*Модел. и анализ информ. систем.* Т. **18**, № **4** (2011) 106–117

УДК 519.681.3

# Простой алгоритм решения задачи покрытия для монотонных счетчиковых систем

Климов Анд. В.[1]

*Институт прикладной математики им. М.В. Келдыша РАН*

*e-mail: klimov@keldysh.ru*

*получена 17 октября 2011 года*

**Ключевые слова:** хорошо-структурированные системы переходов, счетчиковые системы, задача достижимости, задача покрытия, суперкомпиляция.

Предложен алгоритм решения задачи покрытия для монотонных счетчиковых систем. Разрешимость этой задачи хорошо известна, но данный алгоритм интересен своей простотой. Он возник из упрощения некоторой итеративной процедуры применения суперкомпилятора (специализатора программ, основанного на методе суперкомпиляции В.Ф. Турчина) к программе, кодирующей счетчиковую систему и начальное и целевое множества состояний, и из доказательства, что при определенных условиях эта процедура завершается и решает задачу покрытия.

## 1. Введение

*Счетчиковые системы* — популярное средство моделирования параллельных систем, описываемых семействами взаимодействующих однотипных конечных автоматов. Они используются, например, для описания моделей устройств кэш-когерентного доступа к памяти, протоколов взаимодействия параллельных процессов и других параметризованных моделей. Широко известные сети Петри — частный случай счетчиковых систем.

Счетчиковые системы возникают в результате *счетчиковой абстракции*, идея которой состоит в следующем: состояние системы из $n$ одинаковых конечных автоматов с $k$ состояниями отображается в кортеж из $k$ целых неотрицательных чисел $\langle x_1, ..., x_k \rangle$, где $x_i$ — число автоматов в $i$-м состоянии. Возможные переходы в системе из конечных автоматов моделируются соответствующими правилами изменения значений $x_i$. При таком абстрагировании теряется часть информации о поведении исходной системы (происходит аппроксимация множества достижимых состояний сверху), но удается легче решать *задачу достижимости* для практических систем.

[1]Работа поддержана грантом РФФИ № 09-01-00834-а.

Важно, что задача решается сразу для произвольного числа конечных автоматов, то есть число автоматов $n$ является неизвестным параметром. За последние пару десятилетий появилось много программных инструментов для верификации счетчиковых систем, но поскольку в общем случае эта задача алгоритмически неразрешима, продолжается поиск новых алгоритмов для различных классов систем.

Еще в 1980 году В.Ф. Турчин предложил подход [17] к машинному построению доказательств теорем некоторого класса с помощью разработанного им метода преобразования программ — *суперкомпиляции* [18, 19, 20]. Много лет спустя его первым практическим воплощением стала работа А.П. Лисицы и А.П. Немытых, в которой они успешно применили [15, 14, 16] суперкомпилятор SCP4 [2, 18] для языка Рефал для верификации коллекции из около двух десятков протоколов кэш-когерентного доступа к памяти и других моделей, являющихся счетчиковыми системами. Эти эксперименты были затем воспроизведены [9, 10] на суперкомпиляторе JScp [8, 12] языка Java и на других суперкомпиляторах. (Для применения суперкомпилятора счетчиковая система кодируется определенным образом на входном языке суперкомпилятора: на Рефале для суперкомпилятора SCP4 и на языке Java для суперкомпилятора JScp.)

Эти результаты интересны тем, что использовались универсальные системы преобразования программ, предназначенные для их оптимизации, а не ориентированные изначально на анализ счетчиковых систем, как в множестве других работ.

Возник интересный теоретический вопрос: почему так успешно суперкомпилятор справился с задачей верификации данной коллекции протоколов? Ведь в общем случае эта задача алгоритмически неразрешима. Можно ли формально описать некоторый достаточно широкий класс счетчиковых систем, для которого можно доказать, что суперкомпилятор всегда разрешает данную задачу верификации?

В этой статье объясняется, что это действительно так для *монотонных счетчиковых систем*, когда решается задача (не)достижимости множества состояний, замкнутого кверху по стандартному покомпонентному частичному порядку кортежей целых чисел.

Разрешимость этой задачи для вполне структурированных систем переходов (well-structured transition systems), к которым принадлежат счетчиковые системы, хорошо известна. Основные алгоритмы анализа систем переходов делятся на два класса: прямого анализа (forward analysis) и обратного анализа (backward analysis) — по направлению распространения информации о множествах состояний: от начальных состояний вперед по шагам переходов или от конечных (целевых) состояний назад по шагам переходов. Применительно к простейшему классу систем переходов — обыкновенным сетям Петри — история началась с алгоритма прямого анализа в классической работе [7], но первый общий алгоритм решения задачи покрытия для вполне структурированных систем переходов был обратным [3, 1]. Затем появились алгоритмы прямого анализа и недавно была предложена общая схема для некоторого класса прямых алгоритмов, названная EEC (Expand, Enlarge and Check) [5, 6], для которых построено общее доказательство завершаемости и решения задачи покрытия для вполне структурированных систем переходов применительно ко всем алгоритмам, вкладывающимся в эту схему. В этой классификации суперкомпиляция — метод прямого анализа и преобразования программ. Поэтому неудивительно, что

