ВЕРИФИКАЦИЯ МОДЕЛЕЙ ПРОТОКОЛОВ С ПОМОЩЬЮ СУПЕРКОМПИЛЯТОРОВ: ПОЧЕМУ ЭТО УДАЕТСЯ?

А.В. Климов

Аннотация. Рассматриваются результаты экспериментов по верификации с помощью суперкомпиляторов моделей протоколов кеш-когерентного доступа к памяти, протоколов взаимодействия параллельных процессов и других параметризованных моделей. А.П. Лисица и А.П. Немытых провели эти эксперименты с помощью суперкомпилятора SCP4 языка Рефал. Автор воспроизвел их на суперкомпиляторе JScp языка Java. В статье теоретически объясняется, почему эти эксперименты оказались успешными для части моделей. Формально характеризован класс моделей и вид условий верификации, для которых можно доказать, что суперкомпилятор может либо построить доказательство корректности модели, либо опровергнуть, построив контрпример: это класс монотонных счетчиковых систем с множеством «опасных» состояний, являющимся верхним конусом. Сформулированы алгоритмы суперкомпиляции и верификации для этого класса моделей (разрешения проблем *достижимости* и *покрытия*, *reachability and coverability problems*). Приведены схемы доказательств корректности этих алгоритмов.

Счетчиковая абстракция и счетчиковые системы переходов.

Исследования последних двух десятилетий по верификации протоколов и других систем с бесконечным числом состояний показали практическую плодотворность следующего подхода к моделированию, называемому *счетчиковой абстракцией* (*counting* или *counter abstraction*). Рассмотрим систему из семейства однотипных конечных автоматов, которые меняют состояния по некоторым правилам, описывающих синхронное изменение состояний у части автоматов (например, если один из автоматов находится в состоянии A, а другой в состоянии B, то первый переходит в состояние C, а второй в D). Задается множество допустимых начальных состояний I («*initial*») (например, не меньше одного автомата в состоянии A, остальные в B), и множество состояний U («unsafe»), называемых по традиции приложений «опасными» состояниями (например, если в состоянии C находится больше одного автомата, то это «опасно»). Требуется доказать, что «опасные» состояния недостижимы из начальных. Условия такого вида называют *условиями безопасности* (*safety properties*). Иногда рассматриваются системы, включающие еще один автомат, называемый «управляющим». Результаты данной статьи применимы и к этому случаю.

Счетчиковая абстракция — это описание состояний семейства конечных автоматов в виде кортежей целых чисел, указывающих, сколько автоматов находятся в данном состоянии (например, $\langle a,b,c,d\rangle$ означает, что a автоматов находятся в состоянии A, b автоматов — в B и т.д.). Хотя счетчиковая абстракция передает лишь часть информации о системе, на практике, как правило, этого достаточно, чтобы формулировать и доказывать условия безопасности для произвольного числа автоматов.

Определение. Счетчиковые системы (*counter systems*) — это недетерминированные системы переходов (*transition systems*), состояния которых задаются кортежами целых неотрицательных чисел, то есть множество состояний системы $\mathbf{S} = \mathbf{N}^k$ при фиксированном k, где $\mathbf{N}$ — целые неотрицательные числа. Допустимое состояние s', следующее за текущим s, задается отношением перехода $s \Rightarrow s'$.

На множестве кортежей чисел имеется естественный частичный порядок покомпонентного сравнения: $x \le y$, если для всех компонент $x_i \le y_i$, $i = 1,...,k$. Этот порядок играет фундаментальную роль в классификации счетчиковых систем, а также конструировании и изучении свойств алгоритмов их анализа и преобразований.

Определение. Система переходов с отношением квазипорядка $\le$ на ее состояниях называется *монотонной*, если для любых состояний s_1, s_2, s_3 таких, что $s_1 \Rightarrow s_2$, $s_1 \le s_3$, существует s_4, такое, что $s_3 \Rightarrow s_4$, $s_2 \le s_4$.

