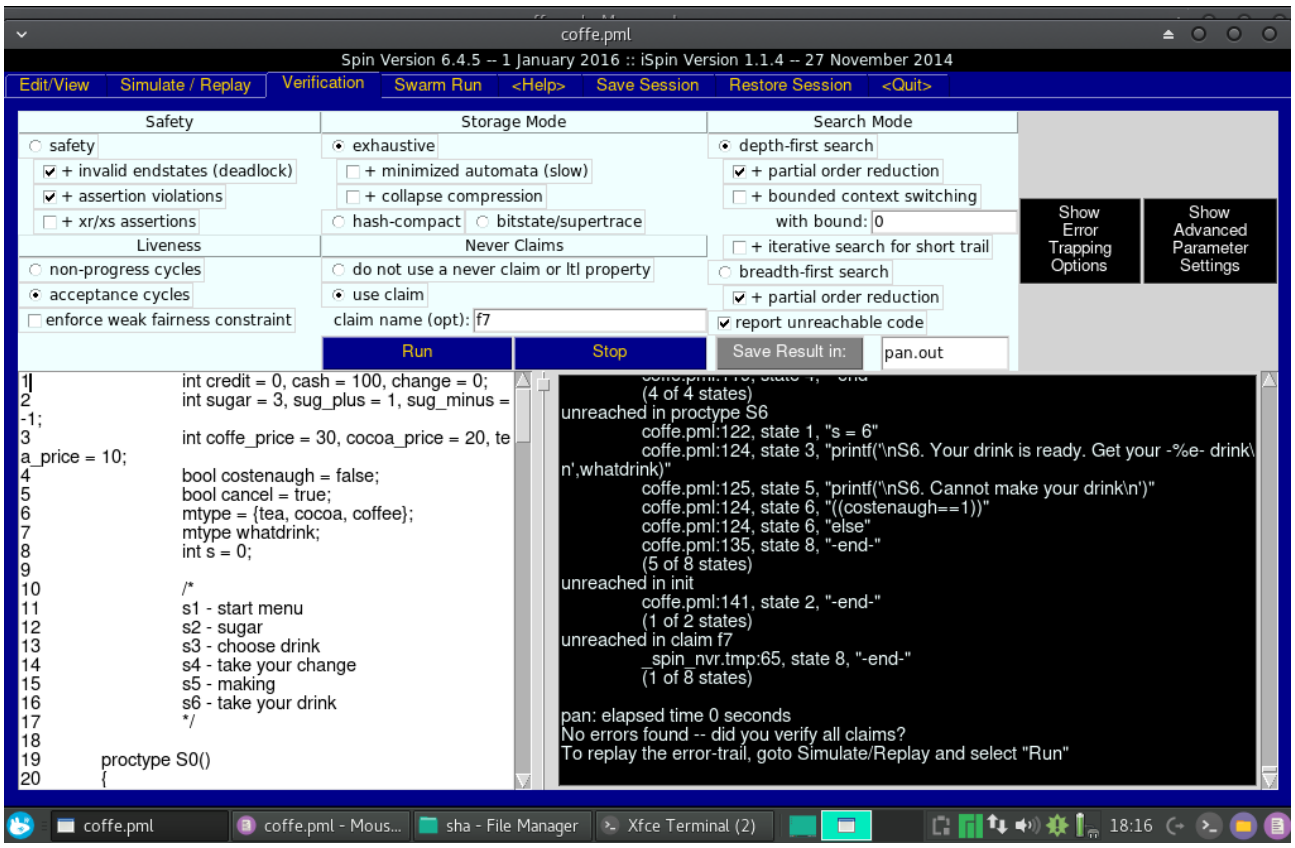

Рисунок 16 - Результат верификации формулы f7

Верификатор Spin сообщает, что при проверке данной формулы ошибки не были найдены («No errors found»), и напоминает, чтобы пользователь проверил и остальные формулы, если таковые имеются.

Подробный результат верификации можно сохранить в отдельный файл (в секции «Search mode» -> «Save result in»). Данный результат проверки формулы f7 приведён в Приложении Б.

В ходе выполнения данной работы были успешно проверены все требования, сформулированные в главе 3.

Однако, если формулу f7 ({credit < tea_price && credit < tea_price && credit < tea_price) -> costenaugh == false}) изменить таким образом, чтобы она оказалось заведомо неверной (например, на { credit < tea_price && credit < tea_price && credit < tea_price) -> costenaugh == true} – то есть, из того, что денег, введёных покупателем не хватает на покупку напитка, следует, что денег хватает), то верификатор выдаст ошибку (рис. 17).

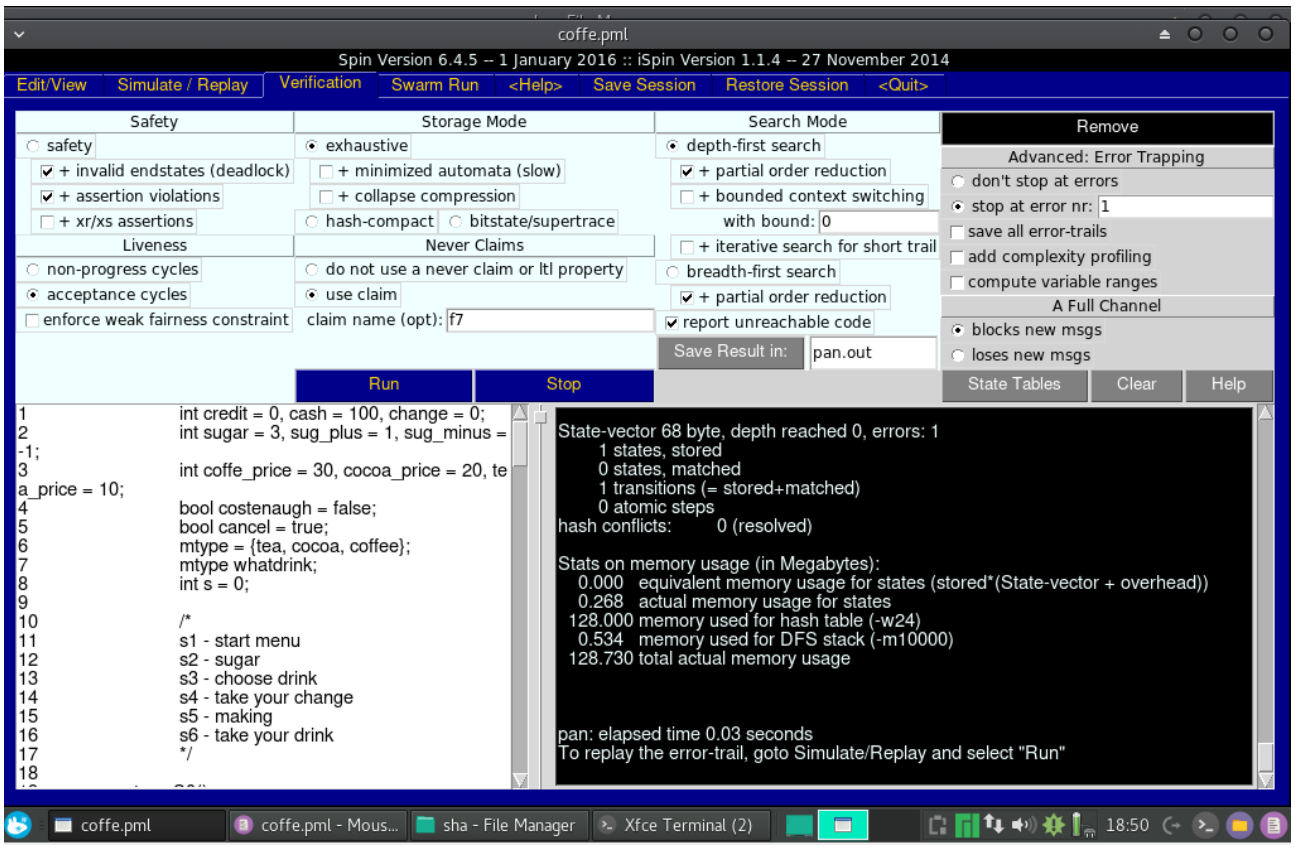

Рисунок 17 - Свойство f7 не прошло верификацию

Результат верификации в данном случае будет таким: «To replay the error-trail, goto Simulate/Replay and select Run»

В этом случае, можно посмотреть контрпример в режиме симуляции.

Для это во вкладке «Simulate/Replay» выбирается режим управляемой симуляции из trial-файла (флаг «Guided, with trail») и запускается симуляция.

Результат такой симуляции показан на рис. 18.

