УДК 004.052.4

Проблемы параметризованной верификации протоколов когерентности памяти

© В.С. Буренков, С.Р. Иванов

МГТУ им. Н.Э. Баумана, Москва, 105005, Россия

Проанализированы основные направления верификации протоколов когерентности и сопутствующие проблемы. Сформулирована проблема когерентности памяти, присущая мультипроцессорным системам, и кратко описаны аппаратные пути ее решения — протоколы когерентности. Рассмотрены методы верификации протоколов когерентности. Особое внимание уделено параметризованной верификации, призванной предоставить доказательство корректности протокола с любым числом агентов. Обозначены достоинства и недостатки существующих методов и определены направления дальнейшей работы по верификации протоколов, разработанных для микропроцессоров архитектуры «Эльбрус».

Ключевые слова: *протокол когерентности памяти, верификация протоколов, формальные методы, проверка модели, доказательство теорем.*

Введение. Протоколы когерентности памяти современных вычислительных систем отличаются высоким уровнем сложности и колоссальной размерностью пространства состояний. Это значительно уменьшает вероятность нахождения замысловатых ошибок в протоколах с помощью традиционных подходов к верификации, основанных на моделировании со случайными воздействиями.

Существующие подходы к верификации, как правило, либо неприменимы к протоколам промышленного масштаба, либо требуют огромного объема ручной работы. Особо можно выделить метод CMP и его модификации, авторы которых заявляют о возможности их работы с протоколами новых архитектур. Однако поскольку в литературе можно найти лишь краткое описание метода, это не позволяет применять его без дополнительных исследований.

1. Проблема когерентности и протоколы когерентности. Мультипроцессорная система с разделяемой памятью состоит из двух или более независимых процессоров, каждый из которых выполняет либо часть большой программы, либо независимую программу. Все процессоры обращаются к командам и данным, хранящимся в общей основной памяти. Для сокращения задержки на доступ к памяти каждый процессор снабжается локальной кэш-памятью. Высокие уровни производительности устройств с мультипроцессорной архитектурой и разделяемой памятью обусловили строгую тенденцию к использо-

ванию именно этих архитектур в современных промышленно выпускаемых многоядерных процессорах. Однако из-за оснащения каждого процессора (ядра) локальной кэш-памятью возникает следующая задача — требуется обеспечить согласованность (когерентность) блоков кэш-памяти различных процессоров.

В решении проблемы когерентности выделяются два подхода: программный и аппаратный. В силу большей производительности аппаратные механизмы, называемые *протоколами когерентности* кэш-памяти, нашли наиболее широкое применение.

Работу протокола когерентности можно кратко описать следующим образом. Адресное пространство разделено на области, каждая из которых принадлежит одному из процессоров, отвечающих за доступ к ней (home-процессор). Получив от ядра исходную команду считывания или записи в память, кэш-память отправляет соответствующий запрос в home-процессор. Home-процессор рассылает запросы, отслеживающие состояние кэшей других процессоров (снуп-запросы), и, возможно, запросы в память. В случае необходимости выдается ответ запросчику, производится запись или считывание данных. Таким образом, home-процессор выполняет роль координатора всех действий.

Постоянно растущая сложность мультипроцессорных систем с разделяемой памятью находит свое отражение и в замысловатости протоколов когерентности этих систем. Более того, для новых поколений микропроцессоров характерно увеличение количества вычислительных ядер в составе процессора. Это, в свою очередь, определяет сложность проверки корректности протоколов и порождает необходимость методов верификации протоколов когерентности систем с любым числом ядер (так называемой параметризованной верификации).

2. Верификация протоколов когерентности. При верификации протоколов когерентности важными являются два аспекта:

- правильность разработки самих протоколов;
- корректность аппаратной реализации этих протоколов, т. е. RTL-описания микропроцессора.

Прежде чем проверять аппаратную реализацию, желательно удостовериться в корректности самого протокола. Распространенная практика — анализ протокола вручную, а затем проверка реализации тестами со случайными воздействиями — не дает возможности говорить о каких-либо гарантиях корректности как протокола, так и системы. Случайная природа тестовых последовательностей не обеспечивает полное покрытие пространства состояний протокола, тем более за приемлемое время. Несмотря на то что данные методы позволяют найти большое количество ошибок, сложные ошибки, свя-

занные с передачей множества сообщений между частями верифицируемой системы, к тому же и в некотором необычном порядке, могут оказаться невыявленными. Формальные методы позволяют получить математическое доказательство корректности протокола (в пределах уровня детализации модели), а также могут содействовать получению тестов, подаваемых на вход RTL-модели верифицируемой микропроцессорной системы.

