Московский государственный технический университет
им. Н.Э. Баумана

На правах рукописи

Буренков Владимир Сергеевич

МЕТОДЫ И СРЕДСТВА ВЕРИФИКАЦИИ ПРОТОКОЛОВ КОГЕРЕНТНОСТИ ПАМЯТИ

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

Диссертация
на соискание ученой степени
кандидата технических наук

Научный руководитель:
кандидат технических наук, доцент
Иванов Сергей Ростиславович

Москва – 2017

Оглавление

Введение

Актуальность темы исследования.

Мультипроцессоры с общей памятью составляют один из базовых классов высокопроизводительных вычислительных систем. В последнее время системы, относящиеся к данному классу, получили развитие в форме многоядерных микропроцессоров, объединяющих несколько процессоров (ядер) на одном кристалле. Характерно, что количество ядер таких микропроцессоров постоянно увеличивается. Разработкой многоядерных микропроцессоров и мультипроцессорных комплексов занимаются как зарубежные (в частности, IBM, Intel и AMD), так и российские компании (в частности, АО «МЦСТ» и ПАО «ИНЭУМ им. И. С. Брука»).

Основной проблемой, возникающей при создании мультипроцессоров с общей памятью, является обеспечение согласованного (когерентного) состояния памяти. Каждое ядро имеет в своем составе локальную кэш-память, из-за чего в системе могут сосуществовать несколько копий одних и тех же данных: одна копия в основной памяти и несколько копий в кэш-памяти процессоров. При изменении какой-либо копии другие копии должны быть либо удалены, либо изменены согласованным образом. За это отвечают так называемые контроллеры когерентности – устройства подсистемы памяти, объединенные в сеть и взаимодействующие друг с другом по специальному протоколу – *протоколу когерентности*.

Разработка механизмов, отвечающих за когерентность памяти, осуществляется в два этапа: проектирование протокола когерентности и реализация протокола в аппаратуре. Ввиду сложности современных протоколов на обоих этапах возможны ошибки. Ошибки в протоколе когерентности особенно критичны и должны быть выявлены до начала реализации подсистемы памяти. Пропуск ошибок в протоколе может привести к ошибкам в реализации, которые не будут найдены до выпуска микропроцессора, и проявят

себя только во время работы выпущенного микропроцессора.

Актуальность разработки методов верификации протоколов когерентности памяти отмечается многими учеными, в число которых входят Э. М. Кларк, Э. А. Эмерсон, А. Пнуэли, Д. Пелед, Л. Лэмпорт, С. Граф, К. МакМиллан, О. Грамберг, П. А. Абдулла, С. Парк, Д. Л. Дилл, М. Талупур, И. В. Коннов и др. В их работах большое внимание уделяется разработке формальных методов, которые позволяют получить математическое доказательство соответствия модели верифицируемого протокола когерентности его спецификации, то есть набору свойств, которым он должен удовлетворять. Несмотря на большое количество исследований в данной области, вопросу верификации протоколов когерентности памяти продолжает уделяться повышенное внимание.

Наиболее известные формальные методы имеют ограниченное применение для верификации протоколов когерентности памяти. Полностью автоматизируемый метод проверки моделей подвержен проблеме комбинаторного взрыва числа состояний и позволяет верифицировать только протоколы систем с четырьмя (или менее) ядрами. Масштабируемый метод доказательства теорем требует чрезмерного объема ручной работы.

В связи с потребностью в верификации систем с большим количеством ядер, в данной работе задача верификации протоколов когерентности памяти поставлена как задача параметризованной верификации, позволяющей исследовать системы с любым числом ядер. Существующими методами решения задачи параметризованной верификации протоколов когерентности памяти являются методы абстракции, методы, основанные на поиске сетевых инвариантов, полные методы редукции, методы композиционной проверки моделей, методы, основанные на поиске инвариантов. По результатам выполненного в диссертации аналитического обзора работ, в которых представлены методы из этих групп, установлено, что данные методы обладают

существенными недостатками, что также определяет невозможность их непосредственного применения для решения задачи верификации протоколов когерентности памяти промышленно разрабатываемых микропроцессоров. Данное обстоятельство определяет чрезвычайную актуальность разработки новых методов и средств верификации протоколов когерентности памяти.

Цель работы – разработка новых методов и средств верификации протоколов когерентности памяти масштабируемых промышленно разрабатываемых микропроцессорных систем.

Решаемые в работе задачи.

Для достижения цели работы были поставлены следующие задачи:

1. Провести анализ существующих методов верификации протоколов когерентности памяти.
2. Определить совокупность математических объектов, необходимых для представления протоколов когерентности памяти и разработать модель протоколов когерентности памяти.
3. Разработать метод верификации протоколов когерентности памяти, который являлся бы масштабируемым и обеспечивал бы высокий уровень автоматизации процесса проверки.
4. Доказать корректность разработанного метода с использованием математической модели протоколов когерентности памяти.
5. Разработать программный инструмент, позволяющий осуществлять верификацию протоколов когерентности с использованием разработанного метода.
6. Провести экспериментальные исследования предложенного метода в применении к микропроцессору Эльбрус-4С.

Научная новизна работы.

Научной новизной обладают следующие результаты работы:

1. Предложенный автором оригинальный метод построения абстрактных

формальных моделей протоколов когерентности памяти, основанный на синтаксических преобразованиях Promela-моделей, позволяющий существенно сократить пространство исследуемых состояний.

2. Сформулированная и доказанная теорема о сохранении абстрактными преобразованиями свойств-инвариантов, определяющая корректность предложенного метода верификации.

3. Разработанные алгоритмы преобразования дерева абстрактного синтаксиса, которое является промежуточным представлением Promela-моделей, позволяющие автоматически выполнять предложенные преобразования моделей.

Теоретическая и практическая значимость работы.

Теоретическая значимость работы заключается в том, что:

1. Разработана многоуровневая математическая модель протоколов когерентности памяти. Уровнями данной модели являются канальная система, граф процесса и система переходов. Модель позволяет установить связь между синтаксическими преобразованиями моделей на языке Promela и результирующими преобразованиями соответствующих систем переходов.

2. Сформулирована и доказана теорема о сохранении абстрактными преобразованиями свойств-инвариантов.

3. Доказана корректность предложенной процедуры уточнения абстрактных моделей, которая используется для устранения ложных сообщений об ошибках.

Практическая значимость работы:

1. Предложен подход к разработке Promela-процессов, моделирующих кэш-контроллеры и координирующий их работу системный коммутатор. Подход позволяет естественным образом описывать модели протоколов когерентности, представленные множествами взаимодействующих конечных автоматов, на языке Promela.

2. Разработан программный инструмент, позволяющий с помощью операций над деревом абстрактного синтаксиса, которое является промежуточным представлением исходной модели, автоматизировать преобразования Promela-моделей.

3. Разработана методика верификации протоколов когерентности памяти, включающая весь процесс от создания формальных моделей протоколов когерентности памяти до их параметризованной верификации.

Разработанные методы и средства реализованы в программной системе, с помощью которой проведена верификация протокола когерентности 16-ядерной системы из микропроцессоров Эльбрус-4С, разработанной в АО «МЦСТ». Кроме того эти результаты могут найти применение в разработке других современных многоядерных микропроцессоров.

Методология и методы исследований.

Результаты диссертационной работы получены на базе использования методов и моделей, применяемых для проведения формальной верификации. Применен формализованный подход, предполагающий использование множества математических объектов для представления верифицируемого объекта.

Математическую основу составляют теория множеств, теория графов, теория алгоритмов, математическая логика, теория формальных языков и теория автоматов.

Положения, выносимые на защиту.

1. Метод верификации протоколов когерентности кэш-памяти, позволяющий проводить верификацию протоколов когерентности современных мультипроцессорных систем с любым количеством ядер, и основанный на предложенных синтаксических преобразованиях моделей, написанных на языке Promela, и предложенной процедуре уточнения получаемых в результате применения преобразований абстрактных моделей.

2. Программный инструмент, автоматизирующий процесс преобразования Promela-моделей к виду, позволяющему проводить верификацию при помощи инструмента Spin.

Степень достоверности и апробация результатов.

Достоверность научных положений обеспечена их строгим математическим обоснованием:

- разработана модель в виде канальной системы, позволяющая корректно формализовать семантику Promela-моделей протоколов когерентности памяти;
- сформулирована и доказана теорема, определяющая корректность предложенного метода верификации.

Основные положения работы обсуждались на:

1. VII Всероссийской молодежной научно-инженерной выставке «Политехника» в МГТУ им. Н.Э. Баумана (г. Москва, 2012 г.).
2. 9-м международном коллоквиуме молодых ученых по программной инженерии SYRCoSE (г. Самара, 2015 г.).
3. 10-м международном коллоквиуме молодых ученых по программной инженерии SYRCoSE (г. Москва, 2016 г.).
4. 7-й Всероссийской научно-технической конференции «Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС-2016)» (г. Москва, 2016 г.).
5. 14-м международном научно-технологическом симпозиуме IEEE East-West Design and Test (EWDTS-2016) (г. Ереван, 2016 г.).
6. Всероссийском конкурсе научно-исследовательских работ в области инженерных и гуманитарных наук, проводимом в рамках Всероссийского инновационного молодежного научно-инженерного форума «Политехника». По итогам конкурса автор диссертации объявлен победителем и награжден дипломом первой степени.

Все основные теоретические и практические результаты работы в виде метода, алгоритмов, методики и программных средств внедрены в процесс проектирования микропроцессоров. Они использованы при выполнении опытно-конструкторских работ по темам «Экскурсовод-2», «Процессор-1» и «Процессор-9» в АО «МЦСТ», о чем имеется соответствующий акт о внедрении.

Публикации. По теме диссертации опубликовано 16 научных работ [1–13, 28–30], в том числе 8 научных статей [1, 2, 4, 7, 10, 13, 29, 30] в рецензируемых журналах, входящих в перечень рекомендованных ВАК РФ. В работе [3] лично автором диссертации описан подход к верификации с помощью метода проверки моделей. В работах [12, 13] лично автором диссертации обозначены и рассмотрены проблемы верификации протоколов когерентности памяти и пути их решения применительно с протоколу Эльбрус-4С. В работах [7, 29] представлен метод верификации, разработанный лично автором диссертации. В работе [28] представлены результаты экспериментов, проведенных лично автором диссертации.

Личный вклад автора. Все представленные в диссертации результаты получены лично автором.

Объем и структура работы. Диссертация состоит из введения, четырех глав, заключения и списка литературы, занимающих 161 страницу. Список литературы включает 122 наименования.

1 Анализ методов и средств верификации протоколов когерентности памяти и постановка задачи

1.1 Проблема когерентности и протоколы когерентности

Развитие микропроцессорных систем характеризуется увеличением числа процессорных ядер. Широкий класс современных вычислительных систем составляют мультипроцессорные системы, полученные объединением нескольких многоядерных микропроцессоров, каждый из которых имеет доступ к общему адресному пространству и при этом физически обладает собственной памятью [53, 44]. Многоядерные микропроцессоры, в свою очередь, состоят из нескольких процессорных элементов (ядер), расположенных на одном кристалле. Время доступа к различным участкам памяти в таких вычислительных системах неодинаково, и для сокращения задержки на обращения к памяти вычислительные ядра снабжаются локальной кэш-памятью, которая во многих случаях предотвращает необходимость доступа к совместно используемой основной памяти. При наличии локальной кэш-памяти в нескольких кэшах могут располагаться копии данных, расположенных в основной памяти по одному и тому же адресу. В случае модификации некоторым ядром данных, расположенных по этому адресу, необходима поддержка согласованности состояния всех копий. Согласованное состояние блоков кэш-памяти различных ядер называется *когерентным*.

Все результаты данной работы применены для верификации 16-ядерной микропроцессорной системы, состоящей из четырех 4-ядерных микропроцессоров Эльбрус-4С, разрабатываемых в АО «МЦСТ».

В публикации [108] введены следующие инварианты, которым должна удовлетворять когерентная система:

- SWMR (один записывающий, много считывающих, single-writer, multiple-read). Для каждого адреса памяти A в каждый момент логического

времени, только один процессор может производить операцию записи по адресу A (а также и операцию считывания), либо некоторое количество процессоров может осуществлять только считывание значения по адресу A. Таким образом, время нахождения каждого блока в кэше разделяется на периоды, в течение каждого из которых либо один процессор имеет доступ по считыванию/записи к этому блоку, либо некоторое число процессоров (возможно, нуль) имеют доступ только по считыванию.

- Инвариант значения данных. Значение данных, хранимое по адресу A, в начале каждого периода является таким же, как и значение по этому адресу, которое было в конце последнего периода с процессорным доступом по считыванию/записи.

В решении проблемы когерентности выделяются два подхода: программный и аппаратный. Ввиду преимуществ в производительности, аппаратные механизмы, реализующие *протоколы когерентности* кэш-памяти [108], нашли наиболее широкое применение. Протоколы когерентности определяют правила взаимодействия устройств распределенной системы.

Цель протоколов когерентности – поддерживать когерентность памяти путем сохранения инвариантов SWMR и значения данных. Для реализации такой поддержки с каждым устройством хранения данных – с каждой кэш-памятью и основной памятью – связывается контроллер поддержания когерентности – конечный автомат. Контроллеры, связанные с кэш-памятью, называются *кэш-контроллерами*. В распределенной системе имеется набор таких контроллеров, и контроллеры обмениваются друг с другом сообщениями, типы которых определяются протоколом когерентности. Обмен сообщениями происходит под управлением координирующего устройства – *системного коммутатора*.

Постоянно растущий уровень сложности мультипроцессорных систем с разделяемой памятью находит свое отражение и в замысловатости протоколов

когерентности этих систем. Это, в свою очередь, определяет высокую сложность задачи проверки корректности протоколов (верификации). В то же время, в связи с колоссальным количеством вариантов взаимодействия контроллеров, полностью предусмотреть которое людям не под силу, высока вероятность совершения ошибок в процессе проектирования протоколов.

Протокол когерентности рассматривает состояние одной строки кэш-памяти и не затрагивает взаимодействие между операциями над различными кэш-блоками. Обращения к памяти, помимо их подразделения на считывание (load) и запись (store), характеризуются *типом памяти*, определяющим, как они будут обработаны в системе. Основными типами памяти, представленными в микропроцессоре Эльбрус-4С, являются тип памяти write back, определяющий режим отложенной записи в память, и тип памяти write through, определяющий режим сквозной записи. Во многом они схожи с одноименными типами памяти, определенными в документации [67], однако имеются некоторые отличия.

Работу протокола когерентности процессора Эльбрус-4С для обращений с типом памяти write back можно кратко описать следующим образом (рисунок 1). Получив от ядра исходную команду считывания или записи в память, контроллер кэш-памяти отправляет в системный коммутатор того процессора, к чьей памяти планируется обращение (home-процессора), соответствующий запрос (считывание и/или уничтожение). Такой запрос называется *исходным*. По приему исходного запроса системный коммутатор формирует и рассылает *снуп*-запросы, отслеживающие состояние кэшей системы, и, возможно, запросы на считывание в основную память. Снуп-запросы также называются когерентными запросами. Ответы на снуп-запросы (подтверждения или данные) отправляются запросчику, который, получив всю необходимую информацию, извещает системный коммутатор о завершении операции.

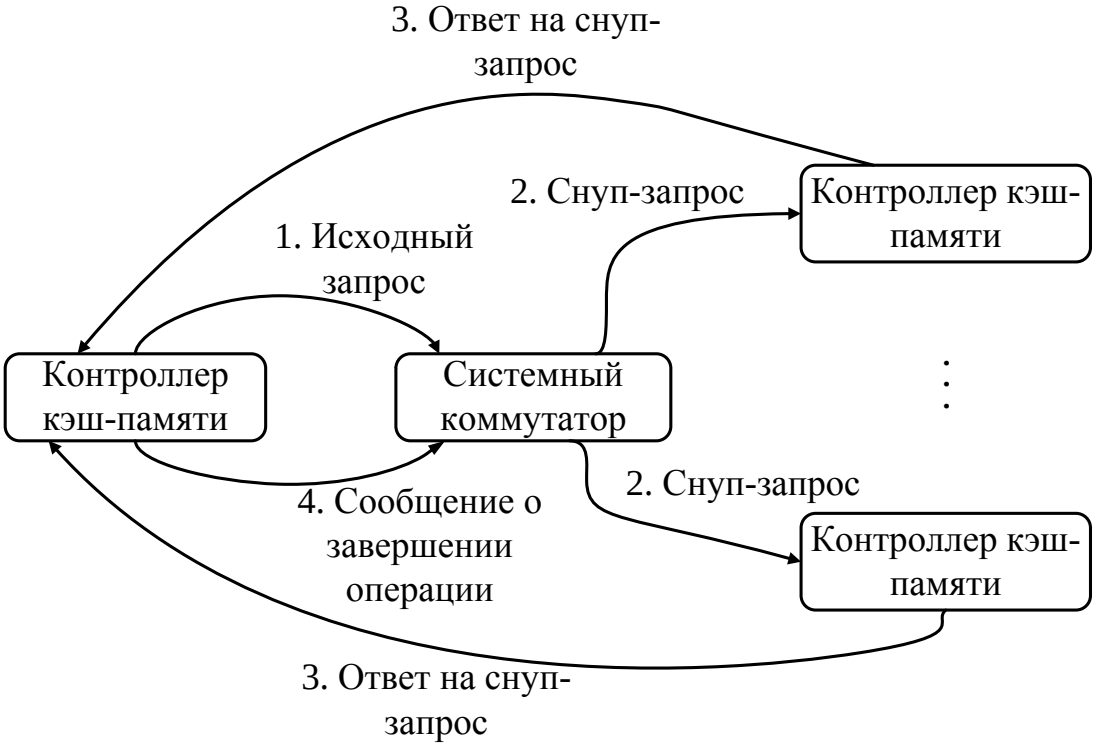


Рисунок 1 – Последовательность этапов выполнения запроса

Порядок исполнения записей с типом write through более сложен и предполагает исполнение запросов других типов, например, запроса на запись в память. Обработка такого запроса требует от системного коммутатора выполнения следующих действий: приема исходного запроса, рассылки снуп-запросов, отправки запроса за данными инициатору исходного запроса, приема этих данных, сбора когерентных ответов, отправки данных в контроллер памяти и информирования запросчика о завершении операции. При этом остальные устройства взаимодействуют с системным коммутатором, отвечая на запросы.

Под состоянием кэш-строки, заведенной в кэш-памяти, понимается состояние соответствующего контроллера кэш-памяти. Среди таких состояний выделяют основные и промежуточные. Основные состояния обычно определяются подмножеством широко известных состояний Modified, Owned, Exclusive, Shared, Invalid (MOESI) [108]. Переходы из одного основного состояния в другое в современных протоколах когерентности памяти происходят не мгновенно, а посредством переходных состояний. Например, при

промахе по считыванию, кэш-контроллер из состояния Invalid перейдет в переходное состояние, в котором проведет некоторое время в ожидании ответа с данными и, возможно, подтверждений, только после чего перейдет в основное состояние Shared. Наличие переходных состояний во многом определяет сложность протоколов когерентности памяти.

Свойства, которым должны удовлетворять корректные протоколы когерентности памяти, обычно формулируют в терминах основных состояний кэш-контроллеров. При этом задаются недопустимые комбинации состояний кэш-строки в различных кэшах.

1.2 Проблема верификации протоколов когерентности

В ходе проектирования микропроцессорных систем протоколы когерентности сначала разрабатываются на концептуальном уровне, в рамках создания микроархитектуры системы. Затем осуществляется их *реализация* путем составления RTL-описания (Register Transfer Level) совокупности устройств, которые должны работать в соответствии с протоколом когерентности памяти (RTL-описания микропроцессора). При этом распространенной практикой является анализ протокола его разработчиками вручную, а затем проверка его реализации тестовыми программами с псевдослучайными воздействиями [19, 107, 66, 4, 5]. Для сокращения времени моделирования микропроцессорной системы разрабатываются прототипы системы, основанные на ПЛИС [17], а также в некоторых случаях применяется интеграция RTL-описания и программных моделей подсистемы памяти [16].

Данный подход откладывает начало верификации на месяцы с момента начала проектирования. Верификация реализации протокола проходит в отсутствие всякой уверенности в корректности самого протокола. Это приводит к тому, что даже спустя годы после начала верификации, все еще обнаруживаются ошибки в самом протоколе (и, соответственно, в реализации

некорректного протокола). Псевдослучайные тестовые последовательности не обеспечивают полноты покрытия пространства состояний протокола за приемлемое время. Несмотря на то, что тестовые программы со случайными воздействиями позволяют найти большое количество ошибок, некоторые ошибки, а в особенности ошибки, связанные с нетривиальной организацией передачи сообщений между частями верифицируемой системы, могут оказаться необнаруженными.

Таким образом, чрезвычайную важность имеет задача верификации самих протоколов когерентности. Ее решение позволит начинать верификацию на ранних этапах проектирования, при взаимодействии разработчиков и верификаторов. Решить задачу верификации протоколов когерентности можно с помощью разработки и анализа математических моделей протоколов.

Формальные методы позволяют получить математическое доказательство соответствия модели верифицируемого объекта его спецификации, то есть набору свойств, которым он должен удовлетворять.

Настоящая работа посвящена разработке методов формальной верификации протоколов когерентности и средств, автоматизирующих применение таких методов.

1.3 Формальные методы

1.3.1 Группы формальных методов

Методика формальной верификации протоколов предполагает наличие:

1. Средств *моделирования* протоколов. Обычно в этой роли выступают языки описания моделей.
2. Средств (языков) *спецификации* протоколов для задания проверяемых свойств.

3. *Методов* верификации, позволяющих установить, соответствует ли модель протокола его спецификации.

Формальные методы разделяются на две группы: методы, основанные на моделях и методы, основанные на доказательствах.

В методах, основанных на моделях, верифицируемая система представляется моделью M формул некоторой логики. Спецификация представлена формулой φ данной логики. Метод верификации устанавливает, является ли M моделью φ ($M \vDash \varphi$). Для конечных моделей такое установление, как правило, может быть автоматизировано.

В методах, основанных на доказательствах (методах дедуктивной верификации), верифицируемая система представляется множеством формул F некоторой логики. Спецификация является другой формулой φ данной логики. Метод верификации состоит в попытке нахождения доказательства того, что φ выводима из F в данной дедуктивной системе ($F \vdash \varphi$). Построение такого вывода обычно не может быть полностью автоматизировано.

Для верификации протоколов когерентности памяти применяются методы, основанные на моделях, методы, основанные на доказательствах, и комбинированные методы. Безотносительно к используемым методам, в большинстве работ рассуждения используют модели протоколов когерентности в виде множеств взаимодействующих процессов, даже если используемые лингвистические средства моделирования не позволяют представлять процессы в явном виде. При этом процессы являются моделями кэш-контроллеров и системного коммутатора. Определение понятия процесса, используемое при разработке методов в данной работе, приведено в главе 2. Далее представлен аналитический обзор методов верификации протоколов когерентности памяти.

Многие работы используют в качестве верифицируемых протоколов протоколы German [99] и FLASH [117]. В работе [32] сообщается, что протокол FLASH является сложным, поэтому если метод может верифицировать этот

протокол, то существует вероятность, что этот метод может быть пригоден для верификации протоколов реальных систем. Протокол German является существенно более простым, однако он также отражает некоторые черты реалистичных протоколов когерентности.

1.3.2 Метод анализа достижимости состояний и метод проверки моделей

Одним из наиболее простых методов верификации протоколов когерентности является анализ достижимости состояний (*reachability analysis*), который полностью исследует пространство глобальных состояний, являющихся композицией состояний всех компонентов системы, то есть основан на алгоритме поиска методом полного перебора. Состояния, в которых ожидаемые свойства корректности протокола не удовлетворены, классифицируются как ошибочные. В противном случае состояния являются допустимыми. Если хотя бы одно из ошибочных состояний достижимо, протокол является некорректным [101, 8]. Метод, основанный на анализе достижимости состояний, подвержен проблеме «взрыва числа состояний» и не применим для сложных систем [102].

Развитием метода анализа достижимости стал распространенный на сегодняшний день формальный метод проверки моделей. Проверка моделей (model checking) – метод, в ходе применения которого систематически исследуется пространство состояний модели верифицируемого протокола с целью проверки выполнения спецификации протокола. Рассматриваемые модели (системы переходов) являются конечными, благодаря чему задача проверки моделей алгоритмически разрешима. Спецификация задается с помощью развитого аппарата темпоральных логик, позволяющих отражать относительный порядок событий без явного указания значений времени. Алгоритмы проверки моделей для наиболее широко используемых

темпоральных логик – логики линейного времени LTL (Linear Time Logic) и логики ветвящегося времени CTL (Computational Tree Logic) – являются эффективными. Вычислительная сложность алгоритма проверки выполнения формулы φ логики LTL равна $O\left(|TS| \cdot 2^{|\varphi|}\right)$, а алгоритма проверки выполнения формулы Ф логики CTL равна $O(|TS| \cdot |\Phi|)$, где $|TS|$ – число состояний и переходов системы переходов, $|\varphi|$, $|\Phi|$ – число подформул соответствующей формулы [21]. Экспоненциальная зависимость от размера формулы φ на практике не носит определяющий характер, так как обычно формулы являются короткими. Определяющим фактором является линейная зависимость от размера системы переходов.

К достоинствам метода проверки моделей относятся:

- полная автоматизация метода;
- генерация диагностической информации – контрпримеров, позволяющих отыскать источник ошибки;
- возможность проведения верификации на ранних стадиях проектирования, до разработки RTL-моделей микропроцессоров;
- возможность предоставления частичных спецификаций свойств, то есть проверки свойств по отдельности.

Основным недостатком метода является «взрыв числа состояний», то есть чрезмерное увеличение $|TS|$.

Существующие инструментальные средства, реализующие метод проверки моделей, в достаточной степени развиты и предоставляют языки описания моделей и спецификации свойств, при этом вся работа по трансляции таких описаний в математические модели и их анализ проводится инструментами автоматически.

Метод проверки моделей позволяет осуществлять проверку свойств различных классов: свойств-инвариантов, свойств безопасности, свойств живучести [21]. Как правило, свойства, с помощью которых формулируется

корректность протоколов когерентности, относятся к классу инвариантов. Поэтому в данной диссертации разрабатываются методы и средства, позволяющие проверять свойства-инварианты.

Метод проверки моделей был успешно применен для верификации различных протоколов когерентности памяти [86, 85, 119, 70, 69, 111, 35, 9, 1]. При этом имеется ограничение на число узлов процессорной системы, которое может быть отражено в модели. В [111, 32, 1] этот параметр ограничивается значением 3–4.

1.3.3 Методы, основанные на доказательстве теорем

Доказательство теорем – процесс определения того, удовлетворяет ли модель заданным свойствам, проводимый путем построения дедуктивного вывода в рамках некоторой формальной системы. В случае успешного построения делается заключение о соответствии модели и спецификации свойств. Этот подход предъявляет высокие требования к верификаторам, поэтому редко используется на практике. Методы, основанные на доказательстве теорем, требуют от пользователя постоянного вмешательства в ход доказательства, с целью продвижения последнего.

Свойства-инварианты могут быть выражены формулой $\mathrm{G}p$ темпоральной логики линейного времени LTL, где p – утверждение – логическая формула от переменных модели. Если утверждение истинно во всех состояниях модели, то, p является инвариантом модели. В соответствии с методом *дедуктивной верификации* для доказательства того, что p – инвариант, необходимо разработать вспомогательное утверждение φ, а затем показать, что p следует из φ. В основе метода лежит следующее правило вывода INV [83]:

$$\frac{\begin{array}{l} I1.\ \varphi \text{ выполняется в начальных состояниях} \\ I2.\ \text{Все переходы между состояниями сохраняют выполнимость } \varphi \\ I3.\ \varphi \to p \end{array}}{\mathrm{G}p}$$

Утверждение φ, удовлетворяющее посылкам $I1$ и $I2$, называется *индуктивным утверждением* или *индуктивным инвариантом*. Если p – инвариант верифицируемой системы, то всегда существует индуктивное утверждение φ, из которого следует p [83]. Исходное утверждение p является индуктивным лишь в редких случаях. Как правило, верификатору необходимо разработать вспомогательное утверждение и установить истинность посылок $I1$–$I3$.

Средство автоматизированного доказательства теорем (theorem prover) – это программный инструмент, определяющий истинность формул в заданной логике. Популярными инструментами являются ACL2, PVS, Coq, Isabelle. Логики, на которых основаны эти средства, очень разнообразны, но общим свойством является их выразительность. Однако выразительность логики влечет за собой ее неразрешимость, что означает невозможность построения такой автоматической процедуры (или алгоритма), что, приняв на вход формулу, она всегда может определить вывод этой формулы в данной логике. Использование средств автоматизированного доказательства теорем подразумевает взаимодействие с пользователем-экспертом и является сложным творческим процессом. Помимо этого, если доказательство теоремы завершается неудачей, то очень сложно на основе этой информации найти ошибку в верифицируемой системе. Преимуществом дедуктивной верификации является возможность работы с системами с бесконечным числом состояний.

В работе [98] описано применение средства автоматизированного доказательства теорем общего назначения PVS [94] для параметризованной верификации протокола когерентности FLASH. В ходе доказательства авторы

[98] многократно вручную проводили поиск кандидатов на индуктивные утверждения. В случае провала попыток доказать их индуктивность, авторы анализировали причины и находили дополнительные условия, превращающие утверждения в индуктивный инвариант. Данный процесс является очень трудоемким, в связи с чем методы, основанные на доказательстве теорем, могут найти лишь ограниченное применение при верификации протоколов когерентности.

В работе [97] предпринята попытка объединения доказательства теорем и автоматической генерации доказательств корректности протоколов когерентности. Достоинством подхода является высокая степень автоматизации, в остальном недостатки те же, что и у доказательства теорем. Данная технология еще недостаточно проработана, в частности, предлагаемый язык описания моделей имеет очень ограниченные возможности.

1.3.4 Сравнение методов проверки моделей и доказательства теорем и постановка задачи верификации параметризованных моделей

Анализ метода проверки моделей и автоматизированного доказательства теорем показал, что метод проверки моделей является полностью автоматизируемым, но применим только к протоколам систем с 3-4 ядрами, а метод автоматизированного доказательства теорем применим для систем с любым количеством ядер, но требует колоссального объема ручной работы.

Промышленность нуждается в методах верификации систем с большим числом процессорных ядер (8, 16 и так далее), что делает метод проверки моделей неприменимым для верификации протоколов когерентности современных систем.

Ограничения автоматизированного доказательства теорем – необходимость чрезмерного объема ручной работы и отсутствие диагностической информации в ходе верификации – делает применение

данного метода нецелесообразным.

Протокол когерентности можно рассматривать как *параметризованную* систему, состоящую из некоторого количества различных процессов и множества идентичных процессов, размер которого является параметром. Другими словами, протокол когерентности можно рассматривать как бесконечное семейство протоколов с конечным числом состояний, в котором первый протокол спроектирован для системы из двух ядер, второй протокол – для системы из трех ядер и так далее. В связи с этим задачу верификации протоколов когерентности памяти целесообразно поставить как задачу параметризованной верификации. Пусть M_n – модель протокола когерентности памяти системы из n процессорных ядер, а φ – темпоральная формула. Моделью протокола когерентности является семейство $\mathcal{F} = \{M_n\}_{n=2}^{\infty}$. Необходимо проверить, что все модели из $\mathcal{F}$ удовлетворяют φ, то есть $\forall n: M_n \vDash \varphi$.

В общем случае задача верификации параметризованных моделей (параметризованной верификации) алгоритмически неразрешима [20, 42]. В связи с этим разработка методов ее решения ведется в двух направлениях:

- разработка полных методов верификации определенных классов моделей с явным указанием всех ограничений на модели;
- разработка неполных методов, для которых нет гарантий завершения.

Классификация методов параметризованной верификации протоколов когерентности памяти приведена на рисунке 2. Указанные группы методов не являются взаимно исключающими и зачастую основаны на общих идеях.

Далее приведен аналитический обзор методов параметризованной верификации протоколов когерентности памяти и, в некоторых случаях, более широкого класса асинхронных реагирующих систем [83]. Большинство работ по верификации протоколов когерентности нацелены на проверку свойств-инвариантов, и лишь некоторые работы также рассматривают проверку свойств

живучести. В дальнейшем под проверяемыми свойствами подразумеваются свойства-инварианты.

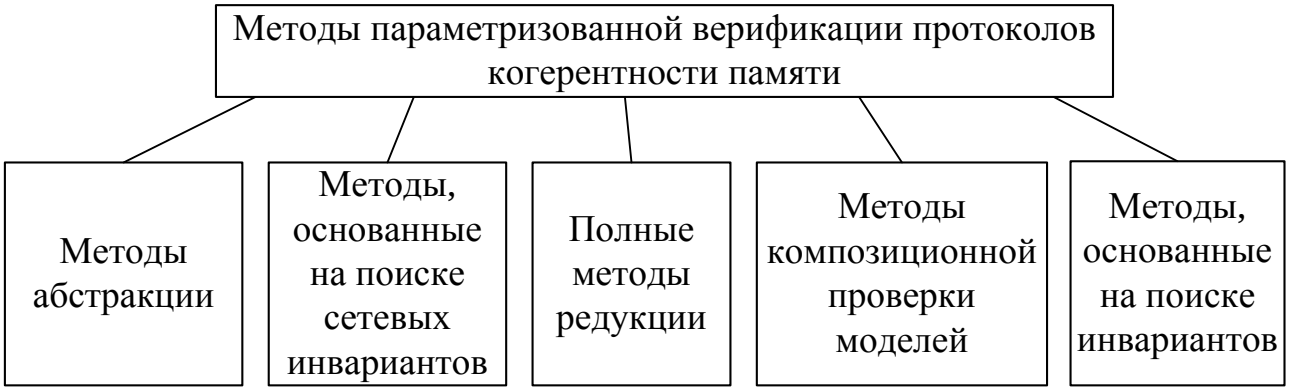


Рисунок 2 – Классификация методов параметризованной верификации протоколов когерентности памяти

1.3.5 Методы абстракции

Одним из основных подходов, позволяющих применить метод проверки моделей к верификации масштабируемых систем, является составление моделей более высокого уровня абстракции. Методы абстракции позволяют сократить количество состояний верифицируемой модели и в то же время сохранить интересующие свойства исходной модели. В методах абстракции происходит отображение параметризованных моделей на одну модель, которая представляет поведение всего параметризованного семейства. Из способа отображения следуют структурные ограничения на параметризованные модели, которые могут быть верифицированы данным методом.

