Полиномиальный алгоритм проверки эквивалентности в модели программ с перестановочными и подавляемыми операторами

[1] *В.В. Подымов*
<valdus@yandex.ru>
[2] *В.А.Захаров*
<zakh@cs.msu.su>
[1] *Факультет ВМК, МГУ имени М.В. Ломоносова,*
119991 ГСП-1 Москва, Ленинские горы, 2-й учебный корпус
[2] *ИСП РАН, 109004, Россия, г. Москва, ул. А. Солженицына, дом 25*

Аннотация. В статье исследована задача проверки эквивалентности последовательных программ, некоторые операторы которых обладают свойствами перестановочности и подавления. Два оператора считаются перестановочными, если результат их последовательного выполнения не зависит от порядка, в котором эти операторы выполняются. Считается, что оператор b подавляет оператор a, если последовательное выполнение операторов a и b дает такой же результат, что и выполнение одного лишь оператора b. Разрешимость проблемы эквивалентности в исследуемой модели программ была установлена в 1971 г. А.А. Летичевским. Однако с тех пор вопрос о сложности проверки эквивалентности таких программ оставался открытым. Основной результат статьи – алгоритм, осуществляющий проверку эквивалентности программ с перестановочными и подавляемыми операторами за время, полиномиально зависящее от размеров анализируемых программ.

Ключевые слова: программа, перестановочные операторы, подавляемые операторы, эквивалентность программ, конечный автомат, разрешающий алгоритм, полиномиальная сложность.

1. Введение

На протяжении своего жизненного пути, от начальной разработки до заключительной модернизации, программа претерпевает разнообразные преобразования. В некоторых случаях, как, например, при компиляции или декомпиляции, преобразования программ проводятся автоматически. В других случаях, как, например, при модернизации программ, такие преобразования проводятся вручную. На каждом этапе, когда программа

подвергается изменению, необходимо следить за тем, чтобы применяемые преобразования не изменяли функций, вычисляемых программой, и не ухудшали вычислительные характеристики программы, такие как быстродействие, объем используемой памяти, размер и др.

Большим подспорьем здесь служат автоматические средства проверки эквивалентности и оптимизации программ. Оптимизирующие компиляторы снабжены эффективными процедурами, которые позволяют обнаруживать и устранять недостижимые и бесполезные фрагменты кода, оптимизировать структуру графов потока управления и линейных участков программ, экономить используемую память и пр.[1] Но все эти средства нуждаются в дальнейшем совершенствовании, поскольку во многих случаях они не способны провести глубокую оптимизацию даже простых фрагментов кода. Например, фрагмент программы π_1:

$$if\ P(z)\ then\ \{x = f(x); y = y + z\}\ else\ \{x = g(x); x = v + z\};$$

$$if\ !P(z)\ then\ \{y = y + z\}\ else\ \{x = v + z\}$$

эквивалентен линейному участку π_2 : $y = y + z; x = v + z$, однако современные компиляторы не способны провести оптимизирующие преобразования подобного вида. Поэтому разработка эффективных алгоритмов глубокого семантического анализа и оптимизации программ по-прежнему остается актуальной задачей теории программирования.

Для разработки таких алгоритмов выбирается и строится подходящая математическая модель программы таким образом, чтобы интересующая задача анализа или оптимизация программ сводилась некоторой общеизвестной математической задаче, наподобие задачи выполнимости логической формулы [2], вычисления хроматического числа графа [3] или минимизации автомата [4]. Алгоритмы решения этой задачи служат основой для построения процедуры решения соответствующей задачи системного программирования.

Этот подход был впервые предложен работах А.А. Ляпунова и Ю.И. Янова [5,6]. В них была введена и исследована первая математическая модель программ, получившая название схем Ляпунова-Янова. Используя систему эквивалентных преобразований, предложенную в [6] и усовершенствованную в [7], а также взаимосвязь схем Ляпунова-Янова с конечными автоматами, А.П. Ершов впервые применил оптимизирующие преобразования в трансляторе программ на языке Алгол [8], показав плодотворность такого подхода и придав тем самым мощный импульс развития теории схем программ.