В ходе формальной верификации протокола проверяется соответствие абстрактной модели протокола (*конечного автомата*) его спецификации, т. е. набору свойств, которым должен отвечать протокол. Формальные методы нацелены на анализ всех достижимых состояний верифицируемой модели.

Простые методы верификации, основанные на алгоритмах полного перебора пространства глобальных состояний протокола (например, анализ достижимости состояний [1–3]), подвержены проблеме «взрыва числа состояний» и неприменимы для сложных систем даже с небольшим числом процессорных ядер, не говоря уже о параметризованных системах. «Взрыв числа состояний» – эффект экспоненциального роста числа состояний с ростом числа компонентов, работающих параллельно.

Существует два класса методов параметризованной верификации протоколов когерентности:

• основанные на *проверке моделей* (*model checking*) и нацеленные на максимальную автоматизацию;

• основанные на *доказательстве теорем* (*theorem proving*) и нацеленные на масштабируемость.

Методы, основанные на model checking. Model checking [4] — метод, в ходе применения которого систематически исследуется пространство состояний модели верифицируемого протокола с целью проверки выполнения свойства, описывающего желаемое поведение протокола. Рассматриваемые модели являются конечными, благодаря чему задача model checking алгоритмически разрешима, а соответствующие алгоритмы, как правило, эффективны. Свойства поведения специфицируются с помощью темпоральной логики, позволяющей отражать относительный порядок событий без явного указания значений времени.

К достоинствам model checking относятся:

• полная автоматизация метода;

• генерация контрпримеров, позволяющих отыскать источник ошибки;

• возможность предоставления частичных спецификаций.

Основным недостатком метода является «взрыв числа состояний».

Существуют инструментальные средства верификации (например, Spin, Murphi), предоставляющие пользователю язык описания моделей и возможность выражения условий корректности. По описанию модели инструменты автоматически строят систему переходов и проводят проверку выполнимости на ней заданной темпоральной формулы. Указанные средства применяются для верификации промышленных протоколов когерентности памяти, и хорошо себя зарекомендовали [5–8]. Однако при этом имеется ограничение на число узлов процессорной системы (процессорных ядер), отраженных в модели. В [5, 7] этот параметр ограничивается значением 3–4. Более того, современные протоколы когерентности настолько сложны, что зачастую невозможна даже верификация систем с тремя процессорами [9]. Таким образом, model checking в чистом виде не масштабируется и непригоден для параметризованной верификации.

В связи с этим применяются методы, позволяющие сократить количество состояний верифицируемой модели и в то же время сохранить интересующие нас свойства исходной модели. Наиболее общими являются методы абстракции (наряду с редукцией на основе симметрии и редукцией частичных порядков).

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. В итоге в случае аппроксимации снизу ошибка в абстрактной системе будет подразумевать ошибку в исходной системе, а в случае аппроксимации сверху корректность абстрактной системы подразумевает корректность исходной системы. В дальнейшем будут затрагиваться только аппроксимации сверху.

Связь модели с ее абстракцией выполняет отношение симуляции [4]. Поскольку абстракция позволяет скрыть некоторые особенности первоначальной модели, она может быть определена на более узком множестве атомарных высказываний. 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.

Построение абстрактных моделей требует нахождения компромисса между двумя конфликтующими целями:

• получение абстрактных моделей небольшого размера, которые могут быть верифицированы методом model checking;

• получение точных абстрактных моделей.

Чем меньше модель, тем больше поведений она допускает. Это может привести к появлению ложных контрпримеров, не обнаруживаемых в исходной модели. Выходов из этой ситуации по крайней

мере два: 1) построение точных абстрактных моделей; 2) анализ контрпримера на ложность и модификация абстрактной модели на основании полученной информации (*counterexample-guided abstraction refinement*).

