УДК 004.052.42 DOI: 10.18698/0236-3933-2017-1-49-66

МЕТОД ПОСТРОЕНИЯ АБСТРАКТНЫХ МОДЕЛЕЙ, ИСПОЛЬЗУЕМЫХ ДЛЯ ВЕРИФИКАЦИИ ПРОТОКОЛОВ КОГЕРЕНТНОСТИ КЭШ-ПАМЯТИ МАСШТАБИРУЕМЫХ СИСТЕМ

В.С. Буренков vansburen@mail.ru
С.Р. Иванов ivanovsr@bmstu.ru

МГТУ им. Н.Э. Баумана, Москва, Российская Федерация

Аннотация

Изложен новый подход к решению задачи верификации протоколов когерентности кэш-памяти масштабируемых микропроцессорных систем. Рассмотрена математическая модель протоколов когерентности. Модель представляет собой отдельные группы устройств, работающих в соответствии с протоколом, в виде графов программ, а протокол в целом — в виде канальной системы, из которой может быть получена и исследована методом *Model checking* система переходов. Предложен подход к описанию моделей протоколов когерентности на языке *Promela* и сформулированы ограничения на модели. Представлен метод построения абстрактных моделей, позволяющий существенно уменьшить их размер и базирующийся на преобразованиях отдельных графов программ исходной модели, или на синтаксических преобразованиях *Promela*-процессов. Приведено математическое доказательство сохранения преобразованиями свойств-инвариантов. Результаты основаны на практике верификации протокола сложной системы на кристалле с архитектурой «Эльбрус»

Ключевые слова

Формальный метод, проверка моделей, преобразование моделей, протокол когерентности кэш-памяти

Поступила в редакцию 19.04.2016

Введение. Проблема верификации протоколов когерентности кэш-памяти является сложной и чрезвычайно актуальной в контексте проектирования многоядерных процессоров и многопроцессорных вычислительных систем с разделяемой памятью.

Не существует единого подхода к решению проблемы, способного адаптироваться к увеличению числа ядер в современных микропроцессорах. В настоящей работе развиваются приведенные в статье [1] направления по разработке формальных методов верификации протоколов когерентности кэш-памяти масштабируемых систем. В основе предлагаемого метода лежит метод, основанный на синтаксической абстракции и способный к масштабированию [2–4]. Однако приведенное описание недостаточно полное, и отсутствует определение ограничений на модели протоколов, что делает невозможным непосредственное применение метода.

Решаемую в этой работе задачу можно сформулировать следующим образом. Дано неформальное описание протокола когерентности наподобие *MOSI* [5], обеспечивающего согласованность данных в n блоках кэш-памяти, каждый из которых принадлежит отдельному процессорному ядру. Сформулирована спецификация протокола когерентности в виде множества свойств-инвариантов, запрещающих определенные комбинации состояний кэш-строки в кэшах системы. Необходимо разработать метод, проверяющий соответствие протокола данному множеству свойств при любом числе $n \in \mathbb{N}$ кэшей.

Синтез математической модели для представления протоколов когерентности. Проведем синтез математической модели.

Модель протоколов когерентности. Язык описания моделей *Promela* предлагает абстракции, представляющие группы устройств, работающих в соответствии с протоколом когерентности (*процессы*, являющиеся конечными автоматами), и их асинхронное взаимодействие (*каналы*, являющиеся *FIFO*-очередями). В связи с этим математическая модель протоколов когерентности, используемая в настоящей работе, основана на формальной семантике *Promela*-моделей и базируется на модели, применяемой в работе [6].

Модель состоит из нескольких уровней. *Promela*-процессам соответствуют графы программ. Асинхронная композиция графов программ описывается канальной системой. Путем «развертывания» канальной системы получается система переходов — стандартная модель аппаратных и программных систем.

Система переходов. Системой переходов TS называется шестерка $TS = (S, Act, \to, I, AP, L)$, где S — множество состояний; Act — множество действий; $\to \subseteq S \times Act \times S$ — отношение переходов; $I \subseteq S$ — множество начальных состояний; AP — множество атомарных высказываний; $L : S \to 2^{AP}$ — функция пометок. Для краткости под обозначением $s \xrightarrow{\alpha} s'$ будем понимать $(s, \alpha, s') \in \to$. Если осуществляемое действие в таком контексте неважно, запишем $s \to s'$.

Граф программы. Поведение процессов описывается с помощью операторов языка *Promela*. Множество всех переменных модели будем обозначать через Var, множество каналов модели — $Chan$.

Обозначим через $dom(x)$ тип переменной x, а через $dom(c)$ — тип переменных, которые могут быть переданы через канал c. Емкость канала c — максимальное число сообщений, которое он способен хранить, — обозначим через $cap(c)$.

Интерпретацией η переменных называется отображение, ставящее в соответствие каждой переменной $v \in Var$ значение $\eta(v) \in dom(v)$. Через $\eta[v := r]$ обозначим интерпретацию переменных, присваивающую значение r переменной v и оставляющую все остальные переменные неизменными. Обозначим через $Eval(Var)$ множество интерпретаций переменных.

Интерпретация ξ каналов — 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 $\xi(c) = v_1 v_2 \ldots v_k$, 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 $\xi[c := v_1 \ldots v_k]$ интерпретацию каналов, присваивающую последовательность $v_1 \ldots v_k$ каналу c и оставляющую все остальные каналы неизменными. Интерпретация ξ_0 отображает канал в пустую последовательность ε, т. е. $\forall c \in Chan : \xi_0(c) = \varepsilon$, $len(\varepsilon) = 0$. Обозначим через $Eval(Chan)$ множество всех интерпретаций каналов.