Описанный ниже алгоритм верификации может применяться как к монотонным, так и к немонотонным счетчиковым системам, но для монотонных мы знаем, что он всегда завершается. А для немонотонных он даст правильный результат, *если* завершится.

В краткой статье мы не можем дать обзор обилия работ по методам верификации счетчиковых систем. Отметим только работы G. Delzanno, который собрал большую коллекцию моделей реальных протоколов [9, 10] (мы использовали ее как экспериментальный материал) и самую близкую к нашей работу [11], описывающих схему алгоритмов, названную авторами «Expand, Enlarge and Check».

В рассмотренных G. Delzanno и нами практических моделях переход в следующее состояние счетчиковой системы описывается правилами вида $L \to R$, где L — конъюнкция условий вида $x_i \ge a_i$ или $x_i = a_i$, где a_i — константы; R — набор присваиваний вида $x_i := \sum_j x_j + b_i$, где суммируются ноль или несколько компонент x_j, а b_i — константы. Если в левой части все условия имеют вид $x_i \ge a_i$, такая система монотонна. Если встречаются условия вида $x_i = a_i$, система немонотонна.

В качестве примера ниже приводится самая простая монотонная модель из коллекции G. Delzanno. Она верифицируется суперкомпиляторами за доли секунды.

Пример. Модель протокола Synapse N+1 кеш-когерентного доступа к памяти в виде монотонной счетчиковой системы [9, 10].

Состояние: тройка из целых неотрицательных чисел *invalid, dirty*, *valid*.

Правила переходов:

1. $invalid \geq 1 \rightarrow dirty := 0, valid := valid + 1, invalid := invalid + dirty - 1$;
2. $invalid \geq 1 \rightarrow valid := 0, dirty := 1, invalid := invalid + dirty + valid - 1$;
3. $valid \geq 1 \rightarrow valid := 0, dirty := 1, invalid := invalid + dirty + valid - 1$.

Множество начальных состояний *I*:

$invalid \geq 1$ & $dirty = 0$ & $valid = 0$.

Множество «опасных» состояний *U*:

$invalid \geq 0$ & $dirty \geq 1$ & $valid \geq 1$.

Для завершаемости наших алгоритмов важно, что отношение $\leq$ на $\mathbf{N}^k$ является *правильным квазипорядком* (*well-quasi-order*). (Русский термин заимствован нами из [3].)

Определение. Квазипорядок $\leq$ на множестве X называется *правильным*, если в любой бесконечной последовательности $\{x_i\}$ элементов из X всегда найдется пара $i < j$ такая, что $x_i \leq x_j$.

Нам понадобятся стандартные понятия верхнего и нижнего конуса.

Определение. Подмножество U множества X с квазипорядком $\leq$ называется *верхним конусом*, если из того, что x принадлежит U и $x \leq y$, следует, что y принадлежит U. *Нижний конус* определяется симметрично.

Суперкомпиляция и ее применение для верификации программ.

Метод преобразования программ, называемый *суперкомпиляцией*, разрабатывается В.Ф Турчиным [8, 17, 18] и его последователями [1, 2, 7, 12 и др.] с 1970-х годов. В последние годы он был доведен до реализации в экспериментальных суперкомпиляторах практических языков программирования Рефал [7, 16] и Java [2, 5, 12]. Основная область использования суперкомпиляторов — специализация и оптимизация программ. Однако еще в самом начале работ по суперкомпиляции В.Ф. Турчин обращал внимание на возможность ее использования для машинного доказательства теорем [17]. Это открывает возможность применения суперкомпиляторов к верификации программ и других формальных систем, которые могут быть промоделированы программами.

А.П. Лисица и А.П. Немытых обнаружили, что практические модели из коллекции G. Delzanno [9, 10], закодированные на языке Рефал-5 [16], успешно верифицируются суперкомпилятором SCP4 [7, 16]. Результаты их экспериментов представлены на сайте [16] и опубликованы в серии статей [6, 15 и др.]. Мы повторили эти результаты на суперкомпиляторе языка Java [2, 5, 13]. Они представлены на сайте [4].