Значительно более развитая алгебраическая модель программ, обобщающая схемы программ Ляпунова-Янова, была предложена в статье В.М. Глушкова и А.А. Летичевского [9]. В рамках этой модели можно, в частности, осуществлять эквивалентные преобразования фрагментов программы π_1 и π_2,

приведенных выше. В статье [9] был также предложен эффективный метод оптимизации программ, в котором задача минимизации программы (как по размеру, так и по быстродействию) сводится к задаче проверке эквивалентности схем программ. К сожалению, рекурсивный алгоритм проверки эквивалентности, предложенный в той же статье для некоторых классов моделей программ, имел экспоненциальную оценку сложность и был непригоден для применения на практике. Отсутствие быстрых и практичных алгоритмов проверки эквивалентности схем программ в моделях программ, учитывающих алгебраические свойства операторов, было главным препятствием дальнейшего внедрения результатов и методов теории схем программ в практику системного программирования [10].

Долгое время задача построения быстрых (полиномиальных по времени) алгоритмов проверки эквивалентности схем программ в алгебраических моделях программ не имела успешного решения. Первый результат в этом направлении был получен в статье [11]. Вскоре были предложены общие методы построения полиномиальных по времени алгоритмов проверки эквивалентности схем программ в моделях программ, учитывающих различные алгебраические свойства операторов [12,13]. Два свойства программных операторов оказались в центре внимания – перестановочность и подавляемость. Операторы a и b перестановочны, если результат вычисления программы не зависит от порядка выполнения этих операторов (т.е. $a;b = b;a$); оператор a подавляет оператор b, если результат выполнения оператора b несущественен, если вслед за ним выполняется оператор a (т.е. $a;b = b$). Интерес к этим алгебраическим свойствам программных операторов обусловлен двумя обстоятельствами.

Во-первых, отношения перестановочности и подавляемости операторов выявляются на основе простого синтаксического анализа операторов. Для каждого оператора a легко вычисляются два множества переменных: множество $Used(a)$, состоящее из всех тех переменных, значения которых используются при выполнении оператора, и множество $Mod(a)$, состоящее из всех тех переменных, значения которых изменяются в результате выполнения оператора. Нетрудно убедиться, что для перестановочности операторов a и b достаточно выполнимости равенств $Used(a) \cap Mod(b) = \emptyset$, $Used(b) \cap Mod(a) = \emptyset$, $Mod(a) \cap Mod(b) = \emptyset$, а для того чтобы оператор a подавлял оператор b, достаточно выполнимости равенств $Used(a) \cap Mod(b) = \emptyset$, $Mod(b) \subseteq Mod(a)$. На основании этих соотношений можно быстро проверить, что операторы $y = y + z$ и $x = v + z$ из примера, приведенного в начале этого раздела, перестановочны, и при этом оператор $x = v + z$ подавляет операторы $x = f(x)$ и $x = h(y)$.

Во-вторых, для многих оптимизирующих преобразований программ достаточно лишь привлечения отношений перестановочности и подавляемости операторов. Так, например, фрагменты программ, π_0 и π_1, приведенные в качестве примеров в начале этого раздела, эквивалентны в

алгебраической модели программ, в которой учитываются только два рассматриваемых свойства операторов.

Проблема эквивалентности программ в алгебраических моделях, учитывающих каждое из этих свойств по отдельности, уже была исследована ранее. В статье [13] описан алгоритм сложности $O(n^3)$, проверяющий эквивалентность программ с частично перестановочными операторами. Если в модели программ учитывается только отношение подавляемости операторов, то, как показано в работе [14], задача проверки эквивалентности принадлежит классу NL. Однако к комбинированным моделям программ общего вида, в которых учитываются оба эти свойства операторов, известные методы построения быстрых алгоритмов проверки эквивалентности оказались неприменимы (за исключением одного специального случая, исследованного в статье [15]).