Обозначим через $Cond(Var)$ множество логических выражений относительно переменных $v \in Var$ [6], 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*() [7] и их объединения с помощью операторов языка *Promela*.

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

Множество действий будем рассматривать как объединение $Act \bigcup Comm$, элементы которого определяются только шестью базовыми операторами языка *Promela*: 1) присваивание; 2) оператор *assert*; 3) оператор *print*; 4) выражение; 5) отправка сообщения в канал; 6) извлечение сообщения из канала. Действия, выраженные первыми четырьмя операторами, составляют множество Act. Действия, выраженные последними двумя операторами, являются коммуникационными действиями:

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

Графом программы (*program graph*) PG *над* $(Var, Chan)$ называется шестерка

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

где Loc — множество состояний (вершин) графа; $Effect : Act \times Eval(Var) \to Eval(Var)$ — функция, определяющая результат действий; $\Rightarrow \in Loc \times Cond(Var, Chan) \times (Act \bigcup Comm) \times Loc$ — отношение переходов; $Loc_0 \subseteq Loc$ — множество начальных состояний; $g_0 \in Cond(Var, Chan)$ — начальное условие.

Переход $(l, g, act, l') \in \Rightarrow$ будем сокращенно обозначать $l \overset{g:act}{\Rightarrow} l'$. Логическое условие g называется защитой перехода $l \overset{g:act}{\Rightarrow} l'$. Действие act может быть осуществлено только в том случае, если защита g истинна, а действие выполнимо.

Далее будем подразумевать соответствие графов программы *Promela*-процессам.

Канальная система. Такая система CS над $(Var,Chan)$ состоит из графов программы PG_i над $(Var_i,Chan)$, где $1 \le i \le n$ и $Var = \bigcup_{1 \le i \le n} Var_i$. Обозначение $CS = [PG_1 | \ldots | PG_n]$.

В языке *Promela* возможны два типа взаимодействия между процессами с помощью каналов: 1) синхронная передача сообщения через канал нулевой емкости; 2) асинхронная передача сообщения через канал ненулевой емкости. При разработке моделей протоколов когерентности нет необходимости использовать синхронную передачу сообщений, поэтому далее такой тип взаимодействия рассматриваться не будет.

Семантика канальной системы формализуется посредством системы переходов. Пусть $CS = [PG_1 | \ldots | PG_n]$ — 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 $CS = [PG_1 | \ldots | PG_n]$ — канальная система над $(Var,Chan)$, и

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

Системой переходов $TS(CS)$, соответствующей данной канальной системе, называется шестерка $TS(CS) = (S, Act, \to, I, AP, L)$, где

$$S = (Loc_1 \times \ldots \times Loc_n) \times Eval(Var) \times Eval(Chan);$$

$Act = \bigcup_{0<i\le n} Act_i \cup \{\tau\}$, τ — специальный символ, представляющий все коммуникационные действия, при которых происходит обмен данными; переход $\to$ определяется правилами, представленными ниже:

- $I = \{l_1,\ldots,l_n,\eta,\xi_0 \mid \forall 0 < i \le n : (l_i \in Loc_{0,i} \wedge (\eta,\xi_0) \models g_{0,i})\}$;
- $AP = \bigcup_{0<i\le n} Loc_i \cup Cond(Var,Chan)$;
- $L(\langle l_1,\ldots,l_n,\eta,\xi\rangle) = \{l_1,\ldots,l_n\} \cup \{g \in Cond(Var,Chan) \mid (\eta,\xi) \models g\}$.

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

Перечислим правила, определяющие отношение переходов $\to$ системы $TS(CS)$.

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

$$\frac{l_i \overset{g:\alpha}{\Rightarrow} l_i' \wedge (\eta,\xi) \models g}{\langle l_1,\ldots,l_i,\ldots,l_n,\eta,\xi\rangle \overset{\alpha}{\to} \langle l_1,\ldots,l_i',\ldots,l_n,\eta',\xi\rangle}, \qquad (1)$$

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:

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

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

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

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

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

Разработка моделей протоколов когерентности на языке *Promela* и определение ограничений для них. Вопрос построения формальных моделей протоколов когерентности изучен в литературе недостаточно. Здесь использована математическая модель протоколов когерентности в виде канальной системы, и на основе практики верификации системы «Эльбрус-4С» определена структура отдельных графов программ и изложены ограничения на используемые операторы, выполнение которых необходимо для работы предлагаемого метода.

Рассмотрены протоколы когерентности, в которых исполнение запросов происходит под управлением координатора. В микропроцессорах с архитектурой «Эльбрус» координатором является системный коммутатор процессора, к памяти которого происходит обращение (*home*-процессора).

В качестве математической модели протокола когерентности будем использовать канальную систему $CS = [PG_0 \,|PG_1|\dots|PG_n]$, где PG_0 — граф программы, соответствующий системному коммутатору *home*-процессора; $PG_1,\dots,PG_n$ — идентичные графы программы, соответствующие контроллерам кэш-строки, находящимся в кэшах верифицируемой системы. Каждый граф программы определен над парой $(Var_i, Chan_i)$ так, что множества Var_i и $Chan_i$ могут пересекаться. При этом $Var = \bigcup_{0\le i\le n} Var_i$, $Chan = \bigcup_{0\le i\le n} Chan_i$.