Отношения эквивалентности, гарантирующие, что модели будут вести себя одинаково, зачастую не приводят к существенному сокращению числа состояний, поэтому на практике, как правило, для связи модели с ее абстракцией используют отношения симуляции [35, 24]. Симуляция гарантирует, что всякое поведение в исходной модели является также поведением в ее абстракции. Однако абстракция при этом может иметь

поведения, которые невозможны в исходной модели.

Пути получения абстрактного пространства состояний могут быть разделены на методы аппроксимации снизу (under-approximation), удаляющие часть поведений, и методы аппроксимации сверху (over-approximation), добавляющие новые поведения. Методы аппроксимации сверху приводят к получению так называемой абстракции существования – абстрактной модели, в которой при существовании перехода между двумя состояниями исходной модели (конкретными состояниями), всегда существует переход между любыми двумя абстрактными состояниями, включающими данные конкретные состояния. Абстракция существования гарантирует сохранение в абстрактной модели достижимых путей в ошибочное состояние. Таким образом, если абстрактная модель не содержит ошибку, то в исходной модели ее тоже нет. С другой стороны, методы аппроксимации снизу ведут к универсальной абстракции, в которой переход из абстрактного состояния будет существовать только тогда, когда из всех состояний исходной модели существует соответствующий переход. Таким образом, в случае аппроксимации снизу корректность абстрактной модели подразумевает корректность исходной модели. В дальнейшем будут затрагиваться только аппроксимации сверху, называемые также консервативными абстракциями.

Построение абстрактных моделей требует нахождения компромисса между двумя конфликтующим целями: получение абстрактных моделей небольшого размера, которые могут быть верифицированы методом проверки моделей, и получением точных абстрактных моделей (рисунки 3–4). Обычно, чем проще модель, тем больше поведений она допускает. Выходов из этой ситуации, по крайней мере, два: построение точной абстрактной модели или анализ контрпримера на ложность и модификация абстрактной модели на основании полученной информации (counterexample-guided abstraction refinement) [38]. В связи с алгоритмической неразрешимостью задачи

параметризованной верификации, нет гарантий завершения цикла детализации абстрактных моделей. Поэтому методы абстракции являются неполными. Однако методы, основанные на абстракции, были успешно применены на практике для верификации протоколов и программно-аппаратных систем. Это связано с тем, что данные методы потенциально могут привести к существенному сокращению пространства состояний модели.

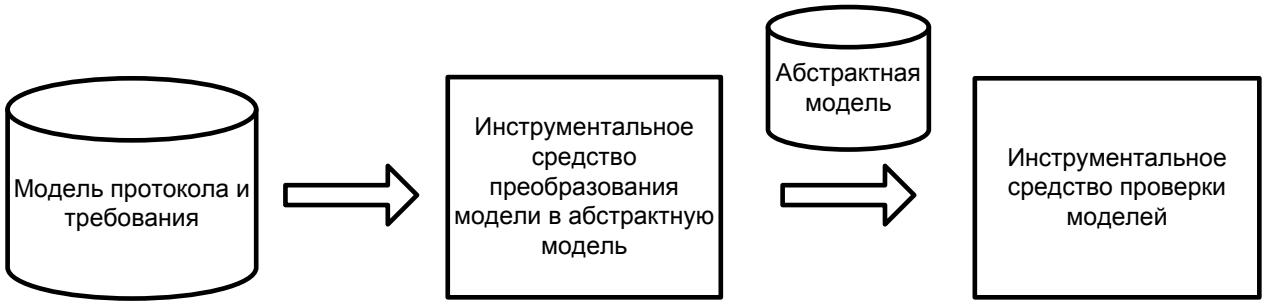


Рисунок 3 – Схема построения точных абстрактных моделей

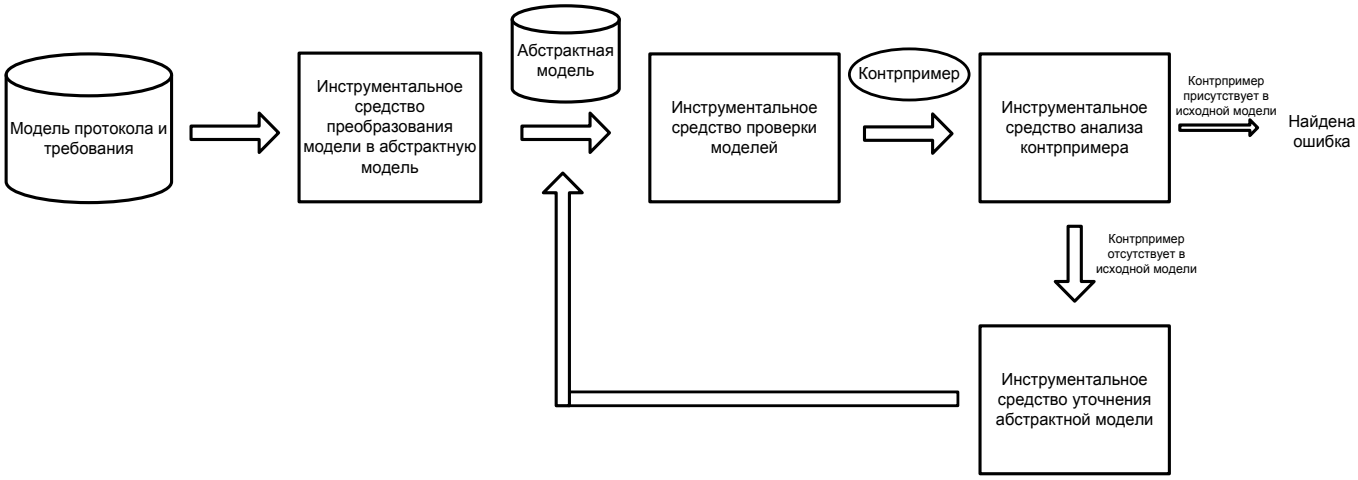


Рисунок 4 – Схема построения абстрактных моделей с циклом детализации

В работе [39] описано применение метода предикатной абстракции, изначально представленного в работе [51], для автоматизированного усиления инвариантов при верификации протокола когерентности памяти FLASH. Данный подход позволяет сократить объем ручной работы по сравнению с дедуктивной верификацией в [98], однако он все еще остается высоким. Авторы [39] отмечают, что им понадобилось пять дней ручной работы и два часа компьютерного времени для проведения доказательства.

В [110] предлагается метод двумерной абстракции, предполагающий построение абстрактной модели за два шага: уменьшение пространства состояний каждого процесса модели путем предикатной абстракции и объединение полученных состояний в классы эквивалентности. Сокращение пространства состояний абстрактной модели продемонстрировано на простых примерах известных протоколов, что не затрагивает важный вопрос верификации протоколов с переходными состояниями.

В работе [100] представлена абстракция со счетчиками ($\{0,1,\infty\}$-counter abstraction) для верификации моделей, состоящих из N одинаковых процессов. Каждый процесс имеет конечное число состояний, и в модели введено конечное число глобальных разделяемых переменных y_j с конечной областью определения. Метод предполагает отображение состояний исходной системы $\langle l_1, l_2, \dots, l_N, \eta \rangle$, где η – интерпретация переменных y_j, в состояния абстрактной системы $\langle \kappa_1, \kappa_2, \dots, \kappa_N, \eta \rangle$, где $\kappa_i, i = 1, \dots, N$ – количество процессов в локальном состоянии l_i: $\kappa_i \in \{0,1,2\}$, $\kappa_i = 0$, если в состоянии l_i нет ни одного процесса, $\kappa_i = 1$, если в данном состоянии находится один процесс, и $\kappa_i = 2$, если более одного процесса. При этом атомарные высказывания позволяют проверять свойства, отражающие количество процессов в данном локальном состоянии. В работе [100] метод применен для верификации протоколов взаимного исключения. Метод может быть автоматизирован, однако определение функции абстракции может потребовать введения дополнительных переменных, при этом не предложено инструкций или алгоритмов по введению таких переменных. Метод подвержен проблеме взрыва числа состояний, если процессы верифицируемой модели имеют большое число локальных состояний, и модель содержит большое число разделяемых переменных.

В работе [71] предложена абстракция, схожая с абстракцией со счетчиками, для верификации протоколов когерентности памяти. Метод является автоматизируемым, однако следующие факторы существенно

затрудняют его применение: привязка к модифицированной версии инструмента проверки моделей Murφ, отсутствие гарантий завершения метода, отсутствие каких-либо указаний по поводу устранения ложных контрпримеров, которые может порождать предложенная абстракция. Работы [101, 120] используют похожую абстракцию.

Метод абстракции среды [34, 36, 114] является обобщением метода абстракции со счетчиками и нацелен на повышение уровня автоматизации процесса верификации нескольких классов протоколов, среди которых находятся протоколы взаимного исключения и протоколы когерентности памяти. Предполагается описание протоколов пользователем с помощью специальных операторов и разработка им спецификации в виде, отличном от вида, предполагаемого известным аппаратом темпоральных логик. При этом инструментальная поддержка этих действий, описываемая авторами [36], носит только характер прототипа. При верификации возможно появление ложных контрпримеров, что требует нетривиального вмешательства пользователя. В работе [36] отмечено, что абстрактные модели носят слишком детальный характер, что в случае сложных протоколов приводит к взрыву числа состояний их абстрактных моделей. С помощью данного метода удалось проверить только упрощенный вариант протокола FLASH.

В работе [22] предложен метод, позволяющий вычислять абстракции параметризованных моделей, разработанных с использованием разрешимой логики WS1S (Weak monadic Second-order logic of One Successor). Метод позволяет получить абстрактную модель автоматически, однако его лингвистическая и инструментальная поддержка носит экспериментальный характер, и вопрос масштабируемости метода в статье [22] не проработан: приведен лишь пример верификации протокола German.

1.3.6 Методы, основанные на поиске сетевых инвариантов

Методы, основанные на поиске сетевых инвариантов [122, 74, 72], используют индуктивные рассуждения для построения доказательства того, что некоторое свойство ϕ истинно для композиции $\wp = P \parallel \cdots \parallel P$ из n процессов P. Идея методов заключается в следующем. Формируется процесс I в качестве кандидата в сетевые инварианты. Выбирается отношение абстракции $\preccurlyeq$ между процессами. Необходимо показать, что I является сетевым инвариантом. Доказательство проводится в два этапа: сначала доказывается, что $P \preccurlyeq I$, а затем, что $P \parallel I \preccurlyeq I$. После установления, что I – сетевой инвариант, показывается, что $I \vDash \phi$, и делается заключение: $\wp \vDash \phi$. Достоинствами методов, основанных на поиске сетевых инвариантов, являются данная несложная схема доказательства, а также возможность верификации моделей, в которых процессы объединены в различные топологии, описываемые сетевыми грамматиками.

Недостатки данных методов определяются тем, что ключевой задачей, решаемой при верификации этими методами, является разработка сетевого инварианта I, которая полностью не автоматизируема: инвариант может не существовать для данной системы, и задача определения того, существует ли инвариант, алгоритмически неразрешима [122]. В работах [72, 80] предложены эвристики, использованные для верификации несложных протоколов.

В статье [52] предложена процедура, автоматизирующая поиск сетевых инвариантов, основанная на алгоритмах для обучающих автоматов. Авторы [52] отмечают, что методы обучения не масштабируются на случаи верификации сложных протоколов. В статье процедура использована для верификации простого протокола взаимного исключения.

В работе [75] предложен частично автоматизированный метод построения сетевого инварианта для протокола когерентности IEEE Futurebus+. В основе метода лежит идея абстракции систем переходов, описанных формулами слабой

логики второго порядка с одним преемником WS1S. Отношение абстракции при этом разрабатывается вручную, на основе чего автоматически вычисляется кандидат в сетевые инварианты. Далее необходима проверка того, действительно ли является полученный кандидат сетевым инвариантом, что требует разработки ряда математических доказательств. Таким образом, для метода характерны сложность представления модели протокола с использованием требуемых формализмов и необходимость многочисленных ручных вмешательств в ходе верификации. Например, при разработке отношения абстракции автор [75] основывался на литературе [35], в которой детально рассматривается нахождение сетевого инварианта для данного протокола вручную. Для протоколов промышленно выпускаемых систем такая литература недоступна.

В [15] предложен метод верификации параметризованных моделей распределенных программ с произвольным числом однотипных асинхронно-взаимодействующих процессов, объединенных в ряд топологий. Введены новые отношения симуляции и предложена модификация метода сетевых инвариантов. В разработанной на основе предложенного подхода системе используется подмножество языка описания моделей Promela и инструмент верификации моделей Spin. Использование лишь подмножества языка Promela накладывает ряд ограничений: взаимодействие процессов может происходить только по синхронным каналам, синхронизация с помощью разделяемых переменных не допускается, что существенно снижает выбор альтернатив при описании протоколов когерентности. Практическая применимость разработанного метода показана на примере верификации параметризованной модели протокола резервирования ресурсов, однако нет сообщений о применении к промышленно разрабатываемым системам.

В работе [84] предложен подход к проектированию заведомо верифицированных компонент протоколов когерентности памяти, которые

могут быть объединены в иерархическую структуру, основанный на поиске сетевых инвариантов и использовании инструмента проверки моделей Cubicle.

1.3.7 Полные методы редукции

Полные методы редукции [49, 46, 48, 47, 118] позволяют сводить задачу параметризованной верификации некоторых классов моделей, в которых процессы удовлетворяют определенным ограничениям, к задаче верификации одного представителя параметризованного семейства методом проверки моделей. Методы предполагают нахождение такого числа K, что верификация модели с K процессами (cutoff) достаточна для гарантии корректности всего семейства параметризованных моделей.

Достоинствами полных методов редукции являются алгоритмизация и гарантия завершения. Недостатками методов является ограниченная область применения, и большие значения числа K в случае протоколов когерентности. В работе [49] рассматривается класс асинхронных моделей, в которых процессы объединены в однонаправленное кольцо и взаимодействуют с помощью синхронной передачи маркера. В работе [118] результаты расширены для произвольных сетей из процессов, обменивающихся маркером. В работе [47] найдено значение $K = 7$ для моделей протоколов когерентности памяти, не учитывающих наличие директории – процесса, синхронизирующего действие процессов, представляющих кэш-контроллеры. Верификация версии протокола German проведена с помощью нетривиального сведения этого протокола к протоколам без директории. В работе [18] найдено значение $K = 6$ для протокола когерентности German. Значения $K > 4$ делают метод неприменимым для верификации протоколов промышленного масштаба.

1.3.8 Методы композиционной проверки моделей

Идея композиционной верификации [33] заключается в использовании естественной декомпозиции сложной распределенной системы на взаимодействующие процессы. Подход предполагает проверку процессов по отдельности с некоторым обобщенным окружением (упрощенных моделей) и последующее объединение результатов, сопровождаемое заключением о корректности исходной модели. Композиционный метод должен сопровождаться доказательством того, что упрощенная модель удовлетворяет тем же свойствам, что и исходная модель.

В работе [89] изложен метод параметризованной верификации протоколов когерентности памяти, основанный на композиционной проверке моделей [87, 88] и реализованный в системе Cadence SMV. Задав свойство, изначально делают попытку проверить его на абстрактной модели, в которой отражены процессы, номера которых содержатся в свойстве, а все остальные процессы заменены абстрактным процессом (в соответствии с абстракцией типов данных [88]). В случае получения контрпримера вероятной причиной его появления является сообщение, порожденное абстрактным процессом. Чтобы избавиться от контрпримера, предлагается две стратегии: выделить отправителя сообщения в окружение в виде отдельного процесса (что не может быть осуществлено много раз, так как ведет к взрыву числа состояний) или ввести лемму, исключающую некорректную отправку сообщения. Процесс повторяется до тех пор, пока не будут исключены все ложные контрпримеры и проверены все свойства спецификации.

Преимущество метода заключается в отсутствии необходимости предоставлять индуктивные инварианты, поскольку соответствующая информация (множество достижимых состояний) получается путем проверки абстрактных моделей. Тем не менее, объем ручной работы по введению лемм остается существенным. Метод был применен для верификации протокола

FLASH, что говорит о масштабируемости метода.

В работе [32] предложен метод параметризованной верификации протоколов когерентности памяти, развивающий идеи работы [89], и основанный на абстракции типов данных и уточнении абстрактных моделей на основе контрпримеров с помощью введения лемм (инвариантов). В качестве основного инструмента, используемого в процессе верификации, выступает средство автоматизированной проверки моделей Murφ. Метод формализован в работе [73]. В [73] для представления моделей введен язык первого порядка. В терминах этого языка приведены преобразования, приводящие к абстрактной модели. Поскольку в [32] для описания моделей используется язык Murφ (а в последующих работах, развивающих данный метод, используется та же самая модель), необходимо в явном виде установить соответствие между языком Murφ и математическим языком из [73], что в [73] не сделано, хотя можно увидеть сходства между подмножеством языка Murφ и данным языком. Достоинствами метода является масштабируемость, что является следствием применения абстракции типов данных, и синтаксический характер абстрактных преобразований. Второй аспект позволяет автоматизировать процесс получения абстрактной модели путем создания инструмента, работающего с описанием модели на языке моделирования. При этом не нужно модифицировать средство автоматизированной проверки моделей, а возможно его использовать напрямую для проверки абстрактных моделей. Более того, в [115] утверждается, что метод корректен для любого симметричного протокола и любой консервативной процедуры абстракции.

Основная сложность метода заключается в нахождении лемм. Добавление лемм – трудоемкий процесс, занимающий много времени и требующий глубокого понимания протокола.

В работе [115] предложено получать леммы на основе потоков сообщений – последовательностей сообщений, пересылаемых между процессорами во

время работы протокола. В работе [93] расширено понятие потоков сообщений до их представления в виде ориентированных ациклических графов. Это позволило достичь частичной автоматизации процесса получения лемм. Расширенный метод был применен для параметризованной верификации сложного протокола LCP (Larrabee Coherence Protocol), разработанного в Intel [93, 79].

В [32] утверждается, что метод может быть использован с любым средством автоматизированной проверки моделей. При этом показан лишь пример описания протокола когерентности German на языке Murφ, полученный непосредственным переводом описания данного протокола из [99]. Не объясняется, каким образом допустимо описывать формальные модели, и какие существуют ограничения, а ограничения существуют, и важность этой проблемы нашла отражение в литературе.

В работе [104, 105] приведен анализ метода с точки зрения пользователя системы Abster, автоматизирующей генерацию абстрактных моделей. Данная система упоминается в [115, 93] и не находится в открытом доступе. В [104, 105] приведен ряд рекомендаций разработчикам протоколов когерентности, и утверждается, что следование данным рекомендациям позволит получать протоколы, совместимые с Abster. Поскольку анализ приведен с пользовательской точки зрения, неясно, какие из указанных ограничений являются фундаментальными ограничениям метода, а какие – ограничениями инструмента Abster.

Таким образом, метод [32, 115, 93] представляет значительный интерес с точки зрения верификации протоколов когерентности, но обладает рядом существенных недостатков, что делает невозможным непосредственное применение метода.

В работах [106, 54, 45, 31] предложены методы верификации иерархических протоколов когерентности памяти, в которых задействованы

несколько уровней кэш-памяти. Методы основаны на абстракции, рассуждениях по схеме «допущение-подтверждение» [35], уточнении абстракций на основе контрпримеров и базируются на [32]. Предложены эвристики для автоматической проверки контрпримеров на ложность. Методы [106, 54, 45, 31] были применены к набору из нескольких иерархических протоколов когерентности, разработанных авторами этих работ на основе таких протоколов, как FLASH и DASH.

Вопросы верификации иерархических протоколов еще не нашли достаточного отражения в литературе, однако необходимость верификации таких протоколов становится все более актуальной в связи с их реализацией в новых моделях микропроцессоров. Например, в микропроцессоре Эльбрус-8С в реализации протокола когерентности участвует кэш-память второго и третьего уровней.

1.3.9 Методы, основанные на поиске инвариантов

Идея методов, основанных на поиске инвариантов, заключается в поиске таких индуктивных инвариантов протокола, из которых следуют проверяемые свойства.

Многие работы, в которых представлены методы, основанные на поиске инвариантов, обладают следующими характерными особенностями. Предполагается, что модель верифицируемого протокола, состоящая из ограниченного до трех или четырех элементов набора процессов, изначально тщательно отлажена, например, с помощью моделирования или метода проверки моделей. Поэтому при разработке таких методов подразумевается, что в случае получения контрпримера, был неверно сгенерирован кандидат в инварианты, а не найдена ошибка в протоколе. Вопросы написания моделей в работах не рассматриваются. Используемые языки описания моделей не имеют процессных типов и примитивов передачи сообщений. Поэтому в случае

использования языков с такими типами (например, Promela) для начальной отладки, что может быть продиктовано удобством моделирования, использование Promela для параметризованной верификации затруднено. Проблему можно решить трансляцией языка Promela во входной язык инструментов, реализующих методы параметризованной верификации, либо разработкой собственных инструментов, поддерживающих язык Promela. Обе задачи нетривиальны. В первом случае на этапе параметризованной верификации придется работать с неестественными моделями. Поскольку требуемые в методах операции (например, возможность вызова решающих процедур для логик) могут существенно отличаться от того, на чем основана реализация инструмента проверки моделей Spin, во втором случае придется проделать колоссальный объем работы по интеграции Promela и необходимых решателей.

В работах [99, 96] изложен неполный метод проверки свойств-инвариантов для определяемого в данных статьях класса моделей асинхронных систем, называемый методом невидимых инвариантов. Количество процессов, составляющих данные модели, определяется параметром. Каждый процесс имеет конечное число состояний. Взаимодействие между процессами осуществляется посредством разделяемых переменных. В работах предложены алгоритм определения истинности посылок правила INV и эвристика для построения индуктивного инварианта («невидимый инвариант»), основанные на алгоритмах символьной проверки моделей. Доказана теорема, согласно которой невидимый инвариант модели из определенного фиксированного количества процессов также является инвариантом параметризованной модели. Для протокола German данное количество процессов равно четырем.

Достоинствами метода являются высокая степень автоматизации и отсутствие необходимости использования средств автоматизированного доказательства теорем. К недостаткам метода относятся:

- необходимость ручной модификации верифицируемой модели в случае, если инструменту, реализующему метод, не удается найти индуктивный инвариант. Отсутствуют алгоритмы проведения такой модификации;
- возможность получения слишком больших значений параметра для моделей, используемых при поиске инвариантов;
- отсутствие промышленных инструментов, реализующих метод; ограниченные лингвистические средства для описания моделей. Алгоритмы реализованы в более не поддерживаемом инструменте TLV (Temporal Logic Verifier).

В работах [76, 77, 78] предложен неполный метод индексных предикатов, являющийся расширением метода предикатной абстракции. Метод позволяет отыскивать наиболее сильные индуктивные инварианты параметризованных моделей, составленные на основе определенного множества атомарных предикатов. При этом некоторые из этих атомарных предикатов должны быть предоставлены пользователем вручную, поскольку для поиска предикатов в работах предложены лишь эвристики. При этом метод позволяет пользователю предоставлять более простые предикаты, чем в методе предикатной абстракции, при этом инструмент верификации способен отыскивать сложные инварианты. Авторами данных статей продемонстрирована работоспособность метода на примере протокола German. В работе [68] сообщается, что инструментальное средство UCLID, в котором реализован данный метод, не способно проверить протокол FLASH. Неприменимость метода к протоколу FLASH также отмечена в статье [95]. В работе [36] сказано, что исследователям не удалось применить методы невидимых инвариантов и индексных предикатов для верификации сложных протоколов.

В работе [95] предложен набор эвристик для поиска инвариантов параметризованных моделей протоколов когерентности памяти, базирующихся на решающих процедурах для фрагмента логики первого порядка с

неинтерпретируемыми функциями и равенством. В работе использованы реализация данных процедур в инструменте UCLID и предоставляемые данным инструментом средства моделирования. В работе акцентируется, что многие методы поиска инвариантов генерируют слишком много инвариантов (или один большой инвариант, являющийся их конъюнкцией), и при этом неясно, как выбирать из этих инвариантов полезные. Метод нацелен на минимизацию количества генерируемых инвариантов и отсеивание ложных предикатов, создаваемых в ходе поиска индуктивных инвариантов. Однако для отсеивания предложены только эвристики и отмечено, что они могут быть автоматизированы. При этом не разработаны инструменты, проводящие такую автоматизацию.

В работе [68] предложен алгоритм, основанный на идее анализа обратной достижимости в параметризованных моделях. Алгоритм способен автоматически находить инварианты, достаточно сильные для проверки протокола FLASH. Метод не был применен к верификации протоколов промышленного масштаба. В работе делается акцент на доказательство корректности протоколов, но не рассматривается вопрос отделения ложных контрпримеров, возникающих в ходе работы метода, и контрпримеров, представляющих ошибки в протоколе.

В работах [81] и [25] предлагаются методы, автоматизирующие вычисление инвариантов, необходимых в методе [32]. Генерируемые первым методом инварианты могут быть слишком большими, что приводит к чрезмерному использованию памяти. Направления по избавлению от лишних переменных не предложены. Второй метод не применим для некоторых протоколов и основан на символьной проверке моделей, что требует трансляции исходной модели протокола в модель на языке инструмента, реализующего требуемые решающие процедуры. Обе работы основаны на тех же моделях, которые использованы в работах [32, 115, 93].

1.4 Постановка задачи

В диссертации разрабатываются новые методы и средства верификации протоколов когерентности кэш-памяти, обеспечивающих согласованность данных в современных масштабируемых микропроцессорных системах промышленного масштаба. Верифицируемым объектом является протокол когерентности 16-ядерной системы из микропроцессоров Эльбрус-4С, разрабатываемой в АО «МЦСТ».

Анализ работ по параметризованной верификации протоколов когерентности памяти показал, что большинство из представленных методов не масштабируется в достаточной степени для их применения к верификации протоколов когерентности памяти промышленно разрабатываемых микропроцессоров. Использование потенциально масштабируемых методов требует разрешения ряда вопросов. Таким образом, не обнаружен метод, который можно непосредственно использовать для верификации протокола системы Эльбрус-4С.

За основу в данной работе взят метод композиционной верификации [32, 115, 93], поскольку метод потенциально является масштабируемым, основан на синтаксической абстракции и предполагает использование средства автоматизированной проверки моделей в качестве инструмента формальной верификации. При этом не нужно модифицировать этот инструмент: всю остальную работу можно выполнить с помощью дополнительно разработанных программ. С другой стороны, базирование на устаревших инструментах, ограниченный анализ метода в существующих работах, отсутствие детального описания метода и ряда соответствующих инструментальных средств в открытом доступе, не позволяют непосредственно применять метод. Необходим дополнительный анализ метода и интеграция с современными инструментальными средствами формальной верификации.

Поскольку в микропроцессоре Эльбрус-4С в реализации протокола

когерентности памяти участвует один уровень кэш-памяти – кэш-память второго уровня – в диссертации рассматривается верификация протоколов, удовлетворяющих данному свойству.

В соответствии с известным подходом, спецификация протоколов когерентности памяти разрабатывается относительно основных состояний кэш-строки [108]. В протоколе когерентности системы Эльбрус-4С основными состояниями кэш-строки являются состояния Modified (модифицированная), Owned (находящаяся во владении), Shared (разделяемая) и Invalid (недействительная). Относительно данных состояний свойства сформулированы следующим образом:

1. Кэш-строка никогда не может находиться в состоянии Modified в двух кэшах одновременно.
2. Кэш-строка никогда не может находиться в состоянии Owned в двух кэшах одновременно.
3. Кэш-строка никогда не может находиться в одном кэше в состоянии Modified, а в другом – в состоянии Shared или Owned.

В связи с тем, что количество ядер системы Эльбрус-4С является большим, и продолжает увеличиваться в новых поколениях микропроцессоров, задача верификации сформулирована как задача параметризованной верификации, не накладывающая ограничений на число ядер. Исходными данными является документация на протокол когерентности. Необходимо проверить соответствие модели протокола когерентности обозначенной спецификации.

В диссертации разрабатывается метод, позволяющий проводить такую проверку. В рамках разработки метода разрешаются следующие вопросы:

- Описание формальных моделей протоколов когерентности памяти и разработка ограничений на модели. Ввиду сложности протоколов когерентности, разрабатываемых для современных микропроцессоров,

необходимо иметь язык, позволяющий описывать протоколы естественным образом, при котором рассуждения разработчиков о протоколе непосредственно отражаются в тексте модели.

- Описание абстрактных преобразований моделей протоколов.
- Разработка математической модели протоколов когерентности, которая отражает семантику выбранного языка описания моделей, и доказательство корректности абстрактных преобразований.
- Разработка процедуры уточнения абстрактных моделей.
- Разработка инструментальных средств, частично автоматизирующих метод.

Выводы по главе 1

1. В связи с постоянным увеличением количества процессорных ядер современных микропроцессорных систем, задача верификации протоколов когерентности памяти, обеспечивающих согласованность данных в таких системах, сформулирована как задача параметризованной верификации.

2. Выполнен аналитический обзор методов параметризованной верификации протоколов когерентности памяти. Обнаружено, что не существует методов, которые могут быть непосредственно применены для решения поставленной задачи. Это связано со следующими особенностями существующих методов и работ. Степень автоматизации методов имеет обратную зависимость от ограничений на область их применения: методы с меньшим количеством ограничений требуют большого объема ручного вмешательства, а методы для узких классов моделей могут быть полностью автоматизированы. Иногда возможно сведение реалистичных протоколов когерентности к таким узким классам, что нетривиально. Инструментальная поддержка зачастую выполнена в виде прототипов, которые не всегда находятся в открытом доступе и основаны на устаревших инструментах проверки

моделей. Вопросы построения моделей, как правило, не рассматриваются. Вопросы ограничений на модели проработаны недостаточно.

3. Осуществлен выбор метода верификации [32, 115, 93] в качестве основы метода, разрабатываемого в данной работе. Критериями выбора явилась масштабируемость, высокий уровень переиспользования существующих широко применяемых инструментов, потенциально высокий уровень автоматизации.

4. Показана необходимость решения следующих задач: поиск языка, позволяющего описывать модели протоколов когерентности естественным образом, разработка абстрактных преобразований, учитывающих особенности такого языка, разработка математической модели протоколов когерентности и доказательство корректности преобразований моделей.

2 Разработка абстрактных моделей протоколов когерентности

2.1 Выбор языка описания и спецификации моделей

2.1.1 Разработка модели протоколов когерентности в виде множества взаимодействующих конечных автоматов

В первой главе в качестве основы для разрабатываемого в данной работе метода параметризованной верификации протоколов когерентности памяти выбран метод, позволяющий использовать инструмент проверки моделей в качестве основного инструментального средства верификации. Это дает возможность выбора такого средства из существующих и последующего использования предоставляемых им лингвистической и алгоритмической поддержки метода проверки моделей для моделирования протоколов когерентности памяти.

В соответствии с моделью микропроцессорной системы, используемой в работе [108] для представления и анализа протоколов когерентности памяти, в данной диссертации выбрана модель протоколов когерентности памяти в виде множества взаимодействующих конечных автоматов [59, 27].

Элементы данного множества разделены на два типа: кэш-контроллеры и системный коммутатор. Каждым устройством памяти микропроцессорной системы управляет контроллер когерентности – конечный автомат. Работу контроллеров когерентности координирует устройство системный коммутатор, которое также может быть представлено в виде конечного автомата. Множество этих конечных автоматов составляет распределенную систему, в которой автоматы обмениваются друг с другом сообщениями с целью постоянного поддержания инвариантов SWMR и значения данных для каждой строки кэш-памяти.

Каждый контроллер когерентности, управляющий кэш-памятью,

логически реализует множество конечных автоматов, элементами которого являются независимые идентичные конечные автоматы, по одному на каждую кэш-строку. Такие автоматы в данной работе называются *кэш-контроллерами*, и рассматриваются модели протоколов когерентности с одной кэш-строкой. Отражать одну кэш-строку в модели без потери общности позволяет независимость и идентичность кэш-контроллеров. Такое представление позволяет ограничить сложность моделей и является типичным для работ по верификации протоколов когерентности памяти.

Взаимодействующие конечные автоматы, составляющие представленную модель (рисунок 5), в литературе по верификации протоколов часто называют процессами и приводят рассуждения с использованием этого понятия, даже в тех работах, где формальные модели в итоге разработаны с помощью языков, не имеющих явного представления процессов (Murφ и другие подобные языки).

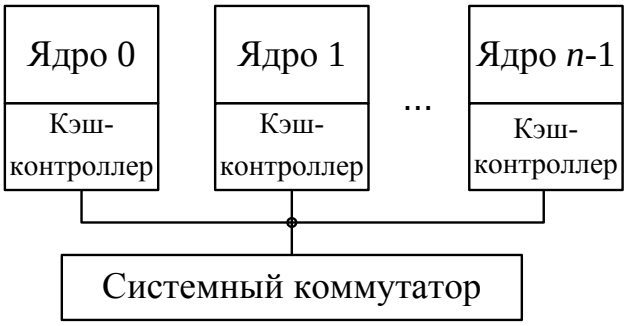


Рисунок 5 – Структура модели протоколов когерентности

Такая модель соответствует структуре микропроцессора Эльбрус-4С. В данном микропроцессоре каждое ядро обладает собственной кэш-памятью первого и второго уровней, и кэш-контроллер управляет кэш-памятью второго уровня. Взаимодействие кэш-контроллеров при этом происходит под управлением системного коммутатора. Инициаторами взаимодействия являются процессорные ядра, исполняющие инструкции обращения к памяти. В главе 4 показано, что такую модель можно использовать не только для моделирования протокола когерентности одного микропроцессора, но и для представления

протокола когерентности всего кластера из микропроцессоров Эльбрус-4С.

2.1.2 Выбор языка и инструментального средства для описания моделей протоколов когерентности

Среди наиболее известных инструментов, автоматизирующих метод проверки моделей, и позволяющих моделировать асинхронные системы, можно выделить Spin [65, 23], Murφ [103] и NuSMV [92]. Математические модели, используемые первыми двумя инструментами, основаны на явном представлении состояний, а NuSMV – на символьном представлении множеств состояний и отношений между ними с помощью бинарных решающих диаграмм (BDDs).

В литературе [32, 43] сообщается, что системы, основанные на неявном представлении состояний модели, менее надежны и эффективны при верификации протоколов когерентности, чем системы с явным представлением. В то же время в системе Spin реализован алгоритм оптимизации, имеющий некоторые сходства с методами, основанными на BDD [55].

В связи с тем, что язык NuSMV основан на параллельных присваиваниях [91], модели протоколов когерентности на языке NuSMV не отражают напрямую то, как протоколы представлены разработчиками при их проектировании и обсуждениях.