Основной результат данной работы – описание полиномиального по времени алгоритма проверки эквивалентности произвольных программ с перестановочными и подавляемыми операторами. Статья состоит из 9 разделов. В начале статьи вводится модель пропозициональных последовательных программ. В разделе 2 приведены основные определения синтаксиса этой модели программ, а в разделе 3 определяется семантика этого класса абстрактных программ. Семантика программ определяется в алгебраических терминах на основе моноидов (полугрупп) с определяющими соотношениями (тождествами) перестановочности и подавления. В следующем разделе 4 исследованы наиболее важные для решения задачи проверки эквивалентности программ алгебраические свойства моноидов с тождествами перестановочности и подавления. В разделе 5 введен граф совместных вычислений – основная структура, при помощи которой проводится проверка эквивалентности программ. В этом же разделе доказана теорема 1 о необходимых и достаточных условиях эквивалентности программ; эти условия представлены в терминах устройства графа совместных вычислений анализируемых программ. В разделе 6 исследованы особенности применения тождеств подавления операторов $a;b = b$ в контексте перестановочности некоторых операторов. Показано, что эффект подавления операторов в операторных цепочках может быть описан при помощи конечных автоматов. Соответствующее утверждение (лемма 8) является ключевым в построении разрешающего алгоритма. В разделе 7 приводятся утверждения, обеспечивающие полиномиальный обход графа совместных вычислений. Существование такого обхода совместно с упомянутой теоремой 1 обеспечивает полиномиальную разрешимость проблемы эквивалентности программ в рассматриваемой модели вычислений. Описанию этого алгоритма и обоснованию его корректности посвящен раздел 8. В заключительном разделе 9 обсуждаются перспективы практического использования результата этой статьи для решения задач системного программирования, а также возможности дальнейших исследований, развивающих полученный результат.

Описание разрешающего алгоритма и Теорема 2 – основные результаты статьи, – являются заслугой первого из авторов статьи.

2. Основные определения

Далее считаем заданными конечные алфавиты операторов $\mathfrak{A}$ и логических условий $\mathfrak{C}$. Слова в алфавите операторов $\mathfrak{A}$ будем называть *термами*. Запись h^- будет обозначать зеркальный образ терма h.

Последовательной программой, или просто программой, будем называть систему $\pi = (L, l^{in}, l^{out}, T, B)$, состоящую из конечного множества состояний управления L, входа $l^{in} \in L$, выхода $l^{out} \in L$, функции переходов $T: (V \setminus lout \times \mathfrak{C} \to L$ и привязки операторов $B: L \to \mathfrak{A}$. Под размером π программы π будем понимать число ее состояний управления. Состояние управления будем называть завершаемым, если из него достижим выход программы.

Пара состояний управления l_1, l_2 программы π образуют шаг вычисления $l_1 \xrightarrow{\delta} l_2$, если $T(l_1, \delta) = l_2$. Путем в программе π назовем всякую последовательность шагов вычисления $pt = l_1 \xrightarrow{\delta_1} l_2 \xrightarrow{\delta_2} \dots$. Путь назовем полным, если он бесконечен или оканчивается в выходе. Путь, начинающийся во входе программы, назовем ее трассой. Полную трассу назовем вычислением. Записью $B(pt)$ обозначим терм $B(l_1)B(l_2)\dots$; для простоты выкладок полагаем $B(l^{in}) = B(l^{out}) = \lambda$, где λ — пустое слово.

Определим семантику программ, основываясь на детерминированных динамических шкалах и моделях [16]. *Шкала* $\mathcal{F} = (S, s^0, R)$ задает множество *состояний данных* S, *начальное состояние* $s^0 \in S$ и *интерпретации операторов* $R: S \times \mathfrak{A} \to S$. Введем следующее обобщение R^* функции R с множества $\mathfrak{A}$ на $\mathfrak{A}^*$: $R^*(s, \lambda) = s$, $R(s, ah) = R^*(R(s, a), h)$. Для краткости вместо $R^*(s^0, h)$ будем писать $[h]$.