Среди переменных из Var выделим множество подмножеств $\{Vstate_i \subseteq Var \,|\, i = 0,\dots,n\}$ переменных, описывающих состояние процессов $PG_i, 0 \le i \le n$. При этом $\bigcap_{0\le i\le n} Vstate_i = \varnothing$. Примерами переменных, входящих в это подмножество, являются локальные переменные процесса PG_i; переменная, хранящая состояние кэш-строки в таком кэш-контроллере; переменная, хранящая признак получения ответа на снуп-запрос от процесса PG_i.

Определим ограничения на структуру графов $PG_i, 0 \le i \le n$.

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 коге-

рентных 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. В коде такие операторы располагаются друг за другом.

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

– логическими утверждениями относительно переменных $v \in Var \setminus \bigcup_{0 \le i \le n} \times$ $\times Vstate_i$, причем значение этих переменных получается либо непосредственно путем исполнения операции извлечения из канала, либо в ходе анализа извлеченного со-общения;

– логическими утверждениями относительно переменных $v \in Vstate_i$, $1 \le i \le n$, значение которых устанавливается в процессе анализа принятых графом сообщений.

Работа кэш-контроллеров осуществляется по-другому. С одной стороны, можно выделить ряд этапов, например, отправка исходного запроса, смена состояния на переходное, прием снуп-запросов и других служебных запросов, прием когерентных ответов, смена переходного состояния 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. Например, если действие команды включает в себя получение сообщения по каналу c, и это действие не должно быть заблокировано, то в защиту с помощью конъюнкции добавляется условие $nempty(c)$. Также любая защита может являться тавтологией.

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

Множество каналов модели представлено тремя подмножествами.

1. Множество каналов C_1 емкостью n, в которые могут отправлять сообщения графы PG_i, $1 \le i \le n$. Извлекать сообщения может только один процесс PG_i, $0 \le i \le n$, на определенном этапе выполнения запроса. 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.

2. Множество каналов $C_2 = \{c_{2,1}, \ldots, c_{2,n}\}$ емкостью $m \in \mathbb{N}$, таких, что из канала $c_{2,i}$, $1 \le i \le n$, может извлекать сообщения только PG_i, а отправлять сообщения по этому каналу может только PG_0. Примером канала из этого множества является канал, по которому процесс, представляющий системный коммутатор *home*-процессора, передает когерентный запрос процессу, представляющему кэш-контроллер.

3. Множество каналов C_3, 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.

Сообщения, передаваемые по каналам, являются парами $m = \langle opc, id \rangle$, где $opc \in \mathbb{N}$ не является номером процесса; $id \in \{0, \ldots, n\}$ — номер процесса (например, идентификатор отправителя сообщения). Для обращения к первому и второму элементам пары m запишем $m.opc$ и $m.id$.

Синтез совокупности преобразований, приводящих к получению абстрактной модели. Проведем синтез совокупности преобразований.

Абстрактная модель протоколов когерентности. Проверяемые свойства протокола затрагивают не более двух кэшей. Поскольку все кэш-контроллеры идентичны и взаимозаменяемы, не имеет значения, какие именно два индекса графов $PG_1, \ldots, PG_n$ рассматривать. Поэтому без потери общности будем полагать, что рассматриваются графы PG_1, PG_2, и все проверяемые свойства сформулированы только относительно части системы, определяемой этими графами.

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

Предполагается, что в исходной системе среди переменных $v \in Vstate_i$, $0 \le i \le n$, хранящих информацию о других процессах, могут быть только переменные, хранящие номер процесса, т. е. переменные, областью определения которых является множество $\{0, \ldots, n\}$. В случае, когда полностью рассматривается только состояние процессов $PG_i, 0 \le i \le 2$, важно сохранять точные значения лишь тогда, когда это значение принадлежит множеству $\{0, 1, 2\}$; все остальные значения неразличимы и их можно представить некоторым абстрактным значением, не находящимся в этом множестве. В связи с этим применим абстракцию типов данных. Множества значений переменных $v \in Var$ и каналов $c \in Chan$ таких, что $dom(v) = \{0, \ldots, n\}$ и $dom(c.id) = \{0, \ldots, n\}$, заменяются $dom_{abs}(v) = \{0, 1, 2, ABS\}$ и $dom_{abs}(c.id) = \{0, 1, 2, ABS\}$, 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 способом систему *CS*.

В соответствии с предлагаемым методом после проведения абстракции типов данных осуществляется замена исходной модели $CS = [PG_0 \,|PG_1|\ldots|PG_n]$ моделью $CS_{abs} = [PG_0|PG_1|PG_2|PG_3]$, в которой изначально графы $PG_i, 0 \le i \le 3$, являются графами $PG_i, 0 \le i \le 3$, исходной системы CS.

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

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

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

coh_answers_chan ? message;

// ack _ list[message.id] $\in \bigcup_{0\le i\le 2} Vstate_i$, если message.id $= 1, 2$

ack_list[message.id] = false;

В ходе исполнения исходного запроса релевантны значения таких переменных *только одного процесса*. Введем такое множество переменных V_{loca}, что его элементы всегда находятся во взаимно-однозначном соответствии с рассмотренными локальными переменными того из процессов, который проводит модификацию переменных из $\bigcup_{0\le i\le 2} Vstate_i$: $V_{loca} \leftrightarrow Vstate_i$ для некоторого $2 < i \le n$.

Следовательно, в качестве пометок состояний абстрактной (TS_{abs}) и исходной (TS) систем будем использовать множество $AP = Cond(V_{AP})$, где $V_{AP} = \bigcup_{0\le i\le 2} Vstate_i \cup V_{loca}$.