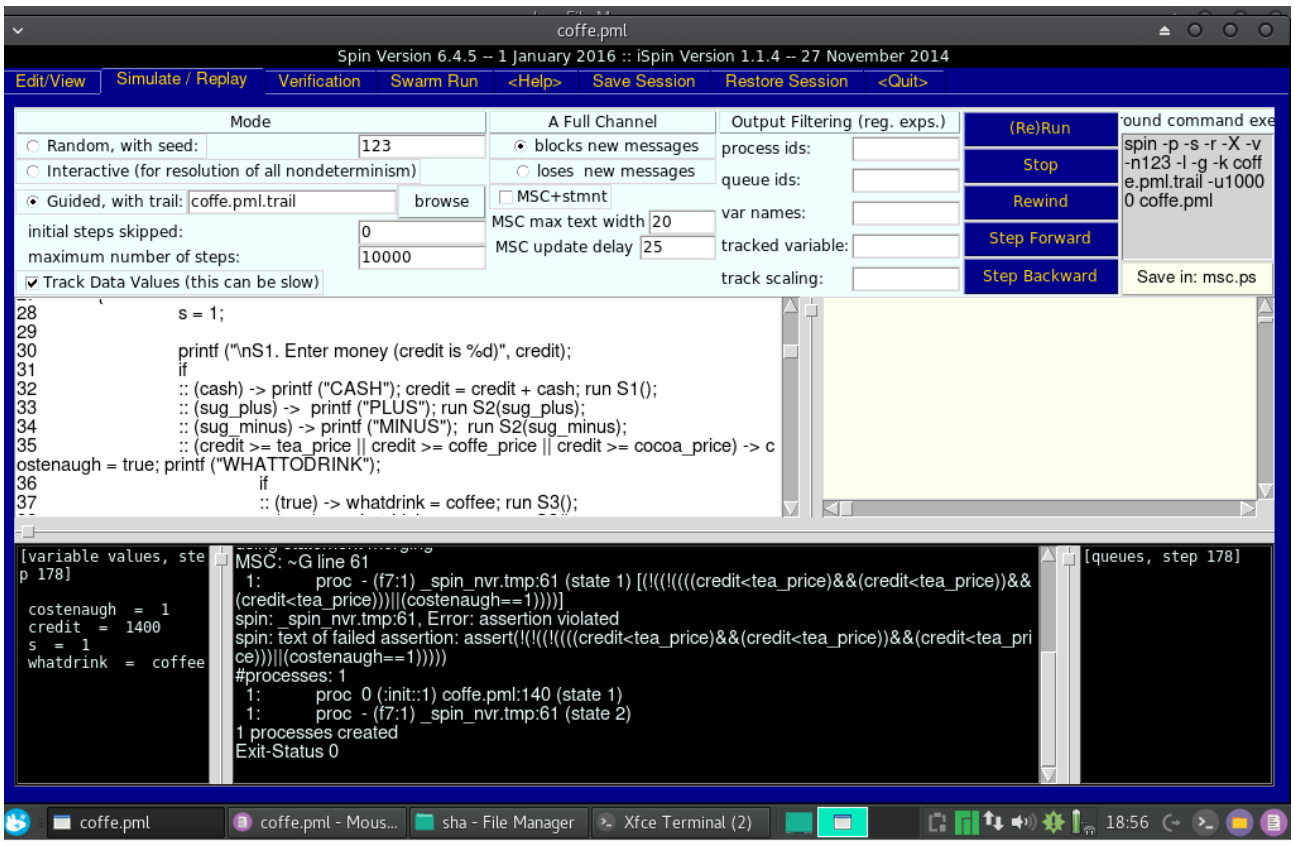

Рисунок 18 - Контрпример заведомо неверного свойства f7

В нижней части данной вкладки располагаются три окна – значения переменных при данных шагах, результат симуляции и очередь сообщений из каналов, если таковые имеются.

Чтобы посмотреть шаги построения контрпримера, необходимо использовать кнопки «Step forward»/«Step Backward». В данном случае, результат симуляции говорит о том, что построение началось с процесса S0. Наблюдая за изменением переменных при пошаговой симуляции можно заключить следующее: в процессе S1 (т.к. переменная s == 1) происходило постоянное наращивание цены на 100 ед. и сравнение результата с отрицанием формулы из файла _spin_nvr.tmp (рис. 19, жёлтым цветом выделен фрагмент, связанный с формулой f7). При каждом шаге подтверждалось отрицание формулы – т.к. напитки в модели кофейного автомата стоят 10, 20 и 30 ед., что даже в сумме дают меньше, чем введённые пользователем деньги. При постоянном увеличении введённых денег сделался вывод о том, что их хватит для оплаты напитка, т.к. такой суммы определённо хватит для оплаты как

минимум одного, и, таким образом, был сделан вывод о том, что формула f7 не верна.

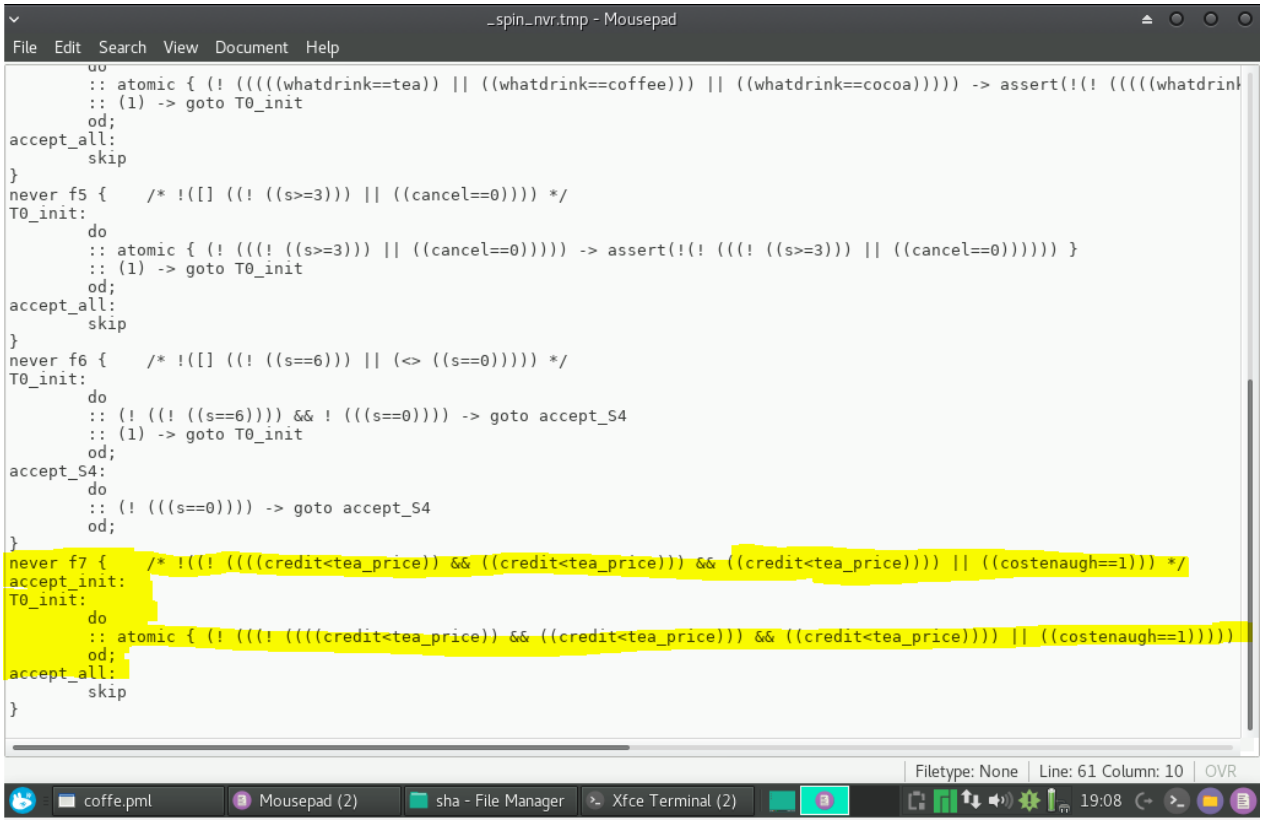

Рисунок 19 - Фрагмент файла _spin_nvr.tmp

При внесении ошибки не в формулу, а в саму спецификацию, верификатор SPIN также найдёт ошибку и построит контрпример.

Пусть при тех же исходных данных (LTL-формулы, описывающие требования к системе, количество и назначение процедур (состояния автомата) и переменные, отвечающие за входные параметры) изменятся процессы, протекающие в процедурах (рис. 20).

```
proctype S0()
{
    s = 0;

    credit = 100; // user's credit
    cash = 100;   // current money
    change = 0;   // change
    sugar = 3;    // level of sugar
    costenaugh = true; // flag: have enaugh money on the account (credit)?
    cancel = true;
    s = 0;

    printf("MSC: Welcome!");
    run S1();
}
proctype S1()
{
    s = 1;

    printf ("MSC: \nS1. Enter money (credit is %d)", credit);
    if
    :: (cash) -> printf ("MSC: CASH"); credit = credit + cash; run S2(sugar); // if user enter the money
    :: (sugar) -> run S2(sugar);                                              // if user control level of sugar
    :: (true) -> whatdrink = coffee; run S3();                                // if user choose coffee
    :: (true) -> whatdrink = cocoa; run S3();                                 // if user choose cocoa
    :: (true) -> whatdrink = tea; run S3();                                   // if user choose tea
    :: (cancel) -> printf ("MSC: Get your money (%d)", credit); credit = 0;  run S6(); // if "cancel" button is pressed
    fi
}
proctype S2(int sug)
{
    s = 2;

    printf("MSC: \nS2. Control level of sugar (now is %d/6)", sugar);
    if
    :: (true) -> sugar = 0;     //if user change level of sugar to 0
    :: (true) -> sugar = 1;     //if user change level of sugar to 1
    :: (true) -> sugar = 2;     //if user change level of sugar to 2
    :: (true) -> sugar = 3;     //if user change level of sugar to 3
```

Рисунок 20 - Фрагмент системы, описанной неверно

В такой заведомо неверной системе, описанной на языке Promela, по сравнению с исходной изменены следующие параметры:

- сумма средств на счету пользователя заранее равна ста единицам – то есть не может быть такого случая, что на счету пользователя будет баланс, равный нулю;
- флаг, отвечающий за достаточность денег на счету пользователя (со значениями: «Достаточно» / «Не достаточно» - «true» и «false» соответственно) всегда равен значению «true»;
- вместо детального описания изменения количества сахара в системе описывается лишь итог – чему равно итоговое количество сахара;
- кнопку «Отмена» можно нажать лишь на этапе ввода денег пользователем, - но не в состоянии изменения количества сахара или выбора напитка.
- после приготовления напитка не происходит присвоение переменным начальных значений.

При данных условиях верификация верной формулы f7 ({credit < tea_price && credit < tea_price && credit < tea_price) -> costenaugh == false}) проходит с ошибкой. Это происходит из-за того, что флаг в данной модели не изменятся и всегда равен true, - т.е. из-за того, что денег на счету пользователя меньше, чем стоимость напитка, никогда не выполнится условие равенства флага значению false.

4.3 Автомат Бюхи

При проверке требований, выраженных с помощью LTL-формул, могут использоваться различные способы представления такой формулы в виде графа переходов. Одним из способов является использование автоматов Бюхи.

При проверке свойств моделируемой системы верификатор SPIN автоматически строит автомат Бюхи по исходной LTL-формуле.