Модель $\mathcal{M} = (\mathcal{F}, \xi)$ задается шкалой $\mathcal{F} = (S, s^0, R)$ и *оценкой логических условий* $\xi: S \to \mathfrak{C}$. Будем говорить, что *трасса реализуется в модели* $\mathcal{M}$, если для любого ее префикса $tr \xrightarrow{\delta} l$ верно равенство $\xi([tr]) = \delta$. Такую трассу для краткости будем называть *трассой в модели* $\mathcal{M}$. Результатом конечного вычисления cp в модели $\mathcal{M}$ будем называть состояние данных $[cp]$. Бесконечные вычисления считаем безрезультатными. Заметим, что в любой модели $\mathcal{M}$ всегда реализуется ровно одно вычисление программы.

Трассы tr_1, tr_2 в произвольных программах π_1, π_2 будем называть $\mathcal{F}$-*совместными*, где $\mathcal{F}$ — произвольная шкала, если они реализуются в какой-либо модели $\mathcal{M} = (\mathcal{F}, \xi)$. Программы π_1, π_2 назовем $\mathcal{F}$-*эквивалентными* ($\pi_1 \sim_{\mathcal{F}} \pi_2$), если все их $\mathcal{F}$-совместные вычисления имеют одинаковый результат.

Свойства перестановочности и подавляемости операторов естественным образом описываются соотношениями перестановочности ($ab = ba$, где $a, b \in \mathfrak{A}$) и подавления ($ab = b$, где $a, b \in \mathfrak{A}$). Рассматриваемые далее шкалы задаются посредством моноидов $\mathfrak{M} = (S_{\mathfrak{M}}, *)$, которые порождаются конечным множеством образующих $\mathfrak{A}$ и определяются разнообразными семействами соотношений перестановочности и подавления. Элементом $\langle h \rangle$ моноида $\mathfrak{M}$, порожденным термом $h \in \mathfrak{A}^*$, является множество всех термов, которые можно получить из h, применив конечное число раз определяющие соотношения этого моноида. Операция умножения элементов моноида определяется следующим образом: $\langle h \rangle * \langle g \rangle = \langle hg \rangle$. Указанные моноиды с тождествами перестановочности и подавления будем называть *CA-моноидами*. Терм h, имеющий наименьшую в классе $\langle h \rangle$ длину, будем называть *минимальным представителем* (класса $\langle h \rangle$). *Нормой* $\|s\|$ элемента s моноида будем называть длину минимальных представителей этого элемента.

Будем говорить, что шкала $\mathcal{F} = (S, s^0, R)$ задается моноидом $\mathfrak{M} = (S_{\mathfrak{M}}, *)$, если верны следующие соотношения: $S = S_{\mathfrak{M}}$; $s^0 = \langle \lambda \rangle$; $R(s, a) = s * \langle a \rangle$. Шкалу назовем *упорядоченной*, если $[h_1 h_2] \neq [h_1]$. Далее будут рассматриваться только упорядоченные шкалы $\mathcal{F}(C, A)$, задаваемые моноидами $\mathfrak{M}(C, A)$ с определяющими соотношениями $C \cup A$, где C содержит только соотношения перестановочности и A – только соотношения подавления. На множестве состояний данных упорядоченной шкалы введем частичный порядок: состояние s_1 предшествует состоянию s_2 (обозначение $s_1 \leq s_2$), если $R(s_1, h) = s_2$ для некоторого терма $h \in \mathfrak{A}^*$. Несравнимость состояний s_1, s_2 по отношению порядка $\leq$ обозначим записью $s_1 \perp s_2$.

Утверждение 1 [13]. Пусть $\mathcal{F}$ - произвольная упорядоченная шкала. Тогда трассы tr_1, tr_2 являются $\mathcal{F}$-совместными в том и только в том случае, если для любых их префиксов $tr_1' \xrightarrow{\delta_1} l_1$, $tr_2' \xrightarrow{\delta_2} l_2$ верно следующее: если $[tr_1] = [tr_2]$, то $\delta_1 = \delta_2$.