Хотя суперкомпиляторы SCP4 и JScp значительно отличаются и по входному языку, и по объему реализованных методов суперкомпиляции, воспроизводимость результата наводит на определенные соображения. Во-первых, это значит, что результат опирается на ядро метода суперкомпиляции, а не на «технические детали». Во-вторых, почти стопроцентная результативность наводит на гипотезу, что рассмотренные программные модели принадлежат некоторому классу, про который можно доказать, что если модель корректна, суперкомпилятор выявляет это, а если модель некорректна, находит контрпример. Ниже эта гипотеза доказана для монотонных систем.

Суперкомпиляция систем переходов.

Сформулируем алгоритм суперкомпиляции применительно к *вполне упорядоченным системам переходов* (*well-quasi-ordered transition systems*), то есть к системам с правильным квазипорядком на состояниях. По ходу изложения будем конкретизировать определения для счетчиковых систем.

Понятие конфигурации. Суперкомпилятор строит дерево всевозможных последовательностей переходов для множеств состояний, описанных некоторым конструктивным способом и называемых *конфигурациями.*

(Мы используем терминологию теорию суперкомпиляции, которая, к сожалению, в этом месте входит в противоречие с терминологией сетей Петри и счетчиковых систем, где слово «конфигурация» означает наше «состояние», а наша «конфигурация» соответствует их термину «ω-конфигурация».)

Применительно к нашим задачам для счетчиковых систем, достаточно рассматривать конфигурации в виде кортежей целых чисел и символов ω, кодирующих все множество **N**. Таким образом, имеем множество конфигураций $\mathbf{C} = (\mathbf{N} \cup \{\omega\})^k$. Конфигурация $C = \langle x_1,...,x_k \rangle$ описывает множество состояний $X_1*...*X_k$, где $X_i = \{x_i\}$, если x_i — целое число, $X_i = \mathbf{N}$, если $x_i = \omega$, и * — декартово произведение.

Начальное множество состояний I и «опасное» U, как правило, не являются множествами такого вида. Во всех рассмотренных моделях множество U является верхним конусом, заданным условиями вида $x_i \geq a_i$, где a_i — константа. Нам достаточно, что можно проверять пустоту пересечения U с конфигурациями. Начальные множества I задаются условиями вида $x_i \geq a_i$ или $x_i = a_i$. Для разрешения проблемы достижимости верхнего конуса монотонной системой достаточно аппроксимировать такое множество I конфигурацией C_0, описывающей минимальное надмножество I, представимое конфигурацией.

Отношение частичного порядка $\leq$ на состояниях продолжим на конфигурации, считая, что ω не сравнимо с числами. При этом оно остается правильным квазипорядком на **C**.

Определение. Имея две конфигурации C_1 и C_2, назовем конфигурацию G их *обобщением*, если G описывает минимальное представимое конфигурациями надмножество множеств, описываемых конфигурациями C_1 и C_2. Применительно к счетчиковым системам, компоненты обобщения G либо равны соответствующим компонентам C_1 и C_2, когда они совпадают, либо ω в противном случае.

Прогонка. Суперкомпилятор выполняет шаги переходов, аппроксимируя сверху множество следующих состояний. Он использует функцию Post(C), которая по текущей конфигурации C выдает конечное множество конфигураций $\{C_i\}$ такое, что для любого состояния s', следующего за некоторым состоянием s из C, существует C_i, включающее s'.

Для счетчиковых систем, заданных правилами $L_i \to R_i$ описанного выше вида, Post выдает по одной конфигурации на каждое правило, у которого вычисление L_i на C дает *true*. Для этого операции в L_i и R_i расширяются на ω так: $a < \omega$ и $\omega + a = \omega - a = \omega$, где a — число.

Алгоритм суперкомпиляции, приведенный ниже, проще классических алгоритмов суперкомпиляции обычных программ из-за недетерминированности систем переходом, отсутствия у них рекурсии и того, что он выдает лишь дерево и множество конфигураций, а не полноценную остаточную программу.