Автоматом Бюхи называется шестёрка $B = (Q, q0, \Sigma, \delta, L, F)$, где:

- Состояния автомата Q;
- Начальное состояние q_0;
- Алфавит Σ здесь не существенен;
- Отношение переходов δ: $Q \times Q$;
- Функция разметки L: $Q \rightarrow 2AP$;
- Множество допускающих состояний F.

Свойство p автоматически конвертируется SPIN в размеченный автомат Бюхи, который соответствует отрицанию проверяемой формулы. Список полученных автоматов Бюхи из формул, доступен на вкладке «Edit/View», они обозначаются iSpin как *claim_названиеСвойства* (рис. 20, отмечены красным цветом)

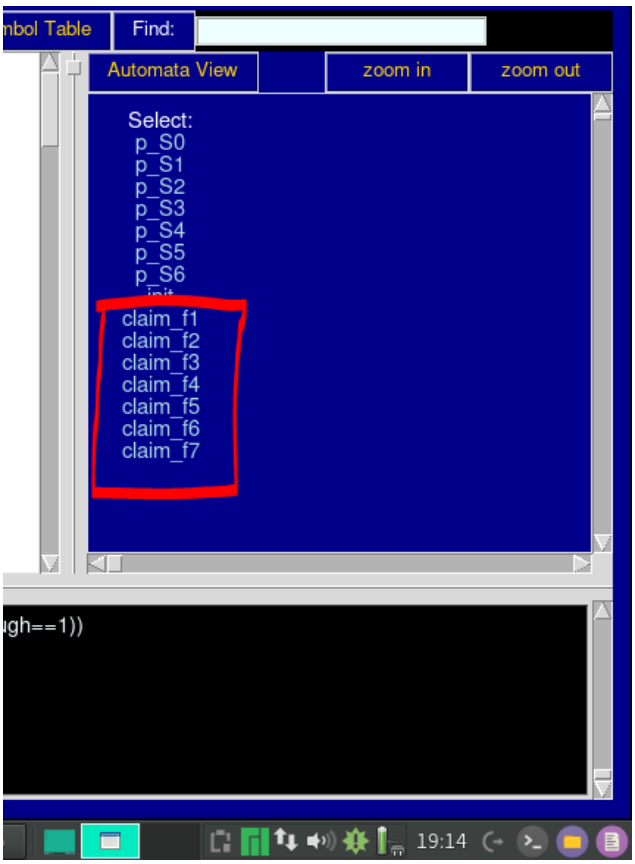

Рисунок 21 - Список автоматов Бюхи в утилите iSpin

Общий алгоритм, который обеспечивает трансляцию LTL-формулы в автомат Бюхи следующий. LTL-формула приводится в негативную нормальную формулу (ННФ), в которой отрицание применяется только к пропозициональным переменным. При этом можно считать, что в формуле не встречаются подформулы вида Fφ и Gφ, так как их можно соответственно заменить на True U φ и False R φ. Для приведения LTL-формулы в ННФ существуют тождества для темпоральных операторов (14 - 16)

$$\neg(\varphi \, U \, \psi) \equiv (\neg\varphi) \, R \, (\neg\psi); \tag{14}$$

$$\neg(\varphi \, R \, \psi) \equiv (\neg\varphi) \, U \, (\neg\psi); \tag{15}$$

$$\neg(X\varphi) \equiv X(\neg\varphi) \tag{16}$$

Из темпоральных операторов используются только U, W и X, а остальные операторы преобразуются к этой форме эквивалентными преобразованиями (описаны выше). Из булевых связок используются ∧ и ∨.

Алгоритм построения автомата Бюхи по LTL-формуле основывается на построение специального графа, каждая вершина в котором соответствует подформуле исходной формулы [18]. Построение заключается в последовательном расщеплении вершин.

После построения, такой граф преобразуется в автомат Бюхи, но, в отличие от классического автомата Бюхи, полученный автомат содержит не одно множество допускающих состояний, а столько, сколько существует подформул вида φ U ψ. Особенность построенного автомата (графа) состоит в том, что для допущения некоторого слова, цикл должен проходить по состояниям из каждого допускающего множества.

Существует множество модификаций данного алгоритма. Они все основываются на преобразовании формулы перед трансляцией с целью уменьшения числа состояний. В iSpin уменьшить число состояний автомата можно, поставив соответствующий флаг на вкладке «Verification» секции «Storage mode» (рис. 21)

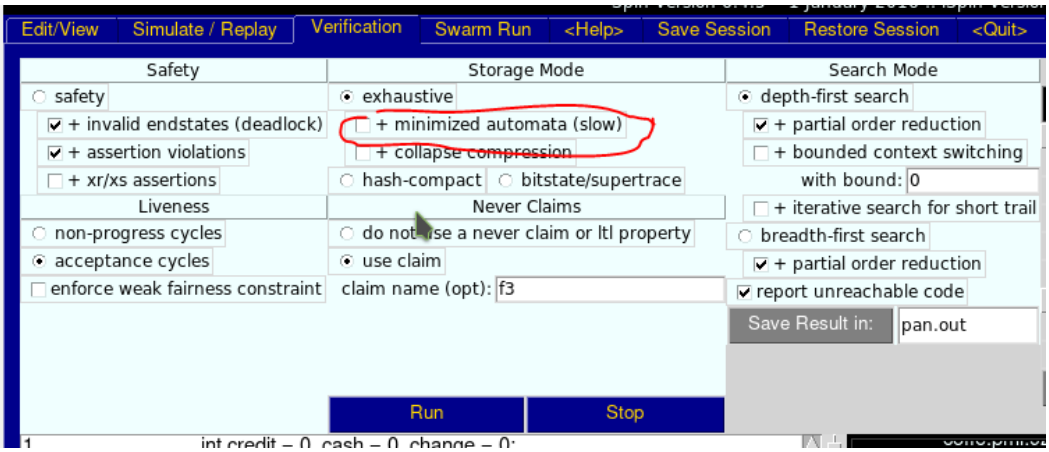

Рисунок 22 - Уменьшение числа состояний автомата в iSpin

В верификаторе Spin алгоритм перевода в автомат Бюхи следующий:

- даны модель Крипке и LTL-формула, выполнение которой требуется проверить на данной модели;

- из отрицания LTL-формулы строится эквивалентный ей автомат Бюхи, Модель Крипке также преобразуется в автомат Бюхи;
- строится третий автомат Бюхи как пересечение первых двух; такой автомат будет допускать пути исходной модели, которые не удовлетворяют LTL-формуле;
- если язык, допускаемый построенным автоматом-пересечением, пуст, то верификация успешна, иначе путь, допускаемый автоматом-пересечением, является контрпримером.

Пример автомата Бюхи, построенного в iSpin показан на рис. 22.

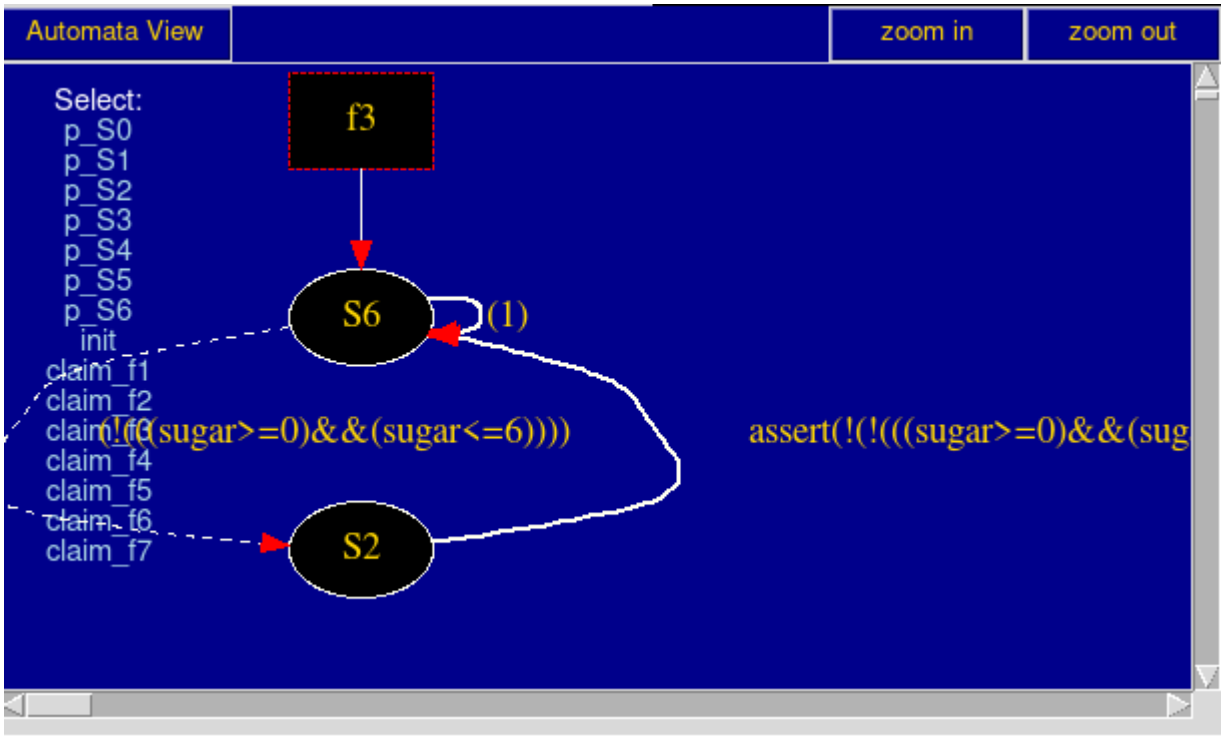

Рисунок 23 - Пример автомата Бюхи, построенного в iSpin

То есть, для построения данного автомата были предприняты следующие шаги (17 - 20)

- Исходная формула

$$G[0 \leq sugar \leq sugar_{max}] \qquad (17)$$

- Избавление от операторов F и G с помощью формул (2) и (3); в данном случае, оператор Gφ становится оператором вида false R φ

$$false\ R\ (0 \leq sugar \leq sugar_{max}) \quad (18)$$

- Отрицание полученной формулы; этот этап необходим для последующей проверки правильности исходной формулы

$$\neg(false\ R\ (0 \leq sugar \leq sugar_{max})) \quad (19)$$

- Приведение полученной формулы к ННФ (негативной нормальной форме); для этого необходимо воспользоваться формулами (14) и (15) – отрицание должно применяться к пропозициональным переменным

$$true\ U \neg\ (0 \leq sugar \leq sugar_{max}) \quad (20)$$

Стоит отметить, что в SPIN явно модель Крипке не строится. Верификатор получает на вход программу, написанную на Promela, и это позволяет ему выделять в программе элементарные действия, поскольку в семантике его языка определено, какие действия совершаются атомарно, как вычислять состояния объектов (переменных), как возвращать систему в предыдущее состояние (откат действий) и какие операторы порождают недетерминированность. Таким образом, верификатор может управлять исполнением программы: совершать и отменять элементарные шаги, вычислять глобальное состояние системы. Каждый элементарный шаг совершается в два хода: сначала происходит элементарный переход в программе, а затем вычисляются необходимые предикаты и совершается переход в автомате Бюхи. Если в автомате Бюхи невозможно выполнить ни один переход, то происходит откат: в данной ветви автомата не удалось найти контрпример.