В работе [13] предложен метод решения проблема эквивалентности пропозициональных последовательных программ на упорядоченных шкалах $\mathcal{F}$: для заданных программ π_1, π_2 и заданной упорядоченной шкалы $\mathcal{F}$ выяснить, верно ли соотношение $\pi_1 \sim_{\mathcal{F}} \pi_2$. Этот метод будет использован в настоящей статье для построения полиномиального по времени работы (относительно размеров программ) разрешающего алгоритма для упорядоченных шкал $\mathcal{F}(C, A)$, задаваемых моноидами $\mathfrak{M}(C, A)$.

3. Алгебраические свойства CA-моноидов

В данном разделе сформулированы основные свойства моноидов $\mathfrak{M}(C, A)$, которые понадобятся далее для построения алгоритма проверки эквивалентности программ с перестановочными и подавляемыми операторами. Обоснование утверждений этого раздела проводится с

использованием традиционной для теории полугрупп техники доказательств; в связи с ограничениями на объем статьи эти доказательства опущены.

Утверждение 2 [17]. СА-моноид $\mathfrak{M}(C,A)$ с множеством определяющих соотношений коммутативности C и подавления A упорядочен тогда и только тогда, когда не существует ни одной пары операторов $a,b \in \mathfrak{A}$, для которой выполняется как равенство $ab = ba$ в моноиде $\mathfrak{M}(C)$ с соотношениями коммутативности C, так и равенство $ab = b$ в моноиде $\mathfrak{M}(A)$ с соотношениями подавления A.

Утверждение 3. Пусть СА-моноид $\mathfrak{M}(C,A)$ упорядочен, и h — такой минимальный представитель, для которого верно $[ah] \neq [h]$ для некоторого оператора $a \in \mathfrak{A}$. Тогда терм ah — также минимальный представитель.

Будем говорить, что моноид $\mathfrak{M}$ обладает свойством левого сокращения, если для любых его элементов s, s_1, s_2 верно следующее: если $s * s_1 = s * s_2$, то $s_1 = s_2$.

Утверждение 4. Всякий упорядоченный СА-моноид $\mathfrak{M}(C,A)$ обладает свойством левого сокращения.

Утверждение 5. Пусть термы h, g являются минимальными представителями некоторого элемента упорядоченного СА-моноида $\mathfrak{M}(C,A)$. Тогда терм g может быть получен из терма h применением только соотношений перестановочности.

Автоматом будем называть систему $\mathbb{A} = (Q, q^0, T, L_{\mathbb{A}}, B_{\mathbb{A}})$, где Q – конечное множество состояний, q^0 – инициальное состояние, $T: Q \times \mathfrak{A} \to Q$ – функция переходов, $L_{\mathbb{A}}$ – множество меток и $B_{\mathbb{A}}$ – разметка состояний. Далее будем использовать следующие обозначения:

- $\mathbb{A}(q,h)$ - состояние, в которое автомат $\mathbb{A}$ переводится словом h из состояния q;
- $\mathbb{A}(h) = \mathbb{A}(q^0,h)$; $B_{\mathbb{A}}(h) = B_{\mathbb{A}}(\mathbb{A}(h))$.

Будем говорить, что автомат $\mathbb{A}$ распознает подавление операторов, если выполнены следующие условия: $L_{\mathbb{A}} = 2^{\mathfrak{A}}$; если $\langle h_1\rangle = \langle h_2\rangle$, то $\mathbb{A}(h_1^-) = \mathbb{A}(h_2^-)$; $B_{\mathbb{A}}(h^-) = \{a \mid \langle ah\rangle = \langle h\rangle\}$.

Утверждение 6. Пусть моноид $\mathfrak{M}(C,A)$ упорядочен. Тогда существует автомат, распознающий подавление операторов.