Последующая модификация графов программ PG_i, $0 \le i \le 3$, осуществляется путем синтаксических преобразований, описанных ниже.

Абстрактные преобразования элементов множеств Act_i. Множество выражений ограничим множеством $Cond(Var, Chan)$. Преобразования присваиваний приведены ниже (символ $\varnothing$ означает отсутствие действия).

1. Оператор в исходной модели $v = val\,(v \in Var, val \in dom(v))$.

2. Оператор в абстрактной модели $v = val$, если $v \in \bigcup_{i=0,1,2} Vstate_i$; $\varnothing$, если $v \in \bigcup_{2<i\le n} Vstate_i$.

Других операторов присваивания нет.

Преобразование выражений $(v, c) \in Cond(V, C)$ осуществляется следующим образом. Если $V = \bigcup_{i=0,1,2} Vstate_i \cup V_{loc}$, то v остается неизменным, иначе v заме-

няется истиной. Если $C = C_1 \bigcup \{c_{2,1}, c_{2,2}\} \bigcup C_3$, то c остается неизменным, иначе c заменяется истиной.

Абстрактные преобразования элементов множеств Comm$_i$. Преобразования коммуникационных действий можно проводить различными способами. Так, 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 число небольшими значениями (3–4), что приводит к удалению некоторых операторов, изменяющих состояние переменных из множества 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 таблице.

Преобразование коммуникационных действий

Оператор в исходной модели	Оператор в абстрактной модели
	Для $c \in C_1$
$c!v\,(v \in dom(c))$	$c!v$, если оператор находится в графах PG_1, PG_2; $\varnothing$, если оператор находится в графе PG_3
$c?x\,(x \in Var, dom(x) \supseteq dom(c))$	Недетерминированный выбор
	Для $c \in C_2$
$c!v\,(v \in dom(c))$	$c!v$, если $c = c_{2,1}$ или $c = c_{2,2}$; $\varnothing$, если $c \in C_2 \setminus \{c_{2,1}, c_{2,2}\}$
$c?x\,(x \in Var, dom(x) \supseteq dom(c))$	$c?x$, если $c = c_{2,1}$ или $c = c_{2,2}$; $\varnothing$, если $c \in C_2 \setminus \{c_{2,1}, c_{2,2}\}$
	Для $c \in C_3$
$c!v\,(v \in dom(c))$	$c!v$
$c?x\,(x \in Var, dom(x) \supseteq dom(c))$	$c?x$

Рассмотрим замену недетерминированным выбором. Если оператор *atomic* {*guard_abs -> c?x;*} находится в PG_0, то он заменяется оператором недетерминированного выбора

```
if
:: atomic { guard_abs -> c?x; }
:: atomic { m.opc = opc1; m.id = ABS; }
…
:: atomic { m.opc = opck; m.id = ABS; }
fi;
```

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

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

Математическое доказательство корректности процедуры абстракции. Проверка выполнимости свойств-инвариантов для данной системы переходов TS эквивалентна проверке выполнения свойства p в каждом состоянии, достижимом из некоторого начального состояния системы TS. Покажем, что множество достижимых состояний системы $TS_{abs} = (S_{abs}, Act_{abs}, \to_{abs}, I_{abs}, AP, L_{abs})$ включает в себя множество достижимых состояний системы $TS = (S, Act, \to, I, AP, L)$.

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 $s \models p$, существует состояние $f(s) \in S_{abs}$, достижимое из некоторого начального состояния $f(s_0) \in I_{abs}$, такое, что $f(s) \models p$.*

◀ Обозначим множество состояний системы переходов TS, достижимых из некоторого начального состояния, через $Reach(TS)$. Докажем утверждение теоремы методом математической индукции по длине пути исходной системы TS [6].

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

Индуктивная гипотеза. Пусть для некоторого $m \ge 0$ верно $\forall k \le m : f(s_k) \in Reach(TS_{abs})$. Это означает, что фрагменту пути $s_0 s_1 \ldots s_m$ системы TS соответствует фрагмент пути $f(s_0) f(s_1) \ldots f(s_m)$ системы TS_{abs}, причем $f(s_i)$ и $f(s_{i+1})$, $0 \le i < m$, не обязательно различны. Кроме того, $L_{abs}(f(s_m)) = L(s_m)$, $m \ge 0$.

Шаг индукции. Рассмотрим переход $s_m \to s_{m+1}$ исходной системы. Состояние s_{m+1} исходной системы TS является следствием наличия в системе перехода из состояния s_m, описываемого одним из правил (1)–(3). 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

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

где $\eta' = Effect(\alpha, \eta)$.

Элемент множества правил абстрактной системы $Rulesi_{abstract} = \{ria_i \,|\, 0 \le i \le 3\}$ имеет вид

$$ria_i = \frac{l_i \overset{g_{abs}:\alpha_{abs}}{\Rightarrow} l_i' \wedge (\eta_{abs}, \xi_{abs})| = g_{abs}}{f(s_m) = \langle l_0, \ldots, l_i, \ldots, l_3, \eta_{abs}, \xi_{abs} \rangle \xrightarrow{\alpha_{abs}} f(s_{m+1}) = \langle l_0, \ldots, l_i', \ldots, l_3, \eta'_{abs}, \xi_{abs} \rangle},$$

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

Рассмотрим функцию $Transi : Rulesi_{concrete} \to Rulesi_{abstract}$. Абстрактные преобразования построены так, что