ЗАКЛЮЧЕНИЕ

Формальная верификация играет ключевую роль в процессе разработки программного обеспечения, так как при данном способе обеспечивается достаточно полная проверка модели с помощью математических алгоритмов.

Формальная верификация позволяет устранять дефекты и обеспечивать качество и надёжность программного обеспечения, выявлять наиболее критичные подверженные ошибкам части создаваемой системы, производить оценку качества программного обеспечения. Особенно актуальным это является в ряде областей, где последствия ошибки в системе могут оказаться чрезвычайно дорогими (например, медицина или самолетостроение), так как формальная верификация способна обнаруживать сложные ошибки, которые сложно выявить с помощью других методов, таких как экспертиза или тестирование.

В ходе выполнения выпускной квалификационной работы была достигнута цель - разработан тестовый пример для верификации автоматных программ и решены поставленные задачи, а именно:

- изучены теоретические сведения об автоматном подходе к программированию;
- изучены принципы моделирования систем;
- изучены теоретические сведения о формальной верификации;
- изучены основы темпоральной логики;
- выбраны средства для верификации системы (метод подхода к проверке модели, утилиты для верификации и графического интерфейса - для работы с верификатором);
- разработана модель системы «Кофейный автомат»;
- сформулированы требования к системе «Кофейный автомат»;
- реализована система и требования к ней на языке верификатора (Promela);
- проведена верификация модели системы «Кофейный автомат» с помощью верификатора SPIN.

Результаты данной работы докладывались и обсуждались на Международной научно-практической конференции «Проблемы управления в социально-экономических и технических системах» (2018 г., доклад: «Моделирование работы кофейного автомата»), Международной научно-практической конференции «Проблемы управления в социально-экономических и технических системах» (2019 г., доклад: «Проверка модели программы с помощью логики LTL», работа отмечена дипломом II степени), VI Международной научной конференции «Проблемы управления, обработки и передачи информации (УОПИ-2018)» (2018 г., доклад: «Моделирование работы кофейного автомата»), Международной научно-практической конференции «Current Trends in History, Culture, Science and Technology» / «Современные направления в истории, культуре, науке и технике» (2019 г., доклад: «Simulation of the coffee vending machine work»), научном стендапе «Эпоха Гагарина: вызовы и приоритеты» в рамках Гагаринской недели (2019 г., тема выступления: «Верификация автоматных программ»).

Статьи по данной работе были опубликованы в сборниках [13, 14, 21, 22].

Также, по данной работе были разработаны методические рекомендации по теме: «Верификация автоматных программ в верификаторе SPIN с помощью графического интерфейса GUI iSpin» (Приложение В).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Теория автоматов и формальных языков [Текст]: учеб. пособие для студ. вузов, обуч. по спец. «Программная инженерия» и «Прикладная информатика» / Т.Э. Шульга. - Саратов: Сарат. гос. техн. ун-т, 2015 – 83 с.
2. Автоматное программирование . [Текст] / Поликарпова Н. И., Шалыто А. А./- СПб.: Санкт-Петербургский государственный университет информационных технологий, механики и оптики, 2008. - 167 с.
3. Шалыто А.А. Парадигма автоматного программирования // Международная научно-техническая мультиконференция «Проблемы информационно-компьютерных технологий и мехатроники». (МВУС`2007). – Таганрог: НИИМВС, 2007. – Т.1. – С.191–194.
4. С. Э. Вельдер, А. А. Шалыто - Верификация простых автоматных программ на основе метода model checking // опубликовано в материалах XV Международной научно-методической конференции «Высокие интеллектуальные технологии и инновации в образовании и науке» - СПбГ ПУ. 2008, с. 285–288.
5. Шишкина Е.А. О верификации автоматных программ // Проблемы управления в социально-экономических и технических системах - Сборник научных статей. 2018. с. 276-277 - Саратов, 2018.
6. Обзор подходов к верификации распределенных систем [Текст]: / И.Б. Бурдонов, А.С. Косачев, В.Н. Пономаренко, В.З. Шнитман. / М.: Российская академия наук. Институт системного программирования. 2003. – 51 с.
7. Тестирование на основе моделей [Текст]: / В. В. Кулямин/ М.: Российская академия наук. Институт системного программирования. – 2010
8. Хворостухина Е.В. Математическая логика [Текст] : учеб. пособие для студентов бакалавриата по направлениям подготовки 09.03.01 «Информатика и вычислительная техника», 09.03.04 «Программная инженерия» / Саратов, 2018.

9. Математические методы верификации схем и программ (семинар 7) [Текст] / Захаров В. А., Подымов В. В., осень 2018

10. Верификация автоматных программ [Текст] : учеб. Пособие / Вельдер С. Э., Лукин М. А., Шалыто А. А., Яминов Б. Р. - СПбГУ ИТМО, 2011. – 242 с.

11. "Spin Model Checker. The Primer and Reference Manual" / Holzmann G. Addison Wesley/ - 2003. - 608 стр.

12. Getting Started: Using iSPIN [Электронный ресурс] / Spin – Formal Ferifiaction - Режим доступа: http://spinroot.com/spin/Man/3_SpinGUI.html

13. Болдырева Ю.Ю. Моделирование работы кофейного автомата // Проблемы управления в социально-экономических и технических системах Сборник научных статей. 2018. с. 215-216. - Саратов, 2018.

14. Болдырева Ю.Ю. Выбор средства верификации модели кофейного автомата // VI Международная научная конференция «Проблемы управления, обработки и передачи информации (УОПИ-2018)» – Саратов, 2018 (в печати)

15. Разработка программной системы для генерации автоматных программ [Текст]: Шульга Т.Э., Иванов Е.А., Сластихина М.Д., Вагарина Н.С. // программирование. 2016. № 3. С. 55-63.

16. Введение в язык Promela и систему комплексной верификации Spin [Текст]: учеб. Пособие / И.В. Шошмина, Ю.Г. Карпов, / СПб: Санкт-Петербургский государственный политехнический университет, 2009 - 66 с.

17. Камкин А. Инструменты и методы программного анализа // крнференция TMPA School, Саратов, 3-4 марта 2018 – 49 с., ИСП РАН

18. LTL 2 BA [Электронный ресурс] / Fast translation from LTL formulae to Büchi automata by Denis Oddoux and Paul Gastin – Режим доступа: http://www.lsv.fr/~gastin/ltl2ba/index.php

19. SPINROOT [Электронный ресурс] / Promela Grammar – Режим доступа: http://spinroot.com/spin/Man/grammar.html

20. "Software engineering economics" / Barry W. Boehm. Prentice Hall / - 1981. - 767 стр.

21. Болдырева Ю.Ю. «Проверка модели программы с помощью логики LTL» // Проблемы управления в социально-экономических и технических системах». - Саратов, 2019 (в печати)

22. Болдырева Ю.Ю. «Simulation of the coffee vending machine work» // Current Trends in History, Culture, Science and Technology» / «Современные направления в истории, культуре, науке и технике». – Саратов, 2019 (в печати)