Алгоритм 1: Суперкомпиляция вполне упорядоченных систем переходов.
Исходные данные:
- Система переходов с отношением правильного квазипорядка ≤ на конфигурациях.
- Начальная конфигурация C_0.

Состояние:
- Дерево, вершины которого помечены конфигурациями. (Вершины и связанные с ними конфигурации будем обозначать одинаково буквами C.)
- Часть листьев образует упорядоченное множество еще необработанных конфигураций *ToDo*, используемое как стек.

Начальное состояние:
- Дерево из одной корневой вершины с начальной конфигурацией C_0.
- *ToDo* = $\{C_0\}$.

Результат:
- Дерево и множество конфигураций, называемые *остаточными*.

Цикл: Пока стек *ToDo* непуст, берем и удаляем из стека *ToDo* очередную конфигурацию C и выполняем один из следующих шагов:
1. *Зацикливание*: Если текущая конфигурация C описывает подмножество одной из уже присутствующих в дереве конфигураций C', то шаг заканчивается.
2. *Свисток и обобщение*: Если на пути от начальной конфигурации C_0 к текущей C есть конфигурация C', такая что $C' \le C$, то обобщаем C' и C, получая G. Удаляем из дерева поддерево ниже C'. Заменяем в дереве конфигурацию C' на G. Удаляем из стека *ToDo* конфигурации из удаленного поддерева. Помещаем G в стек *ToDo*.
3. *Прогонка*: В противном случае вычисляем множество конфигураций Post(C), добавляем их в дерево в качестве потомков C и в стек *ToDo*.

Утверждение 1. Алгоритм 1 всегда завершается, если
- порядок ≤ является правильным квазипорядком,
- отношение ≤ и понятие обобщения таковы, что в п. 2 G ≠ C', если C' ≤ C, и
- у каждой конфигурации есть лишь конечное число обобщений.

Доказательство. Правильный квазипорядок ≤ и проверка в п. 2 не разрешают бесконечные пути. Ветвления конечны: в п. 3 строится конечное число потомков, поскольку такова функция Post; вариантов, порождаемых в п.2, не больше, чем число возможных обобщений первой конфигурации в данном узле. Отмечая, что состояние не может повториться (на состояниях алгоритма можно ввести порядок, по которому следующее состояние строго больше предыдущего), заключаем, что алгоритм всегда завершается.

Утверждение 2. Остаточные конфигурации включают в себя все достижимые состояния.

Доказательство. В п. 3 конфигурации-потомки включают все следующие состояния. В п. 2 конфигурация C' увеличивается (после чего потомки пересчитываются). Корректность п. 1 предполагает, что ранее существовавшая конфигурация C' не будет удалена, пока в дереве присутствует C. Это гарантируется детерминированным порядком обхода деревьев в глубину, используя *ToDo* как стек.

Верификации счетчиковых систем.

Назовем *остаточным множеством состояний R* (*residual set*) объединение множеств состояний, соответствующих остаточным конфигурациям. Если пересечение R с «опасным» множеством U пусто, то очевидно, U не достижимо из C_0. Однако нам требуется противоположное: чтобы *всякий раз*, когда U недостижимо, суперкомпилятор выдавал остаточное множество R, не пересекающееся с U.

А.П. Немытых экспериментально обнаружил, что этого удается достичь, «подкручивая» отношение ≤, запрещая сравнение и обобщение небольших чисел. Например, если *не* считать, что 0 ≤ 1, тем самым запрещая

обобщение чисел 0 и 1 в п. 2, то для всех рассмотренных корректных моделей протоколов существующие суперкомпиляторы SCP4 и JScp строят R, не пересекающееся с U.

Это наблюдение обобщается следующим образом.

Ограничим отношение частичного порядка на множестве конфигураций $\mathbf{C} = (\mathbf{N} \cup \{\omega\})^k$ с помощью целочисленного параметра l следующим образом. Будем считать, что $x \leq_l y$, если i-ые компоненты x_i и y_i либо равны, либо $l < x_i < y_i < \omega$, то есть числа меньшие или равные l объявляются несравнимыми со всеми числами. Ограниченное отношение $\leq_l$ является правильным квазипорядком на $\mathbf{C}$ так же, как и исходное $\leq$.