$Transi(ric_i) = ria_i, i = 0,1,2,$

$Transi(ric_i) = ria_3, i > 2 \wedge L(s_{m+1}) \ne L(s_m),$

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

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

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

$$\forall v \in V_{AP} : \eta_{abs}(v) = \eta(v) \in dom_{abs}(v). \qquad (4)$$

В то же время, защиту g можно представить в виде конъюнкции $g = A_1 \wedge A_2$, где $A_1 \in Cond(V_{AP}, C)$, $A_2 \notin Cond(V_{AP}, C)$. В соответствии с абстрактными преобразованиями $g_{abs} = A_1 \wedge true$, т. е. часть $A_2 \notin Cond(V_{AP}, C)$ заменена истиной. Таким образом, защита g_{abs} является логическим следствием защиты g. С учетом вида g, g_{abs} и соотношения (4) следует, что $((\eta, \xi) |= g) \Rightarrow ((\eta_{abs}, \xi_{abs}) |= g_{abs})$.

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

– $\alpha_{abs} = \alpha$, если $L(s_{m+1}) \ne L(s_m)$, т. е. имеем соответствующий переход $f(s_m) \xrightarrow{\alpha_{abs}} f(s_{m+1})$ в абстрактной системе, вследствие чего $L_{abs}(f(s_{m+1})) = L(s_{m+1})$.

– $\alpha_{abs} = \varnothing$, если $L(s_{m+1}) = L(s_m)$, т. е. в абстрактной системе два состояния исходной системы s_{m+1} и s_m с одинаковыми пометками объединяются в одно. Заключаем, что в TS_{abs} достижимо состояние $f(s_{m+1}) = f(s_m)$.

Если α — выражение, то применение соответствующего правила не изменяет пометку состояния (что имеет место и в исходной системе), а дает переход в системе TS_{abs} с более слабым условием, поскольку $\alpha \Rightarrow a_{abs}$, аналогично защитам g и g_{abs}.

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 $Rulesr_{concrete1} = \{rrc_i \,|\, 0 \le i \le n\}$ имеет вид

$$rrc_i = \frac{l_i \overset{g:c?x}{\Rightarrow} l_i' \wedge (\eta,\xi) \mid = g \wedge len(\xi(c)) = k > 0 \wedge \xi(c) = v_1 ... v_k}{\langle l_0,\ldots, l_i,\ldots, l_n, \eta, \xi\rangle \xrightarrow{\tau} \langle l_0,\ldots, l_i',\ldots, l_n, \eta', \xi'\rangle}, \quad (5)$$

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

В процессе исполнения запроса только одно правило rrc_i, $0 \le i \le n$, может порождать переходы. В этих случаях изменение пометки обусловлено изменением значения переменной *x*. Определим, какие значения может принимать *x* вследствие применения правила rrc_i, $0 \le i \le n$. Для этого рассмотрим фрагмент вычисления $s_0\alpha_1 s_1\alpha_2 \ldots \alpha_m s_m$ системы переходов *TS*. В таком случае переход $s_m \xrightarrow{\tau} s_{m+1}$ получен вследствие применения правила $rrc_i \in Rulesr_{concrete1}$. В моделях либо $i = 0$ (сообщение извлекает процесс, представляющий системный коммутатор *home*-процессора), либо $i > 0$ (сообщение извлекает процесс-запросчик с номером i). Факт применения правила означает, что условие $len(\xi(c)) = k > 0$ выполнено. В свою очередь, это позволяет записать $\xi(c) = v_1 \ldots v_k$. Следовательно, из $\alpha_1,\ldots,\alpha_m$ находится k членов, каждый из которых имеет вид $\alpha_i = \tau$. Таким образом, после осуществления перехода $s_m \xrightarrow{\tau} s_{m+1}$, переменная *x* будет принимать одно из значений из множества *X*, всех ранее отправленных по каналу значений, причем $v_1 \ldots v_k$ 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 $S_1 \le |dom_{abs}(c)|$ — число различных отправляемых значений в операторах отправки сообщения, соответствующих операторам приема сообщения из $c \in C_1$. Выбор правила из этого множества недетерминирован, т. е. $Transr_1 : Rulesr_{concrete1} \to Rulesr_{abstract1}$ — мультиотображение. Первый тип элементов этого множества имеет вид

$$rra_i = \frac{l_i \overset{g_{abs}:c?x}{\Rightarrow} l_i' \wedge (\eta_{abs},\xi_{abs}) \mid = g_{abs} \wedge len(\xi_{abs}(c)) = l > 0 \wedge \xi_{abs}(c) = v_1 ... v_l}{\langle l_0,\ldots, l_i,\ldots, l_3, \eta_{abs}, \xi_{abs}\rangle \xrightarrow{\tau} \langle l_0,\ldots, l_i',\ldots, l_3, \eta_{abs}', \xi_{abs}'\rangle},$$

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

Согласно абстрактным преобразованиям,

$$Transr_1(rrc_i) = rra_i,\ 0 \le i \le 2,\ \ Transr_1(rrc_i) = rra_3, 2 < i \le n.$$

Определим, какие значения при этом может принимать переменная x в абстрактной модели в состоянии $f(s_{m+1})$. Рассмотрим фрагмент вычисления $f(s_0)\,\mathrm{A}(\alpha_1)\,f(s_1)\mathrm{A}(\alpha_2)\ldots\mathrm{A}(\alpha_m)\,f(s_m)$ системы переходов TS_{abs}. Здесь $\mathrm{A} : Act \cup \{\tau\} \to Act_{abs} \cup \{\tau\}$. Условие $len(\xi_{abs}(c)) = l > 0$ может быть выполнено только, если из $\mathrm{A}(\alpha_1),\ldots,\mathrm{A}(\alpha_m)$ 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

$$rsc_i=\frac{l_i \overset{g:c!v}{\Rightarrow} l_i' \ \wedge\ (\eta,\xi)|=g \ \wedge\ len(\xi(c))=k<cap(c) \ \wedge\ \xi(c)=v_1...v_k}{\langle l_0,\ldots,l_i,\ldots,l_n,\eta,\xi\rangle \overset{\tau}{\to}\langle l_0,\ldots,l_i',\ldots,l_n,\eta,\xi'\rangle}, \qquad (6)$$

где $\xi'=\xi[c:=v_1v_2\ldots v_kv]$.