Системы Murφ и Spin предназначены для верификации моделей асинхронных реагирующих систем. В отличие от языка Murφ, язык Promela предоставляет процессные типы и средства синхронного и асинхронного взаимодействия между ними (каналы), что позволяет естественным образом использовать такие типы в качестве абстракций устройств, участвующих в работе протокола, а также удобно моделировать взаимодействие между абстракциями. Более того, система Spin является современной и постоянно развивающейся.

Важным преимуществом системы Spin является то, что в ней реализовано множество алгоритмических оптимизации метода проверки моделей. Данные оптимизации можно разделить на две ортогональные по отношению друг к другу группы:

- Оптимизации, позволяющие уменьшить число достижимых состояний модели. К этой группе относятся редукция частичных порядков [58, 26] и объединение операторов (state merging) [65].
- Оптимизации, позволяющие уменьшить объем памяти, необходимой для хранения каждого состояния. В эту группу входят методы без потерь (компрессия состояний collapse [63], техника с использованием минимального автомата [55]) и методы с потерями (редукция hash-compact [121, 112], bitstate-хэширование [57, 56]). При использовании методов с потерями некоторые контрпримеры могут быть не найдены. Однако если контрпример найден, то он действительно представляет ошибку.

Более того, Spin конструирует множества состояний и переходов «на лету», что, в случае наличия контрпримеров, для их нахождения может потребовать лишь небольшой части того объема памяти, который необходим для построения и исследования всего пространства состояний.

Развитие многоядерных процессоров также находит свое отражение в системе Spin: в последних версиях появились параллельные алгоритмы поиска в глубину [61, 64] и поиска в ширину [62], применяемые в Spin для исследования пространства состояний модели. Несмотря на то, что данные алгоритмы не позволяют сократить объем используемой памяти, они уменьшают время проведения верификации.

В работе [60] предложен алгоритм, который реализован в Spin и позволяет находить контрпримеры, которые более просты для анализа, чем контрпримеры, полученные с помощью традиционных алгоритмов, реализованных в Spin.

Нацеленность инструмента Spin на верификацию протоколов (и

предоставление языка Promela, обладающего необходимыми для этого средствами), богатый выбор оптимизаций и режимов верификации и постоянное развитие инструмента в различных направлениях определили выбор языка Promela в качестве лингвистического средства описания протоколов когерентности памяти, и системы Spin в качестве инструмента проверки моделей.

Инструмент Spin позволяет описывать спецификацию формулами темпоральной логики линейного времени LTL [82, 35, 21, 14]. Формулы LTL интерпретируются (принимают истинное или ложное значение) на путях системы переходов – бесконечных цепочках состояний, в каждом из которых атомарные высказывания имеют конкретное истинностное значение – $true$ или $false$. Формулы LTL должны выполняться на всех возможных путях системы переходов. Требования, приведенные в разделе 1.4, можно сформулировать как требования отсутствия в системе переходов, являющейся моделью протокола когерентности памяти, путей, ведущих в ошибочное состояние, в котором имела бы место запрещенная комбинация состояний кэш-строки. Таким образом, в соответствии с семантикой логики LTL, с помощью LTL-формул, расширенных квантором всеобщности, требуемые свойства можно представить следующим образом:

1. $\forall i, j \in \{1, \dots, n\}, i \neq j$: $\mathrm{G}\neg(cache[i] = M \wedge cache[j] = M)$,
2. $\forall i, j \in \{1, \dots, n\}, i \neq j$: $\mathrm{G}\neg(cache[i] = O \wedge cache[j] = O)$,
3. $\forall i, j \in \{1, \dots, n\}, i \neq j$: $\mathrm{G}\neg\big(cache[i] = M \wedge (cache[j] = O \vee cache[j] = S)\big)$,

где n – количество кэшей в системе, $cache[i]$ – состояние кэш-строки в i-м кэше, $i = 1, \dots, n$.

Данное представление показывает, что логика LTL позволяет естественным образом формализовать требования к протоколам когерентности.

2.2 Выбор математической модели для представления протоколов когерентности

2.2.1 Модель протоколов когерентности

Язык описания моделей Promela предлагает абстракции, естественным образом представляющие группы устройств, работающих в соответствии с протоколом когерентности (*процессы*, являющиеся конечными автоматами), и их асинхронное взаимодействие (*каналы*, являющиеся FIFO-очередями). Поэтому математическая модель протоколов когерентности, используемая в данной работе, основана на формальной семантике Promela-моделей и базируется на модели, применяемой в работе [21].

На этапе верификации система Spin транслирует каждый процессный тип в конечный автомат (граф процесса). Поведение всего протокола строится как асинхронное произведение автоматов, построенных так, что каждому активному экземпляру процесса соответствует один автомат. В результате поведение всей системы процессов задается системой переходов, в которой параллелизм представлен недетерминированным выбором (интерливингом) [82, 21, 14]. Система переходов является стандартной моделью аппаратных и программных систем.

Разработанная в данной работе математическая модель протоколов когерентности состоит из нескольких уровней. Promela-процессам соответствуют графы процессов. Асинхронная композиция графов процессов описывается канальной системой. Таким образом, семантика Promela-моделей формализована посредством канальной системы. Путем «развертывания» канальной системы получается система переходов.

2.2.2 Система переходов

Системой переходов TS называется шестерка

$$TS = (S, Act, \rightarrow, I, AP, L),$$

где

- S – множество состояний,
- Act – множество действий,
- $\rightarrow \subseteq S \times Act \times S$ – отношение переходов,
- $I \subseteq S$ – множество начальных состояний,
- AP – множество атомарных высказываний,
- $L: S \rightarrow 2^{AP}$ – функция пометок.

Для краткости под обозначением $s \overset{\alpha}{\rightarrow} s'$ будем понимать $(s, \alpha, s') \in \rightarrow$. Если осуществляемое действие в данном контексте неважно, будем писать $s \rightarrow s'$.

2.2.3 Граф процесса

Promela-модели состоят из конечного числа процессов, работающих параллельно. Поведение процессов описывается с помощью операторов языка Promela. Множество всех переменных модели будем обозначать через Var, множество каналов модели – $Chan$.

Переменные и каналы в Promela типизированы [65]. Обозначим через $dom(x)$ тип переменной x. Аналогичное обозначение применимо и для каналов $c \in Chan$: в данном случае под типом канала $dom(c)$ понимается тип переменных, которые могут быть переданы через канал. Помимо типа, каждый канал $c \in Chan$ имеет конечную емкость $cap(c)$, то есть максимальное количество сообщений, которое он способен хранить.

Интерпретацией η переменных называется отображение, ставящее в соответствие каждой переменной $v \in Var$ значение $\eta(v) \in dom(v)$. Через $\eta[v \coloneqq r]$ будем обозначать интерпретацию переменных, присваивающую значение r переменной v и оставляющую все остальные переменные неизменными:

$$\eta[v \coloneqq r](v') = \begin{cases} \eta(v'), \text{если } v \neq v' \\ r, \text{ если } v = v' \end{cases}.$$

Обозначим через $Eval(Var)$ множество интерпретаций переменных.

Интерпретацией ξ каналов называется отображение, ставящее в соответствие каждому каналу $c \in Chan$ последовательность $\xi(c) \in dom(c)^*$ такую, что $len(\xi(c)) \leq cap(c)$, где $len(\cdot)$ – длина последовательности, * – звезда Клини.

Запись интерпретации вида $\xi(c) = v_1 v_2 \dots v_k$, где $cap(c) \geq k$, показывает, что элемент v_1 находится в голове очереди c, элемент v_2 – следующий элемент и т.д., элемент v_k находится в хвосте очереди. Обозначим через $\xi[c \coloneqq v_1 \dots v_k]$ интерпретацию каналов, присваивающую последовательность $v_1 \dots v_k$ каналу c и оставляющую все остальные каналы неизменными:

$$\xi[c \coloneqq v_1 \dots v_k\,](\mathrm{c}') = \begin{cases} \xi(\mathrm{c}'), \text{если } c \neq c' \\ v_1 \dots v_k, \text{если } c = c' \end{cases}.$$

Интерпретация ξ_0 отображает канал в пустую последовательность, обозначаемую ε, то есть $\forall c \in Chan: \xi_0(c) = \varepsilon, len(\varepsilon) = 0$.

Обозначим через $Eval(Chan)$ множество всех интерпретаций каналов.

Обозначим через $Cond(Var)$ множество логических выражений относительно переменных $v \in Var$ [21], а через $Cond(Var, Chan)$ множество логических выражений относительно переменных $v \in Var$ и каналов $c \in Chan$.

Выражения относительно каналов строятся из вызовов функций $empty()$, $nempty()$, $full()$, $nfull()$ и их объединения с помощью операторов языка Promela (таблица 1).

Таблица 1 – Предикаты опроса состояния канала $c \in Chan$

Функция	Возвращаемое значение
$empty(c)$	$true$, если $len(\xi(c)) = 0$ $false$, если $len(\xi(c)) \neq 0$
$full(c)$	$true$, если $len(\xi(c)) = cap(c)$ $false$, если $len(\xi(c)) \neq cap(c)$
$nempty(c)$	$false$, если $len(\xi(c)) = 0$ $true$, если $len(\xi(c)) \neq 0$
$nfull(c)$	$false$, если $len(\xi(c)) = cap(c)$ $true$, если $len(\xi(c)) \neq cap(c)$

Отрицания $\neg empty(c)$ и $\neg full(c)$ не определены, вместо них используются функции $nempty(c)$ и $nfull(c)$ соответственно.

Формальная семантика оператора языка Promela с переменными из множества Var и каналами из множества $Chan$ представляется графом процесса над $(Var, Chan)$ – ориентированным графом, ребра которого помечены условиями над элементами $(v, c) \in Var \times Chan$ и действиями. Вершины графа процесса носят управляющую функцию: они показывают возможные переходы.

Множество действий будем рассматривать как объединение $Act \cup Comm$, элементы которого определяются только шестью базовыми операторами языка Promela:

- присваивание,
- оператор assert,
- оператор print,

- выражение,
- отправка сообщения в канал,
- извлечение сообщения из канала.

Действия, выраженные первыми четырьмя операторами, составляют множество Act. Действия, выраженные последними двумя операторами, являются коммуникационными действиями:

- $c!\,v$ – передать значение v по каналу c,
- $c?\,x$ – получить сообщение по каналу c и присвоить его переменной x.

Множество коммуникационных действий:

$$Comm = \{c!\,v, c?\,x \mid c \in Chan, v \in dom(c), x \in Var, dom(x) \supseteq dom(c)\}.$$

Графом процесса PG, или процессом, над $(Var, Chan)$ называется шестерка

$$PG = (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0),$$

где

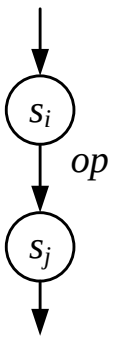
- Loc – множество состояний (вершин) графа,
- Act – множество действий,
- $Effect: Act \times Eval(Var) \to Eval(Var)$ – функция, определяющая результат действий,
- $\hookrightarrow \in Loc \times Cond(Var, Chan) \times (Act \cup Comm) \times Loc$ – отношение переходов,
- $Loc_0 \subseteq Loc$ – множество начальных состояний,
- $g_0 \in Cond(Var, Chan)$ – начальное условие.

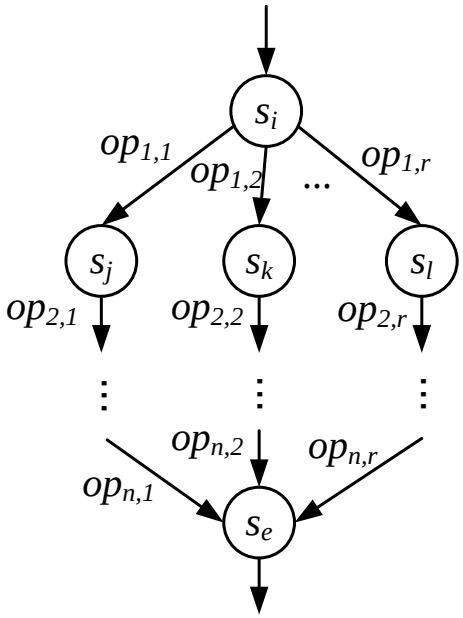
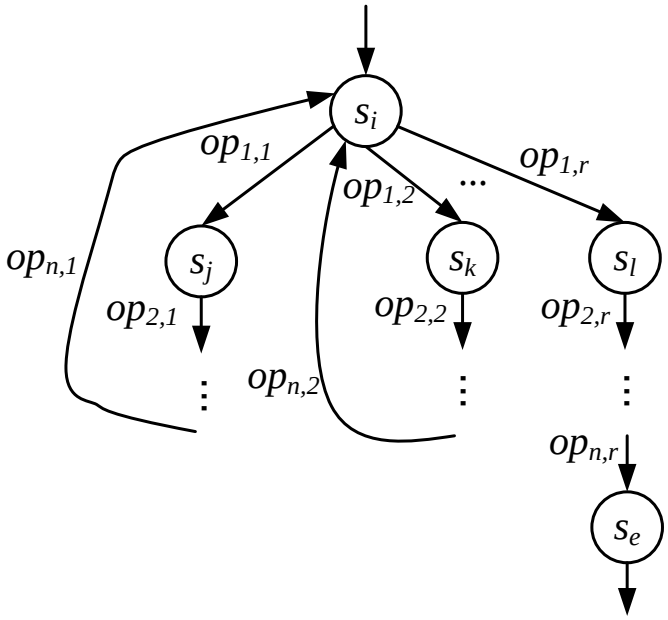
Переход $(l, g, act, l') \in \hookrightarrow$ будем сокращенно обозначать $l \overset{g:act}{\hookrightarrow} l'$.

Логическое условие g называется *защитой* перехода $l \overset{g:act}{\hookrightarrow} l'$. Действие act может быть осуществлено только в том случае, если защита g истинна, а действие выполнимо. Защита может быть тавтологией (например, $g = true$), в случае чего будем писать $l \overset{act}{\hookrightarrow} l'$. В случае, если из некоторого состояния существует несколько переходов с истинными защитами, осуществляется недетерминированный выбор. Если нет ни одного такого перехода, то модель останавливается. Заметим, что для протоколов когерентности останов означает ошибку: протоколы когерентности относятся к классу реагирующих систем, работа которых никогда не завершается.

В таблице 2 приведены фрагменты графов процессов, соответствующие операторам языка Promela. Данная информация будет использована далее при построении фрагментов исходной и абстрактной систем переходов для доказательства корректности абстрактных преобразований.

Таблица 2 – Соответствие фрагментов Promela-моделей фрагментам графов процессов

Фрагмент Promela-модели	Фрагмент графа процесса
op – оператор присваивания, выражение, assert, print, коммуникационное действие, то есть $op \in Act \cup Comm$	

Фрагмент Promela-модели	Фрагмент графа процесса
Оператор недетерминированного выбора *if* *:: $op_{1,1}$; …; $op_{n,1}$* *:: $op_{1,2}$; …; $op_{n,2}$* … *:: $op_{1,r}$; …; $op_{n,r}$* *fi* Здесь для всех i, j: $op_{i,j} \in Act \cup Comm$	
Оператор повторения *do* *:: $op_{1,1}$; …; $op_{n,1}$* *:: $op_{1,2}$; …; $op_{n,2}$* … *:: $op_{1,r}$; …; $op_{n,r}$; break* *do* Здесь для всех i, j: $op_{i,j} \in Act \cup Comm$	

Получение перехода $l \overset{g:act}{\longrightarrow} l'$ с защитой, не являющейся тавтологией, возможно при использовании конструкции atomic так, чтобы вычисление защиты и следующее за ней действие происходили неделимо, например:

$$atomic\,\{(x > y) -> x = x - y\}$$

2.2.4 Канальная система

Канальной системой CS над $(Var, Chan)$ называется множество графов процессов PG_i над $(Var_i, Chan)$, где $1 \leq i \leq n$ и $Var = \bigcup_{1 \leq i \leq n} Var_i$. Обозначение: $CS = [PG_1 \mid \ldots \mid PG_n]$.

В Promela возможны два типа взаимодействия между процессами посредством каналов: синхронная передача сообщения через канал нулевой емкости и асинхронная передача сообщения через канал ненулевой емкости. При разработке моделей протоколов когерентности, не найдено необходимости использования синхронной передачи сообщений, поэтому в дальнейшем такой тип взаимодействия рассматриваться не будет.

Семантика канальной системы формализуется посредством системы переходов. Пусть $CS = [PG_1 \mid \ldots \mid PG_n]$ – канальная система над $(Var, Chan)$. Состояния соответствующей системы переходов $TS(CS)$ являются кортежами вида

$$\langle l_1, \ldots, l_n, \eta, \xi \rangle,$$

где l_i – состояние графа PG_i, $\eta \in Eval(Var)$ – текущая интерпретация переменных, $\xi \in Eval(Chan)$ – интерпретация каналов.

Пусть $CS = [PG_1 \mid \ldots \mid PG_n]$ – канальная система над $(Var, Chan)$, и

$$PG_i = (Loc_i, Act_i, Effect_i, \hookrightarrow_i, Loc_{0,i}, g_{0,i}),\ 1 \leq i \leq n.$$

Системой переходов $TS(CS)$, соответствующей данной канальной системе, называется шестерка

$$TS(CS) = (S, Act, \rightarrow, I, AP, L),$$

где

- $S = (Loc_1 \times \ldots \times Loc_n) \times Eval(Var) \times Eval(Chan)$,
- $Act = \bigcup_{0<i\leq n} Act_i \cup \{\tau\}$, где τ – специальный символ, представляющий все коммуникационные действия, при которых происходит обмен данными,
- $\rightarrow$ определяется правилами, представленными ниже,
- $I = \{\langle l_1, \ldots, l_n, \eta, \xi_0\rangle \mid \forall 0 < i \leq n: \left(l_i \in Loc_{0,i} \wedge (\eta, \xi_0) \vDash g_{0,i}\right)$,
- Множество атомарных высказываний состоит из множества состояний графов процессов (для возможности спецификации того, в какой точке управления находится система) и множества логических условий относительно переменных и каналов:

$$AP = \bigcup_{0<i\leq n} Loc_i \cup Cond(Var, Chan),$$

- Состояния помечаются состояниями отдельных графов процессов и логическими условиями, которые выполняются над (η, ξ):

$$L(\langle l_1, \ldots, l_n, \eta, \xi\rangle) = \{l_1, \ldots, l_n\} \cup \{g \in Cond(Var, Chan) \mid (\eta, \xi) \vDash g\}.$$

В зависимости от интересующих свойств, в качестве пометок может быть использовано любое подмножество указанного множества AP.

Правила, определяющие отношение переходов $\rightarrow$ системы $TS(CS)$:

1. Интерливинг для $\alpha \in Act_i$:

$$\frac{l_i \overset{g:\alpha}{\hookrightarrow} l_i' \quad \wedge \quad (\eta, \xi) \vDash g}{\langle l_1, \ldots, l_i, \ldots, l_n, \eta, \xi\rangle \overset{\alpha}{\rightarrow} \langle l_1, \ldots, l_i', \ldots, l_n, \eta', \xi\rangle}, \qquad (1)$$

где $\eta' = Effect(\alpha, \eta)$.

2. Асинхронная передача сообщения для $c \in Chan, cap(c) > 0$:

2.1. Получение значения по каналу c и присваивание его переменной x:

$$\frac{l_i \xrightarrow{g:c?x} l_i' \quad \wedge \quad (\eta, \xi) \vDash g \quad \wedge \quad len\big(\xi(c)\big) = k > 0 \quad \wedge \quad \xi(c) = v_1 \dots v_k}{\langle l_1, \dots, l_i, \dots, l_n, \eta, \xi\rangle \xrightarrow{\tau} \langle l_1, \dots, l_i', \dots, l_n, \eta', \xi'\rangle}, \qquad (2)$$

где $\eta' = \eta[x \coloneqq v_1]$ и $\xi' = \xi[c \coloneqq v_2 \dots v_k]$.

2.2. Отправка значения $v \in dom(c)$ по каналу c:

$$\frac{l_i \xrightarrow{g:c!v} l_i' \quad \wedge \quad (\eta, \xi) \vDash g \quad \wedge \quad len\big(\xi(c)\big) = k < cap(c) \quad \wedge \quad \xi(c) = v_1 \dots v_k}{\langle l_1, \dots, l_i, \dots, l_n, \eta, \xi\rangle \xrightarrow{\tau} \langle l_1, \dots, l_i', \dots, l_n, \eta, \xi'\rangle}, \qquad (3)$$

где $\xi' = \xi[c \coloneqq v_1 v_2 \dots v_k v]$.

2.3 Разработка моделей протоколов когерентности на языке Promela и определение ограничений для них

Вопрос построения формальных моделей протоколов когерентности изучен в литературе недостаточно. В данном разделе представлена математическая модель протоколов когерентности в виде канальной системы, и на основе практики верификации системы Эльбрус-4С определена структура отдельных графов процессов и изложены ограничения на используемые операторы, выполнение которых необходимо для работы предлагаемого метода.

В данной работе рассматриваются протоколы когерентности, в которых исполнение запросов происходит под управлением координирующего компонента. В микропроцессорах с архитектурой Эльбрус таким компонентом является системный коммутатор процессора-владельца адреса (то есть

процессора, к чей памяти происходит обращение). Такой процессор называется home-процессором.

В качестве математической модели протокола когерентности будем использовать канальную систему

$$CS = [PG_0 \mid PG_1 \mid \dots \mid PG_n],$$

где PG_0 – граф процесса, соответствующий системному коммутатору home-процессора, $PG_1, \dots, PG_n$ – идентичные графы процессов, соответствующие контроллерам кэш-строки, находящимся в кэшах верифицируемой системы. Каждый из графов процессов определен над парой $(Var_i, Chan_i)$ так, что множества Var_i могут пересекаться, и множества $Chan_i$ также могут пересекаться. При этом

$$Var = \bigcup_{0 \le i \le n} Var_i, \;\; Chan = \bigcup_{0 \le i \le n} Chan_i.$$

Множество переменных Var представим в виде множества подмножеств $\{Vstate_i \subseteq Var \mid i = 0, \dots, n\}$ переменных, описывающих состояние процессов $PG_i, 0 \le i \le n$. Примеры переменных, входящих в подмножество $Vstate_i$:

- локальные переменные процесса PG_i;
- переменная, хранящая состояние кэш-строки в данном кэш-контроллере;
- переменная, хранящая признак получения ответа на снуп-запрос от процесса PG_i.

Определим ограничения на структуру графов $PG_i, 0 \le i \le n$, полученные в ходе практики описания протокола системы на кристалле Эльбрус-4С.

Протоколы когерентности обладают свойством, согласно которому в один момент времени в системе может исполняться только один исходный запрос. В

работе системного коммутатора home-процессора можно выделить последовательность этапов, например, получение исходного запроса и его анализ, рассылка когерентных и других служебных запросов по результатам анализа, сбор когерентных ответов и подтверждений. Получение сообщений от других устройств возможно только на определенных этапах. В связи с этим алгоритм работы системного коммутатора удобно представить в виде Promela-процесса, в котором отношение «предыдущий-следующий» между операторами представляется естественным образом (при исполнении модели исполнение предыдущего оператора означает переход к следующему). При этом в коде такие операторы располагаются друг за другом.

Защищенные команды, которыми помечены ребра графа PG_0, строятся таким образом, что их защиты могут являться:

- логическими утверждениями относительно переменных $v \in \bigcup_{0 \le i \le n} Vstate_i$;
- тавтологиями.

Работа кэш-контроллеров осуществляется по-другому. С одной стороны, также можно выделить ряд этапов, например, отправка исходного запроса, смена состояния на переходное, прием снуп-запросов и других служебных запросов, прием когерентных ответов, смена состояния из переходного на основное. С другой стороны, относительный порядок осуществления таких этапов зачастую не фиксирован, и одни и те же сообщения от других устройств могут обрабатываться в различных состояниях кэш-контроллера. В связи с этим, структуру процессов, представляющих кэш-контроллеры, удобно представить в виде бесконечного do-цикла из защищенных команд.

Защита команды графа $PG_i, 1 \le i \le n$, может быть:

- логическим утверждением $e \in Cond(Var_i, Chan_i)$. Предполагается использование логических утверждений относительно каналов для возможности неблокирующего опроса состояния каналов с той целью, чтобы

защищенные команды всегда были выполнимыми. Например, если действие команды включает получение сообщения по каналу c, и это действие не должно быть заблокировано, то в защиту с помощью конъюнкции добавляется условие $nempty(c)$;

- тавтологией.

В связи с асинхронной природой взаимодействий между устройствами, реализующими протокол когерентности, не обнаружено необходимости в использовании синхронного взаимодействия между процессами моделей протоколов когерентности. В Promela синхронное взаимодействие моделируется посредством каналов нулевой емкости. Таким образом, все каналы, используемые в моделях в данной работе, имеют ненулевую емкость (являются FIFO-очередями).

Оператор извлечения сообщения из канала c в графе $PG_i, 0 \leq i \leq n$, имеет вид $c?x$, где $x \in Var \setminus \bigcup_{j \neq i} Vstate_j$.

Исследование путей разработки моделей протоколов когерентности памяти на примере протокола системы Эльбрус-4С показало, что множество каналов модели представимо тремя подмножествами:

- C_1 – множество каналов емкости n, в которые могут отправлять сообщения графы $PG_i, 1 \leq i \leq n$. Извлекать сообщения может только один процесс $PG_i, 0 \leq i \leq n$ на определенном этапе выполнения запроса. Таким образом, для каналов этого множества характерно соответствие множества операторов отправки одному оператору приема. Примеры таких каналов: канал, посредством которого процессы, представляющие кэш-контроллеры, передают исходные запросы процессу, представляющему системный коммутатор home-процессора; канал, по которому некоторому процессу передаются когерентные ответы.

- $C_2 = \{c_{2,1}, \dots, c_{2,n}\}$ – множество каналов емкости $m \in \mathbb{N}$, таких, что из канала $c_{2,i}, 1 \leq i \leq n$ может извлекать сообщения только PG_i, а отправлять

сообщения по этому каналу может только PG_0. Таким образом, для каналов этого множества характерна схема «один отправитель – один получатель», причем оба строго определены. Примером канала из этого множества является канал, по которому процесс, представляющий системный коммутатор home-процессора, передает когерентный запрос процессу, представляющему кэш-контроллер.

- C_3 – множество каналов, по которым может отправлять сообщения только один процесс в специфицированное «время» в ходе выполнения запроса. Для каналов этого множества характерно соответствие каждому оператору приема сообщения ровно одного оператора отправки сообщения. Примером канала из этого множества является канал, по которому процесс-запросчик передает подтверждение о завершении операции процессу-координатору.

Такое разбиение множества каналов позволило на основе особенностей операторов, представляющих коммуникационные действия, разработать различные преобразования таких операторов, приводящие к абстрактным моделям.

Сообщения, передаваемые по каналам, являются парами $m = \langle opc, id \rangle$, где $opc \in \mathbb{N}$ не является номером процесса, $id \in \{0, \dots, n\}$ является номером процесса (например, идентификатором отправителя сообщения). Для обращения к первому и второму элементам пары m будем писать $m.opc$ и $m.id$ соответственно.

2.4 Синтез совокупности преобразований, приводящих к получению абстрактной модели

2.4.1 Абстрактная модель протоколов когерентности

Проверяемые свойства протокола затрагивают не более двух кэшей. Поскольку все кэш-контроллеры идентичны и взаимозаменяемы

(соответственно, в теле Promela-процессов отсутствует код, выполнение которого зависит от конкретных значений индекса процесса), не имеет значения, какие именно два индекса графов $PG_1, \dots, PG_n$ рассматривать. Поэтому без потери общности будем считать, что рассматриваются графы PG_1 и PG_2, и все проверяемые свойства сформулированы только относительно части системы, определяемой этими графами. Например, вместо свойства

$$\forall i, j \in \{1, \dots, n\}, i \neq j: \mathrm{G}\neg(cache[i] = M \wedge cache[j] = M),$$

где $cache[i]$ обозначает состояние кэш-строки в i-м кэше, достаточно проверить выполнение свойства

$$\mathrm{G}\neg(cache[1] = M \wedge cache[2] = M).$$

В отсутствие свойств относительно $PG_3, \dots, PG_n$, с позиции графов PG_0, PG_1, PG_2, конкретное значение индекса графов $PG_3, \dots, PG_n$ неважно. Поэтому можем выполнить следующее консервативное преобразование системы CS.

Предполагается, что в исходной системе среди переменных $v \in Vstate_i, 0 \leq i \leq n$, хранящих информацию о других процессах, могут быть только переменные, хранящие номер процесса, то есть переменные, областью определения которых является множество $\{0, \dots, n\}$. В случае, когда полностью рассматривается только состояние процессов $PG_i, 0 \leq i \leq 2$, важно сохранять точные значения лишь для случаев, когда это значение принадлежит множеству $\{0,1,2\}$; все остальные значения неразличимы, и их можно представить некоторым абстрактным значением, не находящимся в этом множестве. Поэтому применим абстракцию типов данных. Множества значений переменных $v \in Var$ и каналов $c \in Chan$, таких, что $dom(v) = \{0, \dots, n\}$ и

$dom(c.id) = \{0, \dots, n\}$ соответственно, заменяются на $dom_{abs}(v) = \{0,1,2, ABS\}$ и $dom_{abs}(c.id) = \{0,1,2, ABS\}$, соответственно, где $ABS > 2$ – некоторая константа. Для всех остальных случаев $dom_{abs}(v) = dom(v), dom_{abs}(c) = dom(c)$. Далее под исходной системой будем понимать преобразованную таким образом систему CS.

В соответствии с предлагаемым методом, после проведения абстракции типов данных осуществляется замена исходной модели $CS = [PG_0 \mid PG_1 \mid \dots \mid PG_n]$ на модель (рисунок 6)

$$CS_{abs} = [PG_0 \mid PG_1 \mid PG_2 \mid PG_3],$$

где граф PG_3 соответствует абстрактному процессу.

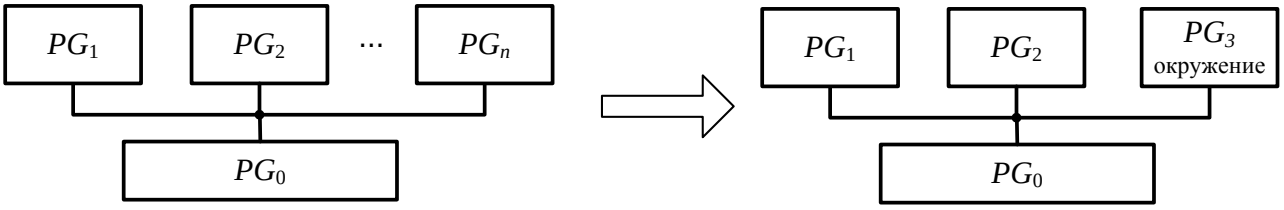


Рисунок 6 – Трансформация канальной системы при абстрактных преобразованиях

Путь получения абстрактной канальной системы основан на следующем положении: абстрактная система переходов допускает все возможные поведения процессов, соответствующих системному коммутатору home-процессора и двум кэш-контроллерам, включая поведения, вызванные воздействиями процессов, представляющих остальные кэш-контроллеры, на выделенные три процесса. Такие поведения моделируются с помощью графа PG_3 в системе CS_{abs}, а также с помощью определенных преобразований графов PG_0, PG_1, PG_2 исходной системы CS.

Изначально графы $PG_i, 0 \leq i \leq 3$ системы CS_{abs} являются графами $PG_i, 0 \leq i \leq 3$ исходной системы CS, соответственно.

Пометим множество состояний системы переходов $TS_{abs} = TS(CS_{abs})$ так, чтобы пометка полностью отражала состояние графов $PG_i, 0 \leq i \leq 2$ исходной системы $TS(CS)$, а также состояние графов $PG_i, i > 2$, непосредственно используемое при модификации состояния графов $PG_i, 0 \leq i \leq 2$.

Выполним пояснения. В графах процессов $PG_i, 2 < i \leq n$ могут быть переменные $v \in \bigcup_{2<i\leq n} Vstate_i$, которые непосредственно участвуют в изменении значений переменных из $\bigcup_{0\leq i\leq 2} Vstate_i$. Обозначим множество таких переменных V_{loc}, $V_{loc} \subseteq \bigcup_{2<i\leq n} Vstate_i$. Значения таких переменных модифицируются операциями извлечения сообщения из канала $c \in C_1$. Например,

// message $\in \bigcup_{2<i\leq n} Vstate_i$

coh_answers_chan ? message;

// ack_list[message.id] $\in \bigcup_{0\leq i\leq 2} Vstate_i$, если message.id $= 1,2$

ack_list[message.id] = false;

В ходе исполнения исходного запроса релевантны значения таких переменных *только одного процесса*. Введем множество переменных V_{loca}, такое, что его элементы всегда находятся во взаимно-однозначном соответствии с множеством рассмотренных локальных переменных того из процессов, который производит модификацию переменных из $\bigcup_{0\leq i\leq 2} Vstate_i$:

$$V_{loca} \leftrightarrow Vstate_i$$

для некоторого $2 < i \leq n$.

Таким образом, в качестве пометок состояния абстрактной TS_{abs} и исходной TS систем выступает следующее множество:

$$AP = Cond(V_{AP}),$$

где $V_{AP} = \bigcup_{0 \le i \le 2} Vstate_i \cup V_{loca}$.

Модификация графов процессов $PG_i, 0 \le i \le 3$ осуществляется посредством синтаксических преобразований, описанных в пп. 2.4.2–2.4.3.

2.4.2 Абстрактные преобразования элементов множеств Act_i

Множество Act, являющееся одним из компонентов графа процесса, состоит из четырех типов элементов: присваивание, оператор assert, оператор print, выражение. Множество выражений ограничим множеством $Cond(Var, Chan)$. Опишем преобразования присваиваний (таблица 3) и выражений. Преобразование операторов assert и print осуществляется посредством преобразования выражений, используемых в этих операторах. Символ $\emptyset$ означает отсутствие действия.

Таблица 3 – Преобразование присваиваний

Оператор в исходной модели	Оператор в абстрактной модели
$v = val\ (v \in Var, val \in dom(v))$	$v = val$, если $v \in \bigcup_{i=0,1,2} Vstate_i$, $\emptyset$, если $v \in \bigcup_{2<i\le n} Vstate_i$

Других операторов присваивания нет.

Выражение $(v, c) \in Cond(V, C)$ в абстрактной модели принимает следующий вид:

(v, c), если $V = \bigcup_{i=0,1,2} Vstate_i \cup V_{loc}$, $C = C_1 \cup \{c_{2,1}, c_{2,2}\} \cup C_3$.

$(v, true)$, если $V = \bigcup_{i=0,1,2} Vstate_i \cup V_{loc}$, $C = Chan \backslash C_1 \cup \{c_{2,1}, c_{2,2}\} \cup C_3$.