4. Граф совместных вычислений

Пусть заданы упорядоченная шкалу $\mathcal{F}(C,A) = (S, s^0, R)$ и программы $\pi_i = \left(L_i, l_i^{in}, l_i^{out}, T_i, B_i\right)$, $i \in \{1,2\}$. Рассмотрим пошаговое построение совместных вычислений программ π_1, π_2, организованное следующим образом. В начале вычисления программ трассы пусты. Предположим, что для программ уже построены трассы tr_1, tr_2. Если $[tr_1] = [tr_2]$, то выбирается произвольное логическое условие δ, и обе программы делают шаг вычисления согласно этому условию. Если программа π_2 «отстает» от программы π_1, т. е. $[tr_2] <$

$[tr_1]$, то программа π_2 делает шаг вычисления. Иначе программа π_1 делает шаг вычисления.

В графе Г совместных вычислений программ π_1, π_2 сохранена информация о такой совместной работе программ, достаточная для исследования проблемы эквивалентности. Вершинами *графа совместных вычислений* Г являются четверки вида (l_1, l_2, s_1, s_2), где $l_i \in L_i$ и $s_i \in S$, $i \in \{1,2\}$. Корнем графа Г объявляется вершина $\left(l_1^{in}, l_2^{in}, s^0, s^0\right)$. Вершину $v = (l_1, l_2, s_1, s_2)$ назовем *опровергающей*, если выполнено одно из трех условий:

1. $l_1 = l_1^{out}$, $s_2 \neq s^0$;
2. $l_2 = l_2^{out}$, $s_1 \neq s^0$;
3. $l_1 = l_1^{out}$, $l_2 = l_2^{out}$, $(s_1, s_2) \neq (s^0, s^0)$.

Опровергающие вершины и вершину $(l_1^{out}, l_2^{out}, s^0, s^0)$ будем называть терминальными.

Дуги графа Г помечены парами $(d_1, d_2) \in (\mathfrak{C} \cup \{\varepsilon\})^2$. Из терминальных вершин не исходит ни одной дуги. Различные дуги, исходящие из нетерминальной вершины, помечены различными парами. Метки дуг, исходящих из вершины $v = (l_1, l_2, s_1, s_2,)$, образуют множество: $\mathfrak{C} \times \{\varepsilon\}$, если $l_2 = l_2^{out}$ или $s_2 \neq s^0$; $\{(\delta,\delta) \mid \delta \in \mathfrak{C}\}$, если $l_1 \neq l_1^{out}$, $l_2 \neq l_2^{out}$ и $s_1 = s_2 = s^0$; $\{\varepsilon\} \times \mathfrak{C}$ в остальных случаях. Вершина (l_1', l_2', s_1', s_2'), достижимая из v по дуге с меткой (d_1, d_2), определяется следующим образом:

- если $d_i = \varepsilon$, то $v_i' = v_i$ и $s_i'' = s_i$, иначе $v_i \xrightarrow{d_i} v_i'$ и $s_i'' = s_i * [B_i(v_i')]$, $i \in \{1,2\}$,
- если $s_1'' = s_2''$, то $s_1' = s_2' = s^0$;
- если $s_1'' \perp s_2''$, то $s_1' = s_1''$ и $s_2' = s_2''$;
- если $s_i'' < s_{3-i}''$, то $s_i' = s^0$ и $s_{3-i}'' = s_i'' * s_{3-i}'$.

Корректность этого описания следует из свойства левого сокращения (утверждение 4).

Корневым маршрутом графа Г назовем маршрут, начинающийся в его корне. Корневой маршрут будем называть *опровергающим* в следующих двух случаях:

1. этот маршрут оканчивается в опровергающей вершине;
2. существуют натуральное число N и индекс i, такие что при отбрасывании N первых вершин маршрута все i-е компоненты меток дуг равны ε и все i-е компоненты вершин завершаемы.

Далее мы покажем, что существование опровергающих маршрутов в графе совместных вычислений Г равносильно неэквивалентности программ.