ПРИЛОЖЕНИЕ А

ЛИСТИНГ ПРОГРАММЫ, ОПИСЫВАЮЩЕЙ МОДЕЛЬ «КОФЕЙНЫЙ АВТОМАТ»

```
int credit = 0, cash = 100, change = 0;              //деньги

int sugar = 3, sug_plus = 1, sug_minus = -1;      //количество сахара; количество, на которое изменяется сахар при нажатии на кнопки +/-

int coffe_price = 30, cocoa_price = 20, tea_price = 10; //цена напитков

bool costenaugh = false;                    // флаг: достаточно ли средств на счету пользователя(credit)

bool cancel = false;                                // флаг: нажата ли кнопка "отмена"

mtype = {tea, cocoa, coffee};    // перечисление: типы напитков

mtype whatdrink = 0;                        // выбранный напиток

int s = 0;                                                  // номер состояния

int sug = 1;

chan ch = [6] of {int};

/*

s0 - начальное меню

s1 - регулировка сахара

s2 - выбор напитка

s3 - выдача сдачи

s4 - приготовление напитка

s5 - выдача напитка

s6 - нажатие кнопки "отмена"

*/


proctype S0()

{


s = 0; //начальное состояние s0

ch ! s; //отправка номера состояния в канал


printf("\nS0. Начальный экран."); //MSC:
```

```
        // если больше 1 позиции в конструкции "if" ИСТИННЫ -
        // недетерминированно выбирается любой из них
        // (чтобы обеспечить вариации действий реального пользователя)

        // Имитация выбора пользователя: нажата ли кнопка "Отмена"
        if
        :: (true) -> cancel = true; run S6(); //если кнопка нажата
        :: (true) -> cancel = false; //если кнопка не нажата
                printf ("MSC: \nВведите купюру! (ваш кредит: %d)", credit);
        s0:
                // Имитация выбора пользователя: номинал вводимой купюры
                if
                :: (true) -> cash = 1; credit = credit + cash; printf ("\n(ваш кредит: %d)", credit);
                :: (true) -> cash = 2; credit = credit + cash; printf ("\n(ваш кредит: %d)", credit);
                :: (true) -> cash = 5; credit = credit + cash; printf ("\n(ваш кредит: %d)", credit);
                :: (true) -> cash = 10; credit = credit + cash; printf ("\n(ваш кредит: %d)", credit);
                :: (true) -> cash = 50;     credit = credit + cash; printf ("\n(ваш кредит: %d)", credit);
                :: (true) -> cash = 100; credit = credit + cash; printf ("\n(ваш кредит: %d)", credit);
                fi
                if
                :: (credit >= tea_price || credit >= coffe_price || credit >= cocoa_price) -> costenaugh = true;
printf("\nCOSTENAUGH == %d, CREDIT = %d\n", costenaugh, credit); //при достаточном количестве
денег
                if
        :: (true) -> run S1(); // имитация выбора пользователя. Нажата кнопка регулировки сахара
                :: (true) -> run S2(); //нажата кнопка выбора напитка
                fi
                :: (credit < tea_price && credit < tea_price && credit < tea_price) -> costenaugh = false;
printf("\nCOSTENAUGH == %d, CREDIT = %d\n", costenaugh, credit);goto s0; // если денег
недостаточно, вернуться к метке s0 и заново запросить ввод денег
                fi
        fi
}
proctype S1()
```

```
{
	ch ? s; //чтение сообщения о сотоянии из канала
	s = 1;
	ch ! s; //отправка номера состояния в канал

	printf("\nS1. Количество сахара: %d/6", sugar);

	if
	:: (true) -> cancel = true; run S6(); //если нажата кнопка отмены
	:: (true) -> cancel = false; //если кнопка отмены не нажата

	s1:
		if
		:: (true) -> sug = sug_plus; sugar = sugar + sug; printf("\n+");    // если пользователь нажал
+
		:: (true) -> sug = sug_minus; sugar = sugar + sug; printf("\n-");  // если пользователь нажал
-
		fi
		if
		:: (sug == 1 && sugar == 6) -> sugar = 6; printf("\nMORE"); run S2(); //больше сахара
нельзя
		:: (sug == -1 && sugar == 0) -> sugar = 0; printf("\nLESS"); run S2(); //меньше сахара
нельзя
		:: (sugar >= 0 && sugar <= 6) -> /*если сахара в норме – имитация выброра
пользователя*/
			if
			:: (true) -> goto s1; //продолжать регулировать сахар
			:: (true) -> run S2(); //выбрать напиток
			fi
		fi
	fi
}
proctype S2()
{
```

```
	ch ? s; //чтение сообщения о сотоянии из канала
	s = 2;
	ch ! s; //отправка номера состояния в канал


	if
	:: (true) -> cancel = true; run S6(); // если нажата кнопка отмены
	:: (true) -> cancel = false; //если кнопка отмены не нажата
		// Имитация выбора пользователя: выбор напитка
		if
		:: (true) -> whatdrink = tea; // пользователь выбрал чай
		:: (true) -> whatdrink = cocoa; //пользователь выбрал какао
		:: (true) -> whatdrink = coffee; //пользователь выбрал кофе
		fi

		printf("\nS2. Вы выбрали: -%e-", whatdrink);

		//вычет цены из кредита
		if
		:: (whatdrink == tea) -> change = credit-tea_price; cancel = false; run S3(); // вычет цены
чая
		:: (whatdrink == cocoa) -> change = credit-cocoa_price; cancel = false; run S3(); // вычет
цены какао
		:: (whatdrink == coffee) -> change = credit-coffe_price; cancel = false;  run S3(); // вычет
цены кофе
		fi
	fi
}
proctype S3()
{
	ch ? s; //чтение сообщения о сотоянии из канала
	s = 3;
	ch ! s; //отправка номера состояния в канал
```

```
        printf("\nS3. Возьмите сдачу! (%d)", change);
        change = 0; //обнуление сдачи, так как выдача произошла

        run S4();
}
proctype S4()
{
        ch ? s; //отправка номера состояния в канал
        s = 4;
        ch ! s; //чтение номера состояния из канала

        printf("\nS4. Ваш напиток готовится");

        run S5();
}
proctype S5()
{
        ch ? s; //отправка номера состояния в канал
        s = 5;
        ch ! s; чтение номера состояния из канала

        printf("\nS5. Напиток готов. Возьмите ваш -%e-", whatdrink);
        if
                :: (whatdrink == tea) -> printf(" (чай) "); // русификация выбора. чай
                :: (whatdrink == cocoa) -> printf(" (какао) "); // русификация выбора. какао
                :: (whatdrink == coffee) -> printf(" (кофе) "); // русификация выбора. кофе
        fi
        printf("\n(сахар: %d/6)", sugar); // вывод информации о количестве сахара
}
proctype S6()
{
        ch ? s; //отправка номера состояния в канал
```

```
	s = 6;
	ch ! s; чтение номера состояния из канала
	printf ("\nS6. Отмена. Возьмите деньги (%d)", credit);
	// блок atomic, так как изменение данных переменных происходит в транзакции
	atomic
	{
		credit = 0;		// кредит (счёт) пользователя
		cash = 0;		// деньги, вводимые пользователем в текущий момент
		change = 0;	// сдача
		costenaugh = false; //достаточно ли денег на счету
		cancel = false;
		sugar = 3;
	}
	ch ? s;
}

init
{
	run S0(); //запуск процедуры s0; с init начинается программа
}



ltl f1 { s == 0 U (s == 1 || s == 2 || s == 6) }
ltl f2 { [](len(ch) == 1 || len(ch) == 0) }
ltl f3 { [](sugar >= 0 && sugar <= 6) }
ltl f4 { [](whatdrink == 0 || whatdrink == tea || whatdrink == cocoa || whatdrink == coffee) }
ltl f5 { [](s >= 3 -> cancel == false) }
ltl f6 { [](s == 6 -> <>(s == 0)) }
ltl f7 { []((credit < tea_price && credit < tea_price && credit < tea_price) -> costenaugh == false) }
```

ПРИЛОЖЕНИЕ Б

РЕЗУЛЬТАТ ВЕРИФИКАЦИИ ТРЕБОВАНИЯ, ОПИСАННОГО LTL-ФОРМУЛОЙ F7

```
spin -a  coffe.pml
ltl f1: ((s==0)) U ((((s==1)) || ((s==2))) || ((s==6)))
ltl f2: [] (((len(ch)==1)) || ((len(ch)==0)))
ltl f3: [] (((sugar>=0)) && ((sugar<=6)))
ltl f4: [] (((((whatdrink==0)) || ((whatdrink==tea))) || ((whatdrink==cocoa))) || ((whatdrink==coffee)))
ltl f5: [] ((! ((s>=3))) || ((cancel==0)))
ltl f6: [] ((! ((s==6))) || (<> ((s==0))))
ltl f7: [] ((! ((((credit<tea_price)) && ((credit<tea_price))) && ((credit<tea_price)))) || ((costenaugh==0)))
  the model contains 7 never claims: f7, f6, f5, f4, f3, f2, f1
  only one claim is used in a verification run
  choose which one with ./pan -a -N name (defaults to -N f1)
  or use e.g.: spin -search -ltl f1 coffe.pml
gcc -DMEMLIM=1024 -O2 -DXUSAFE -w -o pan pan.c
./pan -m10000  -a -c1 -N f7
Pid: 18740
pan: ltl formula f7


(Spin Version 6.4.5 -- 1 January 2016)
        + Partial Order Reduction


Full statespace search for:
        never claim             + (f7)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)
State-vector 156 byte, depth reached 369, errors: 0
   60165 states, stored
    2739 states, matched
   62904 transitions (= stored+matched)
```

```
      0 atomic steps
hash conflicts:       3 (resolved)

Stats on memory usage (in Megabytes):
  10.558      equivalent memory usage for states (stored*(State-vector + overhead))
   7.401      actual memory usage for states (compression: 70.10%)
        state-vector as stored = 101 byte + 28 byte overhead
 128.000      memory used for hash table (-w24)
   0.534      memory used for DFS stack (-m10000)
 135.858      total actual memory usage

unreached in proctype S0
        (0 of 53 states)
unreached in proctype S1
        (0 of 39 states)
unreached in proctype S2
        (0 of 34 states)
unreached in proctype S3
        (0 of 7 states)
unreached in proctype S4
        (0 of 6 states)
unreached in proctype S5
        (0 of 15 states)
unreached in proctype S6
        (0 of 12 states)
unreached in init
        (0 of 2 states)
unreached in claim f7
        _spin_nvr.tmp:65, state 10, "-end-"
        (1 of 10 states)
pan: elapsed time 0.38 seconds
No errors found -- did you verify all claims?
```

ПРИЛОЖЕНИЕ В

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ТЕМЕ: «ВЕРИФИКАЦИЯ АВТОМАТНЫХ ПРОГРАММ В ВЕРИФИКАТОРЕ SPIN С ПОМОЩЬЮ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА GUI ISPIN»

Кофейный автомат – это устройство, взаимодействующее с пользователем, которое должно выполнять следующие функции: прием денег, выбор напитка, регулирование количества сахара, выдача сдачи, приготовление и выдача напитка.

Начальное состояние данной системы предлагает ввести деньги для последующей покупки напитка. Затем можно отрегулировать количество сахара (некоторое количества сахара задано по умолчанию) или вернуть деньги (если принято решение отмены приготовления напитка). После этого следует выбрать напиток. После выбора напитка автомат выдаёт сдачу, если она есть, и начинает приготовление напитка. После того, как напиток готов, автомат выдаёт напиток, и возвращается в начальное состояние.

На рисунке В.1 представлен граф переходов данного автомата.