$(true, c)$, если $V = \bigcup_{2<i\le n} Vstate_i \backslash V_{loc}$, $C = C_1 \cup \{c_{2,1}, c_{2,2}\} \cup C_3$.

$(true, true)$, если $V = \bigcup_{2<i\le n} Vstate_i \backslash V_{loc}$, $C = Chan \backslash C_1 \cup \{c_{2,1}, c_{2,2}\} \cup C_3$.

2.4.3 Абстрактные преобразования элементов множеств $Comm_i$

Преобразования коммуникационных действий можно проводить различными способами. Например, можно удалять только те операторы извлечения сообщений и соответствующие им операторы отправки сообщений, которые не изменяют элементов множества V_{AP}. Однако для возможности проведения верификации на практике необходимо ограничивать емкость каналов и их количество небольшими значениями (3–4), что приводит к необходимости удаления некоторых операторов, изменяющих состояние переменных из V_{AP}.

Соответственно, преобразование коммуникационных действий исходит из следующих соображений. Если оператор приема сообщения не изменяет пометку состояния, то его можно удалить, как и соответствующее множество операторов отправки сообщений. Если удаляется оператор отправки сообщения, такой, что соответствующий ему оператор приема сообщения изменяет пометку, то необходимы дополнительные меры. Разработанные преобразования и принятые меры приведены в таблицах 4–6.

Таблица 4 – Преобразование коммуникационных действий для $c \in C_1$

Оператор в исходной модели	Оператор в абстрактной модели
$c!\,v\ (v \in dom(c))$	$c!\,v$, если оператор находится в PG_1, PG_2 $\emptyset$, если оператор находится в PG_3
$c?\,x\ (x \in Var, dom(x) \supseteq dom(c))$	Недетерминированный выбор

Замена на недетерминированный выбор осуществляется следующим образом. Если оператор

atomic { guard_abs –> c?x; }

находится в PG_0, то он заменяется на оператор недетерминированного выбора

if

:: atomic { guard_abs –> c?x; }

:: atomic { m.opc = opc$_1$; m.id = ABS; }

…

:: atomic { m.opc = opc$_k$; m.id = ABS; }

fi;

так, что множество $\{opc_1, \dots, opc_k\} \subseteq dom_{abs}(m.opc)$ содержит в себе все возможные значения, которые в исходной модели могли быть отправлены посредством соответствующих операторов отправки сообщения, находящихся в $PG_3, \dots, PG_n$. Здесь $guard\_abs$ получается согласно преобразованиям выражений.

Если оператор находится внутри действия защищенной команды do-цикла, то в do-цикл добавляется множество защищенных команд, защита которых получается из защиты исходной команды заменой на истину всех условий относительно канала c, а действия формируются присваиваниями $m.opc = opc_i; m.id = ABS, 1 \le i \le k$.

Таблица 5 – Преобразование коммуникационных действий для $c \in C_2$

Оператор в исходной модели	Оператор в абстрактной модели
$c!\,v\ (v \in dom(c))$	$c!\,v$, если $c = c_{2,1}$ или $c = c_{2,2}$ $\emptyset$, если $c \in C_2 \backslash \{c_{2,1}, c_{2,2}\}$
$c?\,x\ (x \in Var, dom(x) \supseteq dom(c))$	$c?\,x$, если $c = c_{2,1}$ или $c = c_{2,2}$ $\emptyset$, если $c \in C_2 \backslash \{c_{2,1}, c_{2,2}\}$

Таблица 6 – Преобразование коммуникационных действий для $c \in C_3$

Оператор в исходной модели	Оператор в абстрактной модели
$c!\,v\ (v \in dom(c))$	$c!\,v$
$c?\,x\ (x \in Var, dom(x) \supseteq dom(c))$	$c?\,x$

2.5 Математическое доказательство корректности процедуры абстракции

В работе рассматривается проверка свойств-инвариантов, описываемых LTL-формулой $\mathrm{G}p$, где p – формула пропозициональной логики. Проверка выполнимости такого свойства для данной системы переходов TS ($TS \vDash \mathrm{G}p$) эквивалентна проверке выполнения свойства p в каждом из достижимых из некоторого начального состояния состояний системы TS. Обозначим множество состояний системы переходов TS, достижимых из некоторого начального состояния, через $Reach(TS)$.

Путем системы переходов $TS = (S, Act, \rightarrow, I, AP, L)$ без терминальных состояний называется бесконечная последовательность состояний $s_0 s_1 s_2$ ..., такая, что $\forall i > 0$: $(s_{i-1} \rightarrow s_i)$, и $s_0 \in I$.

Вычислением системы переходов $TS = (S, Act, \rightarrow, I, AP, L)$ называется бесконечная чередующаяся последовательность состояний и действий $s_0 \alpha_1 s_1 \alpha_2 s_2$ ..., такая, что $\forall i > 0$: $(s_{i-1} \xrightarrow{\alpha_i} s_i)$, и $s_0 \in I$.

В следующей теореме доказано, что множество достижимых состояний системы $TS_{abs} = (S_{abs}, Act_{abs}, \rightarrow_{abs}, I_{abs}, AP, L_{abs})$ включает в себя множество достижимых состояний системы $TS = (S, Act, \rightarrow, I, AP, L)$.

Теорема. Пусть p – некоторая формула пропозициональной логики, составленная относительно атомарных высказываний из множества AP. Для всех состояний $s \in S$, таких, что s достижимо из некоторого начального состояния $s_0 \in I$ и $s \vDash p$, существует состояние $f(s) \in S_{abs}$, достижимое из некоторого начального состояния $f(s_0) \in I_{abs}$, такое, что $f(s) \vDash p$:

$$(\forall s \in Reach(TS): s \vDash p): (\exists f(s) \in Reach(TS_{abs}): f(s) \vDash p).$$

Доказательство.

Заметим, что формула p выполнима в состоянии s тогда и только тогда, когда подстановка в данную формулу значений из множества пометок этого состояния $L(s)$ превращает ее в истинное высказывание:

$$(s \vDash p) \Leftrightarrow (L(s) \vDash p).$$

Отсюда следует, что если пометки состояний s_1 и s_2 одинаковы, то $(s_1 \vDash p) \Leftrightarrow (s_2 \vDash p)$.

Доказательство утверждения теоремы проведено методом математической индукции по длине пути системы TS.

Базис индукции. Длина пути $m = 0$. По построению абстракции, каждому состоянию $s \in I$ соответствует состояние $f(s) \in I_{abs}$, такое, что $L_{abs}\big(f(s)\big) = L(s)$.

Индуктивная гипотеза. Пусть для некоторого $m \geq 0$ верно:

$$\forall k \leq m: f(s_k) \in Reach(TS_{abs}).$$

Это означает, что фрагменту пути $s_0 s_1 \dots s_m$ системы TS соответствует фрагмент пути $f(s_0)f(s_1) \dots f(s_m)$ системы TS_{abs} (рисунок 7), причем $f(s_i)$ и $f(s_{i+1})$, $0 \leq i < m$, не обязательно различны: если $f(s_i) = f(s_{i+1})$, то $L(s_i) = L(s_{i+1})$. Помимо этого, для всех $k \leq m$ пометки состояния s_k исходной системы и состояния $f(s_k)$ абстрактной системы совпадают:

$$L_{abs}\big(f(s_k)\big) = L(s_k).$$

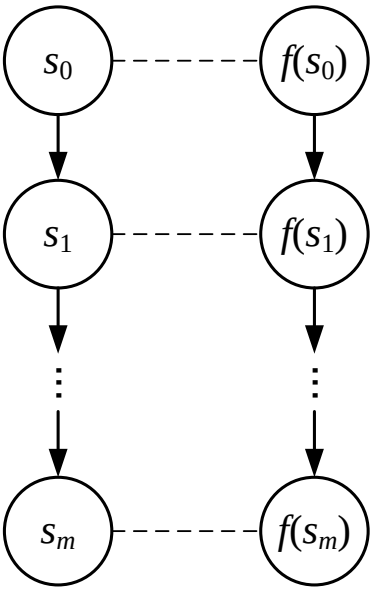


Рисунок 7 – Соответствие фрагментов путей в исходной и абстрактной системах (индуктивная гипотеза)

Шаг индукции.

Рассмотрим переход $s_m \to s_{m+1}$ исходной системы. Покажем, что в абстрактной системе TS_{abs} существует переход $f(s_m) \to f(s_{m+1})$, такой, что $L_{abs}\big(f(s_{m+1})\big) = L(s_{m+1})$.

Состояние s_{m+1} исходной системы TS является следствием наличия в данной системе перехода из состояния s_m, описываемого одним из правил (1) – (3), приведенных в п.2.2.4. Рассмотрим все возможные правила и проверим наличие соответствующих правил в абстрактной системе.

Случай 1. Интерливинг для $\alpha \in Act_i, 0 \le i \le n$. Элемент множества правил исходной системы $Rulesi_{concrete} = \{ric_i \mid 0 \le i \le n\}$ имеет вид:

$$ric_i = \frac{l_i \overset{g:\alpha}{\hookrightarrow} l_i' \quad \wedge \quad (\eta, \xi) \vDash g}{s_m = \langle l_0, \dots, l_i, \dots, l_n, \eta, \xi \rangle \overset{\alpha}{\to} s_{m+1} = \langle l_0, \dots, l_i', \dots, l_n, \eta', \xi \rangle},$$

где $\eta' = Effect(\alpha, \eta)$.

Соответствующий элемент множества правил абстрактной системы

$Rulesi_{abstract} = \{ria_i \mid 0 \le i \le 3\}$ имеет вид:

$$ria_i = \frac{l_i \xrightarrow{g_{abs}:\alpha_{abs}} l_i' \quad \wedge \quad (\eta_{abs}, \xi_{abs}) \vDash g_{abs}}{f(s_m) = \langle l_0, \dots, l_i, \dots, l_3, \eta_{abs}, \xi_{abs} \rangle \xrightarrow{\alpha_{abs}} f(s_{m+1}) = \langle l_0, \dots, l_i', \dots, l_3, \eta_{abs}', \xi_{abs} \rangle},$$

где $\eta_{abs}' = Effect(\alpha_{abs}, \eta_{abs})$.

Для установления соответствия правил исходной и абстрактной систем введем функцию $Transi$: $Rulesi_{concrete} \to Rulesi_{abstract}$.

Действие α может быть присваиванием или выражением. Если α – присваивание, то в соответствии с абстрактными преобразованиями:

- $\alpha_{abs} = \alpha$ (с учетом осуществленной на начальном шаге абстракции типов данных), если присваивание α изменяет значения переменных из V_{AP}, то есть $L(s_{m+1}) \neq L(s_m)$, – такими переменными могут быть только переменные из $\bigcup_{i=0,1,2} Vstate_i$. Если α – действие, совершенное одним из процессов системы CS с номерами $i = 0,1,2$, то α_{abs} – действие, совершенное процессом абстрактной системы CS_{abs} с таким же номером. Если же $i > 2$, то в CS_{abs} действие α_{abs} совершается абстрактным процессом PG_3. Таким образом,

$$Transi(ric_i) = ria_i, \text{ если } i = 0,1,2,$$
$$Transi(ric_i) = ria_3, \text{ если } i > 2 \wedge L(s_{m+1}) \neq L(s_m).$$

- $\alpha_{abs} = \emptyset$, если присваивание α не изменяет значения переменных из V_{AP}, то есть $L(s_{m+1}) = L(s_m)$. Поэтому

$$Transi(ric_i) = \emptyset, \text{ если } i > 2 \wedge L(s_{m+1}) = L(s_m).$$

Покажем, что если второй элемент конъюнкции посылки правила

системы CS истинен, то второй элемент посылки соответствующего правила системы CS_{abs} также истинен.

В исходной системе имеем условие $(\eta, \xi) \vDash g$, причем, согласно ограничениям на модель, в g не входят логические утверждения относительно каналов. В соответствии с индуктивной гипотезой, в состоянии $f(s_m)$ выполняется:

$$\forall v \in V_{AP}: \eta_{abs}(v) = \eta(v) \in dom_{abs}(v).$$

В тоже время защиту g можно представить следующим образом:

$$g = A_1 \wedge A_2,$$

где $A_1 \in Cond(V_{AP}, C)$, и $A_2 \notin Cond(V_{AP}, C)$, а $C = \emptyset$.

В соответствии с абстрактными преобразованиями, защита g_{abs} имеет вид:

$$g_{abs} = A_1 \wedge true,$$

то есть часть $A_2 \notin Cond(V_{AP}, C)$ заменена на истину.

Таким образом, защита g_{abs} является логическим следствием защиты g:

$$\vdash (g \Rightarrow g_{abs}).$$

Из вида g, g_{abs} и соотношений (1) следует, что

$$\vdash \Big(\big((\eta, \xi) \vDash g \big) \Rightarrow \big((\eta_{abs}, \xi_{abs}) \vDash g_{abs} \big) \Big).$$

Доказанное утверждение, а также определение функции $Transi$ позволяют сделать следующие заключения.

1. В случае, когда $\alpha_{abs} = \alpha$, переходу $s_m \xrightarrow{\alpha} s_{m+1}$ исходной системы TS соответствует переход $f(s_m) \xrightarrow{\alpha_{abs}} f(s_{m+1})$ в абстрактной системе TS_{abs}, вследствие чего $L_{abs}\big(f(s_{m+1})\big) = L(s_{m+1})$ (рисунок 8).

2. В случае, когда $\alpha_{abs} = \emptyset$, в абстрактной системе двум состояниям исходной системы s_{m+1} и s_m с одинаковыми пометками соответствует одно состояние. Заключаем, что в TS_{abs} достижимо состояние $f(s_{m+1}) = f(s_m)$ (рисунок 9).

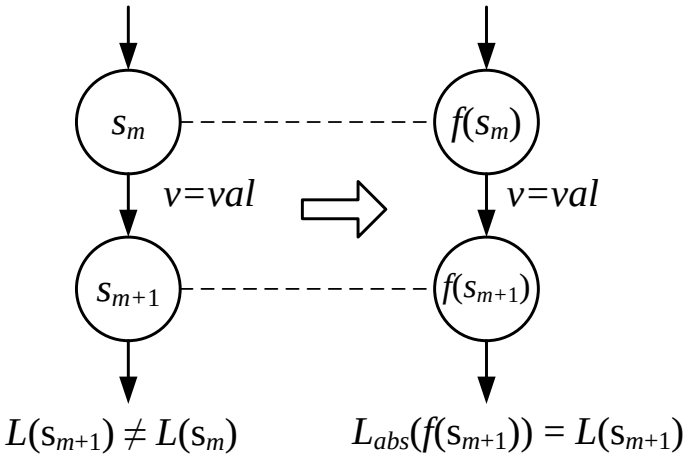


Рисунок 8 – Соответствие перехода между двумя состояниями с разными пометками в исходной системе переходу в абстрактной системе при преобразовании присваиваний

Если α – выражение, то переход, полученный вследствие применения соответствующего правила, не изменяет пометку состояния. В данном случае состояниям s_m и s_{m+1} в абстрактной системе соответствует либо одно состояние, либо два разных состояния с одинаковыми пометками, причем $(\alpha \Rightarrow a_{abs})$, аналогично защитам g и g_{abs}.

Случай 1 рассмотрен.

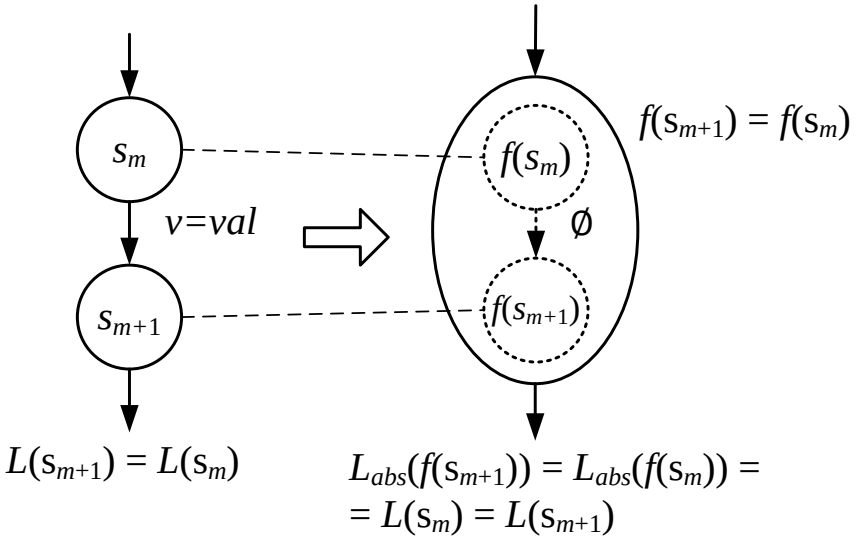


Рисунок 9 – Отображение двух состояний с одинаковыми пометками в исходной системе в одно состояние в абстрактной системе при преобразовании присваиваний

Случай 2. Получение значения по каналу $c \in Chan, cap(c) > 0$ и присваивание его переменной x.

Случай 2.1. $c \in C_1$.

Элемент множества правил исходной системы $Rulesr_{concrete1} = \{rrc_i \mid 0 \le i \le n\}$ имеет вид:

$$rrc_i = \frac{l_i \xrightarrow{g:c?x} l_i' \quad \wedge \quad (\eta, \xi) \vDash g \quad \wedge \quad len\big(\xi(c)\big) = k > 0 \quad \wedge \quad \xi(c) = v_1 \dots v_k}{\langle l_0, \dots, l_i, \dots, l_n, \eta, \xi\rangle \xrightarrow{\tau} \langle l_0, \dots, l_i', \dots, l_n, \eta', \xi'\rangle},$$

где $\eta' = \eta[x := v_1]$ и $\xi' = \xi[c := v_2 \dots v_k]$.

В ходе исполнения запроса только одно правило $rrc_i, 0 \le i \le n$ может порождать переходы. Изменение пометки в этих случаях обусловлено изменением значения переменной x. Определим, какие значения может принимать x вследствие применения правила $rrc_i, 0 \le i \le n$.

Рассмотрим фрагмент вычисления $s_0\alpha_1 s_1\alpha_2 \dots \alpha_m s_m$ системы переходов

TS. В рассматриваемом случае переход $s_m \xrightarrow{\tau} s_{m+1}$ получен вследствие применения правила $rrc_i \in Rulesr_{concrete1}$. В рассматриваемых в диссертации моделях либо $i = 0$ (сообщение извлекает процесс, представляющий системный коммутатор home-процессора), либо $i > 0$ (сообщение извлекает процесс, являющийся запросчиком; i известно с самого начала исполнения запроса). Факт применения правила означает, что условие $len(\xi(c)) = k > 0$ выполнено. (Это, в свою очередь, позволяет записать $\xi(c) = v_1 \dots v_k$ в соответствии с определением $len(\cdot)$.) Следовательно, среди $\alpha_1, \dots, \alpha_m$ находится k членов, каждый из которых имеет вид $\alpha_i = \tau$, для некоторого $1 \le i \le m$.

Таким образом, после осуществления перехода $s_m \xrightarrow{\tau} s_{m+1}$, x будет принимать одно из значений из множества всех ранее отправленных по каналу значений:

$$X = \{v \mid \text{среди } \alpha_i \text{ была пометка } \tau, \text{порожденная действием } c!\, v\},$$

причем $v_1 \dots v_k$ является перестановкой этого множества.

Указанному множеству правил $Rulesr_{concrete1}$ в абстрактной модели соответствует множество правил $Rulesr_{abstract1} = \{rra_i, raa_{i,j} \mid 0 \le i \le 3, 1 \le j \le S_1\}$, где $S_1 \le |dom_{abs}(c)|$ – количество различных отправляемых значений в операторах отправки сообщения, соответствующих операторам приема сообщения из $c \in C_1$. Выбор правила из этого множества недетерминирован, то есть

$$Transr_1 : Rulesr_{concrete1} \to Rulesr_{abstract1}$$

является соответствием. Первый тип элементов этого множества имеет вид:

rra_i

$$= \frac{l_i \xrightarrow{g_{abs}:c?x} l_i' \quad \wedge \quad (\eta_{abs}, \xi_{abs}) \vDash g_{abs} \quad \wedge \quad len\big(\xi_{abs}(c)\big) = l > 0 \quad \wedge \quad \xi_{abs}(c) = v_1 \dots v_l}{\langle l_0, \dots, l_i, \dots, l_3, \eta_{abs}, \xi_{abs}\rangle \xrightarrow{\tau} \langle l_0, \dots, l_i', \dots, l_3, \eta_{abs}', \xi_{abs}'\rangle},$$

где $\eta_{abs}' = \eta_{abs}[x := v_1]$ и $\xi_{abs}' = \xi_{abs}[c := v_2 \dots v_l]$. Здесь l может принимать значения 1,2.

Из абстрактных преобразований следует, что

- операторы извлечения сообщения из канала $c \in C_1$, находящиеся в процессах $PG_0, \dots, PG_2$ исходной модели, остаются в соответствующих процессах абстрактной модели:

$$Transr_1(rrc_i) = rra_i, 0 \le i \le 2,$$

- операторам извлечения сообщения из канала $c \in C_1$, находящимся в процессах $PG_3, \dots, PG_n$ исходной модели, соответствует оператор в процессе PG_3 абстрактной модели:

$$Transr_1(rrc_i) = rra_3, 2 < i \le n.$$

Определим, какие значения при этом может принимать переменная x в абстрактной модели в состоянии $f(s_{m+1})$.

Рассмотрим фрагмент вычисления $f(s_0)\mathrm{A}(\alpha_1)f(s_1)\mathrm{A}(\alpha_2) \dots \mathrm{A}(\alpha_m)f(s_m)$ системы переходов TS_{abs}. Здесь $\mathrm{A}: Act \cup \{\tau\} \to Act_{abs} \cup \{\tau\}$. Условие $len\big(\xi_{abs}(c)\big) = l > 0$ может быть выполнено только в том случае, если среди $\mathrm{A}(\alpha_1), \dots, \mathrm{A}(\alpha_m)$ было l членов, каждый из которых порожден действием $c!v$ и имеет вид $\mathrm{A}(\alpha_i) = \tau$, для некоторого $1 \le i \le m$. Рассмотрим, как $\mathrm{A}(\alpha_i)$ связан с α_i.

Для случая отправки сообщения по каналу $c \in C_1$ множество правил исходной системы $Ruless_{concrete1} = \{rsc_i \mid 1 \le i \le n\}$, где

$$rsc_i = \frac{l_i \overset{g:c!v}{\longrightarrow} l_i' \quad \wedge \quad (\eta, \xi) \vDash g \quad \wedge \quad len\big(\xi(c)\big) = k < cap(c) \quad \wedge \quad \xi(c) = v_1 \dots v_k}{\langle l_0, \dots, l_i, \dots, l_n, \eta, \xi \rangle \overset{\tau}{\to} \langle l_0, \dots, l_i', \dots, l_n, \eta, \xi' \rangle},$$

где $\xi' = \xi[c \coloneqq v_1 v_2 \dots v_k v]$.

Множество правил абстрактной системы $Ruless_{abstract1} = \{rsa_i \mid 1 \le i \le 2\}$, где

$$rsa_i = \frac{l_i \overset{g_{abs}:c!v}{\longrightarrow} l_i' \ \wedge \ (\eta_{abs}, \xi_{abs}) \vDash g_{abs} \ len\big(\xi_{abs}(c)\big) = l < cap_{abs}(c) \wedge \xi_{abs}(c) = v_1 \dots v_l}{\langle l_0, \dots, l_i, \dots, l_3, \eta, \xi \rangle \overset{\tau}{\to} \langle l_0, \dots, l_i', \dots, l_3, \eta, \xi' \rangle},$$

где $\xi'_{abs} = \xi_{abs}[c \coloneqq v_1 v_2 \dots v_l v]$, $cap_{abs}(c) = 2$.

В соответствии с преобразованиями абстракции, отображение $Transs_1 : Ruless_{concrete1} \to Ruless_{abstract1}$ определено следующим образом.

$Transs_1(rsc_i) = rsa_i, i = 1,2$, поскольку соответствующий оператор отправки сообщения сохраняется в процессах PG_1, PG_2.

$Transs_1(rsc_i) = \emptyset, i > 2$ в связи с удалением соответствующих операторов отправки сообщения.

Из этого следует, что $\mathrm{A}(\tau) = \tau$, если соответствующая метка порождена действиями $c!\,v$ графов PG_1, PG_2, и $\mathrm{A}(\tau) = \emptyset$, в противном случае. Другими словами, в тех случаях, когда в исходной системе был переход $s_k \overset{\tau}{\to} s_{k+1}, k < m$, полученный вследствие применения правила $rsc_i, i > 2$, в абстрактной системе соответствующие состояния объединяются: $f(s_k) = f(s_{k+1})$.

Таким образом, переменная x может принимать значения из множества

$$X_{abs} = \{v \mid \text{среди } \mathrm{A}(\alpha_i) \text{ была пометка } \tau, \text{порожденная действием } c!\,v\},$$

то есть только те значения из множества X, которые были отправлены графами PG_1 и PG_2 (это одни и те же значения и в исходной, и в абстрактной моделях).

Все остальные случаи в абстрактной модели представлены оставшимися элементами рассматриваемого множества правил, которые имеют вид:

$$raa_{i,j} = \frac{l_i \xrightarrow{g_{abs}:\, x.opc:=v, x.id:=ABS} l_i' \quad \wedge \quad (\eta_{abs}, \xi_{abs}) \vDash g_{abs}}{\langle l_0, \dots, l_i, \dots, l_3, \eta_{abs}, \xi_{abs}\rangle \xrightarrow{x.opc:=v, x.id:=ABS} \langle l_0, \dots, l_i', \dots, l_3, \eta_{abs}', \xi_{abs}\rangle},$$

где $\eta_{abs}' = \eta_{abs}[x := v]$. v здесь принимает все возможные значения из множества $dom_{abs}(x.opc)$, которые в исходной системе могут быть отправлены по каналу c графами $PG_r, 2 < r \le n$.

Таким образом, полное определение соответствия $Transr_1$ имеет следующий вид.

Пусть $0 \le i \le n$; $t = i$, если $0 \le i \le 2$, $t = 3$, если $2 < i \le n$. Тогда:

$Transr_1(rrc_i) = rra_t$, если значение v_1 было записано графом PG_1 или графом PG_2.

$Transr_1(rrc_i) = \{raa_{t,j} \mid 1 \le j \le S_1\}$, если значение v_1 не было записано ни PG_1, ни PG_2.

Покажем, что если посылка правила системы CS выполнима, то посылка соответствующего правила системы CS_{abs} также выполнима.

Случай а. Правило исходной системы – $rrc_i \in Rulesr_{concrete1}$. Правило абстрактной системы – rra_t, где $t = i$, если $0 \le i \le 2$, $t = 3$, если $2 < i \le n$.

В исходной системе посылка

$$(\eta, \xi) \vDash g \quad \wedge \quad len\big(\xi(c)\big) = k < cap(c) \quad \wedge \quad \xi(c) = v_1 \dots v_k$$

В абстрактной системе посылка

$$(\eta_{abs}, \xi_{abs}) \vDash g_{abs} \quad \wedge \quad len\big(\xi_{abs}(c)\big) = l < cap_{abs}(c) \quad \wedge \quad \xi_{abs}(c) = v_1 \dots v_l$$

Защиту g можно представить следующим образом:

$$g = A_1 \wedge A_2,$$

где $A_1 \in Cond(V_{AP}, C_1)$, и $A_2 \notin Cond(V_{AP}, C_1)$.

В A_1 может входить условие $nempty(c)$. В A_2 не входят условия относительно каналов. В соответствии с $Transs_1$, если v_1 в исходной системе была отправлена графом PG_1 или PG_2, то это же справедливо и в абстрактной системе. Поэтому в данном случае $nempty(c)$ выполняется. Доказательство выполнимости условий относительно переменных приведено при рассмотрении случая 1.

Случай б. Правило исходной системы – rrc_i, правила абстрактной системы – $raa_{t,j}, 1 \leq j \leq S_1$. Посылка в абстрактной системе имеет вид $(\eta_{abs}, \xi_{abs}) \vDash g_{abs}$, где g_{abs} не содержит утверждений относительно каналов. Доказательство его выполнимости такое же, как в случае 1.

На основании определения соответствия $Transr_1$ и доказанного утверждения относительно истинности посылок соответствующих правил, можно заключить, что в абстрактной системе из состояния $f(s_m)$ существует множество переходов, среди которых всегда найдется переход $f(s_m) \rightarrow f(s_{m+1})$, соответствующий переходу $s_m \rightarrow s_{m+1}$, такой, что $L_{abs}\big(f(s_{m+1})\big) = L(s_{m+1})$ (рисунок 10).

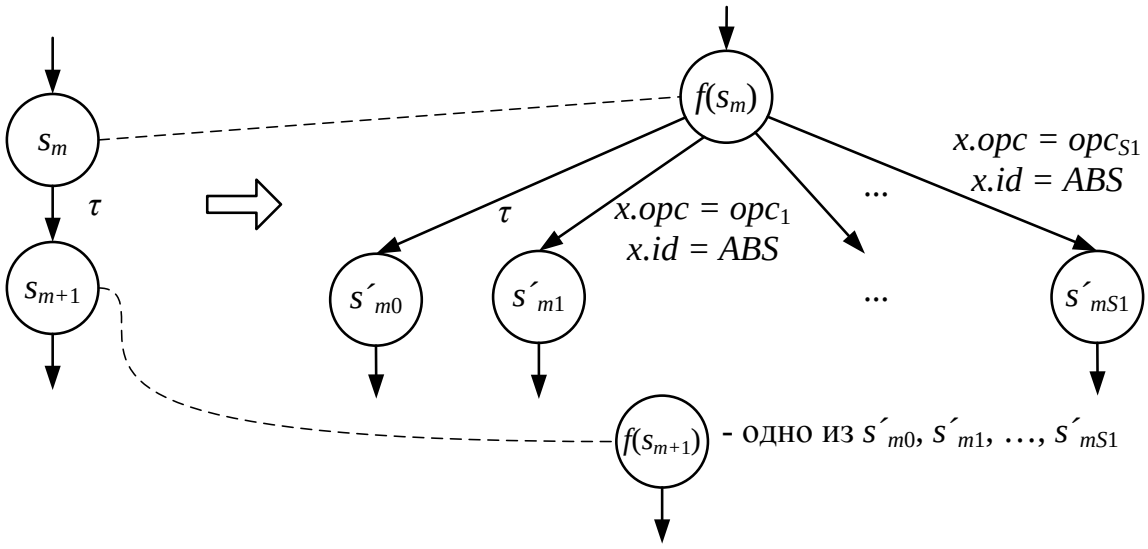


Рисунок 10 – Соответствие перехода в исходной системе, порожденного оператором считывания из канала $c \in C_1$, множеству переходов в абстрактной системе

Случай 2.1 рассмотрен.

Случай 2.2. $c \in C_2$.

Элемент множества правил исходной системы $Rulesr_{concrete2} = \{rrc_i \mid 1 \le i \le n\}$ имеет вид:

$$rrc_i = \frac{l_i \xrightarrow{g:c?x} l_i' \quad \wedge \quad (\eta, \xi) \vDash g \quad \wedge \quad len\big(\xi(c)\big) = k > 0 \quad \wedge \quad \xi(c) = v_1 \dots v_k}{\langle l_0, \dots, l_i, \dots, l_n, \eta, \xi\rangle \xrightarrow{\tau} \langle l_0, \dots, l_i', \dots, l_n, \eta', \xi'\rangle},$$

где $\eta' = \eta[x := v_1]$ и $\xi' = \xi[c := v_2 \dots v_k]$.

Элемент множества правил абстрактной системы $Rulesr_{abstract2} = \{rra_i \mid 1 \le i \le 2\}$ имеет вид:

$$rra_i = \frac{l_i \xrightarrow{g_{abs}:c_{2,i}?x} l_i' \ \wedge (\eta_{abs}, \xi_{abs}) \vDash g_{abs} \wedge len\left(\xi_{abs}\left(c_{2,i}\right)\right) = k > 0 \ \wedge \ \xi_{abs}\left(c_{2,i}\right) = v_1 \dots v_k}{\langle l_0, \dots, l_i, \dots, l_3, \eta_{abs}, \xi_{abs}\rangle \xrightarrow{\tau} \langle l_0, \dots, l_i', \dots, l_3, \eta_{abs}', \xi_{abs}'\rangle},$$

где $\eta'_{abs} = \eta_{abs}[x \coloneqq v_1]$ и $\xi'_{abs} = \xi_{abs}\left[c_{2,i} \coloneqq v_2 \dots v_k\right]$.

Из выполнимости условия $len\big(\xi(c)\big) = k > 0$ в посылке правила $rrc_i \in Rulesr_{concrete2}$, следует выполнимость $len\left(\xi_{abs}\big(c_{2,i}\big)\right) = k > 0$ в посылке правила $rra_i \in Rulesr_{abstract2}$. Покажем это рассмотрением фрагмента вычисления $s_0\alpha_1 s_1\alpha_2 \dots \alpha_m s_m$ системы TS и соответствующего фрагмента вычисления $f(s_0)\mathrm{A}(\alpha_1)f(s_1)\mathrm{A}(\alpha_2) \dots \mathrm{A}(\alpha_m)f(s_m)$ системы TS_{abs}.

Для случая отправки сообщения по каналу $c \in C_2$ множество правил исходной системы $Ruless_{concrete2} = \{rsc_0\}$, $c \in C_2 = \left\{c_{2,1}, \dots, c_{2,n}\right\}$.

$$rsc_0 = \frac{l_0 \overset{g:c!v}{\longrightarrow} l'_0 \quad \wedge \quad (\eta, \xi) \vDash g \quad \wedge \quad len\big(\xi(c)\big) = k < cap(c) \quad \wedge \quad \xi(c) = v_1 \dots v_k}{\langle l_0, \dots, l_i, \dots, l_n, \eta, \xi\rangle \overset{\tau}{\to} \langle l'_0, \dots, l_i, \dots, l_n, \eta, \xi'\rangle},$$

где $\xi' = \xi[c \coloneqq v_1 v_2 \dots v_k v]$.

Множество правил абстрактной системы $Ruless_{abstract2} = \{rsc_0\}$, $c \in \left\{c_{2,1}, c_{2,2}\right\}$.

$$rsa_0 = \frac{l_0 \overset{g_{abs}:c!v}{\longrightarrow} l'_0 \ \wedge \ (\eta_{abs}, \xi_{abs}) \vDash g_{abs} \wedge len\big(\xi_{abs}(c)\big) = k < cap(c) \wedge \xi_{abs}(c) = v_1 \dots v_k}{\langle l_0, \dots, l_i, \dots, l_n, \eta_{abs}, \xi_{abs}\rangle \overset{\tau}{\to} \langle l'_0, \dots, l_i, \dots, l_n, \eta_{abs}, \xi'_{abs}\rangle},$$

где $\xi'_{abs} = \xi_{abs}[c \coloneqq v_1 v_2 \dots v_k v]$.