суперкомпиляция похожа на другие методы прямого анализа и показывает близкие результаты. В работе [11] описаны два алгоритма суперкомпиляции для счетчиковых систем, выведенных из двух классических вариантов суперкомпиляции (так называемые «с обобщением сверху» и «с обобщением снизу») путем их специализации для счетчиковых систем. Оказалось, что для решения данной задачи верификации их можно еще больше упростить. В этой работе приводится алгоритм прямого анализа, полученный упрощением алгоритма суперкомпиляции «с обобщением снизу» вместе с доказательством его корректности.

При проведении экспериментов [14], сталкиваясь с ситуацией, когда суперкомпилятор не справляется с верификацией модели протокола при стандартных режимах суперкомпиляции, А.П. Лисица и А.П. Немытых обнаружили нетривиальный факт, что успехов можно добиться, запрещая *обобщение*[2] небольших целых неотрицательных чисел $i < l$, где $l$ — параметр, заданный пользователем. Для их экспериментов было достаточно использования двух значений $l = 0$ (нет ограничений на обобщение) и $l = 1$ (никогда не обобщается число 0).

Результаты данной статьи показывают, что в общем случае может понадобиться произвольное значение ограничения $l$. Более того, оказалось, что запрет на обобщение целых чисел, меньших $l$, — это единственное, что требуется для завершаемости процедуры и решения задачи покрытия для монотонных счетчиковых систем, и можно использовать самое простое правило обобщения, согласующееся с этим ограничением. На основе этого получаем следующую основную идею алгоритма: итеративно для $l = 1, 2, 3, \ldots$ выполняем прямой анализ счетчиковой системы, обобщая целые числа, большие или равные $l$, и на каждой итерации проверяя, можно ли по результату анализа сделать вывод о покрытии или непокрытии. Мы докажем, что существует такое $l$, при котором (если не раньше) этот цикл заведомо завершится и будет дан ответ.

Данная статья имеет следующую структуру. В разделе 2 напомним используемые понятия счетчиковых систем и введем понятия суперкомпиляции применительно к счетчиковым системам. В разделе 3 приводится алгоритм решения задачи покрытия. В разделе 4 доказывается его корректность.

## 2. Основные понятия и определения

**Системы переходов и счетчиковые системы.** *Система переходов* $\mathcal{S}$ представляет собой пару $\langle S, \Rightarrow \rangle$, где $S$ — множество состояний (быть может, бесконечное), $\Rightarrow \subseteq S \times S$ — отношение перехода между состояниями системы.

Системы переходов являются недетерминированными, то есть из текущего состояния возможен переход в несколько следующих состояний. Для эффективной вычислимости мы полагаем, что множество следующих состояний конечно.

Запись $s_1 \Rightarrow s_2$ означает, что из состояния $s_1$ возможен переход в состояние $s_2$ за один шаг. Запись $s_1 \stackrel{*}{\Rightarrow} s_2$ означает, что состояние $s_2$ *достижимо* из состояния $s_1$ за ноль или любое число шагов.

[2] В терминах теории счетчиковых систем эта фраза звучит так: «запрещая *ускорение* небольших целых неотрицательных чисел $i < l$, то есть их замену на специальное значение $\omega$».

Введем следующие обозначения:

$\mathsf{Post}(\mathcal{S}, s) = \{s' \mid s \Rightarrow s'\}$ — множество следующих состояний для состояния $s$.

$\mathsf{Post}^*(\mathcal{S}, s) = \{s' \mid s \stackrel{*}{\Rightarrow} s'\}$ — множество состояний, достижимых из состояния $s$.

$\mathsf{Reach}(\mathcal{S}, I) = \bigcup_{s \in I} \mathsf{Post}^*(\mathcal{S}, s)$ — множество состояний, достижимых из множества состояний $I$.

Назовем систему переходов $\langle S, \Rightarrow \rangle$ (*квази-, частично*) *упорядоченной*, если на множестве ее состояний $S$ определен некоторый (квази-,[3] частичный[4]) порядок $\preccurlyeq$.

Если $X$ — квази-упорядоченное множество, $\downarrow X$ означает $\{y \mid \exists x \in X \colon x \succcurlyeq y\}$, замыкание книзу множества $X$; $\uparrow X$ означает $\{y \mid \exists x \in X \colon x \preccurlyeq y\}$, замыкание кверху множества $X$.

*Задача покрытия* для квази-упорядоченной системы переходов $\mathcal{S}$, начального множества $I$ и замкнутого кверху целевого множества $U$ отвечает на вопрос: достижимо ли $U$ из $I$, то есть $\exists s \in I, s' \in U \colon s \stackrel{*}{\Rightarrow} s'$? Другими словами, не требуя, чтобы целевое множество $U$ было замкнуто кверху: достижимо ли из $I$ состояние $r$, *покрывающее* некоторое состояние из $U$, то есть $\downarrow\{r\} \cap U \neq \emptyset$?