Основное утверждение. Для любой монотонной счетчиковой системы существует такое значение l параметра отношения частичного порядка $\leq_l$, что остаточное множество R, построенное суперкомпилятором с использованием этого отношения, пересекается с верхним конусом U тогда и только тогда, когда U недостижимо.

Доказательство основывается на монотонности системы и на следующем интересном свойстве отношения $\leq$ на кортежах целых чисел.

Определение. Назовем конфигурацию G *l-ограниченным обобщением конфигурации* C, если у них отличаются только такие i-ые компоненты C_i и G_i, что $l < C_i$ и $G_i = \omega$. Это означает, что запрещено обобщение чисел, меньших или равных l.

Основная лемма. Для множества состояний $\mathbf{S} = \mathbf{N}^k$ с покомпонентным частичным порядком $\leq$ и для любого нижнего конуса D в $\mathbf{S}$, существует целое число l такое, что для любой конфигурации C из $\mathbf{C} = (\mathbf{N} \cup \{\omega\})^k$ и для любого ее l-ограниченного обобщения G, если C описывает подмножество D, то и G описывает подмножество D.

Доказательство. Рассмотрим верхний конус U, являющийся дополнением к нижнему конусу D. Поскольку $\leq$ является правильным квазипорядком, среди элементов U существует конечное число минимальных элементов M, порождающих верхний конус U. Пусть m — максимальное число среди компонент кортежей M. Можно доказать от противного, предполагая существование состояние s из G, не принадлежащего D, что любое $l > m$ удовлетворяет условию леммы.

Приведенное выше основное утверждение подсказывает идею алгоритма 2.

Алгоритм 2: Верификация монотонных счетчиковых систем (разрешение проблем достижимости и покрытия).
Исходные данные:
- Монотонная счетчиковая система с отношением частичного порядка на конфигурациях $\leq_l$ с параметром l.
- Начальная конфигурация C_0.
- Множество состояний U, являющееся верхним конусом по отношению $\leq$.

Результат:
- «Достижимо» или «Недостижимо» множество U из начальной конфигурации C_0.
- *Цикл* для $l = 0, 1, 2, 3, ...$:

1. Вычисляем остаточное множество R по алгоритму 1, используя отношение $\leq_l$ с параметром l.
2. Если пересечение R и U пусто, выдаем ответ «Недостижимо».
3. Если одна из конфигураций C, у которой все родители и она сама не подвергались обобщению по п. 2, пересекается с U, выдаем ответ «Достижимо».

Утверждение. Приведенный алгоритм всегда завершается и выдает правильный ответ о достижимости. Корректность ответа «Достижимо» (п. 3) предполагает, что прогонка «достаточно точна» в следующем смысле: конфигурации-потомки точно описывают *покрытие* множества следующих состояний (нижние конусы, порожденные конфигурациями-потомками и множеством следующих состояний, равны).

Доказательство. Случай достижимости непосредственно вытекает из основного утверждения. Случай недостижимости (п. 3) выше не рассматривался. Это классический прием, восходящий к первой работе В.Ф. Турчина по прогонке [8] и лежащий в основе предложенного им «универсального решающего алгоритма» (УРА) [8]. Легко проверить, что для монотонных счетчиковых систем и используемого представления конфигураций предположение о «точности» прогонки действительно имеет место.

Заключение. Приведен алгоритм суперкомпиляции вполне упорядоченных систем переходов, являющийся конкретизацией на такие системы классического алгоритма суперкомпиляции В.Ф. Турчина [18]. На его основе предложен новый алгоритм проверки достижимости *монотонной* счетчиковой системой множества состояний, являющегося *верхним конусом* по покомпонентному отношению частичного порядка кортежей целых чисел. Этот алгоритм и реализованные суперкомпиляторы также успешно верифицируют многие *немонотонные* модели реальных протоколов. Теоретическое объяснение этого факта пока остается отрытой проблемой.