Отображение $Transs_2 \colon Ruless_{concrete2} \to Ruless_{abstract2}$ определено следующим образом.

$Transs_2(rsc_0) = rsa_0$, если $c \in \left\{c_{2,1}, c_{2,2}\right\}$.

$Transs_2(rsc_0) = \emptyset$, если $c \in C_2 \backslash \{c_{2,1}, c_{2,2}\}$, поскольку отправка сообщений по каналам из данного множества удаляется в ходе абстракции.

Таким образом, $A(\alpha_i) = \alpha_i$, если $\alpha_i = \tau$ и порождено действием $c_{2,1}!\,v$ или $\alpha_i = \tau$ и порождено действием $c_{2,2}!\,v$, $A(\alpha_i) = \emptyset$ в противном случае.

В соответствии с абстрактными преобразованиями отображение

$$Transr_2 : Rulesr_{concrete2} \rightarrow Rulesr_{abstract2}$$

такое, что, во-первых, $Transr_2(rrc_i) = rra_i, i = 1{,}2$, поскольку прием сообщения из канала $c_{2,i}$ сохраняется в процессах $PG_i, i = 1{,}2$ (рисунок 11). Выполнимость посылок правил абстрактной системы при выполнимости посылок соответствующих правил исходной системы показывается аналогично случаю 2.1.

Во-вторых, $Transr_2(rrc_i) = \emptyset, i > 2$, поскольку удаляется прием сообщений процессами $PG_i, i > 2$ из каналов $c_{2,i}$ соответственно. Для случая $i > 2$ правило в исходной системе описывает переход между двумя состояниями с одинаковыми пометками (переменная x является локальной переменной графа PG_i системы CS: $x \in Vstate_i$), которые в абстрактной системе объединяются в одно (рисунок 12). Это означает, что значение переменной x в состоянии $f(s_{m+1})$ является таким же, как и значение x в состоянии s_{m+1}, в тех случаях, когда x входит в V_{AP}.

Случай 2.2 рассмотрен.

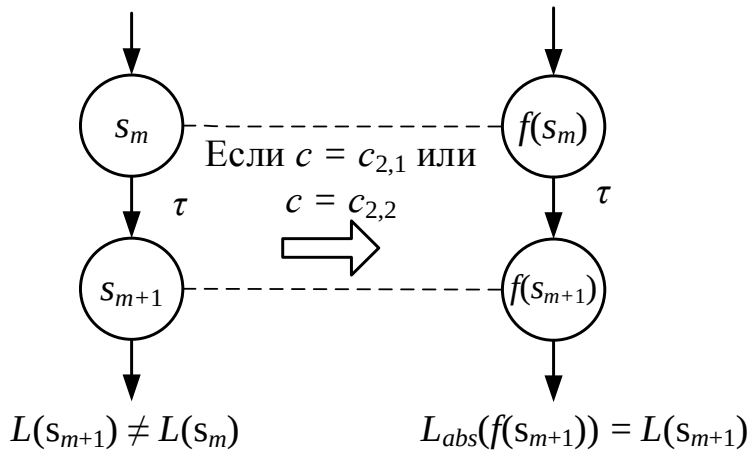


Рисунок 11 – Соответствие перехода между двумя состояниями с разными пометками в исходной системе переходу в абстрактной системе при преобразовании оператора считывания из канала $c \in C_2$

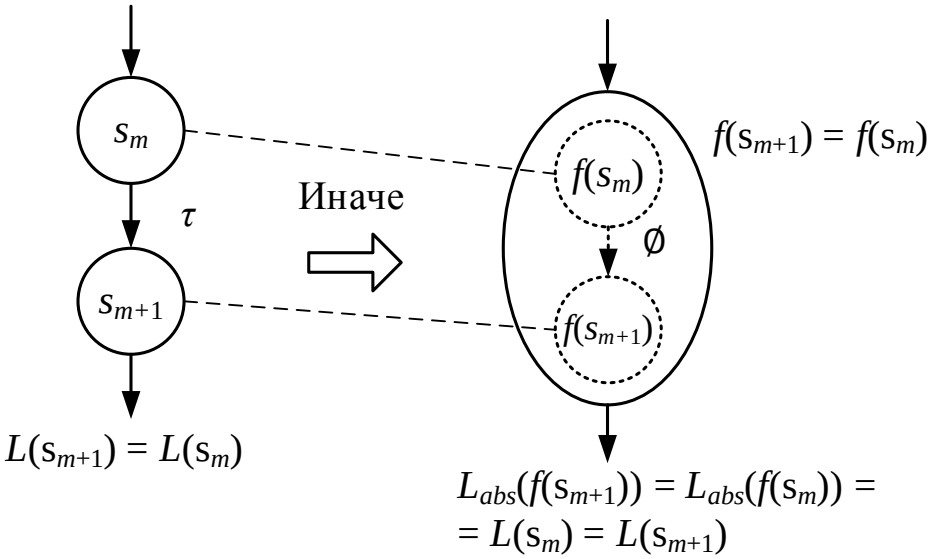


Рисунок 12 – Преобразование перехода исходной системы, порожденного оператором считывания из канала $c \in C_2$

Случай 2.3. $c \in C_3$.

Элемент множества правил исходной системы $Rulesr_{concrete3} = \{rrc_i \mid 1 \leq i \leq n\}$ имеет вид:

$$rrc_i = \frac{l_i \xrightarrow{g:c?x} l_i' \quad \wedge \quad (\eta, \xi) \vDash g \quad \wedge \quad len\big(\xi(c)\big) = k > 0 \quad \wedge \quad \xi(c) = v_1 \dots v_k}{\langle l_0, \dots, l_i, \dots, l_n, \eta, \xi \rangle \xrightarrow{\tau} \langle l_0, \dots, l_i', \dots, l_n, \eta', \xi' \rangle},$$

где $\eta' = \eta[x \coloneqq v_1]$ и $\xi' = \xi[c \coloneqq v_2 \dots v_k]$.

Элемент множества правил абстрактной системы $Rulesr_{abstract3} = \{rra_i \mid 0 \le i \le 3\}$ имеет вид:

$$\frac{l_i \xrightarrow{g_{abs}:c?x} l_i' \quad \wedge \quad (\eta_{abs}, \xi_{abs}) \vDash g_{abs} \quad \wedge \quad len\big(\xi_{abs}(c)\big) = k > 0 \quad \wedge \quad \xi_{abs}(c) = v_1 \dots v_k}{\langle l_0, \dots, l_i, \dots, l_3, \eta_{abs}, \xi_{abs} \rangle \xrightarrow{\tau} \langle l_0, \dots, l_i', \dots, l_3, \eta_{abs}', \xi_{abs}' \rangle},$$

где $\eta_{abs}' = \eta_{abs}[x \coloneqq v_1]$ и $\xi_{abs}' = \xi_{abs}[c \coloneqq v_2 \dots v_k]$.

Выполнимость условия $len\big(\xi_{abs}(c)\big) = k > 0 \quad \wedge \quad \xi_{abs}(c) = v_1 \dots v_k$ в абстрактной модели следует из выполнимости соответствующего условия исходной модели, так как переход, помеченный оператором отправки сообщения по этому каналу от единственного отправителя, также присутствует и в абстрактной модели, и выполнение этого условия осуществимо только при возможности такого перехода. Покажем это.

Для случая отправки сообщения по каналу $c \in C_3$ множество правил исходной системы $Ruless_{concrete3} = \{rsc_i \mid 0 \le i \le n\}$.

$$rsc_i = \frac{l_i \xrightarrow{g:c!v} l_i' \quad \wedge \quad (\eta, \xi) \vDash g \quad \wedge \quad len\big(\xi(c)\big) = k < cap(c) \quad \wedge \quad \xi(c) = v_1 \dots v_k}{\langle l_0, \dots, l_i, \dots, l_n, \eta, \xi \rangle \xrightarrow{\tau} \langle l_0, \dots, l_i', \dots, l_n, \eta, \xi' \rangle},$$

где $\xi' = \xi[c \coloneqq v_1 v_2 \dots v_k v]$.

Множество правил абстрактной модели $Ruless_{abstract3} = \{rsa_i \mid 0 \le i \le 3\}$.

rsa_i

$$= \frac{l_i \xrightarrow{g_{abs}:c!v} l_i' \ \wedge \ (\eta_{abs}, \xi_{abs}) \vDash g_{abs} \wedge len\big(\xi_{abs}(c)\big) = k < cap(c) \wedge \xi_{abs}(c) = v_1 \dots v_k}{\langle l_0, \dots, l_i, \dots, l_3, \eta, \xi \rangle \xrightarrow{\tau} \langle l_0, \dots, l_i', \dots, l_3, \eta, \xi' \rangle},$$

где $\xi'_{abs} = \xi_{abs}[c \coloneqq v_1 v_2 \dots v_k v]$.

Отображение $Transs_3 \colon Ruless_{concrete3} \to Ruless_{abstract3}$ определено следующим образом.

$Transs_3(rsc_i) = rsa_i, i = 0{,}1{,}2$, поскольку отправке сообщения процессом $PG_i, i = 0{,}1{,}2$ исходной модели соответствует отправка сообщения процессом PG_i абстрактной модели. Выполнимость условия $(\eta_{abs}, \xi_{abs}) \vDash g_{abs}$ при выполнимости $(\eta, \xi) \vDash g$ показывается аналогично случаю 2.1.

$Transs_3(rsc_i) = rsa_3, i > 2$, поскольку отправке сообщения процессом $PG_i, i > 2$ исходной модели соответствует отправка сообщения процессом PG_3 абстрактной модели (только один процесс может осуществлять такую отправку в ходе исполнения запроса).

Таким образом, каждому оператору отправки сообщения в исходной модели соответствует оператор отправки сообщения в абстрактной модели.

Отображение

$$Transr_3 \colon Rulesr_{concrete3} \to Rulesr_{abstract3}$$

определено следующим образом.

$Transr_3(rrc_i) = rra_i, i = 0{,}1{,}2$, поскольку приему сообщения процессом $PG_i, i = 0{,}1{,}2$ исходной модели соответствует прием сообщения процессом PG_i абстрактной модели.

$Transr_3(rrc_i) = rra_3, i > 2$, поскольку приему сообщения процессом $PG_i, i > 2$ исходной модели соответствует прием сообщения процессом PG_3

абстрактной модели (только один процесс может осуществлять такой прием в ходе исполнения запроса).

Таким образом, переходу исходной системы, порожденному считыванием сообщения из канала $c \in C_3$, в абстрактной системе всегда соответствует переход, порожденный аналогичным считыванием (рисунок 13), и значение переменной x в состоянии $f(s_{m+1})$ является таким же, как и значение x в состоянии s_{m+1}, в тех случаях, когда x входит в V_{AP}.

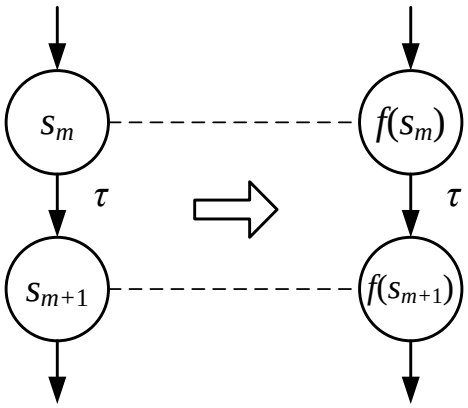


Рисунок 13 – Соответствие перехода в исходной системе, порожденного оператором считывания из канала $c \in C_3$, переходу в абстрактной системе

Случай 2.3 и случай 2 в целом рассмотрены.

Переходы, помеченные операторами отправки сообщения по каналу, не изменяют пометку состояния, и поэтому отдельно не рассматриваются. Однако они оказывают влияние на результаты изменения пометки состояний соответствующими переходами, помеченными операторами приема сообщения из канала, в связи с чем были рассмотрены в рамках случая 2.

Переходы, вызванные наличием в графах программ ребер с метками $goto$ и их эквивалентами, не изменяют пометку состояния и сохраняются в графах процессов абстрактной системы.

Таким образом, для всех возможных видов перехода $s_m \to s_{m+1}$, таких, что $L(s_m) \neq L(s_{m+1})$, в абстрактной системе существует переход $f(s_m) \to$

$f(s_{m+1})$, такой, что $L_{abs}\big(f(s_{m+1})\big) = L(s_{m+1})$.

Теорема доказана.

Выводы по главе 2

1. Осуществлен выбор языка Promela для описания моделей протоколов когерентности памяти. Данный выбор основан на том, что Promela предоставляет процессные типы и примитивы передачи сообщений между процессами, и это позволяет описывать протоколы когерентности памяти естественным образом и согласуется с известной моделью протоколов когерентности в виде множества взаимодействующих конечных автоматов. Более того, инструментальное средство Spin, в котором реализованы алгоритмы проверки моделей на языке Promela, является современным и постоянно развиваемым.

2. Разработана математическая модель протоколов когерентности памяти, базируемая на формальной семантике языка Promela. На низком уровне абстракции модель представлена системой переходов, а на высоком уровне абстракции – процессами и их объединением в канальную систему. Высокий уровень абстракции позволяет формализовать синтаксические преобразования моделей, а низкий – проводить доказательство их корректности.

3. Определены ограничения на модели протоколов когерентности и предложены синтаксические преобразования всех типов операторов языка Promela, позволяющие получать абстрактные модели. Количество процессов в абстрактной модели по сравнению с исходной моделью снижено с $n + 1$ до четырех, что позволяет получить абстрактную систему переходов существенно меньшего размера.

4. Разработан подход к описанию протоколов когерентности памяти, в рамках которого предложена схема организации процессов, соответствующих системному коммутатору и кэш-контроллерам.

5. Сформулирована и доказана теорема, определяющая корректность предложенных преобразований, в которой проведен исчерпывающий анализ соответствия переходов в исходной и абстрактной моделях.

3 Разработка метода верификации протоколов когерентности памяти

3.1 Разработка Promela-моделей протоколов когерентности

3.1.1 Соответствие элементов Promela-моделей математическим абстракциям

В главе 2 модель протоколов когерентности и абстрактные преобразования представлены на высоком уровне, который был необходим для проведения доказательства корректности преобразований. В данной главе показано, как представлять использованные математические объекты на языке Promela, и выполнять преобразования получаемых моделей.

В соответствии с моделью протоколов когерентности памяти, разработанной в главе 2, исходная модель включает в себя следующие процессы:

- один процесс $home$, являющийся абстракцией системного коммутатора home-процессора;
- n процессов $ccon$, являющихся абстракцией кэш-контроллеров.

Абстрактная модель включает в себя следующие процессы:

- один процесс $home_{abs}$, являющийся абстракцией системного коммутатора home-процессора;
- два процесса $ccon_{abs}$, являющихся абстракцией двух кэш-контроллеров;
- один процесс abs, представляющий остальные кэш-контроллеры, или окружение устройств, представляемых процессами $home_{abs}$ и $ccon_{abs}$.

Соответствующие экземпляры процессов абстрактной модели создаются в дополнительном специальном процессе init. При этом каждому экземпляру передается его идентификатор (целое неотрицательное число):

init

```
{
        atomic {
        run home_abs(0);
        run ccon_abs(1);
        run ccon_abs(2);
        run abs(ABS);
        }
}
```

В главе 2 введены множества $Vstate_i$ переменных, описывающих состояние процессов $PG_i, 0 \leq i \leq n$. В таблице 7 приведены соглашения, в соответствии с которыми определяется принадлежность переменных, не являющихся каналами, определенных в Promela-моделях, тому или иному множеству $Vstate_i$.

Таблица 7 – Принадлежность переменных исходной модели множествам $Vstate_i$

Тип переменных	Пример
Глобальные переменные, не являющиеся массивами, описывают исполняемый запрос и находятся в каждом множестве $Vstate_i, 0 \leq i \leq n$. Предполагается, что значения этим переменным присваиваются процессом PG_0 на начальном этапе обработки запроса, а далее в ходе исполнения запроса их может модифицировать только один из процессов $PG_1, \dots, PG_n$ (запросчик)	current_client – идентификатор процесса-запросчика. current_command – исполняемый в данный момент времени исходный запрос

Тип переменных	Пример
Глобальная переменная, являющаяся массивом длины n, рассматривается как множество из n переменных, элементы которого пронумерованы индексами $1, \dots, n$, и i-й элемент которого принадлежит множеству $Vstate_i, i = 1, \dots, n$	cache[n] – массив, i-й элемент которого cache[i] представляет состояние кэш-строки в i-м кэше. snoop_list[n] – массив элементов логического типа, snoop_list[i] фиксирует необходимость отправки снуп-запроса i-му процессу, а после отправки – фиксирует получение снуп-запроса данным процессом. ack_list[n] – аналогично snoop_list, но для когерентных ответов
Локальная переменная процесса PG_i независимо от типа принадлежит множеству $Vstate_i$. Переменная, находящаяся в списке параметров процесса считается локальной переменной данного процесса	message – переменная, значение которой присваивается операцией извлечения сообщения из канала

Нумерация массивов в Promela начинается с нуля, однако здесь для соответствия индексов элементов массивов индексам графов приведенной в главе 2 математической модели протоколов когерентности, используется нумерация с единицы.

Все массивы, используемые в исходной модели (массивы переменных и массивы каналов), имеют длину n и индексируются идентификаторами процессов с номерами $1, \dots, n$. Для перехода от конкретной модели к параметризованной используются следующие правила.

Всякое условие, если оно задействует массив, является либо конъюнкцией, либо дизъюнкцией однотипных условий на все элементы

массива:

- формула $\varphi\{i/1\} \wedge \ldots \wedge \varphi\{i/n\}$ интерпретируется как $\forall i \in \{1, \ldots, n\}: \varphi$;
- формула $\varphi\{i/1\} \vee \ldots \vee \varphi\{i/n\}$ интерпретируется как $\exists i \in \{1, \ldots, n\}: \varphi$.

Всякая цепочка операторов вида $\alpha\{i/1\}; \ldots; \alpha\{i/n\}$ интерпретируется как цикл $for\ (i: 1..n)\ \{\ \alpha\ \}$.

Здесь φ (α) – формула (оператор), содержащая индекс i (идентификатор процесса) в качестве свободной переменной; запись $\varphi\{i/t\}$ $(\alpha\{i/t\})$ означает результат подстановки в φ (α) выражения t вместо всех вхождений переменной i.

3.1.2 Преобразования Promela-моделей

В соответствии с принятыми соглашениями о представлении математических объектов, введенных в главе 2, в Promela-моделях, разработанные в главе 2 преобразования в терминах преобразования операторов языка Promela формулируются следующим образом.

Объявления глобальных переменных, не являющихся массивами, сохраняются. Длина всех глобальных массивов изменяется со значения n на значение 2, то есть среди переменных, входящих в состав массивов, сохраняются только переменные, описывающие состояние процессов PG_1 и PG_2. Объявления локальных переменных, не принадлежащих множеству V_{loca}, удаляются.

Объявления каналов в модели модифицируются следующим образом. В тех случаях, когда емкость каналов зависит от n, что имеет место для каналов $c \in C_1$, их емкость заменяется на 2. Множество каналов C_2 представлено в исходной модели массивом длины n. В абстрактной модели длина этого массива, как и длина массивов глобальных переменных, изменяется на 2. Оставшиеся компоненты объявлений массивов остаются неизменными.

Преобразования операторов, находящихся в теле процессов, состоят из:

- преобразований присваиваний $TA_1, \dots, TA_3$ (Transformations of Assignments, таблица 8);
- преобразований выражений $TE_1, \dots, TE_3$ (Transformations of Expressions, таблица 9);
- преобразований операторов коммуникационных действий $TC_1, \dots, TC_5$ (Transformations of Communication actions, таблица 10).

Таблица 8 – Преобразования присваиваний

№	Тип оператора	Преобразование
	Оператор присваивания значений глобальным переменным	Не изменяется
TA_1	Оператор присваивания значений локальным переменным	Не изменяется, если находится в теле процессов $home_{abs}$, $ccon_{abs}$. Удаляется, если находится в теле процесса abs
TA_2	Оператор присваивания значений элементу массива глобальных переменных $A[k]$, где k – число	Не изменяется, если $k \leq 2$. Удаляется из тела всех процессов, если $k > 2$
TA_3	Оператор присваивания значений элементу массива глобальных переменных $A[g]$, где g – глобальная переменная или локальная переменная, которая может принимать значения из множества $D \subseteq dom(g)$. Оператор имеет вид $A[g] = val$, где $val \in dom(A[g])$	Заменяется на конструкцию if :: g != ABS –> A[g] = val; :: else fi; Данная конструкция устраняет присваивание значений переменным $A[k]$, где $k = 3, \dots, n$

При описании преобразований выражений для простоты предполагается,

что преобразуемые выражения имеют вид $E = E_1 \wedge ... \wedge E_m$ для некоторого m, где $E_i = x \mid x == v \mid x\ != v \mid Pred(c) \mid E_i \vee E_i$, то есть E_i имеет вид одного из указанных выражений (расстановка скобок осуществляется в соответствии с правилами языка Promela). Здесь x – переменная, v – переменная или константа, c – канал, $Pred$ – доступный предикат относительно канала (см. главу 2). В данном случае приведенные преобразования обеспечат замену выражения на истину на верхнем уровне. В противном случае необходим более сложный анализ. Например, выражение $!(E_1 \wedge E_2)$, в котором E_1 и E_2 заменяются на $true$, должно в конечном счете принимать значение $true$. При разработке моделей не было выявлено необходимости использования более сложных выражений, поэтому здесь используется это упрощение.

Таблица 9 – Преобразования выражений

№	Тип оператора	Преобразование
TE_1	Выражение относительно локальной переменной процесса abs	Заменяется на $true$ за исключением случаев, в которых переменная принадлежит множеству V_{loca}: переменная используется в качестве индекса в массив глобальных переменных, а значение ей было присвоено посредством оператора извлечения сообщения из канала
TE_2	Выражение относительно глобальной переменной-массива $A[k]$, где k – число	Не изменяется, если $k \leq 2$. Изменяется на $true$, если $k > 2$
TE_3	Выражение относительно канала $c \in C_2 \backslash \{c_{2,1}, c_{2,2}\}$	Изменяется на $true$

Таблица 10 – Преобразования коммуникационных действий

№	Тип оператора	Преобразование
TC_1	Оператор отправки сообщения в канал $c \in C_1$	Не изменяется, если находится в теле процессов $ccon_{abs}$. Удаляется, если находится в теле процесса abs
TC_2	Оператор приема сообщения из канала $c \in C_1$	Заменяется недетерминированным выбором (глава 2)
TC_3	Оператор отправки сообщения в канал $c \in C_2$. Случай 1. Оператор имеет вид $c[k]!\,v$, где k – число.	Не изменяется, если $k = 1{,}2$, при этом $c = c_{2,1}$ или $c = c_{2,2}$ Удаляется, если $k > 2$, при этом $c \in C_2 \backslash \{c_{2,1}, c_{2,2}\}$
TC_4	Случай 2. Оператор имеет вид $c[g]!\,v$, где g – переменная	Заменяется на конструкцию if :: g != ABS –> c[g]!v; :: else fi;
TC_5	Оператор приема сообщения из канала $c \in C_2$. Оператор, находящийся в теле процесса с идентификатором i, имеет вид $c[i]?\,x$, где x – переменная	Не изменяется, если $i = 1{,}2$, при этом $c = c_{2,1}$ или $c = c_{2,2}$. Удаляется, если $i > 2$, при этом $c \in C_2 \backslash \{c_{2,1}, c_{2,2}\}$

3.2 Процедура уточнения абстрактных моделей

В главе 2 показано, что из выполнимости свойств-инвариантов на абстрактной модели следует их выполнимость на исходной модели. Если же свойство не выполняется на абстрактной модели, то непосредственно из этого нельзя делать никаких заключений о выполнимости свойства на исходной модели. В этом случае необходим анализ контрпримера, который покажет, представляет ли нарушение свойства ошибку в исходной модели, или же нарушение возможно только на абстрактной модели, так как исходная модель такого поведения не допускает.

Предлагаемый подход к описанию протоколов когерентности на языке Promela позволяет явно выделить в коде модели все этапы выполнения запросов. Это делает возможной стратегию уточнения абстрактной модели с помощью введения вспомогательных переменных, позволяющих ограничить временные рамки выполнения действий, которые являются причинами контрпримеров.

Выполнение каждого типа запросов состоит из определенной последовательности действий. Например: отправка кэш-контроллером исходного запроса в системный коммутатор, получение данного исходного запроса системным коммутатором, отправка системным коммутатором когерентных запросов в другие кэш-контроллеры системы и так далее. Каждую такую последовательность будем представлять в виде частично упорядоченного множества $(A, \prec)$, где A – множество действий $a \in A$, $\prec$ – отношение строгого порядка на множестве A.

Процедура уточнения абстрактной модели.

1. Для каждого типа исходных запросов определить на основе документации частично упорядоченное множество $(A, \prec)$ действий ($\prec$ – отношение строгого порядка):

$\forall a_1, a_2 \in A: a_1 \prec a_2$, если действие a_1 осуществляется раньше

действия a_2.

2. Пока имеются ложные контрпримеры:

   2.1. Найти действие a, которое привело к возникновению контрпримера. Найти множество A, содержащее действие a: $a \in A$. В множестве A найти действие b, такое что $b \prec a$.

   2.2. Ввести переменную логического типа aux_b с начальным значением $false$. Заменить b в модели на атомарную последовательность $b; aux_b := true$.

3. Добавить aux_b к защите команды, содержащей действие a, с помощью логического И. Заменить a на атомарную последовательность $a; aux_b := false$.

Доказательство корректности процедуры уточнения.

Из теоремы, сформулированной и доказанной в главе 2, следует, что защиты g_{abs} действий абстрактной модели являются логическими следствиями защит g соответствующих действий исходной модели: $g \Rightarrow g_{abs}$. В соответствии с процедурой уточнения защита g_{abs} заменяется на защиту $g_{abs} \wedge aux_b$. Покажем, что такая защита также является логическим следствием соответствующей защиты исходной модели.

Если переменную aux_b ввести в исходной модели, то, поскольку действие b в соответствии с процедурой уточнения совершено раньше действия a, то в момент совершения действия a выполняется равенство $aux_b = true$. Следовательно, $g \Leftrightarrow g \wedge aux_b$. Тогда $g \Rightarrow g_{abs} \wedge aux_b$, то есть уточненная защита является логическим следствием соответствующей защиты исходной модели. Следовательно, теорема из главы 2 продолжает выполняться при замене абстрактной модели на уточненную абстрактную модель.

Предполагается, что приведенная процедура будет выполняться верификатором вручную. Далее приведено альтернативное описание процедуры уточнения абстрактной модели, которое может быть использовано для

автоматизации процесса уточнения абстрактной модели. Данный вариант предполагает введение большего количества вспомогательных переменных (каждому действию соответствует переменная), чем предыдущий вариант, в котором вводятся только те вспомогательные переменные, которые действительно используются для уточнения модели.

Начало процедуры.

Определить множество множеств действий As. Присвоить $As \coloneqq \emptyset$.

Для каждого типа исходных запросов верифицируемой системы:

1. Определить частично упорядоченное множество $(A, \prec)$ действий следующим образом: $\forall a_1, a_2 \in A: a_1 \prec a_2$, если в ходе выполнения запроса данного типа осуществление действия a_1 предшествует осуществлению действия a_2. Присвоить $As \coloneqq As \cup A$.

2. Определить множество вспомогательных переменных логического типа Aux. Присвоить $Aux \coloneqq \emptyset$.

3. Каждому действию $a \in A$ сопоставить в модели протокола вспомогательную переменную aux_a логического типа с начальным значением $false$. Присвоить $Aux \coloneqq Aux \cup \{aux_a\}$.

4. Для каждого действия $a \in A$ в соответствующем графе процесса абстрактной Promela-модели из множества $\{PG_0, PG_1, PG_2\}$ найти переход, помеченный действием a. Пусть это переход (l_i, g, a, l_j), где l_i и l_j – некоторые состояния, g – защита данного перехода. Заменить пометку этого перехода с $g: a$ на $g: (a \wedge aux_a \coloneqq true)$, что соответствует атомарному выполнению действия a и присваивания $aux_a \coloneqq true$.

Для каждого перехода в графе PG_3, помеченного некоторым действием a, защита которого в ходе абстракции была заменена на $true$:

1. Найти множество $A \subset As$, в котором находится действие a $(a \in A)$. Заменить пометку этого перехода с $true: a$ на $aux_c: a$, где $b, c \in A: c \prec b \prec a$ и действие b было удалено из абстрактной модели, aux_c и aux_b –

вспомогательные переменные, соответствующие действиям c и b, соответственно.

2. Установить $aux_c \coloneqq false$.

Конец процедуры.

3.3 Метод верификации протоколов когерентности памяти

В соответствии с результатами, полученными в предыдущих разделах, разработанный в данной диссертации метод параметризованной верификации протоколов когерентности памяти состоит из двух этапов:

1. Применение разработанных абстрактных преобразований к Promela-моделям протоколов когерентности памяти, представляющим протоколы когерентности в виде множества процессов, обменивающихся сообщениями посредством каналов.

2. Уточнение абстрактных моделей на основе анализа контрпримеров и введения вспомогательных переменных согласно разработанной процедуре уточнения.

3.4 Методика верификации протоколов когерентности памяти

В соответствии с результатами данной работы, полученными в предыдущих разделах, методика верификации протоколов когерентности памяти состоит из следующих этапов (рисунок 14):

1. Разработка исходной модели протокола когерентности на языке Promela. Модель разрабатывается верификатором вручную. На основании документации на протокол когерентности памяти, верификатор в соответствии с предложенным подходом составляет описание процессов, моделирующих кэш-контроллеры и системный коммутатор, и добавляет в модель необходимые инфраструктурные элементы (определение каналов, запуск процессов).

2. Построение абстрактной модели в соответствии с предложенными синтаксическими преобразованиями Promela-моделей. Преобразования могут быть выполнены вручную либо автоматически с использованием системы построения абстрактной модели, разработка которой описана в главе 4. Ручное выполнение требует большого объема скрупулезной работы, при выполнении которой легко совершить ошибку, которая впоследствии может быть не найдена, так как она приведет к некорректному сокращению пространства состояний модели, о чем не будет никаких сигналов. Система построения абстрактной модели разработана для устранения таких ситуаций и также позволяет существенно ускорить процесс верификации за счет значительного уменьшения объема ручной работы.

3. Проверка выполнения свойств на абстрактной модели с помощью инструментального средства Spin. Данный этап идентичен проверке любых Promela-моделей.

4. Анализ верификатором отчета о верификации, сгенерированного инструментом Spin. Если отчет показывает отсутствие ошибок, то верификация завершается с вердиктом о корректности протокола когерентности памяти. Если отчет сигнализирует о наличии ошибки, то проводится анализ соответствующего контрпримера, сгенерированного системой Spin. Если верификатор устанавливает, что данный контрпример является ложным, то есть представленная в нем последовательность действий невозможна в реальной микропроцессорной системе, то производится уточнение абстрактной модели в соответствии с разработанной процедурой и переход к шагу 3. Если верификатор устанавливает, что данный контрпример представляет ошибку в протоколе когерентности, то о данной ошибке сообщается разработчикам протокола. После исправления ошибки разработчиками протокола, верификатор вносит соответствующие исправления в исходную Promela-модель протокола, и верификация начинается заново (происходит переход к шагу 1).

Последовательность указанных этапов повторяется до тех пор, пока не будут устранены все контрпримеры.

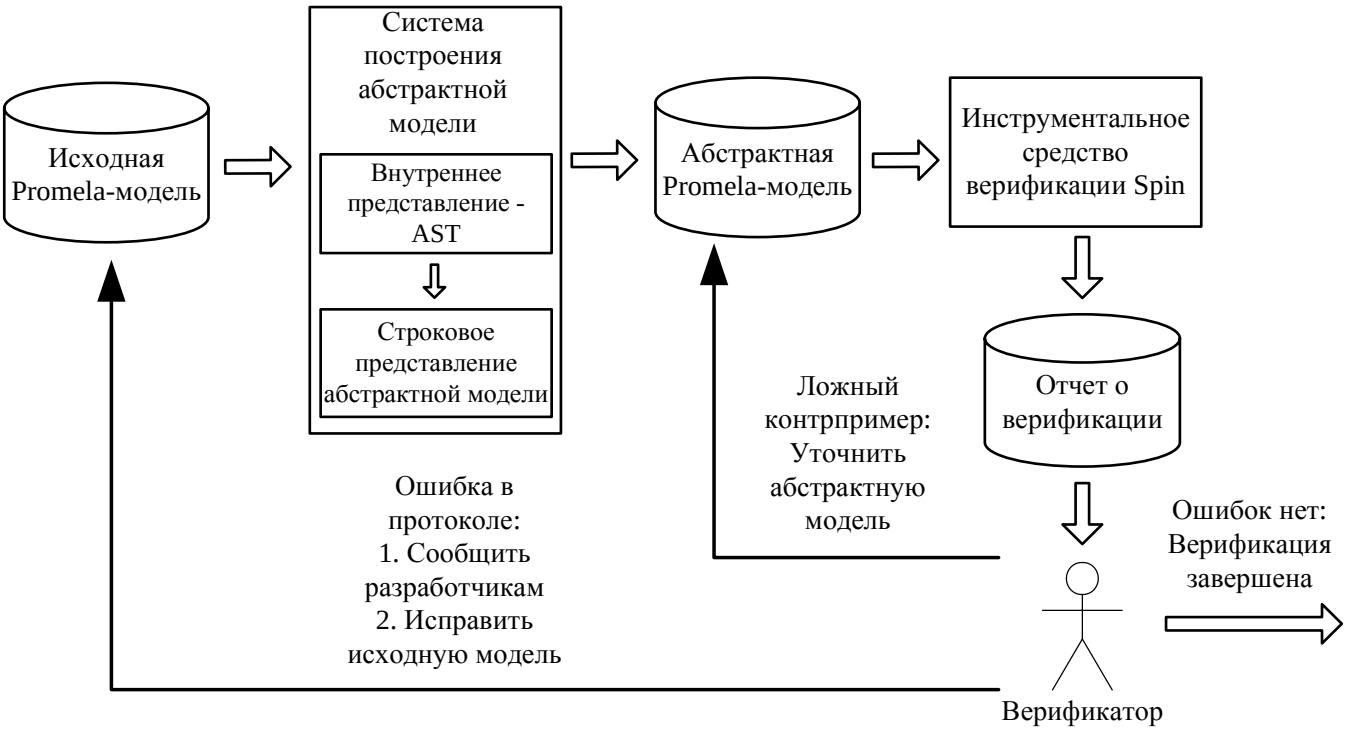


Рисунок 14 – Схема процесса верификации протоколов когерентности с памяти с использованием разработанной методики

Выводы по главе 3

1. Разработаны преобразования Promela-моделей, детально описывающие их более абстрактное представление, приведенное в главе 2. Преобразования разбиты на три группы: преобразования присваиваний, преобразования выражений и преобразования коммуникационных действий и пронумерованы с целью дальнейшей разработки алгоритмов, автоматизирующих выполнение преобразований.