Назовем *покрывающим множеством состояний* (covering set) квази-упорядоченной системы переходов $\mathcal{S}$ по отношению к начальному множеству $I$ множество состояний $\mathsf{Cover}(\mathcal{S}, I) = \downarrow\mathsf{Reach}(\mathcal{S}, I)$ — замыкание книзу множества состояний, достижимых из $I$.

Если бы мы могли определить покрывающее множество, то задача покрытия была бы решена: некоторое состояние множества $U$ покрывается, если $\mathsf{Cover}(\mathcal{S}, I) \cap U \neq \emptyset$, и не покрывается, если $\mathsf{Cover}(\mathcal{S}, I) \cap U = \emptyset$. Однако для достаточно богатых классов систем переходов, включая монотонные счетчиковые системы, задача определения покрывающего множества алгоритмически неразрешима.

Система переходов $\langle S, \Rightarrow \rangle$ с квазипорядком $\preccurlyeq \subseteq S \times S$ называется *монотонной*, если для любых $s_1, s_2, s_3 \in S$ таких, что $s_1 \Rightarrow s_2$ и $s_1 \preccurlyeq s_3$, существует $s_4 \in S$ такой, что $s_3 \stackrel{*}{\Rightarrow} s_4$ и $s_2 \preccurlyeq s_4$.

*$k$-мерной счетчиковой системой* $\mathcal{S}$ назовем систему переходов $\langle S, \Rightarrow \rangle$ с множеством состояний $S = \mathbb{N}^k$ — кортежей из $k$ целых неотрицательных чисел. Счетчиковым системам присущ естественный частичный порядок покомпонентного сравнения $\preccurlyeq$:

$$s_1 \preccurlyeq s_2 \quad \text{тогда и только тогда, когда} \quad \forall i \in [1, k] \colon\ s_1(i) \leq s_2(i).$$

**Понятие конфигурации.** Суперкомпилятор, как и все алгоритмы прямого анализа, строит потенциально бесконечное дерево всевозможных путей переходов от начального множества состояний. При этом шаги переходов выполняются не над конкретными состояниями, а над представлениями множеств состояний, называемыми *конфигурациями*[5]. Язык конфигураций и множество допустимых конфигураций выбираются в зависимости от решаемой задачи.

[3] *Квазипорядок* (*предпорядок*) — рефлексивное и транзитивное отношение.

[4] *Частичный порядок* — антисимметричный квазипорядок.

[5] В этой статье мы используем терминологию теории суперкомпиляции, которая в этом месте входит в противоречие с терминологией сетей Петри и счетчиковых систем, где слово «конфигурация» означает наше «состояние», а наша «конфигурация» соответствует понятию «$\omega$-конфигурация» теории счетчиковых систем.

Применительно к счетчиковым системам и нашей задаче достаточно рассматривать конфигурации в виде кортежей целых неотрицательных чисел и символов $\omega$[6], обозначающих все множество $\mathbb{N}$. Таким образом, имеем множество конфигураций $\mathcal{C} = (\mathbb{N} \cup \{\omega\})^k$. Конфигурация $c = \langle x_1, \ldots, x_k \rangle$ описывает множество состояний $[\![c]\!] = X_1 \times \cdots \times X_k$, где $X_i = \{x_i\}$, если $x_i \in \mathbb{N}$, и $X_i = \mathbb{N}$, если $x_i = \omega$. Здесь $\times$ — декартово произведение.

Мы также будем использовать обозначение $[\![\cdot]\!]$ для множеств конфигураций, понимая под $[\![C]\!]$ объединение множеств, представленных конфигурациями из множества конфигураций $C$: $[\![C]\!] = \bigcup_{c \in C} [\![c]\!]$.

Конфигурации можно сравнивать на принадлежность друг другу как множества состояний, например, конфигурация $\langle \omega, 0, 1, 0 \rangle$ изображает подмножество множества, изображенного конфигурацией $\langle \omega, 0, \omega, 0 \rangle$.

Множество начальных состояний $I$ и целевое множество $U$, как правило, не изобразимы конфигурациями. Во всех практических моделях, с которыми мы проводили эксперименты, множество $U$ задано условиями вида $x_i \geq a_i$, где $a_i$ — константа, и тем самым замкнуто кверху. Нам достаточно проверять пустоту пересечения $U$ с конфигурациями.

Начальные множества $I$ задаются условиями вида $x_i \geq a_i$ или $x_i = a_i$. Для разрешения задачи достижимости замкнутого кверху множества $U$ достаточно аппроксимировать такое множество $I$ конфигурацией $c_0$, описывающей минимальное надмножество $I$, представимое конфигурацией, то есть использовать символ $\omega$ для компонент $x_i$, заданных неравенством $x_i \geq a_i$. Символ $\omega$ представляет множество $x_i \geq 0$, то есть при $a_i \neq 0$ «захватываются» лишние состояния, меньшие состояний из $I$. Это допустимо для решения задачи покрытия для монотонных систем.