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

$$rsa_i=\frac{l_i \overset{g_{abs}:c!v}{\Rightarrow} l_i'\wedge(\eta_{abs},\xi_{abs})|=g_{abs}\wedge len(\xi_{abs}(c))=l<cap_{abs}(c)\wedge\xi_{abs}(c)=v_1...v_l}{\langle l_0,\ldots,l_i,\ldots,l_3,\eta_{abs},\xi_{abs}\rangle \overset{\tau}{\to}\langle l_0,\ldots,l_i',\ldots,l_3,\eta_{abs},\xi_{abs}'\rangle}$$

где $\xi_{abs}^{'}=\xi_{abs}[c:=v_1v_2\ldots v_lv]$, $cap_{abs}(c)=2$.

В соответствии с преобразованиями абстракции отображение $Transs_1: Ruless_{concrete1}\to Ruless_{abstract1}$ определено так, что $Transs_1(rsc_i)=rsa_i$, $i=1,2$, $Transs_1(rsc_i)=\varnothing$, $i>2$. Следовательно, $\mathrm{A}(\tau)=\tau$, если соответствующая метка порождена действиями $c!v$ процессов PG_1, PG_2, и $\mathrm{A}(\tau)=\varnothing$ в противном случае. Таким образом, переменная x может принимать только те значения из множества X, которые были отправлены процессами PG_1 и PG_2.

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

$$raa_{i,j}=\frac{l_i \overset{g_{abs}:x.opc:=v,x.id:=ABS}{\Rightarrow} l_i'\wedge(\eta_{abs},\xi_{abs})|=g_{abs}}{\langle l_0,\ldots,l_i,\ldots,l_3,\eta_{abs},\xi_{abs}\rangle \overset{x.opc:=v,x.id:=ABS}{\to}\langle l_0,\ldots,l_i',\ldots,l_3,\eta_{abs}',\xi_{abs}\rangle}.$$

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

Мультиотображение $Transr_1$ определено следующим образом, как следует из абстрактных преобразований. 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 $2<i\le n$.

Защита g имеет вид $g = A_1 \wedge A_2$, где $A_1 \in Cond(V_{AP}, C_1)$; $A_2 \notin Cond(V_{AP}, C_1)$. 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 было отправлено процессом PG_1 или PG_2, то это же справедливо и в абстрактной системе. В таком случае условие $nempty(c)$ выполняется. Доказательство выполнимости условий относительно переменных аналогично случаю 1.

Случай б. Правило исходной системы rrc_i, правила абстрактной системы $raa_{t,j}$, $1 \le j \le S_1$. Доказательство выполнимости посылки правила абстрактной системы такое же, как и в случае 1.

Следовательно, в абстрактной системе из состояния $f(s_m)$ существует множество переходов, в которых всегда найдется переход $f(s_m) \to f(s_{m+1})$, соответствующий переходу $s_m \to s_{m+1}$, такой, что $L_{abs}(f(s_{m+1})) = L(s_{m+1})$.

Случай 2.2: $c \in C_2$. Элемент множества правил исходной системы $Rulesr_{concrete2} = \{rrc_i \mid 1 \le i \le n\}$ имеет вид (5), элемент множества правил абстрактной системы $Rulesr_{abstract2} = \{rra_i \mid 1 \le i \le 2\}$ — вид

$$rra_i = \frac{l_i \overset{g_{abs}:c_{2,i}?x}{\Rightarrow} l_i' \ \wedge \ (\eta_{abs}, \xi_{abs})| = g_{abs} \ \wedge \ len(\xi_{abs}(c_{2,i})) = k > 0 \ \wedge \ \xi_{abs}(c_{2,i}) = v_1 \ldots v_k}{\langle l_0, \ldots, l_i, \ldots, l_3, \eta_{abs}, \xi_{abs}\rangle \xrightarrow{\tau} \langle l_0, \ldots, l_i', \ldots, l_3, \eta'_{abs}, \xi'_{abs}\rangle},$$

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

Из выполнимости условия $len(\xi(c)) = k > 0$ в посылке правила $rrc_i \in Rulesr_{concrete2}$, следует выполнимость $len(\xi_{abs}(c_{2,i})) = k > 0$ в посылке правила $rra_i \in Rulesr_{abstract2}$. Покажем это рассмотрением фрагмента вычисления $s_0\alpha_1 s_1\alpha_2 \ldots \alpha_m s_m$ системы TS и соответствующего фрагмента вычисления $f(s_0)A(\alpha_1)f(s_1)A(\alpha_2)\ldots A(\alpha_m)f(s_m)$ системы TS_{abs}. Для случая отправки сообщения по каналу $c \in C_2$ множество правил исходной системы $Ruless_{concrete2} = \{rsc_0\}$, $c \in C_2 = \{c_{2,1}, ..., c_{2,n}\}$:

$$rsc_0 = \frac{l_0 \overset{g:c!v}{\Rightarrow} l_0' \ \wedge \ (\eta, \xi)| = g \ \wedge \ len(\xi(c)) = k < cap(c) \ \wedge \ \xi(c) = v_1 \ldots v_k}{\langle l_0, \ldots, l_i, \ldots, l_n, \eta, \xi\rangle \xrightarrow{\tau} \langle l_0', \ldots, l_i, \ldots, l_n, \eta, \xi'\rangle},$$

где $\xi' = \xi[c := v_1 v_2 \ldots v_k v]$.