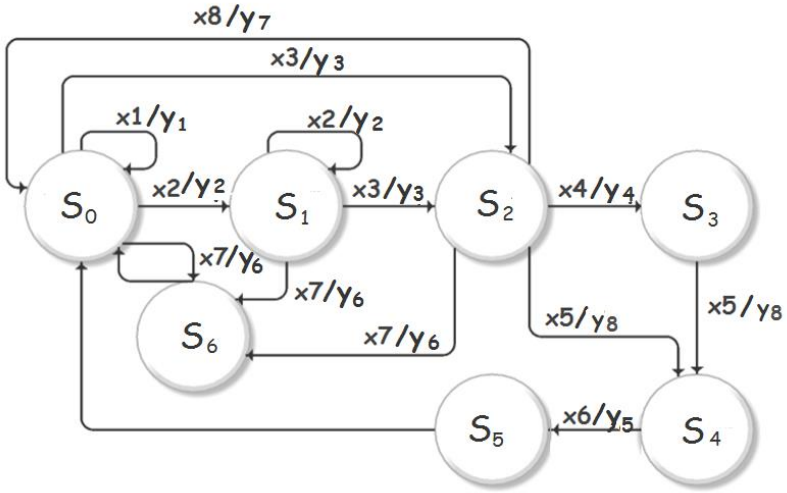

Рисунок В.1 - Граф переходов автомата «Кофейный автомат»

Входные сигналы автомата:

- X1 – введена купюра;
- X2 – нажата кнопка «+» или «-» (регулировка сахара);

- X3 – нажата кнопка выбора напитка;
- X4 – выдать сдачу (сдача больше 0);
- X5 – сдача равна 0 (сдача выдана);
- X6 – напиток готов;
- X7 – нажата кнопка «Возврат денег»;
- X8 – денег на счету меньше, чем стоимость напитка

Выходные сигналы автомата:

- Y1 – вывести сообщение «Введите купюру»;
- Y2 – перейти к регулировке сахара
- Y3 – вывести название выбранного напитка;
- Y4 – вывести сообщение «Возьмите сдачу»; вернуть сдачу;
- Y5 – вывести сообщение «Возьмите напиток»; выдать напиток;
- Y6 – вернуть деньги;
- Y7 – вывести сообщение «Недостаточно денег»;
- Y8 – вывести сообщение: «Идет приготовление напитка»

Состояния автомата:

- S0 – начальный экран;
- S1 – регулировка количества сахара;
- S2 – выбор напитка;
- S3 – выдача сдачи;
- S4 – приготовление напитка;
- S5 – выдача напитка;
- S6 – выдача остатка денег

Основные требования к модели системы «Кофейный автомат» были сформулированы следующим образом и представлены в виде формул линейной темпоральной логики LTL (логика, в которой учитывается временной аспект):

- «Начальный экран» (7). В момент времени t_0 (начало цикла работы кофейного автомата) показан начальный экран. Начальный экран показывается сколь угодно долго до того, как нажата кнопка, переводящая автомат в состояние приготовления напитка (регулировка сахара или выбор напитка);

- В любой момент времени на экране кофейного автомата показывается только одно (текущее) состояние (8);
- «Регулировка количества сахара» (9). Количество сахара не может быть меньше 0 и больше максимальной нормы на порцию напитка;
- «Выбор напитка» (10). В один цикл работы автомата можно выбрать только один напиток;
- После выбора и оплаты напитка пользователь не может отменить покупку (11);
- После выдачи остатка денег пользователю, автомат переходит в начальное состояние («Начальный экран») (12);
- Автомат не может приготовить выбранный напиток, если на счету нет достаточного количества денег для оплаты (13).

После разработки модели системы и определения требований к ней, проводится формальная верификация на основе метода model checking (проверка модели).

Формальная верификация – это метод проверки систем, базирующийся на математическом доказательстве их корректности, - т.е. предполагает под собой формальное доказательство на абстрактной математической модели системы

Один из подходов к формальной верификации – метод проверки модели (model checking). Основная идея данного метода заключается в проверке корректности систем с помощью запуска алгоритмов, исполняемых компьютером. При использовании данного подхода пользователь вводит описание модели системы (возможное поведение, описанное при помощи конечного автомата) и описание спецификации требований (желаемое поведение), а верификацию проводит специальное инструментальное средство - верификатор. Если ошибка найдена, верификатор предоставляет контрпример, который показывает при каких путях исполнения программы обнаруживается ошибка. Контрпример представляет собой сценарий, в котором модель ведет себя нежелательным образом - т.е. свидетельствует о том, что построенная модель неверна и должна быть исправлена.

Методом проверки модели могут быть эффективно верифицированы автоматные программы, созданные на основе switch-технологии (способ программирования, при котором ветвления в программе создаются с помощью оператора ветвления), т.к. в таких программах управляющие состояния явно выделены, а их количество обозримо.

Таким образом, для начала проведения верификации системы, необходимо перевести граф переходов автомата системы «Кофейный автомат» на язык программирования, понятный для верификатора. В данном случае, для используемого в работе верификатора SPIN таким языком является язык Promela.

Т.к. кодирование графа переходов системы «Кофейный автомат» является автоматным подходом к программированию, то каждое состояние автомата – это отдельная процедура (рис. В.2).

```
21   proctype S0()
22   { ···
57   }
58   proctype S1()
59   { ···
85   }
86   proctype S2()
87   { ···
111  }
112  proctype S3()
113  { ···
122  }
123  proctype S4()
124  { ···
132  }
133  proctype S5()
134  { ···
148  }
149  proctype S6()
150  { ···
169  }
170
171
172  init
173  {
174      run S0();
175  }
```

Рисунок В.2 – Кодирование состояний автомата

В языке Promela выполнение программы начинается с процедуры init, поэтому в данном случае, в init содержится единственная операция – вызов процедуры S0() (аналогична состоянию S_0).

Процессы в процедурах S1 – S6 описаны согласно графу переходов разработанного ранее автомата (рис. В.1). Процессы вызывают друг друга (рис. В.3, для наглядности вызов процедур показан красными стрелками), что обеспечивает переходы между состояниями автомата.

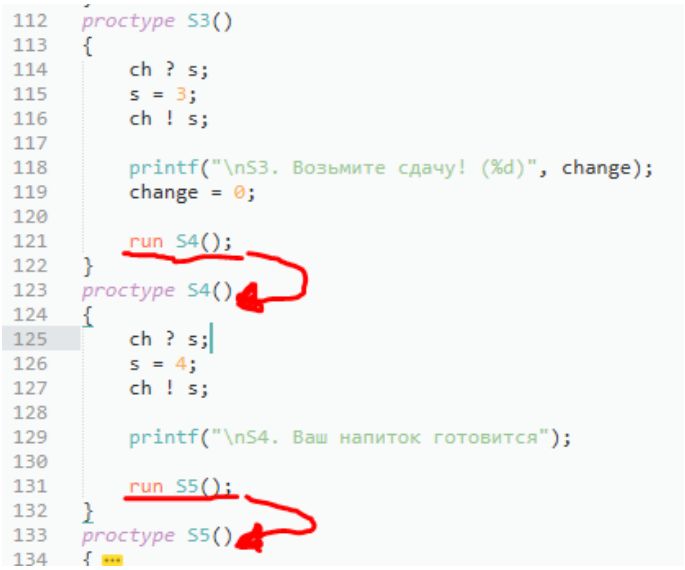

Рисунок В.3 – Взаимный вызов процедур в программе

Все необходимые переменные (включая каналы и перечисления), общие для всей системы описаны глобально, что обеспечивает доступ к ним из любой процедуры.

Требования к системе также необходимо перевести на язык Promela (рис. В.4).

Они описываются после всех процедур, и имеют следующий синтаксис: *ltl название_свойства { LTL-формула }*.

```
ltl f1 { s == 0 U (s == 1 || s == 2 || s == 6) }
ltl f2 { [](len(ch) == 1 || len(ch) == 0) }
ltl f3 { [](sugar >= 0 && sugar <= 6) }
ltl f4 { [](whatdrink == 0 || whatdrink == tea || whatdrink == cocoa || whatdrink == coffee) }
ltl f5 { [](s >= 3 -> cancel == false) }
ltl f6 { [](s == 6 -> <>(s == 0)) }
ltl f7 { []((credit < tea_price && credit < tea_price && credit < tea_price) -> costenaugh == false) }
```

Рисунок В.4 – Требования к системе на языке Promela

Полный листинг данной программы приведён в приложении А.

Верификацию системы можно проводить как в консольном режиме, так и с помощью разнообразных графических интерфейсов. Второй вариант намного

удобнее и понятней пользователю, поэтому верификация проводится в одном из доступных графических интерфейсов – GUI iSpin.

Данный интерфейс работает только с сохранённой версией программы, поэтому сначала необходимо загрузить исходный код с помощью кнопки Open вкладки «Edit/View» (рис. В.5).

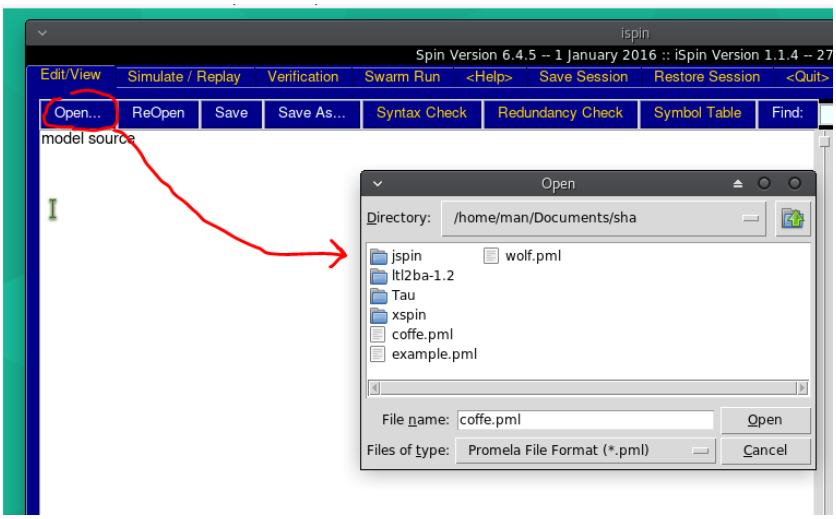

Рисунок В.5 – Загрузка исходного кода программы

После загрузки исходного кода, в данной вкладке «Edit/View» можно редактировать код в окне редактора (без подсветки синтаксиса; удобнее работать с кодом в редакторе Sublime), сохранять, открывать заново, проверять код на наличие синтаксических ошибок. Все действия пользователя отображаются в консоли внизу экрана. В данном случае, ошибок в программе нет, но есть 7 формул, которые необходимо проверить (рис. В.6).