**Покрывающее множество конфигураций.** Назовем *покрывающим множеством конфигураций* (coverability set) конечное множество конфигураций $C$, представляющее покрывающее множество состояний (covering set) следующим образом: $\downarrow [\![C]\!] = \mathsf{Cover}(\mathcal{S}, I)$. Отметим, что покрывающее множество *состояний* одно, но имеется много его представлений в виде покрывающих множеств *конфигураций*[7].

Приведенный ниже алгоритм вычисляет аппроксимации покрывающего множества конфигураций сверху и снизу. Мы докажем, что существует значение параметра $l$, когда в качестве аппроксимации сверху он выдаст в точности покрывающее множество конфигураций, если задача покрытия целевого множества $U$ не будет решена раньше при меньших значениях $l$. Однако какое именно $l$ соответствует покрывающему множеству, алгоритмически неразрешимо.

**Прогонка.** Шаги выполнения правил перехода в общем виде над конфигурациями называются в теории суперкомпиляции *прогонкой* (driving). Функция прогонки Drive является обобщением функции перехода Post с состояний на конфигурации.

Применительно к счетчиковым системам с правилами перехода указанного выше вида и множеству конфигураций $\mathcal{C} = (\mathbb{N} \cup \{\omega\})^k$ функция Drive определяется путем

[6]Такое обозначение используется в теории сетей Петри и счетчиковых систем.

[7]Мы будем говорить просто «покрывающее множество», когда будем надеяться, что из контекста ясно, о каком из них идет речь — множестве состояний или множестве конфигураций.

обобщения операций $+$ и $\geq$ для чисел на расширенную область $\mathbb{N} \cup \{\omega\}$:

$$\forall a \in \mathbb{N}\colon \omega > a,\ \omega + a = \omega - a = \omega + \omega = \omega.$$

Это определение удовлетворяет следующим свойствам, которые в общем случае должны выполняться для функции $\mathsf{Drive}$ (здесь $s \in S$, $c \in \mathcal{C}$):

1. $\mathsf{Drive}(\mathcal{S}, s) = \mathsf{Post}(\mathcal{S}, s)$ — конфигурация, представляющая одно состояние, переходит в множество конфигураций, совпадающих по представлению с его следующими состояниями;
2. $[\![\mathsf{Drive}(\mathcal{S}, c)]\!] \supseteq \bigcup\{\mathsf{Post}(\mathcal{S}, s) \mid s \in [\![c]\!]\}$ — конфигурации, выданные функцией $\mathsf{Drive}$, аппроксимируют сверху множество следующих состояний. Аппроксимации сверху достаточно для применения суперкомпиляции к оптимизации программ, но для верификации счетчиковых систем на результат $\mathsf{Drive}$ накладывается еще одно условие:
3. $[\![\mathsf{Drive}(\mathcal{S}, c)]\!] \subseteq \downarrow\bigcup\{\mathsf{Post}(\mathcal{S}, s) \mid s \in [\![c]\!]\}$ — для решения задачи покрытия достаточно потребовать, чтобы конфигурации, выданные функцией $\mathsf{Drive}$, были подмножествами замыкания книзу множества следующих состояний.

**Обобщение конфигураций.** Когда множество конфигураций, представленное конфигурацией $c$, является подмножеством множества конфигураций, представленного конфигурацией $g$, $[\![c]\!] \subseteq [\![g]\!]$, говорят, что вторая конфигурация *более общая*, чем первая, или что вторая конфигурация является *обобщением* первой.

Через $\sqsubseteq \in \mathcal{C} \times \mathcal{C}$ обозначим *отношение обобщения*: $c \sqsubseteq g$ эквивалентно $[\![c]\!] \subseteq [\![g]\!]$, $c \sqsubset g$ эквивалентно $[\![c]\!] \subsetneq [\![g]\!]$.

Нам понадобится следующая функция обобщения с целочисленным параметром $l$, которая заменяет на $\omega$ компоненты конфигурации $c$, большие или равные $l$:

$$\mathsf{Generalize}_l(c) = g \text{ такое, что } \forall i \in [1, k]\colon g(i) = c(i), \text{ если } c(i) < l, \text{ иначе } \omega.$$

Такая функция выдает правильное обобщение: $\forall c \in \mathcal{C}, l \in \mathbb{N}\colon c \sqsubseteq \mathsf{Generalize}_l(c)$.

Параметр $l$ играет важную роль, поэтому мы его записываем в особой позиции в виде индекса у имен функций, которые принимают его в качестве аргумента.