Авторам неизвестно о существовании подхода, применимого к протоколам когерентности промышленного масштаба, позволяющего сразу строить точную абстракцию модели протокола. Например, в литературе отражены абстракция counter abstraction и ее обобщение environment abstraction [10–12], которые получены путем выделения набора идентичных процессов в классы эквивалентности на основе предикатов, которым они удовлетворяют, при этом только один представитель каждого класса принимается во внимание. Абстрактные модели, полученные с помощью этих методов, являются очень подробными и, следовательно, будут слишком большими в случае сложных протоколов. В источниках сообщается о применении этих абстракций к академическим протоколам. Роль пользователя в этих методах относительно небольшая. Так, в [11] предлагается подход, при котором пользователь только описывает протокол на предлагаемом входном языке. Далее специальное средство на основе environment abstraction извлекает из этого описания соответствующую абстрактную модель, представляет ее на языке SMV и подает на вход инструментального средства SMV, проверяющего выполнение свойств на абстрактной модели.

Существуют методы, нацеленные на нахождение такого числа K, что верификации системы с K процессами будет достаточно, чтобы гарантировать корректность параметризованной системы с N процессами для любого N. На основании работ в этом и смежном направлениях [13–16] можно сделать вывод, что получаемые значения K являются слишком большими. Это делает метод неприменимым на практике. Так, в [14] было найдено $K = 7$ для протокола когерентности, основанного на справочнике.

Методы, основанные на доказательстве теорем. Эффективность метода model checking ограничена используемыми для проведения верификации механизмами. Иногда сложность задачи верификации может быть понижена, если имеется более общее математическое описание верифицируемого объекта. Такое описание может быть получено в рамках дедуктивного метода — доказательства теорем.

В ходе доказательства теорем посредством математических рассуждений определяется, удовлетворяет ли спецификации данная реализация проекта. Реализация и спецификация должны быть представлены в виде формул некоторой формальной логики. Взаимосвязи между реализацией и спецификацией рассматриваются как теоремы в логике. Заключение о соответствии затем выносится путем доказательства теорем.

Средство автоматизированного доказательства теорем (theorem prover) — это программный инструмент, определяющий истинность формул в заданной логике. Популярными инструментами являются ACL2, Coq, HOL, Isabelle, NuPrl, PVS. Логики, на которых основаны эти средства, очень разнообразны, но общим аспектом является их выразительность. Однако выразительность логики влечет за собой ее неразрешимость, что означает невозможность построения такой автоматической процедуры (или алгоритма), которая, приняв на вход формулу, 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, очень сложно на основе этой информации найти ошибку в верифицируемой системе. Преимуществом дедуктивной верификации является возможность работы с системами с бесконечным числом состояний. В дальнейшем под доказательством теорем будем понимать любой метод, требующий от пользователя вспомогательных утверждений о верифицируемой системе.

В [17] Парк применил средство автоматизированного доказательства теорем общего назначения PVS [18] для параметризованной верификации протокола когерентности FLASH [19]. Данный процесс очень сложен, так как требует от пользователя индуктивных инвариантов. Маловероятно, что методы, основанные только на доказательстве теорем, могут найти применение при верификации протоколов современных мультипроцессоров.

В [20] изложен метод параметризованной верификации протоколов когерентности памяти, основанный на композиционной проверке моделей и реализованный в системе Cadence SMV. Задав свойство относительно некоторого кэша системы, изначально делают попытку проверить это свойство на грубой абстрактной модели, в которой отражен соответствующий процесс, а все остальные процессы заменены абстрактным процессом. Вероятной причиной полученного контрпримера является сообщение от абстрактного процесса. Чтобы избавиться от контрпримера, предлагается две стратегии: добавить отправителя сообщения в абстракцию 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 следующем: отсутствует необходимость 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.

Метод CMP. В [5] предлагается метод, базирующийся на комбинации model checking и доказательства теорем, в частности, основанный на идеях из [20]. Автор работы [21] формализует описание этого метода, а также доказывает его корректность.

Метод [5], получивший название CMP (по фамилиям авторов — Chou, Mannava, Park), принимает на вход описание симметричного протокола для N процессоров, отношение переходов которого задано с помощью набора правил (rules). Правило является защищенной командой (guarded command), условие которой (защита) определяет, будут ли выполняться действия, отраженные в команде.