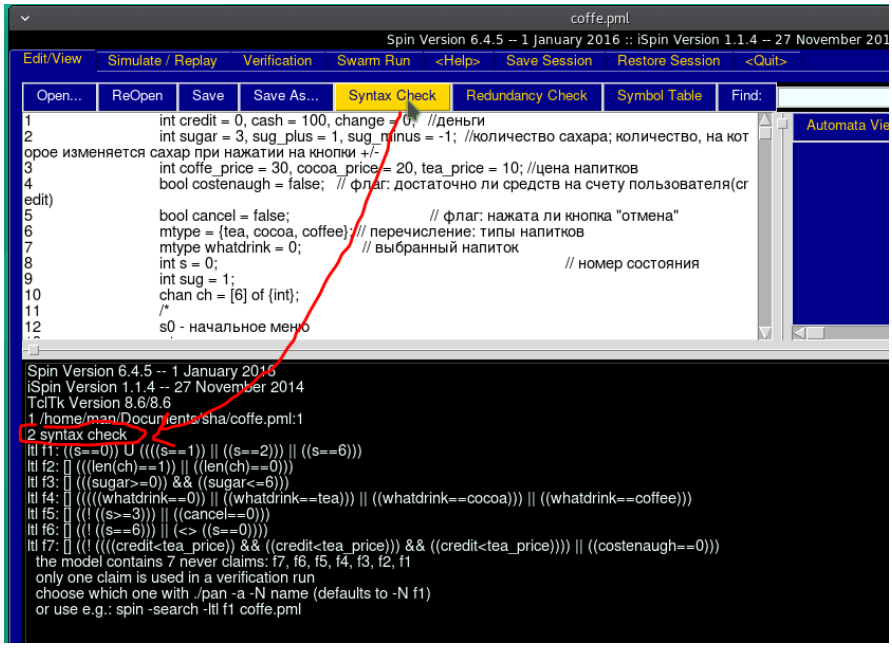

Рисунок В.6 – Проверка на синтаксические ошибки

Также, во вкладке «Edit/View» доступен просмотр графов переходов (окно «Automata view»), построенных по загруженной программе. Каждый автомат представляет собой либо процедуру, либо LTL-формулу, при чём SPIN сам размечает состояния. Так, на рисунке В.7 представлен граф переходов внутри процедуры S0.

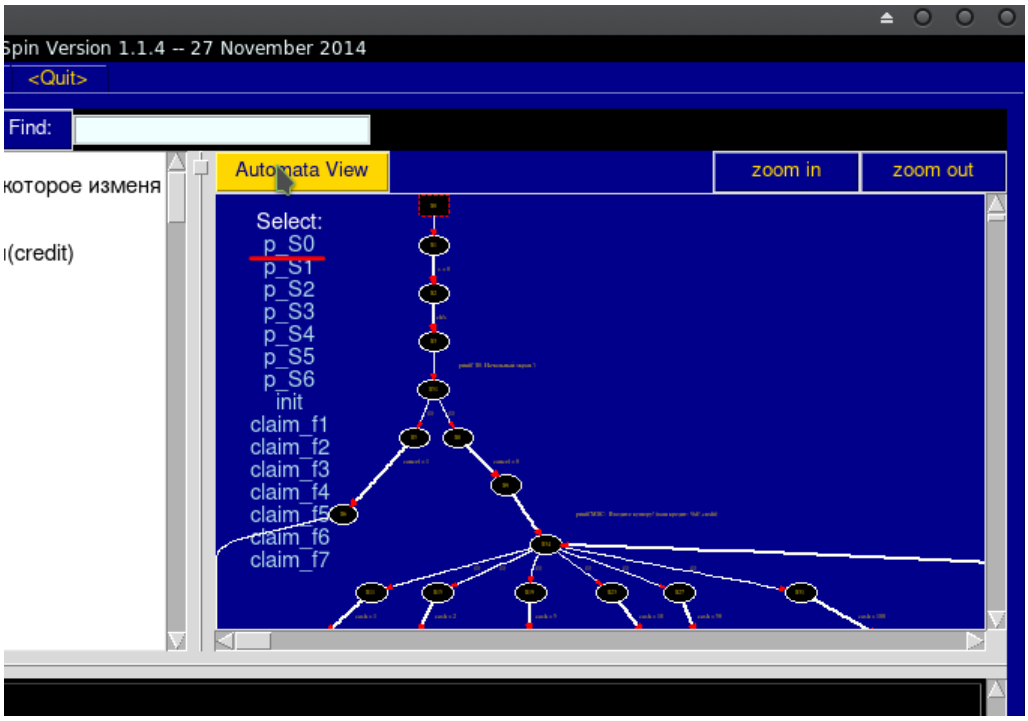

Рисунок В.7 – Граф переходов в процедуре S0

На вкладке «Simulate/Replay» можно запустить процесс симуляции – запуск одного из возможных путей выполнения программы. Симуляция может происходить в трёх режимах: случайный (выбор исполняемой трассы происходит случайно), интерактивный (выбор исполняемой трассы предоставляется пользователю пошагово) и из сгенерированного файла (заранее известная трасса, записанная в файл с расширением *.trail).

На рисунке В.8 показан пример случайной симуляции. Причём, т.к. в программе есть переменная типа chan (канал), то на диаграмме взаимодействия (в окне слева) показан обмен сообщениями в канале.

В нижней части окна представлены данные о симуляции – значение переменных, порядок выполнения трассы и значение переменных в канале.

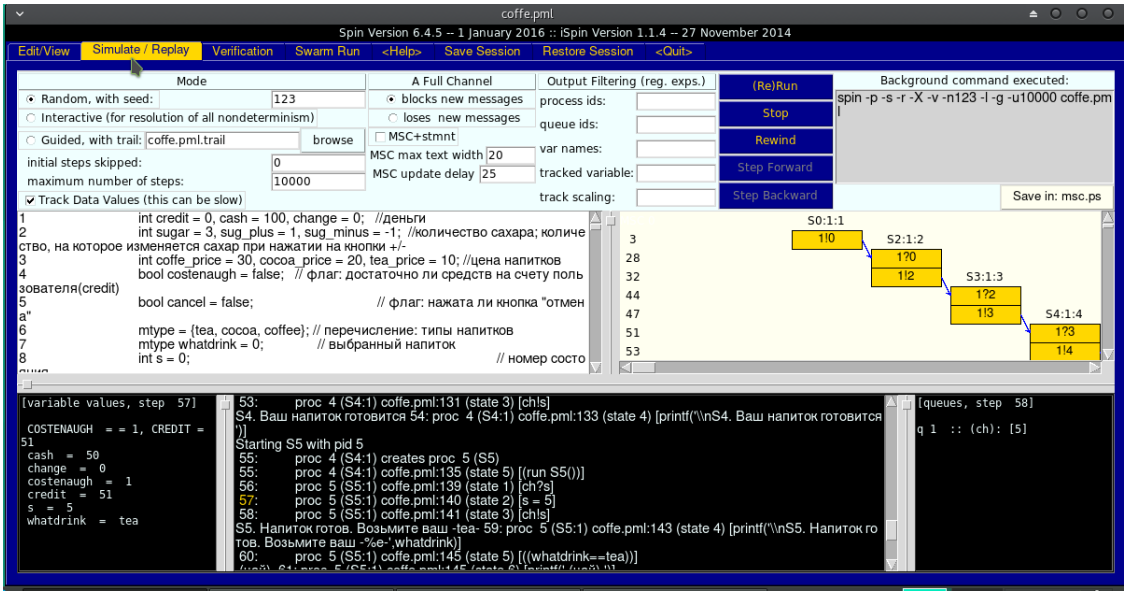

Рисунок В.8 – Режим симуляции

Для проведения верификации необходимо перейти во вкладку «Verification».

Систему можно проверить как в целом по свойствам безопасности, живости и пр., так и проверить отдельные свойства (требования к системе). В iSpin существует возможность выбора разнообразных настроек для проверки свойств, включая и расширенные параметры (рис. В.9, красным обведены расширенные параметры, появляющиеся/скрывающиеся при нажатии соответствующих кнопок).

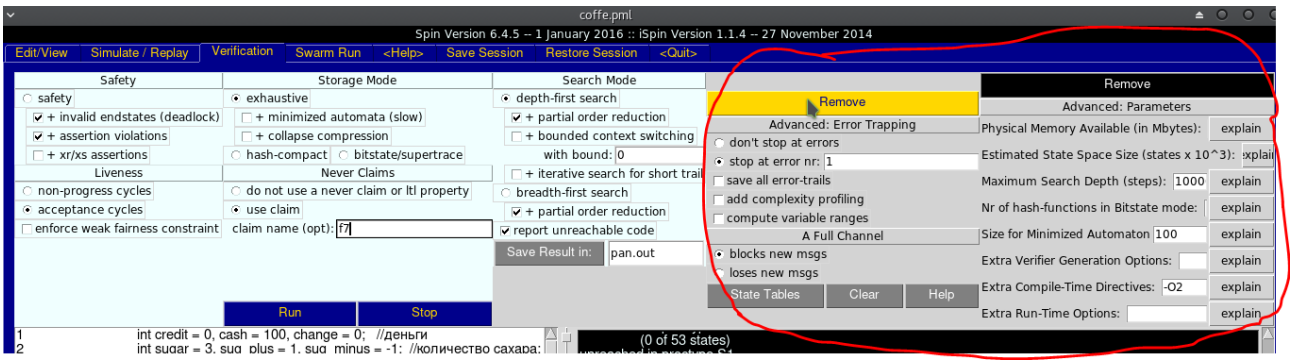

Рисунок В.9 – Параметры для верификации в iSpin

Свойства в iSpin (и в SPIN соответственно) верифицируются каждое в отдельности (т.е. по одному за раз). Так, для верификации конкретного свойства, необходимо в группе «Never claims» (в окне «Verification») указать опцию Use claim, а в поле ввода ввести название формулы, описанной в коде программы на языке Promela. Результат верификации будет доступен в правой части окна. В случае, представленном на рисунке В.10, формула f7 успешно прошла верификацию.