# 3. Алгоритм решения задачи покрытия

В этом разделе представлен алгоритм решения задачи покрытия для монотонных счетчиковых систем. Он состоит из головной функции $\mathsf{Reachable}(\mathcal{S}, I)$, которая содержит цикл по параметру $l$ и выдает ответ, и функций $\mathsf{Under}_l(\mathcal{S}, I)$ и $\mathsf{Over}_l(\mathcal{S}, I)$, вычисляющих аппроксимации сверху и снизу покрывающего множества, вызывая центральную функцию $\mathsf{Approximate}_l(over, \mathcal{S}, I)$ — упрощенную суперкомпиляцию.

**Головная функция** $\mathsf{Reachable}(\mathcal{S}, I, U)$ берет в качестве аргументов счетчиковую систему $\mathcal{S}$, начальную конфигурацию $I$ и замкнутое кверху целевое множество $U$ и итеративно применяет функции $\mathsf{Under}_l(\mathcal{S}, I)$ и $\mathsf{Over}_l(\mathcal{S}, I)$, выдающие аппроксимации снизу и сверху покрывающего множества, с возрастающими значениями параметра

**Алгоритм 1:** $\mathsf{Reachable}(\mathcal{S}, I, U)$: Решение задачи покрытия для монотонных счетчиковых систем.

**input** : $\mathcal{S}$ — монотонная счетчиковая система
$I$ — начальная конфигурация
$U$ — замкнутое кверху целевое множество
**result**: Достижимо ли $U$ из $I$?

```
Reachable(S, I, U)
    for l = 1, 2, 3, ... do
        if [[Under_l(S, I)]] ∩ U ≠ ∅ then
            return 'Достижимо'
        if [[Over_l(S, I)]] ∩ U = ∅ then
            return 'Недостижимо'
```

**Алгоритм 2:** $\mathsf{Approximate}_l(\mathcal{S}, I)$, $\mathsf{Under}_l(\mathcal{S}, I)$, $\mathsf{Over}_l(\mathcal{S}, I)$: Построение аппроксимаций снизу и сверху покрывающего множества счетчиковой системы.

**input** : $\mathcal{S}$ — счетчиковая система
$I$ — начальная конфигурация
$l$ — целочисленный параметр обобщения
$over$ = **true** — вычислять аппроксимацию сверху, **false** — снизу
**result**: $R$ — аппроксимация покрывающего множества сверху или снизу

```
Approximate_l(over, S, I)
    R ← {I}
    T ← {I}
    while T ≠ ∅ do
        выбрать любую конфигурацию из c ∈ T
        T ← T \ {c}
        foreach c' ∈ Drive(S, c) do
            g ← Generalize_l(c')
            if ∄c̄ ∈ R: c̄ ⊒ c' ∧ (over ∨ g = c') then
                R ← R ∪ {g} \ {c̄ ∈ R | c̄ ⊏ g}
                T ← T ∪ {g} \ {c̄ ∈ T | c̄ ⊏ g}
    return R

Under_l(S, I) = Approximate_l(false, S, I)
Over_l(S, I) = Approximate_l(true, S, I)
```

обобщения $l = 1, 2, 3, \ldots$ Цикл завершается ответом ‘Достижимо’ или ‘Недостижимо’, когда аппроксимация снизу пересекается с $U$ или аппроксимация сверху не пересекается с $U$ соответственно[8].

[8]Этот алгоритм близок к алгоритмам 4.1 и 5.2 в [5, с. 122 и с. 141] и проще их.

**Функция** $\mathsf{Approximate}_l(\boldsymbol{over}, \mathcal{S}, I)$ вызывается с булевским параметром *over*, указывающим, какую аппроксимацию надо вычислить — сверху или снизу.

Множество конфигураций, являющееся аппроксимацией покрывающего множества, накапливается в переменной $R$ («residual set», «остаточное множество»), начиная с начальной конфигурации $I$. Конфигурации, подлежащие обработке, берутся из множестве $T$ («to treat», «обработать»). Алгоритм завершается, когда $T = \emptyset$.

На каждом шаге из множества $T$ берется произвольная необработанная конфигурация $c$ и выполняется *шаг прогонки*: функция $\mathsf{Drive}$ вычисляет множество следующих конфигураций и каждая из них $c'$ обрабатывается следующим образом:

1. Проверяется, не покрыта ли конфигурация $c'$ одной из пройденных конфигураций из $R$. Если $c'$ покрыта некоторой конфигурацией $\bar{c} \in R$: $\bar{c} \sqsupseteq c'$, она больше не обрабатывается, поскольку все ее потомки покрыты потомками $\bar{c}$.
2. Когда вычисляется аппроксимация снизу, проверяется, не обобщается ли конфигурация $c'$ на текущем шаге, сравнением с $g = \mathsf{Generalize}_l(c')$. Если обобщается, обработка этой ветви прекращается, чтобы в $R$ накапливались только конфигурации, порожденные без обобщения.
3. Конфигурация $g = \mathsf{Generalize}_l(c')$, совпадающая с $c'$ или являющаяся ее обобщением, добавляется в множества $R$ и $T$. Кроме того, из этих множеств удаляются старые конфигурации, покрытые новой, за ненадобностью.