Метод CMP основан на композиционных рассуждениях и состоит из двух основных шагов: получения абстрактной модели (abstraction) и ее уточнения (strengthening), которые итеративно применяются к описанию протокола. Пусть задано свойство относительно двух процессов i и j, которое должно выполняться на модели. Метод строит абстрактную модель, в которой сохраняется описание двух процессов (например, 1 и 2), а описание остальных процессов заменяется недетерминированным процессом Other. Такая замена правомерна, поскольку в симметричной системе истинность свойства для процессов 1, 2 подразумевает его истинность и для любой другой пары процессов.

Процедура абстракции может быть следующей. Каждое условие в коде модели, затрагивающее процессоры 3, ..., N, заменяется на true, false или недетерминированное булево значение. Каждое присваивание переменных, соответствующих процессорам 3, ..., N, удаляется. Абстракция свойства осуществляется похожим образом; любое условие, в которое входят процессоры 3, ..., N, заменяется на true, false или недетерминированное булево значение.

Получаемая абстрактная модель является очень грубой из-за отсутствия ограничений на поведение процесса Other, и проверка выполнения свойств на этой модели будет сопровождаться ложными контрпримерами. Для устранения таких контрпримеров метод требует от пользователя введения лемм (non-interference lemmas), или инвариантов. Инварианты представляют собой выражения, сформулированные в терминах переменных модели протокола. По результатам анализа трассы контрпримера, предоставленной инструментальным средством проверки моделей, определяется правило абстрактной модели, вызвавшее ложный переход. Условия, отраженные в леммах, добавляются в защиту найденного правила, тем самым ограничивая поведение процесса Other. Данный процесс повторяется итеративно, пока не

будет найден контрпример, проявляющийся и в исходном протоколе, либо не будет доказана истинность проверяемого свойства.

В [22] доказывается, что метод CMP корректен для любого симметричного протокола и любой консервативной процедуры абстракции.

Основная сложность метода заключается в нахождении лемм. В [5] отражен опыт применения метода к протоколам FLASH и German. В [23] сообщается, что в Intel метод был применен к протоколу когерентности, сложность которого на несколько порядков выше, чем у протокола FLASH. В протоколе FLASH возможны 16 различных типов сообщений, а в протоколе Intel — 54. Верификация заняла около месяца и потребовала 25 разработанных вручную лемм. Добавление лемм — трудоемкий процесс, занимающий много времени и требующий глубокого понимания протокола. На этом этапе в Intel были автоматизированы такие процессы, как составление исходной абстракции и уточнение абстракции на основе предоставленных лемм.

В [22] описан подход на основе потоков сообщений (message flows), призванный сократить объем ручной работы при поиске инвариантов. Потоки сообщений — это последовательности сообщений, пересылаемые между процессорами во время работы протокола. Введенные в [22] потоки отражают линейный порядок событий, в которых участвуют два агента. В [9] сообщается, что такое определение потоков, в частности принятие во внимание двух агентов, недостаточно для случая реальных протоколов когерентности, поскольку в модели протокола, помимо кэшей и home-процессора, должен быть отражен и контроллер памяти. Более того, некоторому событию могут предшествовать несколько других событий, что не может быть зафиксировано линейными потоками, поэтому в [9] вводятся потоки в виде ориентированных ациклических графов. Авторы предлагают язык, позволяющий специфицировать события наряду с участниками.

Отслеживание активных потоков может привести к порождению полезных лемм. Модель протокола должна быть написана так, чтобы каждое правило соответствовало определенному событию, отраженному в потоках. Тогда в модель могут быть добавлены вспомогательные переменные, позволяющие следить за состояниями потоков.

На основе значений этих переменных могут быть получены два класса лемм: ограничения предшествования, устанавливающие порядок срабатываний правил, и ограничения по конфликтам, запрещающие одним потокам переходить в активное состояние, пока не завершатся другие. Заметим, что возможны и другие виды лемм, причем не все они одинаково полезны. В связи с этим необходимо отыскивать такие классы, которые позволяют эффективно проводить верификацию.

В [9] сообщается о параметризованной верификации протокола LCP (Larrabee coherence protocol), разработанного в Intel и имеющего около 50 различных типов сообщений. На модели, описанной на языке Murphi, проверялось выполнение свойства безопасности (если в системе есть кэш с эксклюзивным доступом к данным, то никакой другой кэш не имеет доступа к этим данным) и свойств, определяющих согласованность между списком кэшей с доступом к данным, который хранится в справочнике home-процессора, и реальным набором кэшей с таким доступом.