2. Разработана процедура уточнения абстрактных моделей, позволяющая устранить ложные контрпримеры, возникающие при верификации абстрактных моделей, построенных с использованием предложенных преобразований.

3. Разработана методика параметризованной верификации протоколов когерентности памяти, включающая в себя подход к разработке формальных Promela-моделей протоколов когерентности, метод разработки и процедуру уточнения абстрактных моделей, процесс верификации абстрактных моделей с использованием инструментального средства Spin.

4 Реализация и экспериментальные исследования системы верификации протоколов когерентности памяти

4.1 Составление формальных моделей протоколов когерентности на примере протокола системы на кристалле Эльбрус-4С

4.1.1 Анализ системы на кристалле Эльбрус-4С

Разработка формальных моделей протоколов когерентности памяти осуществляется на основе документации на протокол когерентности памяти. В тех случаях, когда такая документация отсутствует, необходим анализ структуры верифицируемой системы (осуществляемый на основе документации на микропроцессорную систему), на основе которого выделяются устройства, участвующие в реализации протокола когерентности. Далее эти устройства представляются в виде конечных автоматов. При этом один автомат может представлять группу устройств. Описание данных автоматов дает документацию на протокол когерентности.

Далее приведен анализ микропроцессора Эльбрус-4С и системы из таких микропроцессоров, и определены устройства (группы устройств), моделируемые с помощью абстракций, предоставляемых языком Promela.

Процессорный узел системы на кристалле Эльбрус-4С включает в себя четыре ядра, каждое со своей кэш-памятью первого и второго уровней, а также системный коммутатор, выполняющий роль связующего центра, различные контроллеры и прочее оборудование распределенной интерфейсной логики. В состав кластера входят четыре процессорных узла (рисунок 15), связанных высокоскоростными межпроцессорными каналами.

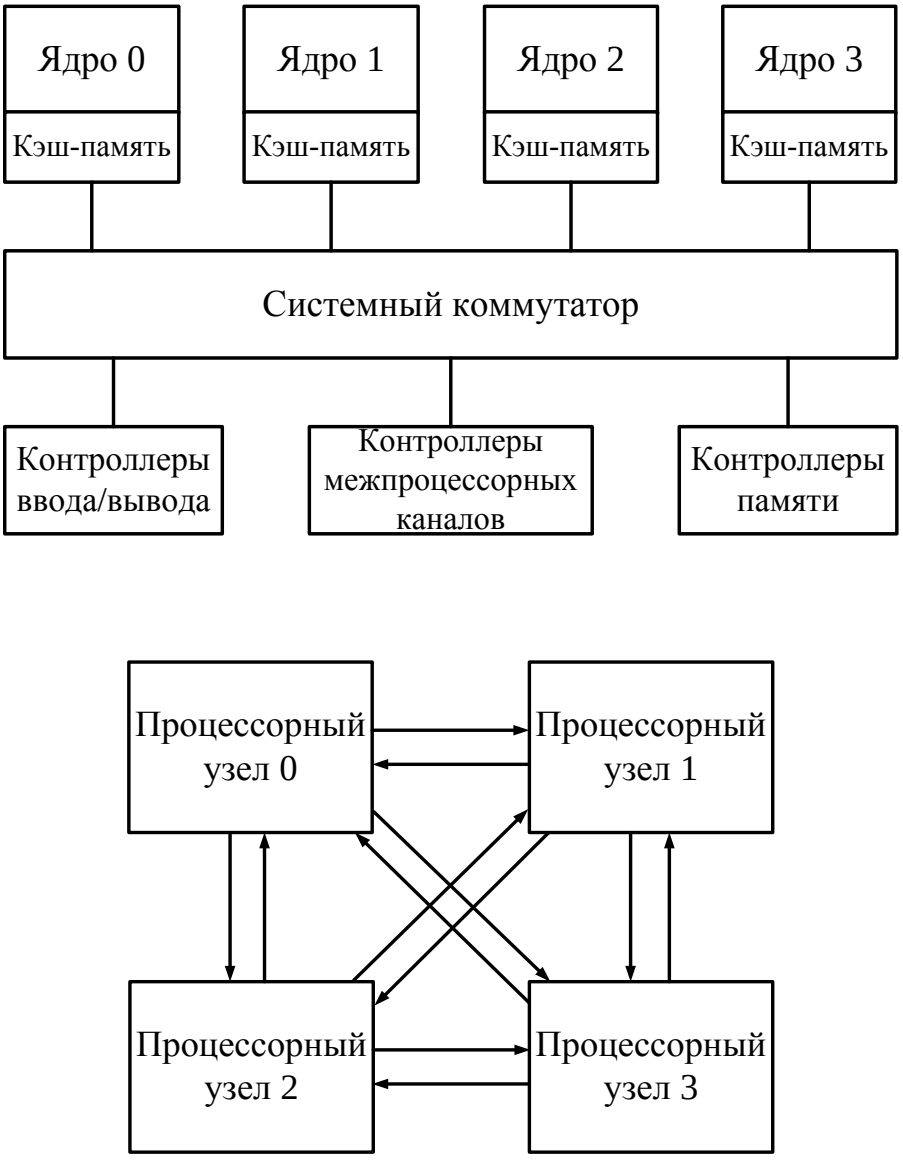


Рисунок 15 – Схема процессорного узла (вверху) и кластера (внизу)

Эльбрус-4С

Эльбрус-4С является системой с неоднородным доступом к памяти (NUMA, Non-Uniform Memory Access): каждый процессорный узел включает в себя контроллер основной памяти, работающий с памятью, подключенной к этому узлу. С программной точки зрения, адресное пространство кластера является общим. Когерентность поддерживается на аппаратном уровне. Процессор, к чьей памяти происходит обращение, называется home-процессором.

В системе Эльбрус-4С реализован протокол когерентности памяти MOSI.

Состояния MOSI кэш-строки означают следующее:

- Modified – строка модифицирована и имеется в единственном экземпляре;
- Owned – строка модифицирована и может быть у других ядер;
- Shared – строка не модифицирована и может быть у других ядер;
- Invalid – строка недействительна.

В ядре протокол когерентности реализуют кэш-память второго уровня L2 и устройство обращения к памяти MAU (Memory Access Unit). Основным устройством, участвующим в реализации протокола когерентности и находящимся вне ядра, является системный коммутатор.

Основной задачей системного коммутатора является обеспечение обмена между процессорными ядрами, локальной оперативной памятью и внешними по отношению к данному процессорному модулю абонентами, включая контроллер канала ввода/вывода.

Важной функцией системного коммутатора является поддержка очередности исполнения запросов. Запросы в системном коммутаторе обслуживаются по одному: если выполняется операция над ячейкой памяти, последующие запросы с таким же адресом блокируются системным коммутатором.

Получение информации о состоянии кэш-контроллеров системы происходит на основе широковещательных сообщений с аннулированием при записи. Для сокращения количества рассылаемых сообщений предусмотрен режим фильтрации запросов когерентности с помощью справочника – устройства, хранящего информацию о состоянии кэш-строк в системе.

При использовании широковещательных сообщений каждой операции, требующей изменения состояния кэш-строки, предшествует рассылка соответствующего когерентного запроса всем кэш-контроллерам: инициатор должен дождаться получения от каждого из них ответа, свидетельствующего о

выполнении необходимых действий, затем изменить состояние своей копии и выполнить операцию над ней.

В данной работе рассматривается режим работы протокола с широковещательной рассылкой сообщений, поскольку модель протокола со справочником будет иметь ту же структуру, что и модель с широковещательной рассылкой. При этом процесс, соответствующий системному коммутатору home-процессора, будет дополнен алгоритмом работы справочника.

4.1.2 Протокол когерентности системы на кристалле Эльбрус-4С

Работу протокола когерентности системы Эльбрус-4С можно представить следующим образом. Процессорное ядро в соответствии с исполняемой программой формирует запрос на считывание или запись, имеющий атрибут – тип памяти. Запрос процессорного ядра поступает в контроллер кэш-памяти второго уровня, принадлежащей данному ядру. Если состояние контроллера кэш-памяти позволяет заключить, что запрашиваемые действия могут быть осуществлены локально, то они осуществляются. В противном случае контроллер кэш-памяти формирует *исходный запрос* и отправляет его в системный коммутатор. Системный коммутатор рассылает в остальные контроллеры кэш-памятей системы *снуп-запросы*. При получении снуп-запросов контроллеры кэш-памяти формируют *ответы* (в виде данных или подтверждения) и отправляют их либо в контроллер кэш-памяти ядра-запросчика, либо в системный коммутатор. Адресат в данном случае зависит от состояния кэш-контроллера, которое, в свою очередь, было определено типом исходного запроса. В случае, если сбором ответов занимается кэш-контроллер ядра-запросчика, то по окончании сбора он отправляет в системный коммутатор соответствующее сообщение, получив которое, системный коммутатор может принять следующий исходный запрос.

Конкретные действия контроллеров кэш-памяти и системного

коммутатора зависят от типа полученного сообщения, состояния контроллера. Часть автомата состояний контроллера кэш-памяти второго уровня, отвечающая за обработку запросов от процессорного ядра на считывание и запись с типом памяти write back, приведена на рисунке 16.

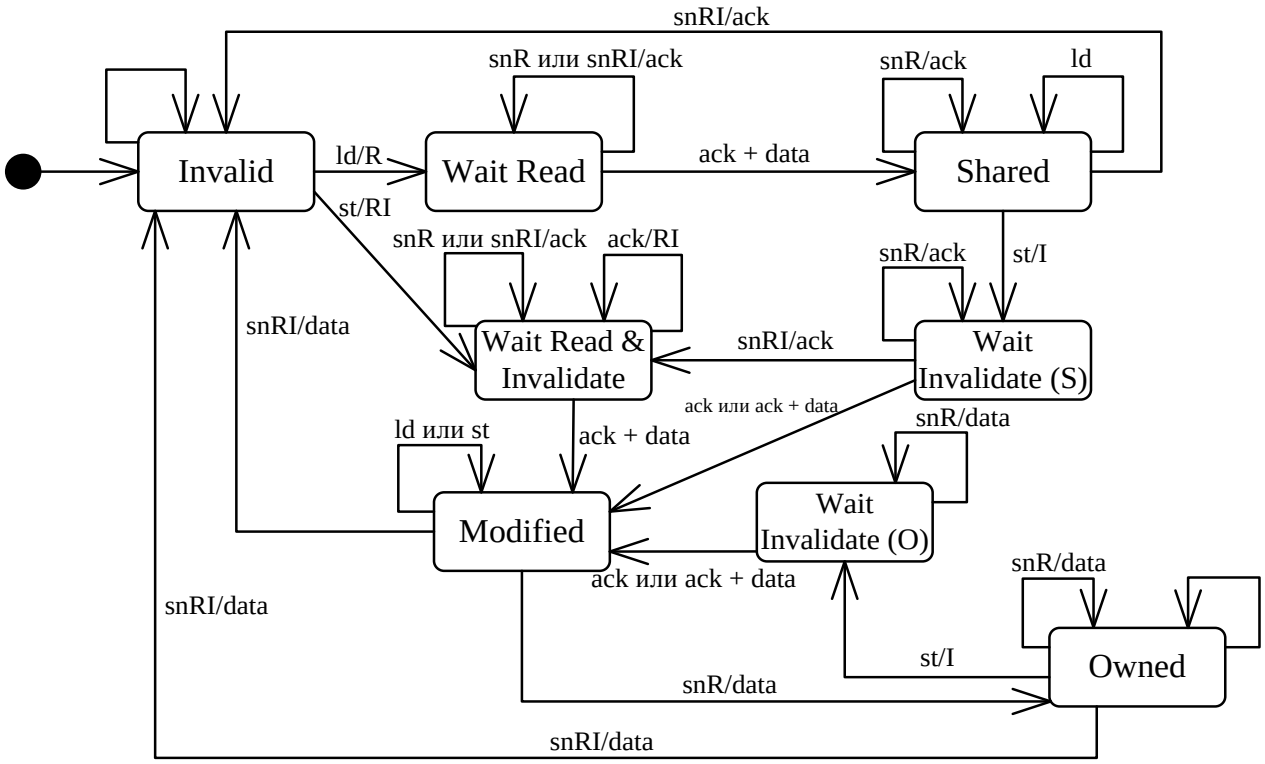


Рисунок 16 – Часть автомата состояний контроллера кэш-памяти второго уровня микропроцессора Эльбрус-4С

При реализации в аппаратуре некоторые из описанных функций разделены между различными устройствами (например, контроллером кэш-памяти второго уровня и устройством обращения к памяти), а некоторые (передача сообщений) могут происходить посредством дополнительных устройств. Например, сбором ответов на когерентные запросы в микропроцессоре Эльбрус-4С занимается не контроллер банка кэш-памяти второго уровня (данная кэш-память разделена на четыре банка; банк, к которому происходит обращение, определяется отведенными разрядами

физического адреса), а устройство доступа к памяти MAU, которое затем передает их в контроллер кэш-памяти. Однако в соответствии с преследуемыми целями – разработка и проверка формальных моделей протоколов когерентности памяти – необходимо описание протокола когерентности, абстрагирующееся от таких деталей реализаций. Повышение уровня абстракции позволяет уменьшить количество процессов результирующей формальной модели, что необходимо для борьбы с проблемой взрыва числа состояний. Данный подход к разработке формальных моделей соответствует рекомендациям, приведенным в работе [65], по устранению процессов, которые являются только источниками, фильтрами или приемниками сообщений.

4.1.3 Разработка формальной модели протокола когерентности Эльбрус-4С

На основе анализа протокола когерентности системы Эльбрус-4С и рекомендаций по разработке моделей, приведенных в публикации [65], для моделирования протокола когерентности выделены следующие процессы:

- $home$, представляющий системный коммутатор home-процессора,
- $cl2m$, объединяющий функции процессорного ядра, кэш-памяти второго уровня и устройства доступа к памяти.

Модель протокола состоит из одного процесса $home$ и n процессов $cl2m$, взаимодействующих друг с другом в соответствии с протоколом. Данная модель представляет как один процессор, так и систему из процессоров. В последнем случае дополнительно использованы рекомендации из [65]. В системе Эльбрус-4С ядра процессоров, отличных от home-процессора, не отправляют ответы в системный коммутатор home-процессора непосредственно: ответы собирает специальное устройство и отправляет в системный коммутатор один обобщенный ответ. От данных деталей реализации можно абстрагироваться и считать, что все ядра напрямую отправляют ответы в системный коммутатор

home-процессора.

Взаимодействие процессов организовано с помощью четырех объектов: трех каналов и одного массива каналов (рисунок 17). В первый канал процессы $cl2m_i$ отправляют исходные запросы. Процесс $home$ извлекает запросы из этого канала для их последующей обработки. Посредством массива каналов, $home$ отправляет когерентные запросы процессам $cl2m_i$. В третий канал процессы $cl2m_i$ отправляют ответы на когерентные запросы, которые собирает ответственный за сбор ответов процесс. Четвертый канал служит для отправки в $home$ сообщений о завершении обработки запроса.

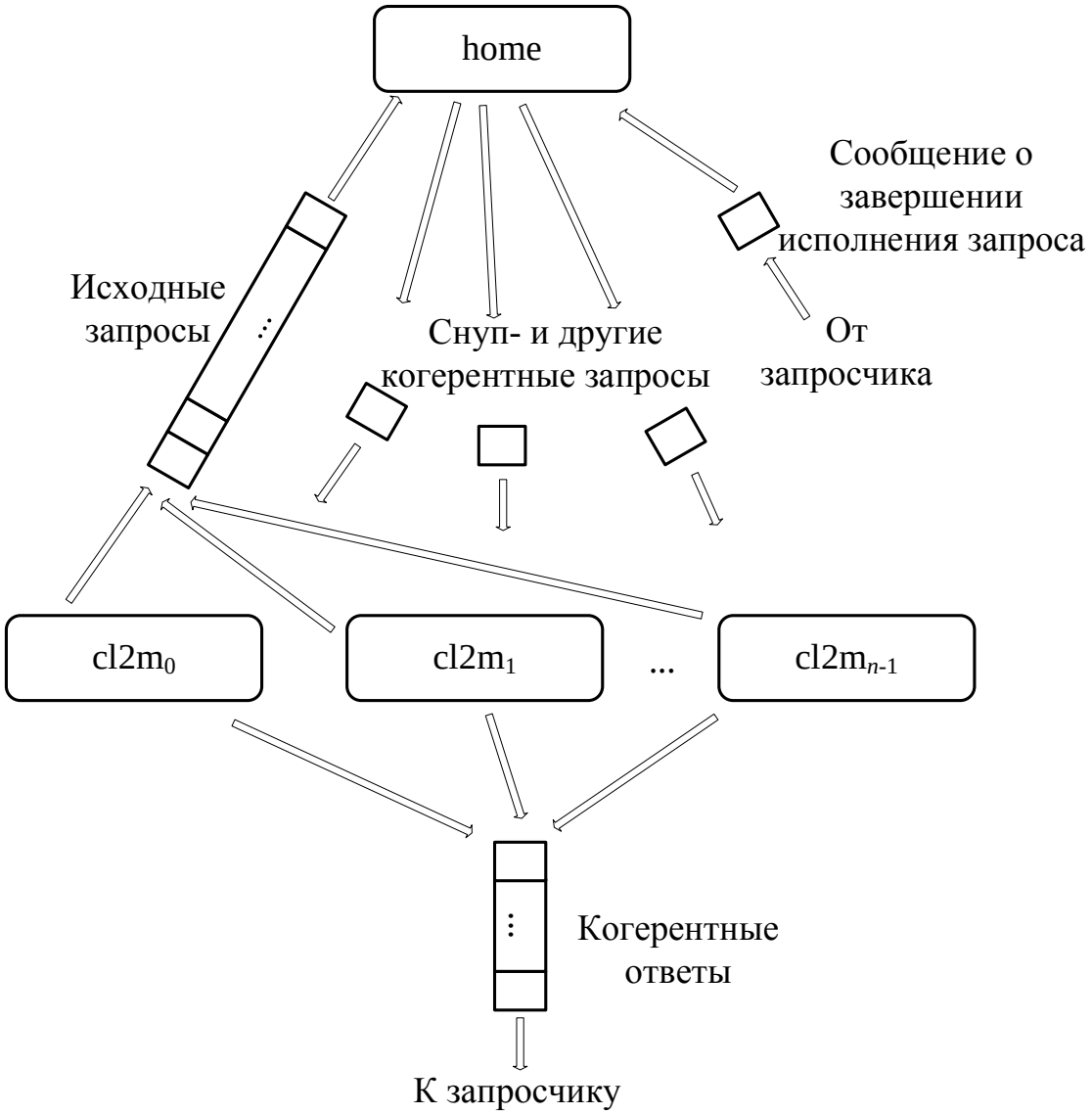


Рисунок 17 – Организация взаимодействия между процессами модели

Поскольку работа протоколов когерентности не предполагает останова, процессы $home$ и $cl2m$ организованы в виде бесконечных циклов. В соответствии с предложенными в главе 2 способами организации процессов, моделирующих системный коммутатор и кэш-контроллеры, структура процессов $home$ и $cl2m_i$ имеет вид, показанный на рисунках 18, 19.

```
proctype home()
{
again:
 <получение нового исходного запроса>
 <анализ исходного запроса>
 <отправка когерентного запроса>
<получение сообщения о завершении исполнения запроса> или
 <сбор когерентных ответов на некоторые типы когерентных запросов>
 <завершение обработки соответствующих исходных запросов>
goto again
}
```

Рисунок 18 – Организация процесса $home$

```
proctype cl2m()
{
 do
 :: <отправка исходных запросов из основных состояний>
 :: <получение и обработка снуп-запроса>
 :: <сбор когерентных ответов>
 :: <отправка сообщения о завершении исполнения запроса в home>
 od
}
```

Рисунок 19 – Организация процессов $cl2m_i$

Как видно из рисунков, предлагаемая организация процессов в явном виде отражает все стадии выполнения запроса.

4.1.4 Уточнение абстрактной модели протокола Эльбрус-4С

Абстрактная модель получается автоматически на основании исходной модели путем использования разработанного инструментального средства.

После применения абстрактных преобразований к исходной модели, защиты части множества команд абстрактного процесса $cl2m\_abs$ становятся истинными. Ограничение поведения данного процесса осуществляется путем уточнения абстракции с помощью процедуры, описанной в главе 3.

В микропроцессоре Эльбрус-4С обращениям к памяти с типами памяти write back, write through и write combined от процессорных ядер соответствуют четыре типа исходных запросов. Запросы одного типа обрабатываются в соответствии с одним и тем же алгоритмом. Тип (R, RI, I) включает запросы на когерентное считывание (R), когерентное считывание с вычеркиванием строки из кэша (RI) и вычеркивание строки из кэша (I). Тип W1 представляет запрос на запись данных в память по маске. Тип W2 представляет запрос на запись данных в память. Тип WB представляет запрос на запись модифицированной строки из кэш-памяти в основную память.

Далее представлены множества A, сформированные на основании документации на микропроцессор Эльбрус-4С для всех четырех типов исходных запросов: (R, RI, I), W1, W2, WB.

Для типа (R, RI, I). Множество $(A, \prec)$:

$\{a_0 =$ завершение исполнения предыдущего запроса от процесса $cl2m_i, 1 \leq i \leq n$,

$a_1 =$ отправка исходного запроса запросчиком $cl2m_i$,

$a_2 =$ получение исходного запроса процессом $home$,

$a_3 =$ отправка процессом $home$ снуп-запросов во все процессы $cl2m_j, 1 \leq j \leq n, j \neq i$,

$a_4 =$ получение снуп-запроса процессом $cl2m_j, 1 \leq j \leq n, j \neq i$,

a_5 = отправка процессом $cl2m_j$ ответа на снуп-запрос запросчику,

a_6 = получение запросчиком когерентного ответа от процесса $cl2m_j$,

a_7 = отправка запросчиком сообщения о завершении исполнения запроса в $home$,

a_8 = получение процессом $home$ сообщения о завершении исполнения запроса}.

Отношение $\prec$ определяется следующим образом: $\forall i, j = 0, \dots, |A| - 1: i < j \Rightarrow a_i \prec a_j$. Отождествим вспомогательные переменные с самими элементами множества A.

Для типа W1. Множество $(A, \prec)$:

$\{a_0 =$ завершение исполнения предыдущего запроса от процесса $cl2m_i, 1 \leq i \leq n$,

a_1 = отправка исходного запроса запросчиком $cl2m_i$,

a_2 = получение исходного запроса процессом $home$,

a_3 = отправка процессом $home$ снуп-запросов во все процессы $cl2m_j, 1 \leq j \leq n$,

a_4 = отправка процессом $home$ сообщения «Выдать данные» запросчику,

a_5 = получение снуп-запроса процессом $cl2m_j$,

a_6 = получение запросчиком сообщения «Выдать данные»,

a_7 = отправка процессом $cl2m_j$ ответа на когерентный запрос в $home$,

a_8 = получение процессом $home$ когерентного ответа от процесса $cl2m_j$,

a_9 = отправка процессом $home$ сообщения о завершении запроса запросчику,

a_{10} = получение запросчиком сообщения о завершении исполнения запроса}.

Отношение $\prec$ определяется следующим образом: $\forall i, j = 0, \dots, |A| - 1: i < j \Rightarrow a_i \prec a_j$. Отождествим вспомогательные переменные с самими элементами

множества A.

Для типа W2 множество A аналогично соответствующему множеству для запросов типа W1 с тем исключением, что для типа W2 в множестве A отсутствуют элементы a_9 и a_{10}.

Для типа WB. Множество $(A, \prec)$:

$\{a_0 =$ завершение исполнения предыдущего запроса от процесса $cl2m_i, 1 \leq i \leq n$,

$a_1 =$ отправка исходного запроса запросчиком $cl2m_i$,

$a_2 =$ получение исходного запроса процессом $home$,

$a_3 =$ отправка процессом $home$ сообщения «Выдать данные» запросчику,

$a_4 =$ получение запросчиком сообщения «Выдать данные»,

$a_5 =$ отправка запросчиком ответа на сообщение «Выдать данные» в $home$,

$a_6 =$ получение процессом $home$ когерентного ответа от запросчика$\}$.

Отношение $\prec$ определяется следующим образом: $\forall i, j = 0, \dots, |A| - 1: i < j \Rightarrow a_i \prec a_j$. Отождествим вспомогательные переменные с самими элементами множества A.

Уточнение абстрактной модели потребовало введения двух вспомогательных переменных.

Анализ первого контрпримера показал, что абстрактный процесс $cl2m\_abs$ отправляет в $home$ подтверждение о завершении обработки запроса первого типа (R, RI, I) еще до получения первого когерентного ответа. Анализ соответствующего множества A позволяет найти в качестве действия, которое привело к возникновению контрпримера, действие a_7 («отправка запросчиком сообщения о завершении исполнения запроса в $home$»). В соответствии с процедурой уточнения, в множестве A находим действие a_6 («получение запросчиком когерентного ответа от процесса $cl2m_j$») и в качестве локальной переменной процесса $cl2m\_abs$ вводим вспомогательную переменную

ack_received с начальным значением $false$. Заменяем в модели оператор, соответствующий действию a_6, на атомарную последовательность, состоящую из данного оператора и оператора присваивания переменной ack_received значения $true$. Добавляем ack_received к защите команды процесса $cl2m\_abs$, содержащей действие a_7, и заменяем оператор, соответствующий данному a_7 на атомарную последовательность из этого оператора и присваивания переменной ack_received значения $false$. Таким образом обеспечивается, что процесс $cl2m\_abs$ не отправит подтверждение до получения первого когерентного ответа.

Анализ второго контрпримера показал, что абстрактный процесс $cl2m\_abs$ спонтанно отправляет в $home$ ответ на сообщение «Выдать данные». Такое действие находится в множестве A для запросов типа WB: a_5 («отправка запросчиком ответа на сообщение «Выдать данные» в $home$»). В данном множестве находим действие $a_3 \prec a_5$ («отправка процессом $home$ сообщения «Выдать данные» запросчику»). Вводим вспомогательную переменную wb_from_abs. В абстрактной модели оператор, соответствующий a_3, удален, поэтому на его место записываем присваивание переменной wb_from_abs значения $true$. Выполняем модификацию защиты и присваивание wb_from_abs значения $false$ в соответствии с процедурой. Теперь абстрактный процесс может отправить рассматриваемый ответ только в то время, когда $home$ действительно ожидает этот ответ.

После введения двух вспомогательных переменных верификация модели протокола когерентности памяти системы Эльбрус-4С завершилась без возникновения контрпримеров.

4.2 Реализация инструмента построения внутреннего представления Promela-моделей

4.2.1 Выбор внутреннего представления Promela-моделей

Синтаксические преобразования Promela-моделей, приведенные в главе 3, могут быть выполнены автоматически. Для автоматизации преобразований выбран часто используемый для такого рода задач подход, в соответствии с которым на основе исходного кода модели конструируется дерево абстрактного синтаксиса [37], представляющее синтаксическую структуру кода, а далее при обходе этого дерева выполняются преобразования. Далее описана разработка инструментального средства, реализующего такой подход (рисунок 20).

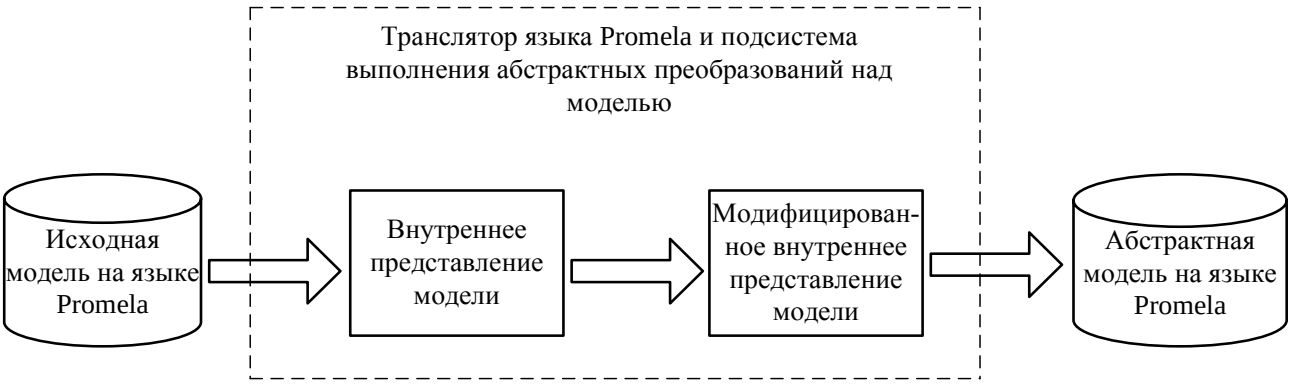


Рисунок 20 – Схема автоматизированного выполнения абстрактных преобразований

Конструирование дерева абстрактного синтаксиса обычно осуществляется на этапе синтаксического анализа исходного кода. Реализовать синтаксический анализатор можно либо вручную, либо с использованием генераторов синтаксических анализаторов, таких как Bison, ANTLR, Boost.Spirit. Ввиду чрезмерной трудоемкости первого пути выбран второй путь.

В качестве инструментального средства построения дерева абстрактного синтаксиса выбрана библиотека Boost.Spirit (в данной работе использована ее

часть Boost.Spirit.Qi версии V2), реализованная на языке C++ и являющаяся частью известного рецензируемого специалистами набора библиотек Boost [116, 90]. Данная библиотека с помощью таких техник программирования, как переопределение операторов, шаблоны выражений (expression templates) и шаблонное метапрограммирование (template metaprogramming), реализует проблемно-ориентированный встроенный язык, который позволяет описывать грамматику рассматриваемого языка на языке C++ и без использования дополнительных инструментов строить синтаксический анализатор языка, определяемого данной грамматикой.

Грамматика в Boost.Spirit описывается с помощью множества правил. Каждое правило соответствует нетерминалу грамматики рассматриваемого языка. Правила могут принимать параметры (наследуемые атрибуты) и возвращать значение (синтезируемые атрибуты). Типы наследуемых и синтезируемых атрибутов должны быть явно указаны при определении правила. Boost.Spirit реализует определенные правила распространения типов атрибутов для составных правил, таких как последовательности, альтернативы, звезда Клини и опциональные элементы. Это позволяет разбить задачу построения дерева абстрактного синтаксиса на две последовательно выполняемые задачи:

1. Разработка грамматики, ее тестирование и отладка. На данном этапе внимание уделяется только способности Spirit-грамматики правильно отвечать на вопрос о синтаксической корректности Promela-модели. Аспекты, касающиеся синтезируемых атрибутов правил, игнорируются.

2. Разработка структур данных для узлов дерева абстрактного синтаксиса и назначение типов атрибутов правил. С помощью механизма атрибутов построение дерева абстрактного синтаксиса выполняется автоматически, без явного добавления операторов конструирования узлов дерева в грамматику.

Таким образом, факторами, определившими выбор Boost.Spirit, являются:

- возможность программирования на языке C++ с использованием современных подходов к программированию, позволяющих работать с удобными для данной проблемной области абстракциями без потери производительности относительно более низкоуровневых стилей программирования (например, практиковавшихся ранее на языке C);
- отсутствие необходимости использования отдельных инструментальных средств, таких как Bison и ANTLR (для разработки необходимы только компилятор языка C++ и библиотека Boost);
- атрибутивность Spirit-грамматик, обеспечивающая удобную поддержку процесса построения деревьев абстрактного синтаксиса;
- наличие множества встроенных синтаксических анализаторов;
- эффективность получаемых синтаксических анализаторов [41].

Использование дерева абстрактного синтаксиса, построение которого выполняется с помощью библиотеки Boost.Spirit, в качестве промежуточного представления Promela-моделей позволяет разбить процесс решения задачи автоматического выполнения абстрактных преобразований на три этапа (рисунок 21).

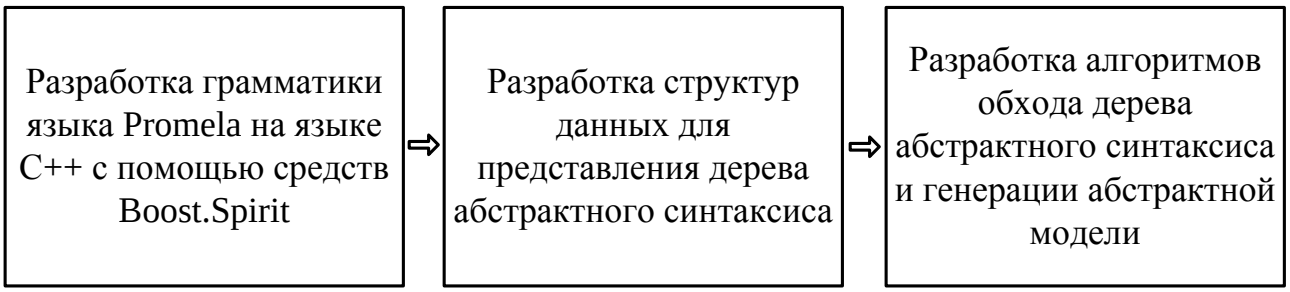


Рисунок 21 – Этапы решения задачи автоматического построения абстрактной модели

4.2.2 Разработка грамматики языка Promela с помощью средств Boost.Spirit

Первым этапом построения инструмента является разработка грамматики языка Promela.

В качестве основы при разработке Spirit-грамматики языка Promela использовано существующее описание грамматики [109]. Поскольку Spirit – библиотека генерации синтаксических анализаторов, работающих по методу рекурсивного спуска, данное описание было модифицировано таким образом, чтобы получаемые синтаксические анализаторы могли с ним корректно работать. В частности, были устранены все случаи использования левой рекурсии.