# 4. Корректность алгоритма

Если алгоритм завершается, то достаточно очевидно, что ответ правильный, поскольку можно показать, что для любой монотонной счетчиковой системы $\mathcal{S}$, любой начальной конфигурации $I$ и всех $l \in \mathbb{N}$:

$$[\![\mathsf{Under}_l(\mathcal{S}, I)]\!] \subseteq \mathsf{Cover}(\mathcal{S}, I) \subseteq {\downarrow}[\![\mathsf{Over}_l(\mathcal{S}, I)]\!].$$

**Завершаемость алгоритма** $\mathsf{Approximate}_l(\boldsymbol{over}, \mathcal{S}, I)$ следует из ограниченности множества конфигураций, которые могут быть выданы функцией $\mathsf{Generalize}_l(c)$ при фиксированном $l$: множества $R$ и $T$ являются подмножествами конечного множества $\{0, \ldots, l-1, \omega\}^k$. На множестве конфигураций можно ввести отношение частичного порядка, по которому значения $R$ не будут убывать. В тех же случаях, когда $R$ не изменяется, убывает $T$.

**Завершаемость алгоритма** $\mathsf{Reachable}(\mathcal{S}, I, U)$ доказывается для двух случаев.

*Случай 1 (Достижимо).* Если существует путь из начального множества $I$ в целевое множество $U$, при некотором $l$ (равном максимуму компонент состояний на этом пути плюс 1) весь этот путь будет пройден алгоритмом без обобщений и его концевая конфигурация пересечется с $U$. Учитывая, что прогонка может обобщать конфигурации только вниз, получаем следующее утверждение.

**Теорема 1** (Разрешение достижимости)**.** *Для любой монотонной счетчиковой системы $\mathcal{S}$, начальной конфигурации $I$ и замкнутого кверху множества $U$ таких,*

*что* $\mathsf{Reach}(\mathcal{S}, I) \cap U \neq \emptyset$ *существует* $l \in \mathbb{N}$*, при котором алгоритм* Reachable *завершается и выдает ответ 'Достижимо'.*

*Случай 2 (Недостижимо).* Случай $U = \emptyset$ тривиален. Следовательно, будем считать, что $U \neq \emptyset$.

Требуется доказать, что существует $l \in \mathbb{N}$ такое, что аппроксимация сверху окажется покрывающим множеством: $\downarrow[\![\mathsf{Over}_l(\mathcal{S}, I)]\!] = \mathsf{Cover}(\mathcal{S}, I)$. Таким значениям $l$ можно дать верхнюю оценку благодаря следующей лемме.

**Лемма 1.** *Для любого замкнутого книзу множества* $D = \downarrow D \subseteq S = \mathbb{N}^k$*, где используется покомпонентный частичный порядок* $\preccurlyeq$*, существует* $l \in \mathbb{N}$ *такое, что для любых* $c, g \in \mathcal{C} = (\mathbb{N} \cup \{\omega\})^k$:

$$[\![c]\!] \subseteq D \;\; и \;\; c \sqsubseteq_l g \;\; влечет \;\; [\![g]\!] \subseteq D.$$

*Доказательство.* Рассмотрим нетривиальный случай, когда $c \neq g$, $c \sqsubset_l g$.

Пусть $U = \overline{D}$ — дополнение к замкнутому книзу множеству $D$. Множество $U$ замкнуто кверху. Поскольку частичный порядок $\preccurlyeq$ является вполне-квази-порядком[9], существует конечное множество $\min(U)$ минимальных элементов $U$ (*генератор* $U$) такое, что $\uparrow \min(U) = U$. Пусть $m$ равно максимуму координат элементов $\min(U)$:

$$m = \max_{s \in \min(U)} \max_{i \in [1,k]} s(i).$$

Можно показать, что $l = m$ удовлетворяет условию леммы. □

**Следствие 1.** *Для любой счетчиковой системы* $\mathcal{S}$ *и начальной конфигурации* $I$ *существует* $l \in \mathbb{N}$ *такой, что для всех конфигураций* $c, g \in \mathcal{C} = (\mathbb{N} \cup \{\omega\})^k$:

$$[\![c]\!] \subseteq \mathsf{Cover}(\mathcal{S}, I) \;\; и \;\; c \sqsubseteq_l g \;\; влечет \;\; [\![g]\!] \subseteq \mathsf{Cover}(\mathcal{S}, I).$$

**Лемма 2.** *Для любой монотонной системы переходов* $\mathcal{S}$*, любой начальной конфигурации* $I$ *и любой конфигурации* $c \in \mathcal{C}$:

$$[\![c]\!] \subseteq \mathsf{Cover}(\mathcal{S}, I) \;\; влечет \;\; \mathsf{Cover}(\mathcal{S}, c) \subseteq \mathsf{Cover}(\mathcal{S}, I).$$