Авторами [9] был разработан инструмент, принимающий на вход параметризованное описание протокола на Murphi и описание потоков в отдельном файле. Инструмент строит промежуточное представление Murphi-описания в виде абстрактного синтаксического дерева. Операции абстракции и уточнения далее выполняются над этим представлением. Инструмент автоматически строит леммы на основе потоков и добавляет их к защитам подходящих правил (правило, чья защита находится в инварианте потока, модифицируется этим инвариантом). Авторы отмечают использование 15 потоков, полученных из документов на протокол и породивших 36 лемм, 25 из которых разработаны на основе ограничений предшествования, а остальные — на основе ограничений по конфликтам. Еще 5 лемм пришлось добавить вручную. Таким образом, предложенный метод позволил намного сократить объем ручной работы, выполненной при верификации похожего протокола [23].

В [24] предлагается подход к полной автоматизации процесса получения лемм, основанный на бинарных решающих диаграммах. Однако отсутствие масштабируемости этого подхода является камнем преткновения.

Заключение. Достоинства метода model checking — полная автоматизация и порождение контрпримеров, позволяющих отыскивать источники ошибок, — делают его наиболее удобным формальным методом верификации протоколов когерентности памяти. Однако метод подвержен проблеме комбинаторного «взрыва числа состояний», а потому может быть применен лишь к системам с тремя-четырьмя процессорными ядрами.

Ключ к решению проблемы видится в комбинации методов, основанных на model checking, с методами, основанными на доказательстве теорем.

Наиболее интересным из упомянутых в литературе представляется метод CMP, дополненный средствами автоматизации получения необходимых для его работы лемм. Предполагается применить данный метод к протоколам микропроцессоров архитектуры «Эльбрус», предварительно разрешив ряд проблем. В литературе нет детального

описания метода CMP, и средства, реализующие этот метод, не находятся в открытом доступе. В оригинальных статьях [5, 22] упоминается использование языка Murphi для описания моделей. Авторы настоящей статьи при верификации протоколов используют постоянно развивающуюся (в отличие от Murphi) систему Spin, предоставляющую пользователю язык Promela, который нацелен на моделирование протоколов. Поэтому предполагается адаптация метода к Promela, для этого, в частности, требуется разработать средство построения внутреннего представления моделей, пригодного для абстрагирования исходной модели и уточнения абстрактной модели. Также необходимо спроектировать способы представления потоков и механизмы работы с ними.

ЛИТЕРАТУРА

[1] Pong F., Dubois M. A New Approach for the Verification of Cache Coherence Protocols. *IEEE Transactions on Parallel and Distributed Systems*, 1995, vol. 6, no. 8, pp. 773–787.

[2] Буренков В.С. Метод перебора состояний для верификации протоколов когерентности памяти. *Тр. 54-й науч. конф. МФТИ*, 2011, с. 22, 23.

[3] Pong F., Dubois M. Verification Techniques for Cache Coherence Protocols. *ACM Computing Surveys*, 1997, vol. 29, no. 1, pp. 82–126.

[4] Clarke E., Grumberg O., Peled D. *Model Checking*. MIT Press, 1999, 314 p.

[5] Chou C., Mannava P., Park S. A Simple Method for Parameterized Verification of Cache Coherence Protocols. *Formal Methods in Computer-Aided Design*. 2004, vol. 3312, pp. 382–398.

[6] Harrison J. Formal Methods at Intel — An Overview. *Second NASA Formal Methods Symposium*, 2010. URL: http://www.cl.cam.ac.uk/~jrh13/slides/nasa-14apr10/slides.pdf (дата обращения 09.11.2013).

[7] Буренков В.С. Инструмент верификации протокола когерентности памяти. *Молодежный научно-технический вестник*, 2013, № 1. URL: http://sntbul.bmstu.ru (дата обращения 09.11.2013).

[8] Буренков В.С. Анализ применимости инструмента Spin к верификации протоколов когерентности памяти. *Вопросы радиоэлектроники*, 2013, вып. 3, с. 126–134.

[9] O'Leary J., Talupur M., Tuttle M. Protocol Verification Using Flows: An Industrial Experience. *Formal Methods in Computer-Aided Design*, 2009, pp. 172–179.