Как правило, при использовании Boost.Spirit.Qi не предполагается использование лексических анализаторов, и работа, которая обычно выполняется на этапе лексического анализа, осуществляется с помощью специальных средств библиотеки. Для реализации функциональности пропуска некоторых символов и их последовательностей – например, пробелов и комментариев, – правила в качестве одного из шаблонных параметров принимают тип объекта, реализующего такую функциональность (skipper). Для этого была описана грамматика конструкций, которые необходимо пропустить:

```
skip =
        qi::space
        | ("/*" >> *(qi::char_ - "*/") >> "*/")
        | ("//" >> *(qi::char_ - qi::eol) >> qi::eol)
        ;
```

В некоторых случаях при распознавании неделимых компонентов (лексем) необходимо отключить функциональность пропуска символов, для чего использована директива qi::lexeme[].

В целом, Boost.Spirit предоставляет достаточно удобные средства описания грамматик, основанные на PEG (Parsing Expression Grammar) [50], что позволило реализовать грамматику языка Promela. Помимо этого, в библиотеке имеются средства отладки [40], позволяющие зафиксировать интересующие правила грамматики и с помощью печати выводить информацию об их применении. Данные средства позволили определить источники ошибок, которые возникали при разработке и приводили к неверному заключению о корректности модели.

4.2.3 Разработка структур данных для дерева абстрактного синтаксиса

Вторым этапом построения инструмента является разработка структур данных для представления дерева абстрактного синтаксиса. На данном этапе учитываются правила распространения типов атрибутов для составных правил Spirit-грамматики. Для каждой конструкции языка определяется, какая информация будет представлена в соответствующем узле дерева. Несущественная пунктуация и разделители (скобки, запятые, точки с запятой и т. п.) в узлах дерева не представляется.

Основные правила распространения атрибутов, которые были использованы при разработке структур данных для дерева абстрактного синтаксиса, приведены в таблице 11.

Таблица 11 – Правила распространения атрибутов

Тип синтаксического анализатора из библиотеки Boost.Spirit.Qi	Тип синтезируемого атрибута
Литеральный: ‘a’, “str”, lit	Отсутствует
Анализатор-примитив: char_, string(“str”	Соответствующий тип языка C++ (символ, строка символов)
Нетерминал: rule<Type()>, grammar<Type()>	Type

Тип синтаксического анализатора из библиотеки Boost.Spirit.Qi	Тип синтезируемого атрибута
Оператор-последовательность: a >> b	boost::fusion::vector<A, B>, где A – тип синтезируемого атрибута анализатора a, B – тип синтезируемого атрибута анализатора b
Оператор-альтернатива: a | b	boost::variant<A, B>
Оператор-звезда Клини («нуль или больше»): *a	std::vector<A>
Оператор «один или больше»: +a	std::vector<A>
Оператор «нуль или один»: -a	boost::optional<A>
Предикаты: &a, !a	Отсутствует

В соответствии с таблицей, основными типами данных, использованными при построении дерева абстрактного синтаксиса, являются:

- примитивные типы языка C++;
- контейнер из стандартной библиотеки языка C++ std::vector (все типы, находящиеся в пространстве имен std, являются частью стандартной библиотеки языка C++ [113]);
- контейнер объектов символьного типа std::string;
- Boost.Fusion-последовательность;
- размеченное объединение (discriminated union) boost::variant, позволяющее работать с объектами, конструируемыми на основе гетерогенного множества типов. Разрешение циклических зависимостей и возможность использования рекурсивных типов в разработанной системе осуществлено с помощью использования объектов класса boost::recursive_wrapper<T> вместо объектов класса T;
- класс boost::optional, позволяющий представлять объекты, которые могут не содержать значения.

Все указанные типы данных имеют семантику значений (value semantics), что делает реализацию синтаксического анализатора достаточно высокоуровневой так, что она не предполагает использование указателей и ручного управления памятью. Это позволяет сфокусировать внимание на абстракциях данной предметной области (синтаксического анализа).

Контейнер std::vector позволяет работать с множеством значений одного типа, а объекты типа boost::variant могут хранить данные различных типов, но иметь только одно значение.

Boost.Fusion-последовательности использованы для постановки в соответствие правилам грамматики C++-структуры (struct). При этом для распознания библиотекой Boost.Spirit этой структуры как Fusion-последовательности, использована ее адаптация с помощью макроса BOOST_FUSION_ADAPT_STRUCT.

Важно, чтобы тип синтезируемого атрибута, назначенный правилу Spirit-грамматики, соответствовал правилам распространения атрибутов. В остальном, выбор информации, представляемой в узлах дерева абстрактного синтаксиса, осуществляется в соответствии с преследуемыми целями. В разработанном инструменте в структурах данных, соответствующих синтезируемым атрибутам правил, фиксируется вся информация о входящих в состав правила нетерминалах.

Характерные примеры применения указанных правил приведены в таблице 12. Предполагается, что реализация соответствующих структур данных находится в пространстве имен ast. При определении правила указывается три шаблонных параметра: тип итератора, тип синтезируемого атрибута и тип объекта, пропускающего символы. Обычно тип синтезируемого атрибута правила R имеет название ast::R, в некоторых случаях это std::string.

Таблица 12 – Правила грамматики и соответствующие типы синтезируемых атрибутов

Правило грамматики	Реализация типа синтезируемого атрибута на языке C++ и комментарии
qi::rule<Iterator, ast::program(), Skipper> program; program = +module;	struct program : std::vector<module> {};
qi::rule<Iterator, ast::module(), Skipper> module; module = proctype | init | ltl | utype | mtype | decl_lst | ';' ;	using module = boost::variant< proctype, init, ltl, utype, mtype, decl_lst >;

qi::rule<Iterator, ast::proctype(), Skipper> proctype;
proctype =
qi::lit("proctype")
>> name >> '('
>> -decl_lst >> ')' >> '{'
>> sequence
>> '}' ;
	struct proctype
	{
	std::string name;
	boost::optional<decl_lst> decls;
	sequence seq;
	};
	
	Адаптация структуры (осуществляется в глобальном пространстве имен):
	BOOST_FUSION_ADAPT_STRUCT(
	ast::proctype,
	(std::string, name)
	(boost::optional<ast::decl_lst>, decls)
	(ast::sequence, seq)
	)
	
	Примечание. Тип синтезируемого атрибута правила name – std::string.

Правило грамматики	Реализация типа синтезируемого атрибута на языке C++ и комментарии

qi::rule<Iterator, ast::mtype(), Skipper> mtype;
mtype =
qi::lit("mtype") >> -qi::lit('=')
>> '{' >> name
>> *(',' >> name)
>> '}';
	struct mtype
	{
	std::string first;
	std::vector<std::string> rest;
	};
	
	Представление множества в виде первого элемента и вектора из оставшихся элементов используется в тех случаях, когда обработка первого и остальных элементов в некоторой степени различается.

qi::rule<Iterator, ast::assign(), Skipper> assign;
assign =
varref >> '='
>> expr;
	struct assign
	{
	varref lhs;
	expr rhs;
	};

qi::rule<Iterator, ast::statement(), Skipper> statement;
qi::rule<Iterator, ast::labeled_statement(), Skipper> labeled_statement;
 	
statement =
// …
| labeled_statement
// …
 	
labeled_statement =
name >> ':'
>> statement
;
	struct labeled_statement;
	
	using statement = boost::variant<
	// …
	recursive_wrapper<labeled_statement>,
	// …
	>;
	
	struct labeled_statement
	{
	std::string name;
	statement stmt;
	};
	
	Данный пример иллюстрирует разрешение циклических зависимостей.

Конструирование дерева абстрактного синтаксиса на основе разработанной Boost.Spirit-грамматики языка осуществляется с помощью вызова функции qi::phrase_parse(), находящейся в библиотеке Boost.Spirit.

Основная работа по осуществлению синтаксических преобразований $TA_1, \dots TA_3, TE_1, \dots, TE_3, TC_1, \dots, TC_5$ производится при обработке узлов дерева абстрактного синтаксиса, имеющих определенные типы, причем множество этих типов является собственным подмножеством множества всех типов узлов. Ниже описаны такие типы объектов, разработанные в соответствии с грамматикой языка Promela. Описание состоит из имени типа, написанного с заглавной буквы, далее в фигурных скобках – списка составляющих компонентов. Необязательные элементы данного списка заключены в квадратные скобки.

Обращение к переменной в Promela может быть представлено строкой, удовлетворяющей правилам именования идентификаторов в Promela, после которой может быть два опциональных элемента: индекс обращения к элементу массива, заключенный в квадратные скобки, и имя элемента структуры, предваряемое точкой:

Переменная
{
    Строка Имя
    [Выражение Индекс]
    [Переменная Элемент]
}

Оператор присваивания в Promela специфицирует слева объект и справа, после знака «=», значение, которое ему присваивается:

Присваивание
{
    Переменная Лев

Выражение Прав

}

Оператор отправки сообщения по каналу указывает идентификатор канала и список отправляемых аргументов:

Отправка

{

Переменная Имя_канала

Список_аргументов_отправки Аргументы

}

Список аргументов отправки является списком выражений, представленным следующим образом:

Список_аргументов_отправки

{

Выражение Начало

Вектор<Выражение> Остаток

}

Здесь Вектор – тип, соответствующий типу std::vector; в угловых скобках указан тип хранимых в векторе элементов.

Оператор приема сообщения по каналу указывает идентификатор канала и список принимаемых аргументов:

Прием

{

Переменная Имя_канала

Список_аргументов_приема Аргументы

}

Список аргументов приема является списком переменных:

Список_аргументов_приема

{

Переменная Начало

Вектор<Переменная> Остаток

}

Выражение представляется списком компонентов выражения (что представлено здесь в несколько упрощенном виде), которые могут быть объектами одного из следующих типов:

Компонент_выражения =

Переменная

| Выражение

| Константа

| Опрос_состояния_канала

| Создание_экземпляра_процесса

Выражение

{

Компонент_выражения Начало

Вектор<(Оператор, Компонент_выражения)> Остаток

}

Здесь Оператор представляется с помощью типа Строка.

4.2.4 Обход дерева абстрактного синтаксиса и разработка алгоритмов, осуществляющих абстрактные преобразования

Третьим этапом построения инструмента является разработка алгоритмов обхода дерева абстрактного синтаксиса и генерации абстрактной модели.

Построенное дерево состоит из узлов различных типов, и обход такого дерева, то есть посещение всех узлов, осуществлен в единообразном стиле с использованием посетителей.

Разработка посетителя осуществляется следующим образом. Определяется класс, который назовем Visitor (имя класса не имеет значения, и

может быть несколько таких классов), в котором для каждого типа узлов дерева абстрактного синтаксиса переопределяется оператор вызова функции operator()(). Выбирается тип возвращаемого значения таких операторов. Для этого типа в классе должен быть определен псевдоним result_type. В разработанном инструменте для всех посетителей объекты типов, представляющих узлы дерева, в перегруженные операторы передаются по константной ссылке, а возвращаемым значением операторов является строка std::string, представляющая ту или иную конструкцию Promela-модели:

using result_type = std::string;

Для обхода дерева необходимо создать объект класса Visitor и вызвать с использованием этого объекта перегруженный оператор вызова функции, передав в него объект типа, представляющего дерево абстрактного синтаксиса (тип, соответствующий конструкции самого верхнего уровня в грамматике языка).

Алгоритмы, осуществляющие абстрактные преобразования, реализованы в перегруженных операторах вызова функции. Преобразование выполняется за три этапа (рисунок 22).

В приведенных ниже алгоритмах использованы следующие обозначения. Присваивание отмечено знаком ≔. Оператор возврата $return$ аналогичен соответствующему оператору языков C и C++. $guarded$ – переменная логического типа, изначально принимающая значение $false$. $Locals$ – множество локальных переменных процесса $ccon$, используется при построении процесса abs. $Received$ – множество полученных сообщений, элементы которого представляют элементы множества V_{loca}. $Sent$ – отношение, с помощью которого устанавливается соответствие операторов приема и отправки сообщений.

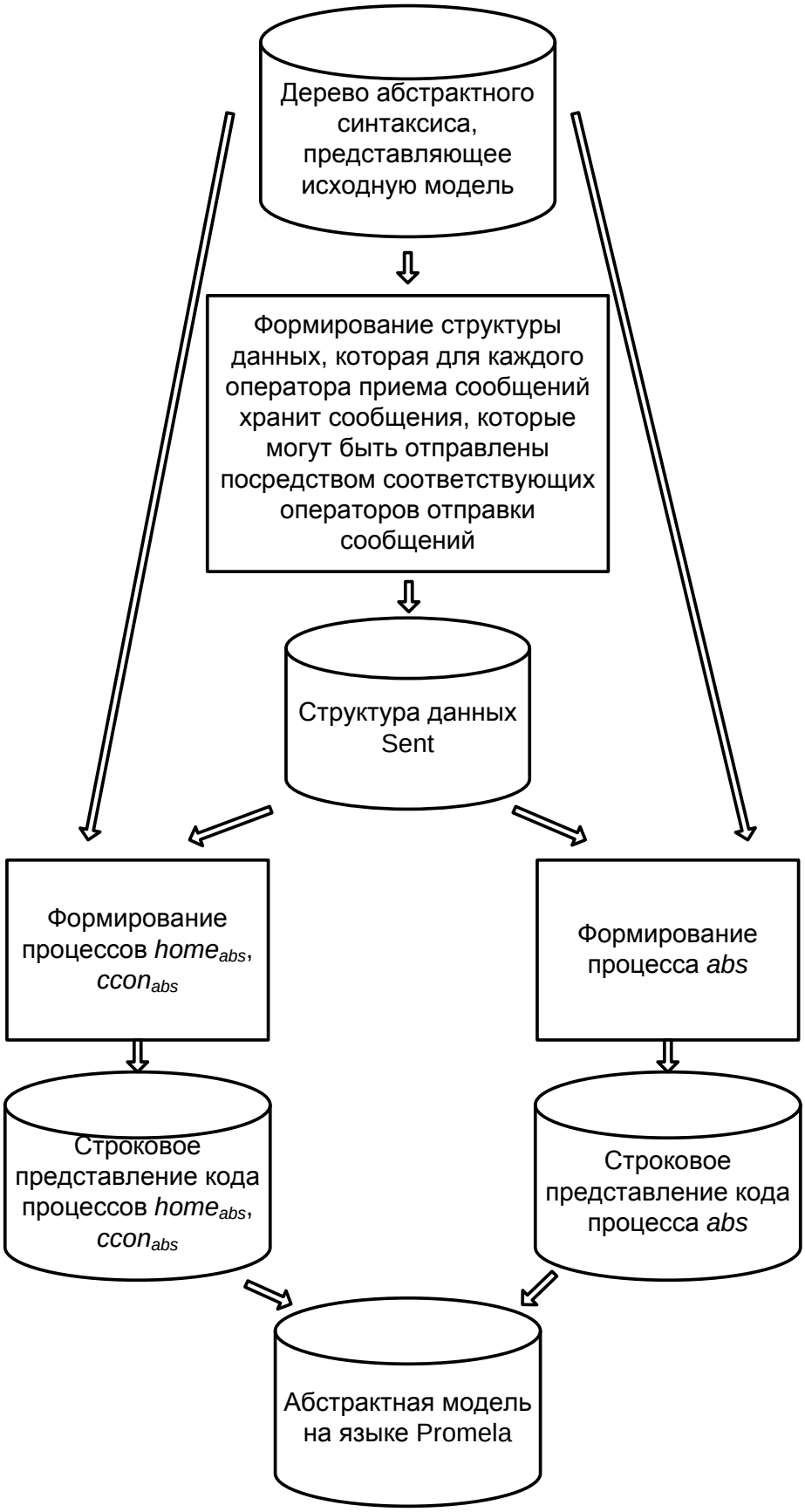


Рисунок 22 – Этапы формирования абстрактной модели

Отношение $Sent$ строится таким образом, что при анализе оператора приема сообщения по каналу c, $Sent[c]$ является множеством сообщений, которые могут быть отправлены посредством операторов отправки сообщений, соответствующих данному оператору приема сообщений.

Условия, позволяющие определить, какой в настоящий момент рассматривается оператор, реализуются анализом значения дополнительно введенных переменных логического типа, изначально принимающих значение $false$. При входе в оператор обработки некоторой конструкции, значение соответствующей переменной устанавливается равным $true$, а при выходе из него – равным $false$.

Конструкции в скобках, например, (П. Имя), означают вызов подходящего оператора с соответствующим параметром. Вызываемый оператор определяется типом параметра.

Алгоритм обработки объектов типа Переменная создает предпосылки для осуществления множества из рассматриваемых преобразований.

Алгоритм обработки объектов типа Переменная

Вход: объект П типа Переменная.

Выход: строка, соответствующая обработке данного объекта П.

1. Присваиваем пустую строку в качестве начального значения переменной, хранящей результат работы алгоритма:

   Результат := ""

2. Получаем имя переменной:

   Имя := (П. Имя)

3. Если анализируется тело процесса abs, и рассматривается индекс обращения к элементу массива, и Имя $\in Locals$, и Имя $\notin Received$ (условие осуществления $\boldsymbol{TE_1}$), то возвращаем пустую строку:

   $return$ ""

4. Если определен П. Индекс, то

4.1. Получаем индекс:

Индекс := П. Индекс

4.2. Если Индекс является числом и Индекс > 2 (условие осуществления $\boldsymbol{TA_2}$, $\boldsymbol{TE_2}$, $\boldsymbol{TC_3}$ и $\boldsymbol{TC_5}$), то возвращаем пустую строку:

$return$ ""

4.3. Если Индекс не является числом или параметром процесса – условие выполнения $\boldsymbol{TA_3}$ и $\boldsymbol{TC_4}$ – то устанавливаем значение переменных, которые будут использованы при выполнении преобразований:

$guarded := true$

$lhs\_idx :=$ Индекс

5. Если определен П. Элемент, то добавляем его к результату:

Результат := "." + (П. Элемент)

6. Возвращаем результат:

$return$ Результат

Данный алгоритм также создает предпосылку для осуществления преобразования $\boldsymbol{TE_3}$ одновременно с предпосылкой для TE_1, поскольку множество каналов C_2 представлено массивом, и канал $c_{2,i}$ – i-й элемент этого массива. Индекс i при модификации процесса abs с помощью данного алгоритма заменяется на пустую строку, что приводит к замене выражений относительно соответствующего канала на $true$.

Алгоритм обработки объектов типа Присваивание

Вход: объект П типа Присваивание.

Выход: строка, представляющая результат обработки соответствующего оператора присваивания согласно преобразованиям TA_1, TA_2, TA_3.

1. Присваиваем пустую строку в качестве начального значения переменной, хранящей результат работы алгоритма:

Результат ≔ ""

2. Получаем имя переменной, которой присваивается значение:

Лев ≔ П. Лев

3. Если Лев = "", то выполняем $\boldsymbol{TA_2}$:

$return$ ""

4. Если анализируется тело процесса abs и Лев ∈ $Locals$, то выполняем преобразование $\boldsymbol{TA_1}$:

$return$ ""

5. Если $guarded = true$, то выполняем первую часть $\boldsymbol{TA_3}$:

Результат ≔ Результат + *“if::”* + *lhs_idx* + *“!=ABS->”*

6. Фиксируем имя переменной, которой присваивается значение:

Результат ≔ Результат + Лев + " = "

7. Получаем присваиваемое значение:

Результат ≔ Результат + (П. Прав)

8. Если $guarded = true$, то выполняем вторую часть $\boldsymbol{TA_3}$:

$guarded ≔ false$

Результат ≔ Результат + *“::else fi”*

9. Возвращаем результат:

$return$ Результат

Алгоритм обработки объектов типа Выражение

Вход: объект B типа Выражение, полученный при разборе процесса abs.

Выход: строка, полученная на основе объекта B в соответствии с преобразованиями TE_1 и TE_3.

1. Присваиваем пустую строку в качестве начального значения переменной, хранящей результат работы алгоритма:

Результат ≔ ""

2. Анализируем первый компонент объекта B:

Начало ≔ (B. Начало)

3. Если (Начало – пустая строка) или (Начало $\in Locals$, и Начало $\notin Received$, и не рассматривается оператор отправки сообщения, поскольку процесс abs может отправить свой идентификатор), то:

3.1. Если не рассматривается оператор присваивания, приема сообщения или предикат опроса состояния канала, то производим замену компонента на $true$ (осуществление $\boldsymbol{TE_1}$):

Результат := Результат + *"true"*

3.2. Иначе возвращаем пустую строку:

$return$ ""

4. Иначе добавляем данный компонент к результату:

Результат := Результат + Начало

5. Для всех элементов (Оп, К) структуры данных В. Остаток осуществляем их исследование и добавляем соответствующий результат к строке-результату:

Результат := Результат + ((Оп, К))

6. Возвращаем результат:

$return$ Результат

Алгоритм обработки объектов типа (Строка, Компонент_выражения)

Вход: пара объектов (Оп, К) типа (Строка, Компонент_выражения).

Выход: строка, полученная при разборе данной пары.

1. Присваиваем пустую строку в качестве начального значения переменной, хранящей результат работы алгоритма:

Результат := ""

2. Добавляем оператор к результату:

Результат := Результат + Оп

3. Добавляем к результату результат исследования второго компонента пары:

Результат := Результат + (К)

4. Возвращаем результат:

$return$ Результат

Алгоритм обработки объектов типа Отправка

Вход: объект От типа Отправка.

Выход: строка, полученная на основе объекта От в соответствии с преобразованиями TC_1, TC_3, TC_4.

1. Присваиваем пустую строку в качестве начального значения переменной, хранящей результат работы алгоритма:

Результат ≔ ""

2. Получаем имя канала:

Имя ≔ (От. Имя_канала)

3. Если Имя – пустая строка, то возвращаем пустую строку в качестве результата (выполнение $\boldsymbol{TC_3}$):

$return$ ""

4. Если анализируется тело процесса abs и Имя $\in C_1$, то возвращаем пустую строку (выполнение $\boldsymbol{TC_1}$):

$return$ ""

5. Если $guarded = true$, то дополняем оператор защитой (первый этап выполнения $\boldsymbol{TC_4}$):

Результат ≔ Результат + *“if::”* + *lhs_idx* + *“!=ABS->”*

6. Добавляем имя канала и знак оператора отправки сообщения к результату:

Результат ≔ Результат + Имя + *“!”*

7. Добавляем к результату представление аргументов:

Результат ≔ Результат + (От. Аргументы)

8. Если $guarded = true$, то осуществляем второй этап выполнения $\boldsymbol{TC_4}$:

$guarded \coloneqq false$

Результат ≔ Результат + *“::else fi”*

9. Возвращаем результат:

$return$ Результат

Алгоритм осуществления трансформаций TC_2 в процессе $home_{abs}$

Вход: объект П типа Прием.

Выход: строка, полученная на основе объекта П в соответствии с преобразованием TC_2.

1. Присваиваем начальное значение переменной, хранящей результат работы алгоритма:

Результат ≔ "if"

2. Получаем имя канала:

Имя ≔ (П. Имя_канала)

3. Добавляем имя канала и символ оператора приема сообщения к результату:

Результат ≔ Результат + Имя + *“?”*

4. Добавляем представление сообщения к результату:

Сообщение = (П. Аргументы)

Результат ≔ Результат + Сообщение

5. Для всех элементов $message$ структуры данных $Sent$[Имя], хранящей сообщения из всех операторов отправки, соответствующих данному оператору приема, выполняем дополнение результата альтернативами:

Результат ≔ Результат + *“;::atomic{”* + Сообщение + *“.opc=”* + $message$ + *“;”* + Сообщение + *“.id=ABS}”*

6. Завершаем формирование конструкции недетерминированного выбора:

Результат ≔ "fi"

7. Возвращаем результат:

$return$ Результат

Алгоритм выполнения преобразования TC_2 в процессах $ccon_{abs}$ и abs использует предположение, что модифицируемый оператор $c?$Сообщение находится в блоке (скорее всего, атомарном), представляющем альтернативу do-цикла, с помощью которого организована структура этих процессов. Первым элементом блока является условие его выполнения, в которое входит конструкция $nempty(c)$. Вторым элементом является сам оператор, далее идут несколько операторов, вероятно, использующих Сообщение.

Алгоритм осуществления трансформаций TC_2 в процессах $ccon_{abs}$ и abs

Вход: объект, представляющий блок с преобразуемым оператором отправки сообщения по каналу Имя.

Выход: множество строк, представляющих альтернативы недетерминированного выбора в соответствии с TC_2.

1. Создаем множество строк A, представляющих дополнительные альтернативы do-цикла:

   $A \coloneqq \emptyset$

2. Для всех элементов $message$ структуры данных $Sent$[Имя]:

   2.1. Создаем копию A_i текущего блока.

   2.2. В условии выполнения A_i заменяем вхождения $nempty$(Имя) на $true$.

   2.3. Аналогично алгоритму осуществления трансформаций TC_2 в процессе $home_{abs}$, оператор отправки сообщения в A_i заменяем на Сообщение + *“.opc=”* + $message$ + *“;”* + Сообщение + *“.id=ABS”*.

   2.4. Добавляем результат в A: $A \coloneqq A \cup A_i$.

3. Возвращаем множество A в качестве результата:

   $return\ A$

Алгоритм осуществления преобразования TC_5 в процессе abs

Вход: объект П типа Прием.

Выход: строка, полученная на основе объекта П в соответствии с преобразованием TC_5.

1. Присваиваем пустую строку в качестве начального значения переменной, хранящей результат работы алгоритма:

   Результат ≔ ""

2. Получаем имя канала:

   Имя ≔ (П. Имя_канала)

3. Если Имя – пустая строка, то возвращаем пустую строку (выполняем $\boldsymbol{TC_5}$):

   $return$ ""

4. Добавляем имя канала и символ оператора приема сообщения к результату:

   Результат ≔ Результат + Имя + “?”

5. Добавляем представление сообщения к результату:

   Сообщение ≔ (П. Аргументы)

   Результат ≔ Результат + Сообщение

6. Добавляем полученное сообщение в множество $Received$:

   $Received \coloneqq Received \cup \{$Сообщение$\}$

7. Возвращаем результат:

   $return$ Результат

Множество $Received$ хранит сообщения до окончания текущего блока. Перед выходом из блока (в соответствующем операторе) все элементы из множества удаляются: $Received \coloneqq \emptyset$.

4.3 Сбор и обработка информации по результатам испытаний

4.3.1 Анализ требуемых ресурсов

Для проведения испытаний разработаны несколько версий моделей протокола когерентности системы Эльбрус-4С: модель с поддержкой обращений к памяти с типом write back, модель с поддержкой обращений к памяти с типами write back и write through и модель с поддержкой всех когерентных типов памяти (write back, write through и write combined).

В таблицах 13–16 приведены количество состояний различных моделей и объем ресурсов, требуемые (после устранения всех контрпримеров) для проверки свойства

$$\mathrm{G}\neg(cache[1] = M \wedge cache[2] = M)$$

при использовании различных опций оптимизации, предоставляемых системой Spin. Эксперименты проводились на сервере Intel Xeon E5-2697 (тактовая частота 2,6 ГГц) с 264 Гб оперативной памяти.

Обозначим количество процессорных ядер, отраженных в модели, через n.

Таблица 13 – Результаты верификации модели с поддержкой обращений к памяти с типом памяти Write Back и $n = 3$

Оптимизация	Количество исследованных состояний модели	Объем использованной памяти	Время верификации
Отсутствует	$2{,}1 \cdot 10^4$	7,1 Мб	0,03 с
COLLAPSE	$2{,}1 \cdot 10^4$	6,3 Мб	0,05 с
MA=90	$2{,}1 \cdot 10^4$	6 Мб	0,33 с

Оптимизация	Количество исследованных состояний модели	Объем использованной памяти	Время верификации
HC	$2{,}1 \cdot 10^4$	5,7 Мб	0,02 с
BITSTATE	$2{,}1 \cdot 10^4$	2,3 Мб	0,03 с

Таблица 14 – Результаты верификации модели с поддержкой обращений к памяти с типами памяти Write Back, Write Through и $n = 3$

Оптимизация	Количество исследованных состояний модели	Объем использованной памяти	Время верификации
Отсутствует	$9{,}1 \cdot 10^5$	123 Мб	1,5 с
COLLAPSE	$9{,}1 \cdot 10^5$	58 Мб	2,4 с
MA=97	$9{,}1 \cdot 10^5$	89 Мб	32 с
HC	$9{,}1 \cdot 10^5$	2 Мб	1,1 с
BITSTATE	$9{,}1 \cdot 10^5$	31 Мб	1,4 с

Таблица 15 – Результаты верификации модели с поддержкой обращений к памяти с типами памяти Write Back, Write Through, Write Combined и $n = 3$

Оптимизация	Количество исследованных состояний модели	Объем использованной памяти	Время верификации
Отсутствует	$5{,}1 \cdot 10^6$	682 Мб	9 с
COLLAPSE	$5{,}1 \cdot 10^6$	328 Мб	15 с
MA=97	$5{,}1 \cdot 10^6$	218 Мб	3,5 мин
HC	$5{,}1 \cdot 10^6$	293 Мб	7 с
BITSTATE	$5{,}1 \cdot 10^6$	314 Мб	11 с

Таблица 16 – Результаты верификации модели с поддержкой обращений к памяти с типами памяти Write Back, Write Through, Write Combined и $n = 4$

Оптимизация	Количество исследованных состояний модели	Объем использованной памяти	Время верификации
COLLAPSE	$1{,}3 \cdot 10^9$	81 Гб	1,5 часа
MA=120	$1{,}3 \cdot 10^9$	12,5 Гб	35 часов
HC	$1{,}3 \cdot 10^9$	66 Гб	40 мин
BITSTATE	$1{,}3 \cdot 10^9$	8,2 Гб	1 час

Анализ полученных результатов показывает, что с добавлением алгоритмов обработки запросов новых типов, соответствующих новым типам памяти, сложность моделей и ресурсы, требуемые для их верификации, увеличиваются значительно: с добавлением поддержки каждого нового типа памяти количество состояний модели увеличивалось на порядок. Однако еще более существенным является увеличение количества процессорных ядер, отраженных в модели.

Таблица 16 показывает, что верификация модели, отражающей четыре ядра, с использованием точных методов приводит к значительным затратам памяти или времени. Однако использование приближенного метода (который в данном случае позволил исследовать все пространство состояний) для этой модели позволило провести верификацию при приемлемых емкостных и временных затратах.

Анализ результатов верификации абстрактной модели (таблица 17), из корректности которой следует корректность всего параметризованного семейства моделей, показывает, что количество состояний этой модели меньше количества состояний исходной модели, отражающей три процессора. Таким образом, модель можно еще усложнить, применив метод к более сложному протоколу когерентности, чем протокол системы Эльбрус-4С.

Таблица 17 – Результаты верификации абстрактной модели с поддержкой обращений к памяти с типами памяти Write Back, Write Through, Write Combined

Оптимизация	Количество исследованных состояний модели	Объем использованной памяти	Время верификации
Отсутствует	$2{,}2 \cdot 10^6$	256 Мб	3,7 с
COLLAPSE	$2{,}2 \cdot 10^6$	108 Мб	6,2 с
MA=83	$2{,}2 \cdot 10^6$	43 Мб	43 с
HC	$2{,}2 \cdot 10^6$	107 Мб	2,8 с
BITSTATE	$2{,}2 \cdot 10^6$	141 Мб	4 с

Оптимизация COLLAPSE является оптимизацией, которую можно применять в первую очередь, поскольку она позволяет значительно и без потерь уменьшить объем используемой памяти при не очень существенном увеличении времени верификации. Основные результаты верификации при использовании данной оптимизации сведены в таблицы 18–19.

Таблица 18 – Требуемые ресурсы для верификации исходной модели

Количество ядер, представленных в модели	Число состояний модели	Объем памяти	Время верификации
3	$5{,}1 \cdot 10^6$	328 Мб	15 с
4	$1{,}3 \cdot 10^9$	81 Гб	1,5 часа

Таблица 19 – Требуемые ресурсы для верификации абстрактной модели

Количество ядер, представленных в модели	Число состояний модели	Объем памяти	Время верификации
любое, большее 2	$2{,}2 \cdot 10^6$	108 Мб	6,2 с

Таким образом, предложенный метод верификации Promela-моделей позволил сократить требование ресурсов (памяти) с чрезмерно большого при

количестве ядер, большем, чем 3–4, что свойственно методу проверки моделей, до незначительного. Объем ручной работы при верификации приемлем: для уточнения полученной абстрактной модели потребовалось ввести две вспомогательные переменные логического типа.

4.3.2 Найденные ошибки и верификация протокола с ошибками

В ходе процесса анализа документации и написания формальных моделей была обнаружена серьезная ошибка в протоколе когерентности. В определенной ситуации, контроллер кэш-памяти второго уровня, получив ответ без данных по завершении последовательности событий, инициированной исходным запросом типа Read-Invalidate (считывание и аннулирование), должен повторно выполнить запрос Read-Invalidate. Повторное исполнение в этом случае не было предусмотрено протоколом когерентности, в то время как в других схожих случаях повтор присутствовал. Проявление ошибки в реальной системе приводило к зависанию. Данная ошибка была найдена на поздней стадии верификации, до которой протокол когерентности и его реализация проверялись другими методами в течение двух лет.

Для проверки способности предложенного метода верификации находить ошибки была проведена верификация нескольких ошибочных версий протокола, разработанных на основе анализа ошибок, найденных в микропроцессорах Эльбрус.

Ошибочные версии 1–3 моделируют некорректную модификацию состояния. В состоянии $State$, при получении снуп-запроса, на него отправлен ответ, однако не осуществлена смена состояния на $Invalid$. Версии отличаются состоянием $State$ (таблица 20).

Таблица 20 – Состояния моделей, которые модифицируются некорректно

Версия	Состояние $State$	Снуп-запрос
1	$Shared$	$snRI$ или snI
2	$Owned$	$snRI$
3	$Modified$	$snRI$

Ошибочная версия 4 моделирует потерю снуп-запроса. В состоянии $Shared$ процесс $cl2m$ после получения снуп запроса не отправляет на него ответ и не модифицирует состояние.

Ошибочная версия 5 моделирует некорректное разделение ответственности между L2 и MAU. В ходе исполнения запроса, имеющего тип WB, процесс $cl2m$ отправляет два ответа на снуп-запрос вместо одного.

При верификации всех ошибочных версий были зафиксированы ошибки.

Выводы по главе 4

1. Проведен анализ микропроцессора Эльбрус-4С, в ходе которого выделены устройства данного микропроцессора, реализующие протокол когерентности памяти. Показано соответствие абстракций, рассмотренных в главах 1–3, – кэш-контроллеров и системного коммутатора – устройствам или группам устройств микропроцессора. Разработана формальная модель протокола, обеспечивающего когерентность памяти в системе из четырех микропроцессоров Эльбрус-4С.