*Доказательство.* Непосредственно следует из монотонности. Это утверждение известно в литературе по системам переходов, например, лемма 2.2 в [5, с. 182]. □

**Следствие 2.** *Для любой монотонной счетчиковой системы* $\mathcal{S}$ *и начальной конфигурации* $I$ *существует* $l \in \mathbb{N}$ *такой, что для всех конфигураций* $c, g \in \mathcal{C} = (\mathbb{N} \cup \{\omega\})^k$:

$$[\![c]\!] \subseteq \mathsf{Cover}(\mathcal{S}, I) \;\; и \;\; c \sqsubseteq_l g \;\; влечет \;\; \mathsf{Cover}(\mathcal{S}, g) \subseteq \mathsf{Cover}(\mathcal{S}, I).$$

**Лемма 3.** *Для любой монотонной счетчиковой системы* $\mathcal{S}$ *и начальной конфигурации* $I$ *существует* $l \in \mathbb{N}$ *такой, что*

$$\downarrow[\![\mathsf{Over}_l(\mathcal{S}, I)]\!] = \mathsf{Cover}(\mathcal{S}, I).$$

[9] Квазипорядок $\preccurlyeq$ называется *вполне-квази-порядком*, если в любой бесконечной последовательности $\{x_i\}_{i \in \mathbb{N}}$ найдутся два члена таких, что $x_i \preccurlyeq x_j$, $i < j$.

*Доказательство.* Возьмем любое значение $l$, удовлетворяющее условию леммы 1 и, следовательно, следствиям 1 и 2. В алгоритме Approximate каждое обобщение, ограниченное этим параметром $l$, расширяет замыкание книзу множества состояний, достижимых из конфигураций текущего множества $R$, не больше, чем до покрывающего множества. Следовательно, аппроксимация сверху, полученная при этом $l$, является покрывающим множеством. □

**Теорема 2** (Разрешение недостижимости)**.** *Для любой монотонной счетчиковой системы $\mathcal{S}$ с покомпонентным частичным порядком $\preccurlyeq$, для любой начальной конфигурации $I$ и замкнутого кверху целевого множества $U$ такого, что* $\mathsf{Reach}(\mathcal{S}, I) \cap U = \emptyset$*, существует $l \in \mathbb{N}$ такой, что алгоритм* Reachable *завершается и выдает ответ 'Недостижимо'.*

Отметим, что мы не знаем заранее значение параметра $l \leq m$, при котором алгоритм завершит свою работу. Мы также не можем вычислить его верхнюю оценку $m$, поскольку задача нахождения покрывающего множества даже для монотонных счетчиковых систем алгоритмически неразрешима (это следует из результатов статьи [4]). Тем не менее, мы знаем, что такое $l$ существует для любой монотонной счетчиковой системы с непустым множеством недостижимых состояний. Поэтому разрешимость задачи покрытия для монотонных систем переходов [3] не противоречит неразрешимости нахождения покрывающего множества.

# 5. Заключение

Предложенный алгоритм является, насколько известно автору, простейшим среди решающих проблему покрытия для монотонных счетчиковых систем. Он вкладывается в известную схему для класса алгоритмов, решающих задачу покрытия, называемую EEC (Expand, Enlarge and Check) [5, 6]. Можно было бы дать доказательство корректности алгоритма сведением к ECC, однако приведенное здесь прямое доказательство намного проще.

Эксперименты [9, 10, 14, 16] показывают, что этот и другие алгоритмы (в частности, [11]), основанные на суперкомпиляции, часто решают задачу покрытия и для немонотонных систем. Приведенное доказательство позволяет легче анализировать причины, почему эти алгоритмы завершаются. Например, можно предположить, что алгоритм $\mathsf{Over}_l(\mathcal{S}, I)$ завершается для немонотонных счетчиковых систем, когда существует аппроксимация сверху покрывающего множества, замкнутая относительно операции прогонки Drive и непересекающаяся с целевым множеством $U$.

Описанный алгоритм основан на упрощенной суперкомпиляции, и его можно рассматривать как пример *многорезультатной суперкомпиляции*, предложенной И.Г. Ключниковым и С.А. Романенко [13]. Они показали плодотворность многорезультатной суперкомпиляции для доказательства эквивалентности программ и в двухуровневой суперкомпиляции, а в данной работе демонстрируется задача, не решаемая однократной суперкомпиляцией, но решаемая перебором потенциально бесконечного множества результатов суперкомпиляции.