Проблема (не)достижимости в счетчиковых системах имеет большое практическое значение, поскольку к ней сводятся задачи верификации многих реальных систем: протоколов кеш-когерентного доступа к памяти, широковещательных протоколов, взаимодействия семейств параллельных процессов и других параметризованных систем.

Работа выполняется при поддержке РФФИ по грантам № 08-07-00280-а и № 09-01-00834-а.

ЛИТЕРАТУРА:

1. С.М. Абрамов *Метавычисления и их приложения*. М.: Наука. Физматлит, 1995
2. Анд.В. Климов Особенности построения суперкомпилятора языка Java // *Научный сервис в сети Интернет: решение больших задач: Труды Всероссийской научной конференции (21–27 сентября 2008 г., г. Новороссийск).* М.: Изд-во Московского университета, 2008. С. 252–256
3. Е.В. Кузьмин, В.А. Соколов *Вполне структурированные системы помеченных переходов.* М.: Физматлит, 2005
4. Анд.В. Климов *Проект JVer: Верификация программ на языке Java с помощью суперкомпилятора JScp*. Электронный ресурс. http://pat.keldysh.ru/jver
5. Анд.В. Климов, Арк.В. Климов, А.Б. Шворин *Проект Java-суперкомпилятор*. Электронный ресурс. http://www.supercompilers.ru
6. А.П. Лисица, А.П. Немытых Верификация как параметризованное тестирование (эксперименты с суперкомпилятором SCP4) // *Программирование*, N 1, 2007. С. 22–34
7. А.П. Немытых *Суперкомпилятор SCP4: общая структура*. М.: Editorial URSS, 2007
8. В.Ф. Турчин Эквивалентные преобразования рекурсивных функций, описанных на языке Рефал // *Труды симпозиума «Теория и методы построения систем программирования», Киев-Алушта*. 1972. С. 31–42
9. G. Delzanno *Automatic verification of cache coherence protocols via infinite-state constraint-based model checking*. Электронный ресурс. http://www.disi.unige.it/person/DelzannoG/protocol.html
10. G. Delzanno Automatic verification of parameterized cache coherence protocols // E. Allen Emerson, A. Prasad Sistla (eds.), *Computer Aided Verification. Proc. 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000. LNCS* 1855, 2000. P. 53–68
11. G. Geeraerts, J.-F. Raskin, L. van Begin. Expand, Enlarge and Check: New algorithms for the coverability problem of WSTS // *Journal of Computer and System Sciences*, 72(1):180–203, 2006
12. And.V. Klimov An approach to supercompilation for object-oriented languages: the Java Supercompiler case study // *Proceedings of the First International Workshop on Metacomputation in Russia (July 2–5, 2008, Pereslavl-Zalessky, Russia)*. Pereslavl-Zalessky: Ailamazyan University of Pereslavl, 2008. P. 43–53. http://meta2008.pereslavl.ru/accepted-papers/paper-info-4.html
13. And.V. Klimov A Java supercompiler and its application to verification of cache-coherence protocols // *Proc. 7th International Andrei Ershov Memorial Conference "Perspectives of Systems Informatics", PSI 2009, Akademgorodok, Novosibirsk, Russia, June 5–19, 2009. Revised Papers. LNCS* 5947, 2010. P. 185-192
14. A.P. Lisitsa, A.P. Nemytykh *Experiments on verification via supercompilation*. Электронный ресурс. http://refal.botik.ru/protocols
15. A.P. Lisitsa, A.P. Nemytykh Reachability analysis in verification via supercompilation // *International Journal of Foundations of Computer Science* 19, N 4, 2008. P. 953–970
16. A.P. Nemytykh *Refal-5 and the Supercompiler SCP4 home page*. Электронный ресурс. http://www.botik.ru/pub/local/scp/refal5
17. V.F. Turchin The use of metasystem transition in theorem proving and program optimization // Proc. 7th Colloquium on Automata, Languages and Programming. LNCS 85, 1980. P. 645–657
18. V.F. Turchin The concept of a supercompiler // *ACM Transactions on Programming Languages and Systems* 8, No. 3, 1986. P. 292–325