[10] Clarke E., Talupur M., Veith H. Environment Abstraction for Parameterized Verification. *Verification, Model Checking, and Abstract Interpretation*, 2006, vol. 3855, pp. 126–141.

[11] Talupur M. *Abstraction Techniques for Parameterized Verification,* Thesis … PhD, 2006. URL: http://reports-archive.adm.cs.cmu.edu/anon/2006/CMU-CS-06-169.pdf (дата обращения 09.11.2013).

[12] Clarke E., Talupur M., Veith H. Proving Ptolemy Right: The Environment Abstraction Framework for Model Checking Concurrent Systems. *Proceedings of the Theory and practice of software, 14th international conference on Tools and algorithms for the construction and analysis of systems*, 2008, pp. 33–47.

[13] Emerson A., Kahlon V. Reducing Model Checking of the Many to the Few. *Automated Deduction*, 2000, vol. 1831, pp. 236–254.

[14] Emerson A., Kahlon V. Model Checking Large-Scale and Parameterized Resource Allocation Systems. *Tools and Algorithms for the Construction and Analysis of Systems*, 2002, vol. 2280, pp. 251–265.

[15] Emerson A., Namjoshi K. Reasoning about Rings. *Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 1995, pp. 85–94.
[16] Clarke E., Talupur M., Touili T., Veith H. Verification by Network Decomposition. *CONCUR*, 2004, vol. 3170, pp. 276–291.
[17] Park S., Dill D. Verification of FLASH Cache Coherence Protocol by Aggregation of Distributed Transactions. *Proceedings of the 8th annual ACM symposium on parallel algorithms and architectures,* 1996, pp. 288–296.
[18] Owre S., Rushby J., Shankar N. PVS: A prototype verification system. *Automated Deduction*, 1992, vol. 607, pp. 748–752.
[19] Kuskin J., Ofelt D., Heinrich M., Heinlein J., Simoni R., Gharachorloo K., Chapin J., Nakahira D., Baxter J., Horowitz M., Gupta A., Rosenblum M., Hennessy J. The Stanford FLASH multiprocessor. *ISCA Proceedings of the 21st annual international symposium on Computer architecture*, 1994, pp. 302–313.
[20] McMillan K. Parameterized Verification of the FLASH Cache Coherence Protocol by Compositional Model Checking. *Proceedings of the 11th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, 2001, pp. 179–195.
[21] Krstic S. *Parameterized System Verification with Guard Strengthening and Parameter Abstraction. Automated Verification of Infinite State Systems,* 2005.
[22] Talupur M., Tuttle M. Going with the Flow: Parameterized Verification Using Message Flows. *Formal Methods in Computer-Aided Design*, 2008, pp. 1–8.
[23] Talupur M., Krstic S., O'Leary J., Tuttle M.R. Parametric Verification of Industrial Strength Cache Coherence Protocols. *In Proc. Workshop on Design of Correct Circuits (DCC),* 2008.
[24] Bingham J. Automatic Non-interference Lemmas for Parameterized Model Checking. *Formal Methods in Computer-Aided Design*, 2008, pp. 77–85.

Статья поступила в редакцию 28.06.2013

Ссылку на эту статью просим оформлять следующим образом:

Буренков В.С., Иванов С.Р. Проблемы параметризованной верификации протоколов когерентности памяти. *Инженерный журнал: наука и инновации,* 2013, вып. 11. URL: http://engjournal.ru/catalog/it/hidden/1013.html

Буренков Владимир Сергеевич родился в 1989 г., окончил МГТУ им. Н.Э. Баумана в 2012 г. Аспирант МГТУ им. Н.Э. Баумана. Автор более семи научных работ. Области научных интересов: формальная верификация вычислительных систем, алгоритмы и структуры данных, архитектура микропроцессорных систем. e-mail: VanSBuren@mail.ru

Иванов Сергей Ростиславович родился в 1937 г. Окончил МВТУ им. Н.Э. Баумана в 1960 г. Канд. техн. наук, доцент кафедры «Компьютерные системы и сети» МГТУ им. Н.Э. Баумана. Автор более 60 научных работ. Область научных интересов: системы автоматизированного проектирования электронных схем. e-mail: ivanovsr@bmstu.ru