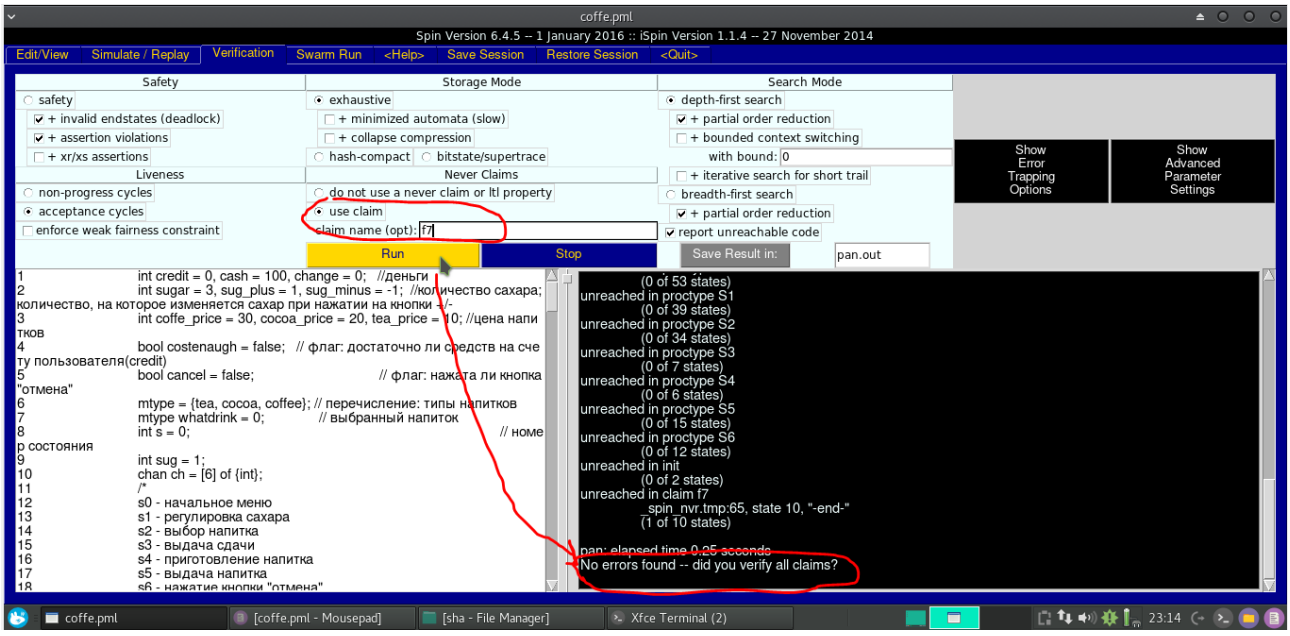

Рисунок В.10 – Верификация свойства f7

Допустим ошибку в программе: в процедуре S1() изменим цикл таким образом, чтобы значение переменной sugar выходило за установленный требованиями диапазон [0..6] (рис. В.11 – в структуре if убраны ограничения при изменении количества сахара).

```
68      :: (true) -> cancel = false;
69
70      s1:
71          if
72          :: (true) -> sug = sug_plus; sugar = sugar + sug; printf("\n+");    // если пользователь нажал +
73          :: (true) -> sug = sug_minus; sugar = sugar + sug; printf("\n-");   // если пользователь нажал -
74          fi
75          if
76          :: (sug == 1 && sugar == 6) -> sugar = 6; printf("\nMORE"); run S2();
77          :: (sug == -1 && sugar == 0) -> sugar = 0; printf("\nLESS"); run S2();
78          :: (sugar >= 0 && sugar <= 6) ->
79              if
80              :: (true) -> goto s1;
81              :: (true) -> run S2();
82              fi
83          fi
```

Рисунок В.11 – Фрагмент кода программы с допущенной ошибкой

В этом случае, верифкатор сообщит об ошибке (рис. В.12) и предложит посмотреть построенный контрпример в режиме симуляции, доказывающий, что формула не выполняется на найденном пути.

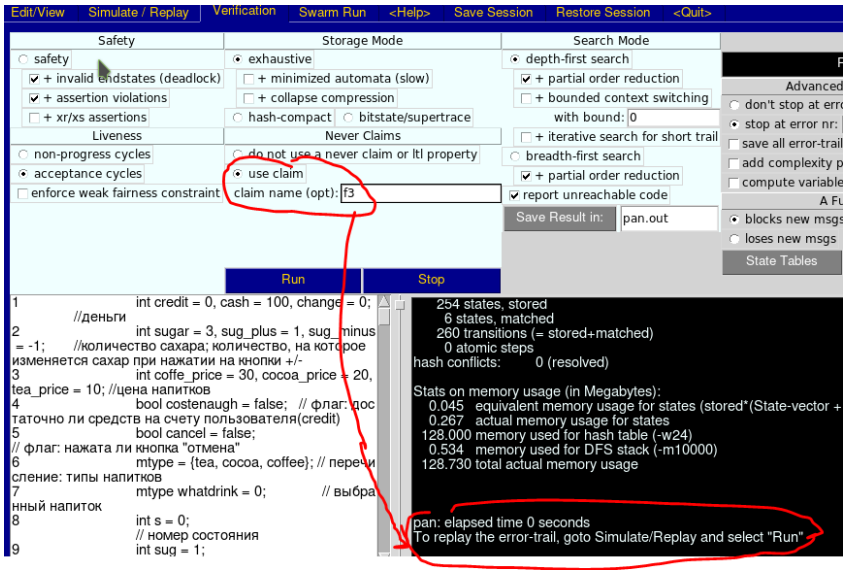

Рисунок В.12 – Верификация свойства f3

На рисунке В.13 показан найденный контрпример в режиме симуляции при проверке формулы f3.

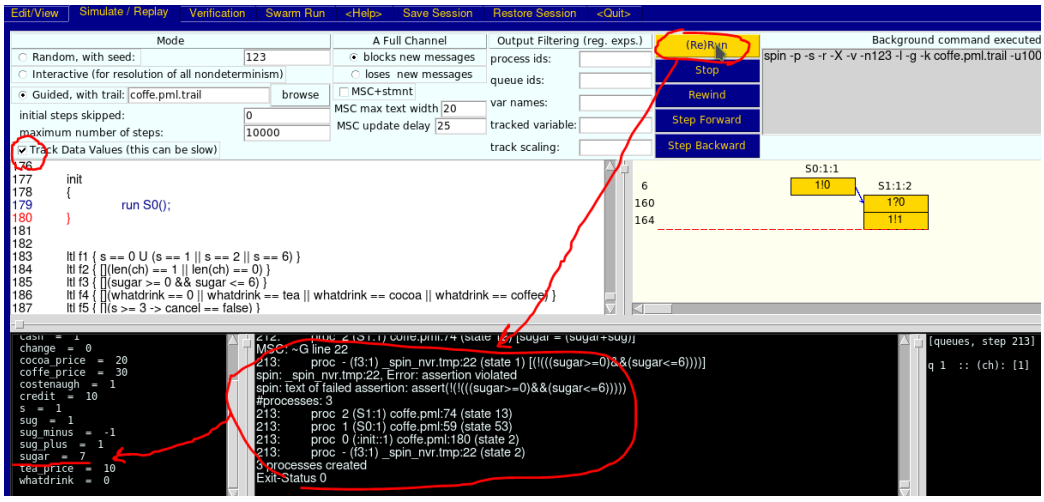

Рисунок В.13 – Контрпример для свойства f3

Так, при запуске симуляции (кнопка «(Re)Run»), отметив флажок «Track Data Values» можно заметить, что переменная sugar стала равна 7, хотя формулой f3 её значение определяется от 0 до 6. В нижнем окне, расположенном по центру, показаны шаги, при соблюдении которых появляется данная ошибка. Теперь, мы видим, что ошибка возникла из-за достижения переменной sugar значения 7 (т.е. выхода из диапазона [0..6], описанного в LTL-формуле f3) в процедуре S1(). Подробные шаги выполнения можно увидеть, нажав на кнопки «Step Forward» / «Step Backward» («Шаг вперёд» / «Шаг назад» соответственно) После устранения ошибки формула успешно проходит верификацию (рис. В.14)

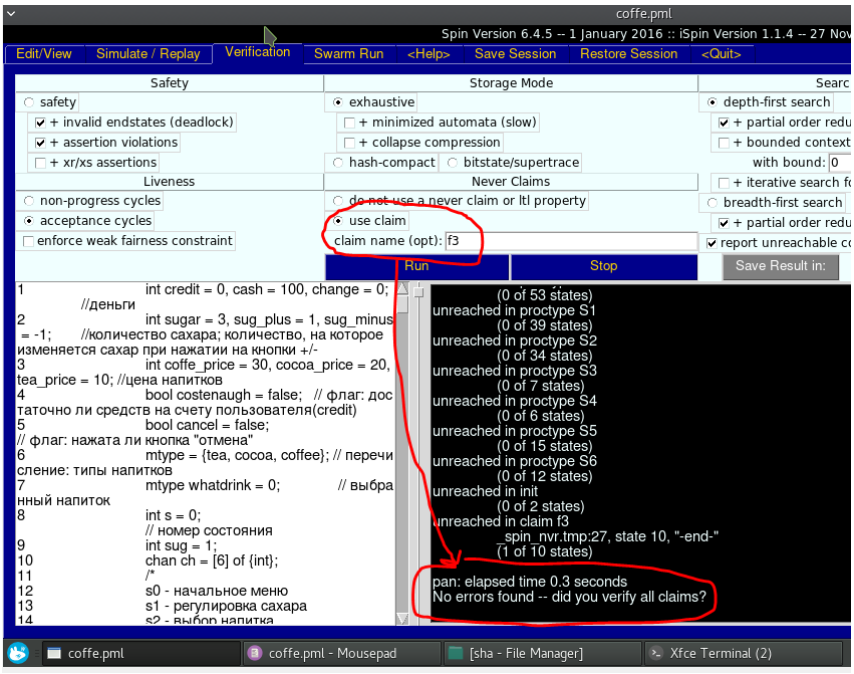

Рисунок В.14 – Успешная верификация свойства f3 после устранения ошибки