2. Разработан инструмент, автоматизирующий преобразования Promela-моделей, описанные в главе 3. Разработаны алгоритмы, выполняющие все указанные преобразования. Данные алгоритмы реализованы в виде алгоритмов модификации дерева абстрактного синтаксиса, построенного на основе исходной Promela-модели.

3. Описан процесс уточнения абстрактной модели протокола

когерентности системы Эльбрус-4С, полученной автоматически с помощью разработанного инструмента. Уточнение проведено в соответствии с процедурой, представленной в главе 3, и потребовало введения двух вспомогательных переменных.

4. Приведены результаты экспериментальных исследований по параметризованной верификации протокола 16-ядерной системы из микропроцессоров Эльбрус-4С, проведенной с помощью разработанных методов и средств. Результаты продемонстрировали незначительные затраты памяти и незначительные затраты машинного времени. Поскольку затраченный объем ручной работы ограничен, такие затраты являются приемлемыми.

Выводы по диссертации

1. Выполнен аналитический обзор методов параметризованной верификации протоколов когерентности памяти. Проведена классификация методов, указаны достоинства и недостатки как отдельных методов, так и групп методов.

2. Выбран язык Promela, позволяющий описывать протоколы когерентности памяти естественным образом, и разработана математическая модель протоколов когерентности памяти, основанная на формальной семантике языка Promela.

3. Разработан новый метод верификации протоколов когерентности памяти, базирующийся на преобразованиях Promela-моделей, приводящих к абстрактным моделям, и позволяющий существенно сократить число состояний модели протокола и проводить верификацию без ограничений на количество процессорных ядер. Поскольку все преобразования осуществляются на синтаксическом уровне, метод позволяет переиспользовать все множество алгоритмов верификации и оптимизаций, реализованное в инструменте Spin.

4. Сформулирована и доказана теорема, определяющая корректность предложенного метода, в которой проведен исчерпывающий анализ соответствия переходов в исходной и абстрактной моделях.

5. Сформулированы ограничения на модели протоколов когерентности и разработан новый подход к описанию моделей, базирующийся на предложенной автором диссертации организации Promela-процессов и использовании примитивов передачи сообщений.

6. Разработан программный инструмент, позволяющий с помощью операций над деревом абстрактного синтаксиса, которое является промежуточным представлением исходной модели, автоматизировать преобразования Promela-моделей. Инструмент устраняет необходимость проведения большого объема ручной работы, при выполнении которой высока

вероятность совершения серьезных ошибок, которые впоследствии могут остаться незамеченными.

7. С использованием разработанных методов и средств проведена верификация протокола когерентности 16-ядерной системы из микропроцессоров Эльбрус-4С, разработанной в АО «МЦСТ», при незначительных затратах памяти и приемлемых временных затратах.

Направления дальнейших исследований по теме диссертации включают:

1. Разработку методов верификации протоколов когерентности памяти, в реализации которых задействованы несколько уровней кэш-памяти. Необходимость разработки таких методов обусловлена введением дополнительного уровня кэш-памяти (кэш-памяти третьего уровня), участвующего в реализации протокола когерентности, в новейших микропроцессорах архитектуры Эльбрус, проектируемых в настоящее время.

2. Разработку методов верификации аппаратных реализаций протоколов когерентности памяти. Сложность аппаратных реализаций значительно превышает сложность самих протоколов, и некоторые нетривиальные ошибки не удается найти на этапе верификации. Необходимы новые методы верификации, позволяющие повысить уверенность в корректности аппаратных реализаций. В этом направлении у автора диссертации имеются определенные результаты. Разработан программный инструмент, позволяющий получать тестовые программы на языке ассемблера Эльбрус на основе формальных моделей протоколов когерентности памяти. Данный инструмент использован для верификации реализаций протоколов когерентности памяти в микропроцессорах Эльбрус-4С, Эльбрус-8С и Эльбрус-8С2, проектируемых в настоящее время в АО «МЦСТ». Инструмент позволил найти более 60 ошибок, включая ошибки, найденные при проведении верификации на поздних стадиях проектирования и не обнаруженные ранее (в течение нескольких лет) другими методами.

Список литературы

1. Буренков, В. С. Анализ применимости инструмента Spin к верификации протоколов когерентности памяти / Буренков В. С. // Вопросы радиоэлектроники. – 2013. – Выпуск 3. – Сер. ЭВТ. – С. 126–134.

2. Буренков, В. С. Анализ применимости формальных методов к верификации протоколов когерентности кэш-памяти масштабируемых систем / Буренков В. С. // Вопросы радиоэлектроники. – 2015. – Выпуск 1. – Сер. ЭВТ. – С. 105–116.

3. Буренков, В. С. Верификация технических систем методом проверки моделей / Буренков В. С., Иванов С. Р. // Современные компьютерные системы и технологии: сб. трудов каф. «Компьютерные системы и сети» МГТУ им. Н.Э. Баумана. – М. : Изд-во НИИ Радиоэлектроники и лазерной техники, 2012. – С. 54–59.

4. Буренков, В. С. Генератор тестов для верификации протокола когерентности кэш-памяти / Буренков В. С. // Вопросы радиоэлектроники. – 2014. – Выпуск 3. – Сер. ЭВТ. – С. 56–63.

5. Буренков, В. С. Генератор тестов для верификации протокола когерентности кэш-памяти [Электронный ресурс] / В. С. Буренков // Молодежный научно-технический вестник. – 2015. – № 2. – Режим доступа: http://sntbul.bmstu.ru/doc/759420.html.

6. Буренков, В. С. Инструмент верификации протокола когерентности памяти [Электронный ресурс] / В. С. Буренков // Молодежный научно-технический вестник. – 2013. – № 1. – Режим доступа: http://sntbul.bmstu.ru/doc/532989.html.

7. Буренков, В. С. Метод масштабируемой верификации PROMELA-моделей протоколов когерентности кэш-памяти / В. С. Буренков, А. С. Камкин // Сб. трудов VII Всероссийской научно-технической конференции «проблемы

разработки перспективных микро- и наноэлектронных систем – 2016». – 2016. – Часть II. – С. 54–60.

8. Буренков, В. С. Метод перебора состояний для верификации протоколов когерентности памяти / В. С. Буренков // Труды 54-й научной конференции МФТИ «Проблемы фундаментальных и прикладных естественных и технических наук в современном информационном обществе». – Москва–Долгопрудный–Жуковский : МФТИ, 2011. – Том 1. – С. 22–23.

9. Буренков, В. С. Метод проверки модели для верификации протоколов когерентности памяти / В. С. Буренков // Труды 55-й научной конференции МФТИ «Проблемы фундаментальных и прикладных естественных и технических наук в современном информационном обществе». – Москва–Долгопрудный–Жуковский : МФТИ, 2012. – Радиотехника и кибернетика : Том 1. – С. 60–61.

10. Буренков, В. С. О консервативном преобразовании формальных моделей, используемых применительно к масштабируемым системам для верификации протоколов когерентности памяти / Буренков В. С. // Вопросы радиоэлектроники. – 2016. – № 3. – Сер. ЭВТ. – С. 48–52.

11. Буренков, В. С. Походы к верификации протоколов когерентности памяти [Электронный ресурс] / В. С. Буренков // Молодежный научно-технический вестник. – 2013. – № 8. – Режим доступа: http://sntbul.bmstu.ru/doc/603343.html.

12. Буренков, В. С. Проблемы параметризованной верификации протоколов когерентности памяти [Электронный ресурс] / Буренков В. С., Иванов С. Р. // Инженерный журнал: наука и инновации. – 2013. – № 11. – Режим доступа: http://www.engjournal.ru/catalog/it/hidden/1013.html.

13. Буренков, В. С. Проблемы формальной верификации технических систем / Буренков В. С., Иванов С. Р., Савельев А. Я. // Наука и образование: научное издание МГТУ им. Н.Э. Баумана. – 2012. – № 4.

14. Карпов, Ю. Г. MODEL CHECKING. Верификация параллельных и распределенных программных систем / Ю. Г. Карпов. – СПб. : БХВ-Петербург, 2010. – 560 с.

15. Коннов, И. В. Верификация параметризованных моделей распределенных систем : дис. … канд. физико-математических наук : 05.13.11 / Коннов Игорь Владимирович. – М., 2008. – 203 с.

16. Куцевол, В. Н. Методология верификации протокола когерентности «Эльбрус-2S» / Куцевол В. Н., Мешков А. Н., Петроченков М. В. // Вопросы радиоэлектроники. – 2013. – Выпуск 3. – Сер. ЭВТ. – С. 107–117.

17. Слесарев, М. В. Определение расчетной частоты эмуляции микропроцессора в прототипе на основе ПЛИС / Слесарев М. В., Юрлин С. В. // Вопросы радиоэлектроники. – 2014. – Выпуск 3. – Сер. ЭВТ. – С. 119–130.

18. Abdulla, P. Parameterized verification through view abstraction / Parosh Abdulla, Frederic Haziza, Lukas Holik // International Journal on Software Tools for Technology Transfer. – 2016. – Vol. 18, Issue 5. – P. 495–516.

19. Adir, A. Generating Concurrent Test-Programs with Collisions for Multi-Processor Verification / A. Adir, G. Shurek // Seventh IEEE International High-Level Design Validation and Test Workshop. – IEEE, 2002. – P. 77–82.

20. Apt, K. Limits for Automatic Verification of Finite-State Concurrent Systems / Krzystof R. Apt, Dexter C. Kozen // Information Processing Letters. – 1986. – Vol. 22, Issue 6. – P. 307–309.

21. Baier, C. Principles of Model Checking / Christel Baier, Joost-Pieter Katoen. – Cambridge, Massachusetts : The MIT Press, 2008. – 984 p.

22. Baukus, K. Parameterized Verification of a Cache Coherence Protocol: Safety and Liveness / Kai Baukus, Yassine Lakhnech, Karsten Stahl // Verification, Model Checking, and Abstract Interpretation. – Springer Berlin Heidelberg, 2002. – P. 317–330.

23. Ben-Ari, M. Principles of the Spin Model Checker / Mordechai Ben-Ari. – London : Springer-Verlag, 2008. – 220 p.

24. Bensalem, S. Abstraction as the Key for Invariant Verification / Saddek Bensalem, Susanne Graf, Yassine Lakhnech // Verification: Theory and Practice. – Springer Berlin Heidelberg, 2003. – P. 67–99.

25. Bingham, J. Automatic Non-Interference Lemmas for Parameterized Model Checking / Jesse Bingham // Formal Methods in Computer-Aided Design. – IEEE, 2008. – P. 1–8.

26. Bosnacki, D. Improving Spin's Partial-Order Reduction for Breadth-First Search / Dragan Bosnacki, Gerard J. Holzmann // Model Checking Software. – Springer Berlin Heidelberg, 2005. – P. 91–105.

27. Brand, D. On Communicating Finite-State Machines / Daniel Brand, Pitro Zafiropulo // Journal of the ACM. – 1983. – Vol. 30, Issue 2. – P. 323–342.

28. Burenkov, V. Applying Parameterized Model Checking to Real-Life Cache Coherence Protocols / Vladimir Burenkov, Alexander Kamkin // Proc. of IEEE East-West Design & Test Symposium (EWDTS). – 2016. – P. 1–4.

29. Burenkov, V. Checking Parameterized PROMELA Models of Cache Coherence Protocols / Burenkov V. S., Kamkin A. S. // Proc. of the Institute for System Programming. – 2016. – Vol. 28, Issue 4. – P. 57–76.

30. Burenkov, V. On the Implementation of a Formal Method for Verification of Scalable Cache Coherent Systems / V. Burenkov // Proc. of the Institute for System Programming. – 2015. – Vol. 27, Issue 3. – P. 183–196.

31. Chen, X. Verification of Hierarchical Cache Coherence Protocols for Futuristic Processors : Doctoral Dissertation / Xiaofang Chen. – University of Utah, Salt Lake City, UT, USA, 2008. – 142 p.

32. Chou, C. A Simple Method for Parameterized Verification of Cache Coherence Protocols / Ching-Tsun Chou, Phanindra K. Mannava, Seungjoon Park //

Formal Methods in Computer-Aided Design. – Springer Berlin Heidelberg, 2004. – P. 382–398.

33. Clarke, E. Compositional Model Checking / E. M. Clarke, D. E. Long, K. L. McMillan // Proc. of the Fourth Annual Symposium on Logic in Computer Science. – IEEE, 1989. – P. 353–362.

34. Clarke, E. Environment Abstraction for Parameterized Verification / Edmund Clarke, Muralidhar Talupur, Helmut Veith // Verification, Model Checking, and Abstract Interpretation. – Springer Berlin Heidelberg, 2006. – P. 126–141.

35. Clarke, E. Model Checking / Edmund M. Clarke, Jr., Orna Grumberg, Doron A. Peled. – Cambridge, Massachusetts : The MIT Press, 1999. – 314 p.

36. Clarke, E. Proving Ptolemy Right: The Environment Abstraction Framework for Model Checking Concurrent Systems / Edmund Clarke, Murali Talupur, Helmut Veith // Tools and Algorithms for the Construction and Analysis of Systems. – Springer Berlin Heidelberg, 2008. – P. 33–47.

37. Compilers: Principles, Techniques, and Tools / Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. – 2nd ed. – Boston, MA : Addison-Wesley, 2008. – 1000 p.

38. Counterexample-Guided Abstraction Refinement / Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, Helmut Veith // Computer-Aided Verification. – Springer Berlin Heidelberg, 2000. – P. 154–169.

39. Das, S. Experience with Predicate Abstraction / Satyaki Das, David L. Dill, Seungjoon Park // Computer-Aided Verification. – Springer Berlin Heidelberg, 1999. – P. 160–171.

40. de Guzman, J. Debugging : Preliminary documentation for Spirit2 debugging support [Электронный ресурс] / Joel de Guzman. – Режим доступа: http://boost-spirit.com/home/articles/doc-addendum/debugging/.

41. de Guzman, J. Fastest numeric parsers in the world! [Электронный ресурс] / Joel de Guzman. – 2014. – Режим доступа: http://boost-spirit.com/home/2014/09/03/fastest-numeric-parsers-in-the-world/.

42. Decidability of Parameterized Verification / Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. – San Rafael : Morgan & Claypool, 2015. – 172 p.

43. Dill, D. A Retrospective on Murφ / David L. Dill // 25 Years of Model Checking. – Springer Berlin Heidelberg, 2008. – P. 77–88.

44. Dubois, M. Parallel Computer Organization and Design / Michel Dubois, Murali Annavaram, Per Stenstrom. – New York : Cambridge University Press, 2012. – 562 p.

45. Efficient Methods for Formally Verifying Safety Properties of Hierarchical Cache Coherence Protocols / Xiaofang Chen, Yu Yang, Ganesh Gopalakrishnan, Ching-Tsun Chou // Formal Methods in System Design. – 2010. – Vol. 36, Issue 1. – P. 37–64.

46. Emerson, E. A. Reducing Model Checking of the Many to the Few / E. Allen Emerson, Vineet Kahlon // Automated Deduction. – Springer Berlin Heidelberg, 2000. – P. 236–254.

47. Emerson, E. A. Exact and Efficient Verification of Parameterized Cache Coherence Protocols / E. Allen Emerson, Vineet Kahlon // Correct Hardware Design and Verification Methods. – Springer Berlin Heidelberg, 2003. – P. 247–262.

48. Emerson, E. A. Model Checking Large-Scale and Parameterized Resource Allocation Systems / E. Allen Emerson, Vineet Kahlon // Tools and Algorithms for the Construction and Analysis of Systems. – Springer Berlin Heidelberg, 2002. – P. 251–265.

49. Emerson, E. A. Reasoning about Rings / E. Allen Emerson, Kedar S. Namjoshi // Principles of Programming Languages. – ACM New York, 1995. – P. 85–94.

50. Ford, B. Parsing Expression Grammars: A Recognition-Based Syntactic Foundation / Bryan Ford // Principles of Programming Languages. – ACM New York, 2004. – P. 111–122.

51. Graf, S. Construction of Abstract State Graphs with PVS / Susanne Graf, Hassen Saidi // Computer-Aided Verification. – Springer Berlin Heidelberg, 1997. – P. 72–83.

52. Grinchtein, O. Inferring Network Invariants Automatically / Olga Grinchtein, Martin Leucker, Nir Piterman // Automated Reasoning. – Springer Berlin Heidelberg, 2006. – P. 483–497.

53. Hennessy, J. Computer Architecture: A Quantitative Approach / John L. Hennessy, David A. Patterson. – 5th ed. – Waltham, MA : Morgan Kaufmann, 2011. – 856 p.

54. Hierarchical Cache Coherence Protocol Verification One Level at a Time Through Assume Guarantee / Xiaofang Chen, Yu Yang, Michael Delisi, Ganesh Gopalakrishnan, Ching-Tsun Chou // Proc. of IEEE International High Level Design Validation and Test Workshop. – IEEE, 2007. – P. 107–114.

55. Holzmann, G. A Minimized Automaton Representation of Reachable States / Gerard J. Holzmann, Anuj Puri // International Journal on Software Tools for Technology Transfer. – 1999. – Vol. 2, Issue 3. – P. 270–278.

56. Holzmann, G. An Analysis of Bitstate Hashing / Gerard J. Holzmann // Formal Methods in System Design. – 1998. – Vol. 13, Issue 3. – P. 289–307.

57. Holzmann, G. An Improved Protocol Reachability Analysis Technique / Gerard J. Holzmann // Software – Practice and Experience. – 1988. – Vol. 18, Issue 2. – P. 137–161.

58. Holzmann, G. An Improvement in Formal Verification / Gerard J. Holzmann, Doron Peled // Formal Description Techniques VII. – Springer US, 1995. – P. 197–211.

59. Holzmann, G. Design and Validation of Computer Protocols / Gerard J. Holzmann. – Upper Saddle River, NJ : Prentice-Hall, 1991. – 512 p.

60. Holzmann, G. Model checking with bounded context switching / Gerard J. Holzmann, Mihai Florian // Formal Aspects of Computing. – 2011. – Vol. 23, Issue 23. – P. 365–389.

61. Holzmann, G. Multi-Core Model Checking with Spin / Gerard J. Holzmann, Dragan Bosnacki // Proc. of 21st IEEE International Parallel and Distributed Processing Symposium. – IEEE, 2007. – P. 1–8.

62. Holzmann, G. Parallelizing the Spin Model Checker / Gerard J. Holzmann // Model Checking Software. – Springer Berlin Heidelberg, 2012. – P. 155–171.

63. Holzmann, G. State Compression in Spin / Gerard J. Holzmann // Proc. of Third Spin Workshop. – 1997. – P. 1–10.

64. Holzmann, G. The Design of a Multicore Extension of the Spin Model Checker / Gerard J. Holzmann, Dragan Bosnacki // IEEE Transactions on Software Engineering. – 2007. – Vol. 33, Issue 10. – P. 659–674.

65. Holzmann, G. The Spin Model Checker: Primer and Reference Manual / Gerard J. Holzmann. – Boston, MA : Addison-Wesley, 2004. – 608 p.

66. Hudson, J. A Configurable Random Instruction Sequence (RIS) Tool for Memory Coherence in Multi-processor Systems / John Hudson, Gunaranjan Kurucheti // 15th International Workshop on Microprocessor Test and Verification. – IEEE, 2014. – P. 98–101.

67. Intel® 64 and IA-32 Architectures Software Developer’s Manual [Электронный ресурс]. – 2016. – Vol. 3: System Programming Guide. – 1998 p.

68. Invariants for Finite Instances and Beyond / Sylvain Conchon, Amit Goel, Sava Krstic, Alain Mebsout, Fatiha Zaidi // Formal Methods in Computer-Aided Design. – IEEE, 2013. – P. 61–68.

69. Ip, C. N. Better Verification through Symmetry / C. Norris Ip, David L. Dill // Formal Methods in System Design. – 1996. – Vol. 9, Issue 1. – P. 41–75.

70. Ip, C. N. Efficient Verification of Symmetric Concurrent Systems / C. Norris Ip, David L. Dill // Proc. of IEEE 1993 International Conference on Computer Design: VLSI in Computers and Processors. – IEEE, 1993. – P. 230–234.

71. Ip, C. N. Verifying Systems with Replicated Components in Murϕ / C. Norris Ip, David L. Dill // Formal Methods in System Design. – 1999. – Vol. 14, Issue 3. – P. 273–310.

72. Kesten, Y. Control and data abstraction: the cornerstones of practical formal verification / Yonit Kesten, Amir Pnueli // International Journal on Software Tools for Technology Transfer. – 2000. – Vol. 2, Issue 4. – P. 328–342.

73. Krstic, S. Parameterized System Verification with Guard Strengthening and Parameter Abstraction [Электронный ресурс] / Sava Krstic // Proc. of Workshop on Automated Verification of Infinite State Systems. – 2005. – Режим доступа: http://www.csee.ogi.edu/~krstics/psfiles/ParamSys.pdf.

74. Kurshan, R. A Structural Induction Theorem for Processes / R. P. Kurshan, K. McMillan // Proceedings of the eights annual ACM Symposium on Principles of distributed computing. – ACM New York, 1989. – P. 239–247.

75. Kyas, M. Verifying a Network Invariant for All Configurations of the Futurebus+ Cache Coherence Protocol / Marcel Kyas // Electronic Notes in Theoretical Computer Science. – 2001. – Vol. 50, Issue 4. – P. 357–370.

76. Lahiri, S. K. Constructing Quantified Invariants via Predicate Abstraction / Shuvendu K. Lahiri, Randal E. Bryant // Verification, Model Checking, and Abstract Interpretation. – Springer Berlin Heidelberg, 2004. – P. 267–281.

77. Lahiri, S. K. Indexed Predicate Discovery for Unbounded System Verification / Shuvendu K. Lahiri, Randal E. Bryant // Computer-Aided Verification. – Springer Berlin Heidelberg, 2004. – P. 135–147.

78. Lahiri, S. K. Predicate Abstraction with Indexed Predicates / Shuvendu K. Lahiri, Randal E. Bryant // ACM Transactions on Computational Logic. – 2007. – Vol. 9, Issue 1, Article 4.

79. Larrabee: A Many-Core x86 Architecture for Visual Computing / Larry Seiler, Doug Carmean, Eric Sprangle, Tom Forsyth, Pradeep Dubey, Stephen Junkins, Adam Lake, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan, Michael Abrash, Jeremy Sugerman, Pat Hanrahan // IEEE Micro. – 2009. – Vol. 29, Issue 1. – P. 10–21.

80. Lesens, D. Automatic verification of parameterized networks of processes / David Lesens, Nicolas Halbwachs, Pascal Raymond // Theoretical Computer Science. – 2001. – Vol. 256, Issues 1–2. – P. 113–144.

81. Lv, Yi. Computing Invariants for Parameter Abstraction / Yi Lv, Huimin Lin, Hong Pan // Formal Methods and Models for Codesign. – IEEE, 2007. – P. 29–38.

82. Manna, Z. The Temporal Logic of Reactive and Concurrent Systems: Specification / Zohar Manna, Amir Pnueli. – New York : Springer-Verlag, 1992. – 427 p.

83. Manna, Z. The Temporal Logic of Reactive and Concurrent Systems: Safety / Zohar Manna, Amir Pnueli. – New York : Springer-Verlag, 1995. – 512 p.

84. Matthews, O. Verifiable Hierarchical Protocols with Network Invariants on Parametric Systems / Opeoluwa Matthews, Jesse Bingham, Daniel J. Sorin // Proc. of the 16th Conference on Formal Methods in Computer-Aided Design. – 2016. – P. 101–108.

85. McMillan, K. L. Formal Verification of the Gigamax Cache Consistency Protocol / Kenneth McMillan, James Schwalbe // Shared Memory Multiprocessing. – Cambridge, Massachusetts : London, England : The MIT Press, 1992. – P. 111–134.

86. McMillan, K. L. Symbolic Model Checking: An Approach to the State Explosion Problem : Doctoral Dissertation / Kenneth Lauchlin McMillan. – Carnegie Mellon University, PA, USA, 1992. – 214 p.

87. McMillan, K. L. Verification of an Implementation of Tomasulo's Algorithm by Compositional Model Checking / K. L. McMillan // Computer-Aided Verification. – Springer Berlin Heidelberg, 1998. – P. 110–121.

88. McMillan, K. L. Verification of Infinite State Systems by Compositional Model Checking / K. L. McMillan // Correct Hardware Design and Verification Methods. – Springer Berlin Heidelberg, 1999. – P. 219–237.

89. McMillan, K. L. Verification of the FLASH Cache Coherence Protocol by Compositional Model Checking / K. L. McMillan // Correct Hardware Design and Verification Methods. – Springer Berlin Heidelberg, 2001. – P. 179–195.

90. Mukherjee, A. Learning Boost C++ Libraries / Arindam Mukherjee. – Birmingham : Packt Publishing, 2015. – 558 p.

91. NuSMV 2.5 Tutorial [Электронный ресурс] / Roberto Cavada, Alessandro Cimatti, Gavin Keighren, Emanuele Olivetti, Marco Pistore, Marco Roveri. – Режим доступа: http://nusmv.fbk.eu/NuSMV/tutorial/v25/tutorial.pdf.

92. NuSMV2: An OpenSource Tool for Symbolic Model Checking / Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, Armando Tacchella // Computer-Aided Verification. – Springer Berlin Heidelberg, 2002. – P. 359–364.

93. O'Leary, J. Protocol verification using flows: An industrial experience / John O'Leary, Murali Talupur, Mark R. Tuttle // Formal Methods in Computer-Aided Design. – IEEE, 2009. – P. 172–179.

94. Owre, S. PVS: A Prototype Verification System / S. Owre, J. M. Rushby, N. Shankar // Automated Deduction. – Springer Berlin Heidelberg, 1992. – P. 748–752.

95. Pandav, S. Counterexample Guided Invariant Discovery for Parameterized Cache Coherence Verification / Sudhindra Pandav, Konrad Slind, Ganesh Gopalakrishnan // Correct Hardware Design and Verification Methods. – Springer Berlin Heidelberg, 2005. – P. 317–331.

96. Parameterized Verification with Automatically Computed Inductive Assertions / Tamarah Arons, Amir Pnueli, Sitvanit Ruah, Ying Xu, Lenore Zuck // Computer-Aided Verification. – Springer Berlin Heidelberg, 2001. – P. 221–234.

97. ParaVerifier: An Automatic Framework for Proving Parameterized Cache Coherence Protocols / Yongjian Li, Jun Pang, Yi Lv, Dongrui Fan, Shen Cao, Kaiqiang Duan // Automated Technology for Verification and Analysis. – Springer International Publishing, 2015. – P. 207–213.

98. Park, S. Verification of Cache Coherence Protocols by Aggregation of Distributed Transactions / S. Park, D. L. Dill // Theory of Computing Systems. – 1998. – Vol. 31, Issue 4. – P. 355–376.

99. Pnueli, A. Automatic Deductive Verification with Invisible Invariants / Amir Pnueli, Sitvanit Ruah, Lenore Zuck // Tools and Algorithms for the Construction and Analysis of Systems. – Springer Berlin Heidelberg, 2001. – P. 82–97.

100. Pnueli, A. Liveness with (0, 1, ∞)-Counter Abstraction / Amir Pnueli, Jessie Xu, Lenore Zuck // Computer-Aided Verification. – Springer Berlin Heidelberg, 2002. – P. 107–122.

101. Pong, F. A New Approach for the Verification of Cache Coherence Protocols / Fong Pong, M. Dubois // IEEE Transactions on Parallel and Distributed Systems. – 1995. – Vol. 6, Issue 8. – P. 773–787.

102. Pong, F. Verification Techniques for Cache Coherence Protocols / Fong Pong, Michel Dubois // ACM Computing Surveys. – 1997. – Vol. 29, Issue 1. – P. 82–126.

103. Protocol Verification as a Hardware Design Aid / David L. Dill, Andreas J. Drexler, Alan J. Hu, C. Han Yang // Proc. of IEEE 1992 International Conference on Computer Design: VLSI in Computers and Processors. – IEEE, 1992. – P. 522–525.

104. PVCoherence: Designing Flat Coherence Protocols for Scalable Verification / Meng Zhang, Jesse D. Bingham, John Erickson, Daniel J. Sorin // Proc. of 2014 IEEE 20th International Symposium on High Performance Computer Architecture. – IEEE, 2014. – P. 392–403.

105. PVCoherence: Designing Flat Coherence Protocols for Scalable Verification / Meng Zhang, Jesse D. Bingham, John Erickson, Daniel J. Sorin // IEEE Micro. – 2015. – Vol. 35, Issue 3. – P. 84–91.

106. Reducing Verification Complexity of a Multicore Coherence Protocol Using Assume/Guarantee / Xiaofang Chen, Yu Yang, Ganesh Gopalakrishnan, Ching-Tsun Chou // Formal Methods in Computer-Aided Design. – IEEE, 2006. – P. 81–88.

107. Singh, P. Test Generation for CMP Designs / Padmaraj Singh, David L. Landis // Eleventh International Workshop on Microprocessor Test and Verification. – IEEE, 2010. – P. 67–70.

108. Sorin, D. A Primer on Memory Consistency and Cache Coherence / Daniel J. Sorin, Mark D. Hill, David A. Wood. – San Rafael : Morgan & Claypool, 2012. – 210 p.

109. Spin Version 6 – Promela Grammar [Электронный ресурс]. – Режим доступа: http://spinroot.com/spin/Man/grammar.html.

110. State space reduction in modeling checking parameterized cache coherence protocol by two-dimensional abstraction / Yang Guo, Wanxia Qu, Long Zhang, Weixia Xu // The Journal of Supercomputing. – 2012. – Vol. 62, Issue 2. – P. 828–854.

111. Stern, U. Automatic Verification of the SCI Cache Coherence Protocol / Ulrich Stern, David L. Dill // Correct Hardware Design and Verification Methods. – Springer Berlin Heidelberg, 1995. – P. 21–34.

112. Stern, U. Improved Probabilistic Verification by Hash Compaction / Ulrich Stern, David L. Dill // Correct Hardware Design and Verification Methods. – Springer Berlin Heidelberg, 1995. – P. 206–224.

113. Stroustrup, B. The C++ Programming Language / Bjarne Stroustrup. – 4th ed. – Boston, MA : Addison-Wesley, 2013. – 1368 p.

114. Talupur, M. Abstraction Techniques for Parameterized Verification : Doctoral Dissertation / Muralidhar Talupur. – Carnegie Mellon University, PA, USA, 2006. – 278 p.

115. Talupur, M. Going with the Flow: Parameterized Verification Using Message Flows / Murali Talupur, Mark R. Tuttle // Formal Methods in Computer-Aided Design. – IEEE, 2008. – P. 1–8.

116. The Boost C++ Libraries: Documentation [Электронный ресурс]. – Режим доступа: http://www.boost.org/doc.

117. The Stanford FLASH Multiprocessor / J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, J. Hennessy // Proc. of the 21st Annual International Symposium on Computer Architecture. – IEEE, 1994. – P. 302–313.

118. Verification by Network Decomposition / Edmund Clarke, Muralidhar Talupur, Tayssir Touili, Helmut Veith // Concurrency Theory. – Springer Berlin Heidelberg, 2004. – P. 276–291.

119. Verification of the Futurebus+ Cache Coherence Protocol / Edmund M. Clarke, Orna Grumberg, Hiromi Hiraishi, Somesh Jha, David E. Long, Kenneth L. McMillan, Linda A. Ness // Formal Methods in System Design. – 1995. – Vol. 6, Issue 2. – P. 217–232.

120. Verifying Distributed Directory-based Cache Coherence Protocols: S3.mp, a Case Study / Fong Pong, Andreas Nowatzyk, Gunes Aybay, Michel Dubois // EURO-PAR’95 Parallel Processing. – Springer Berlin Heidelberg, 1995. – P. 287–300.

121. Wolper, P. Reliable Hashing without Collision Detection / Pierre Wolper, Denis Leroy // Computer-Aided Verification. – Springer Berlin Heidelberg, 1993. – P. 59–70.

122. Wolper, P. Verifying Properties of Large Sets of Processes with Network Invariants / Pierre Wolper, Vinciane Lovinfosse // Automatic Verification Methods for Finite State Systems. – Springer Berlin Heidelberg, 1989. – P. 68–80.

Приложения

Акционерное общество «МЦСТ»
117105, Москва, ул. Нагатинская, д. 1, стр.23
телефон: +7 (495) 363 96 65, факс: +7 (495) 363 95 99
электронная почта: mcst@mcst.ru

«УТВЕРЖДАЮ»
Генеральный директор АО «МЦСТ»
А. К. Ким
«__» _____ 2016 г.

АКТ

о внедрении результатов диссертационной работы
Буренкова Владимира Сергеевича

Комиссия в составе председателя – к.т.н. Волконского В. Ю., членов комиссии – к.т.н. Мешкова А. Н., к.т.н. Груздова Ф. А. составила настоящий акт о том, что разработанные в диссертационной работе Буренкова В. С. методы и программные средства верификации протоколов когерентности кэш-памяти и их аппаратных реализаций использованы при выполнении опытно-конструкторских работ по темам «Экскурсовод-2», «Процессор-1» и «Процессор-9» в АО «МЦСТ».

Указанные методы позволили преодолеть проблему экспоненциального роста числа состояний протокола, возникающую при возрастании количества процессорных ядер, и провести верификацию протокола когерентности 16-ядерной системы, разрабатываемой в рамках работы «Экскурсовод-2», без ограничений на количество процессорных ядер и при незначительных затратах памяти и приемлемых временных затратах. При использовании разработанных Буренковым В. С. средств верификации аппаратных реализаций протоколов когерентности обнаружено более 50 ошибок реализации протоколов когерентности в 16- и 32-ядерных микропроцессорных системах, разрабатываемых в рамках работ по темам «Экскурсовод-2», «Процессор-1» и «Процессор-9», среди которых имеются ошибки, не найденные ранее с помощью других инструментов.

Зам. ген. директора АО «МЦСТ», к.т.н., В. Ю. Волконский
Начальник отдела АО «МЦСТ», к.т.н., А. Н. Мешков
Начальник отдела АО «МЦСТ», к.т.н., Ф. А. Груздов

Московский государственный технический университет имени Н. Э. Баумана

Студенческое научно-техническое общество имени Н. Е. Жуковского

ДИПЛОМ

I степени

награждается

Победитель

Всероссийского конкурса научно-исследовательских работ в области инженерных и гуманитарных наук

в номинации «Информационно-телекоммуникационные технологии и моделирование»

Буренков Владимир Сергеевич

аспирант кафедры «Компьютерные системы и сети»

с проектом «Методы и средства верификации протоколов когерентности памяти»

(Научный руководитель: Иванов С.Р., к.т.н., доцент).

Россия, Москва, МГТУ им. Н.Э. Баумана, сентябрь 2016 г.

Ректор А.А. Александров