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

$$rsa_0 = \frac{l_0 \overset{g_{abs}:c!v}{\Rightarrow} l_0' \ \wedge \ (\eta_{abs}, \xi_{abs})| = g_{abs} \ \wedge \ len(\xi_{abs}(c)) = k < cap(c) \ \wedge \ \xi_{abs}(c) = v_1 \ldots v_k}{\langle l_0, \ldots, l_i, \ldots, l_3, \eta_{abs}, \xi_{abs}\rangle \xrightarrow{\tau} \langle l_0', \ldots, l_i, \ldots, l_3, \eta_{abs}, \xi'_{abs}\rangle},$$

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

Отображение $Transs_2 : Ruless_{concrete2} \to Ruless_{abstract2}$ определено следующим образом: $Transs_2(rsc_0) = rsa_0, c \in \{c_{2,1}, c_{2,2}\}$, $Transs_2(rsc_0) = \varnothing, c \in C_2 \setminus \{c_{2,1}, c_{2,2}\}$. 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 $c_{2,2}!v$, $A(\alpha_i) = \varnothing$ в противном случае.

В соответствии с абстрактными преобразованиями, отображение $Transr_2 : Rulesr_{concrete2} \to Rulesr_{abstract2}$ такое, что, во-первых, $Transr_2(rrc_i) = rra_i$, $i = 1, 2$. Выполнимость посылок правил абстрактной системы при выполнимости посылок соответствующих правил исходной системы показывается аналогично, как и в случае 2.1. Во-вторых, $Transr_2(rrc_i) = \varnothing$, $i > 2$. 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. 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 множество V_{AP}.

Случай 2.3: $c \in C_3$. Элемент множества правил исходной системы $Rulesr_{concrete3} = \{rrc_i \mid 0 \le i \le n\}$ имеет вид (5). Элемент множества правил абстрактной системы $Rulesr_{abstract3} = \{rra_i \mid 0 \le i \le 3\}$:

$$rra_i = \frac{l_i \overset{g_{abs}:c?x}{\Rightarrow} l_i' \ \wedge \ (\eta_{abs}, \xi_{abs}) \mid = g_{abs} \ \wedge \ len(\xi_{abs}(c)) = k > 0 \ \wedge \ \xi_{abs}(c) = v_1 \ldots v_k}{\langle l_0, \ldots, l_i, \ldots, l_3, \eta_{abs}, \xi_{abs} \rangle \xrightarrow{\tau} \langle l_0, \ldots, l_i', \ldots, l_3, \eta'_{abs}, \xi'_{abs} \rangle},$$

где $\eta'_{abs} = \eta_{abs}[x := v_1]$ и $\xi'_{abs} = \xi_{abs}[c := v_2 \ldots v_k]$.

Выполнимость условия $len(\xi_{abs}(c)) = k > 0 \ \wedge \ \xi_{abs}(c) = v_1 \ldots v_k$ 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. Это доказывается рассмотрением для случая отправки сообщения по каналу $c \in C_3$ функции из множества правил исходной системы во множество правил абстрактной системы (аналогично случаям 2.1 и 2.2).

Отображение $Transr_3 : Rulesr_{concrete} \to Rulesr_{abstract3}$ определено следующим образом: $Transr_3(rrc_i) = rra_i$, $i = 0,1,2$, $Transr_3(rrc_i) = rra_3$, $i > 2$. Следовательно, переходу исходной системы, порожденному считыванием сообщения из канала $c \in C_3$, в абстрактной системе всегда соответствует переход, порожденный аналогичным считыванием, 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}.

Отметим, что переходы, вызванные наличием в графах программ ребер с метками *goto* и их эквивалентами, не изменяют пометку состояния и сохраняются в графах программ 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 $L_{abs}(f(s_{m+1})) = L(s_{m+1})$. ►

Заключение. Предложенные преобразования формальных моделей протоколов когерентности памяти позволяют уменьшить число процессов верифицируемой модели с большого числа до нескольких (в настоящей работе четырех). Синтаксическая природа преобразований позволяет их автоматизировать, а всю работу по созданию и анализу соответствующей системы переходов предоставить средству автоматизированной проверки моделей *Spin*. Это позволяет использовать все реализованные в нем алгоритмы, включая множество алгоритмов оптимизации, с помощью которых можно сократить число состояний исследуемой системы переходов.

Все приведенные особенности и ограничения моделей соответствуют разработанному авторами подходу к составлению моделей протоколов когерентности, который позволил лаконично описать протокол когерентности системы «Эльбрус-4С» и провести верификацию системы из трех ядер. Предложенные абстрактные преобразования позволяют сократить число процессов и размеры используемых структур данных так, что после устранения ложных контрпримеров, размер полученной модели будет находиться в пределах от размера модели для трех ядер до размера модели для четырех ядер, что позволит провести верификацию. Автоматизация процесса получения абстрактных моделей, описанного в настоящей работе, а также разработка метода уточнения абстрактной модели для устранения ложных контрпримеров являются направлениями дальнейшей работы и исследований.