## Список литературы


1. Кузьмин Е.В., Соколов В.А. *Вполне структурированные системы помеченных переходов*. М.: Физматлит, 2005.
2. Немытых А.П. *Суперкомпилятор SCP4: общая структура*. М.: Editorial URSS, 2007.
3. Parosh Aziz Abdulla, Kārlis Čerāns, Bengt Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems // *The 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, July 27–30, 1996*. IEEE Computer Society, 1996. P. 313–321.
4. Catherine Dufourd, Alain Finkel, and Philippe Schnoebelen. Reset nets between decidability and undecidability //*Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings*. Springer, 1998. *LNCS*. Vol. 1443. P. 103–115.
5. Gilles Geeraerts. *Coverability and Expressiveness Properties of Well-Structured Transition Systems*. PhD thesis, Université Libre de Bruxelles, Belgique, May 2007. `http://www.ulb.ac.be/di/verif/ggeeraer/thesis.pdf`.
6. Gilles Geeraerts, Jean-François Raskin, and Laurent Van Begin. Expand, Enlarge and Check: New algorithms for the coverability problem of WSTS // *Journal of Computer and System Sciences*. 2006. 72(1). P. 180–203.
7. Richard M. Karp and Raymond E. Miller. Parallel program schemata // *J. Comput. Syst. Sci.* 1969. 3(2). P. 147–195.
8. Klimov Andrei V. An approach to supercompilation for object-oriented languages: the Java Supercompiler case study // *The First International Workshop on Metacomputation in Russia, Proceedings. Pereslavl-Zalessky, Russia, July 2–5, 2008*. Pereslavl-Zalessky: Ailamazyan University of Pereslavl, 2008. P. 43–53.
9. Klimov Andrei V. *JVer Project: Verification of Java programs by Java Supercompiler*, 2008. Электронный ресурс: `http://pat.keldysh.ru/jver/`.
10. Klimov Andrei V. A Java Supercompiler and its application to verification of cache-coherence protocols // *Perspectives of Systems Informatics, 7th International Andrei Ershov Memorial Conference, PSI 2009, Novosibirsk, Russia, June 15-19, 2009. Revised Papers*. Springer, 2010. *LNCS*. Vol. 5947. P. 185–192.
11. Klimov Andrei V. Solving coverability problem for monotonic counter systems by supercompilation // *The 8th Andrei Ershov Informatics Conference, PSI 2011, Akademgorodok, Novosibirsk, Russia, June 27 – July 01, 2011*. Novosibirsk: Ershov Institute of Informatics Systems, 2011. P. 92–103.
12. Klimov Andrei V., Klimov Arkady V., Shvorin Artem B. *The Java Supercompiler Project*. Электронный ресурс: `http://www.supercompilers.ru`.
13. Klyuchnikov I., Romanenko S. Multi-result supercompilation as branching growth of the penultimate level in metasystem transitions. // *The 8th Andrei Ershov Informatics Conference, PSI 2011, Novosibirsk, Russia, June 27 – July 01, 2011*. Novosibirsk: Ershov Institute of Informatics Systems, 2011. P. 104–115.

14. Lisitsa Alexei P., Nemytykh Andrei P. *Experiments on verification via supercompilation*, 2007. Электронный ресурс: `http://refal.botik.ru/protocols/`.

15. Lisitsa Alexei P., Nemytykh Andrei P. Verification as a parameterized testing (experiments with the SCP4 supercompiler) //*Programming and Computer Software*. 2007. 33(1). P. 14–23.

16. Lisitsa Alexei P., Nemytykh Andrei P. Reachability analysis in verification via supercompilation //*Int. J. Found. Comput. Sci.* 2008. 19(4). P. 953–969.

17. Turchin Valentin F. The use of metasystem transition in theorem proving and program optimization // *ICALP*. Springer, 1980. *LNCS*. Vol. 85. P. 645–657.

18. Turchin Valentin F. The concept of a supercompiler // *Transactions on Programming Languages and Systems*. 1986. 8(3). P. 292–325.

19. Turchin Valentin F. Metacomputation: Metasystem transitions plus supercompilation // *Dagstuhl Seminar on Partial Evaluation*. Springer, 1996. *LNCS*. Vol. 1110. P. 481–509.

20. Turchin Valentin F. Supercompilation: techniques and results // *Perspectives of System Informatics, Second International Andrei Ershov Memorial Conference, Akademgorodok, Novosibirsk, Russia, June 25-28, 1996. Proceedings*. Springer, 1996. *LNCS*. Vol. 1181. P. 227–248.

## A Simple Algorithm for Solving the Coverability Problem for Monotonic Counter Systems

Klimov And.V.

**Keywords:** well-structured transition systems, counter systems, reachability, coverability, supercompilation.

An algorithm for solving the coverability problem for monotonic counter systems is presented. The solvability of this problem is well-known, but the algorithm is interesting due to its simplicity. The algorithm has emerged as a simplification of a certain procedure of a supercompiler application (a program specializer based on V.F. Turchin's supercompilation) to a program encoding a monotonic counter system along with initial and target sets of states and from the proof that under some conditions the procedure terminates and solves the coverability problem.

**Сведения об авторе:**
**Климов Андрей Валентинович**,
Институт прикладной математики им. М.В. Келдыша РАН,
заведующий сектором методов анализа и преобразования программ.