ЛИТЕРАТУРА

1. *Буренков В.С.* Анализ применимости формальных методов к верификации протоколов когерентности кэш-памяти масштабируемых систем // Вопросы радиоэлектроники. 2015. № 3. С. 105–117.

2. *Chou C., Mannava P., Park S.* A simple method for parameterized verification of cache coherence protocols // Proc. Formal Methods in Computer-Aided Design. 2004. P. 382–398.

3. *Talupur M., Tuttle M.* Going with the flow: Parameterized verification using message flows // Proc. Formal Methods in Computer-Aided Design. 2008. P. 1–8.

4. *O'Leary J., Talupur M., Tuttle M.* Protocol verification using flows: An industrial experience // Proc. Formal Methods in Computer-Aided Design. 2009. P. 172–179.

5. *Sorin D.J., Hill M.D., Wood D.A.* A primer on memory consistency and cache coherence. San Rafael: Morgan & Claypool Publishers, 2012. 210 p.

6. *Baier C., Katoen J.-P.* Principles of model checking. Cambridge: MIT Press, 2008. 984 p.

7. *Holzmann G.* The spin model checker: Primer and reference manual. Boston: Addison-Wesley Professional, 2003. 608 p.

Буренков Владимир Сергеевич — аспирант кафедры «Компьютерные системы и сети» МГТУ им. Н.Э. Баумана (Российская Федерация, 105005, Москва, 2-я Бауманская ул., д. 5).

Иванов Сергей Ростиславович — канд. техн. наук, доцент кафедры «Компьютерные системы и сети» МГТУ им. Н.Э. Баумана (Российская Федерация, 105005, Москва, 2-я Бауманская ул., д. 5).

Просьба ссылаться на эту статью следующим образом:

Буренков В.С., Иванов С.Р. Метод построения абстрактных моделей, используемых для верификации протоколов когерентности кэш-памяти масштабируемых систем // Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение. 2017. № 1. С. 49–66.
DOI: 10.18698/0236-3933-2017-1-49-66

METHOD OF CONSTRUCTING ABSTRACT MODELS FOR PROTOCOL VERIFICATION OF CACHE COHERENCE IN SCALABLE SYSTEMS

V.S. Burenkov vansburen@mail.ru
S.R. Ivanov ivanovsr@bmstu.ru

Bauman Moscow State Technical University, Moscow, Russian Federation

Abstract

The article presents a novel approach to solving the verification problem for cache coherence in scalable microprocessor systems. The article examines a mathematical model of cache coherence protocols. The model uses program graphs as an abstraction for groups of devices operating in accordance with a given cache coherence protocol. The whole protocol is represented as a channel system that can be unfolded into a transition system to be explored by model checking algorithms. The paper proposes an approach to the construction of *Promela* models of cache coherence protocols and defines the limitations on models. The article presents a method for constructing abstract models of cache coherence protocols that produces models of significantly smaller size and therefore subjected to model checking. The method is based on program graphs transformations, or, equivalently, on syntactical transformations of *Promela* models. The paper contains a mathematical proof of invariance preservation by the abstract models. All the results are based on verification practice of a complex industrial *Elbrus* system-on-a-chip

Keywords

Formal method, model checking, model transformations, cache coherence protocol

REFERENCES

[1] Burenkov V.S. An analysis of applicability of formal methods to verification of cache coherence protocols of scalable systems. *Voprosy radioelektroniki* [Questions of Radio-Electronics], 2015, no. 3, pp. 105–117 (in Russ.).

[2] Chou C., Mannava P., Park S. A simple method for parameterized verification of cache coherence protocols. *Proc.* "*Formal Methods in Computer-Aided Design*", 2004, pp. 382–398.

[3] Talupur M., Tuttle M. Going with the flow: Parameterized verification using message flows. *Proc.* "*Formal Methods in Computer-Aided Design*", 2008, pp. 1–8.

[4] O'Leary J., Talupur M., Tuttle M. Protocol verification using flows: An industrial experience. *Proc.* "*Formal Methods in Computer-Aided Design*", 2009, pp. 172–179.

[5] Sorin D.J., Hill M.D., Wood D.A. A primer on memory consistency and cache coherence. San Rafael, Morgan & Claypool Publishers, 2012. 210 p.

[6] Baier C., Katoen J.-P. Principles of model checking. Cambridge, MIT Press, 2008. 984 p.

[7] Holzmann G. The spin model checker: Primer and reference manual. Boston, Addison-Wesley Professional, 2003. 608 p.

Burenkov V.S. — post-graduate student of Computer Systems, Complexes and Networks Department, Bauman Moscow State Technical University (2-ya Baumanskaya ul. 5, Moscow, 105005 Russian Federation).

Ivanov S.R. — Cand. Sci. (Eng.), Assoc. Professor of Computer Systems, Complexes and Networks Department, Bauman Moscow State Technical University (2-ya Baumanskaya ul. 5, Moscow, 105005 Russian Federation).

Please cite this article in English as:

Burenkov V.S., Ivanov S.R. Method of Constructing Abstract Models for Protocol Verification of Cache Coherence in Scalable Systems**.** *Vestn. Mosk. Gos. Tekh. Univ. im. N.E. Baumana, Priborostr.* [Herald of the Bauman Moscow State Tech. Univ., Instrum. Eng.], 2017, no. 1, pp. 49–66. DOI: 10.18698/0236-3933-2017-1-49-66