Чтобы установить соответствие между корневыми маршрутами графа Г и совместными трассами программ π_1, π_2, введем понятие *проекции* $pr(\Omega)$ корневого маршрута Ω графа Г:

- $pr(\Omega) = \left(pr_1(\Omega), pr_2(\Omega)\right)$;
- $pr_i\left(l_1^{in}, l_2^{in}, s^0, s^0\right) = l_i^{in}$;
- $pr_i\left(\Omega \xrightarrow{d_1,d_2} (l_1, l_2, s_1, s_2)\right) = pr_i(\Omega)$, если $d_i = \varepsilon$;
- $pr_i\left(\Omega \xrightarrow{d_1,d_2} (l_1, l_2, s_1, s_2)\right) = pr_i(\Omega) \xrightarrow{d_i} l_i$, если $d_i \neq \varepsilon$.

Лемма 1. Пусть Ω — корневой маршрут графа Г. Тогда:

1. $pr_1(\Omega)$ и $pr_2(\Omega)$ — совместные трассы программ π_1, π_2;
2. если маршрут Ω оканчивается в вершине (l_1, l_2, s_1, s_2), то:
   1. для некоторого состояния данных s верны соотношения $[pr_i(\Omega)] = s * s_i$, $i \in \{1,2\}$;
   2. $s_2 \in \{[a] \mid a \in \mathfrak{A}\} \cup \{s^0\}$;
   3. если $s_1 \neq s^0$ и $s_2 \neq s^0$, то $s_1 \perp s_2$.

Лемма 1 легко обосновывается индукцией по длине маршрута Ω с привлечением утверждения 1. Если маршрут Ω бесконечен, то совместность его проекций вытекает из совместности любых конечных префиксов его проекций.

Лемма 2. Пусть cp_1, cp_2 — совместные вычисления программ π_1, π_2. Тогда в графе совместных вычислений Г существует корневой маршрут Ω, который либо является бесконечным, либо оканчивается в терминальной вершине, такой что $pr_i(\Omega)$ – это префикс вычисления cp_i для $i \in \{1,2\}$.

Для обоснования леммы 2 достаточно построить маршрут согласно описанию совместной работы программ и убедиться, что ни на каком шаге построения не нарушается совместность трасс (утверждение 1, лемма 1). Если процесс построения маршрута бесконечен, то в качестве результата берется бесконечный маршрут, префиксами которого являются маршруты на всех шагах построения.

Лемма 3. Пусть Ω — корневой маршрут графа Г с проекцией (tr_1, tr_2) и для некоторого $i \in \{1,2\}$ трасса tr_i является префиксом трассы tr. Тогда трассы tr и tr_{3-i} совместны.

Доказательство. Без ограничения общности считаем $i = 1$. Предположим, что трассы tr и tr_2 несовместны. По утверждению 1 равенство $[tr_1'] = [tr_2']$ выполнено для некоторых собственных префиксов tr_1', tr_2' трасс tr, tr_2. По утверждению 1 трассы tr_1 и tr_2 совместны, а значит, $tr_1' = tr_1 \to pt$. Рассмотрим кратчайший префикс Ω′ маршрута Ω, для которого $pr_2(\Omega') = tr_2'$. Привлекая утверждение 1, лемму 1 и описание совместной работы программ, легко убедиться в невозможности достроить маршрут Ω′ до Ω.

Теорема 1. $\pi_1 \sim_{\mathcal{F}} \pi_2$ тогда и только тогда, когда в графе Г не содержится ни одного опровергающего маршрута.

Доказательство. Достаточность. Рассмотрим опровергающий маршрут Ω. По определению он либо бесконечен, либо оканчивается в опровергающей вершине. Если он оканчивается в вершине, содержащей выходы программ π_1, π_2, то по лемме 1 получим совместные конечные вычисления с различными результатами. Если он оканчивается в вершине, содержащей выход l_i^{out} ровно одной программы, то, продолжив трассу $pr_{3-i}(\Omega)$ произвольным образом до вычисления и используя утверждение 4 и леммы 1, 3, получим либо совместные конечные вычисления с различными результатами, либо совместные конечное и бесконечное вычисление. Пусть теперь маршрут Ω бесконечен. По лемме 1 его проекцией являются бесконечное вычисление программы π_i и конечная трасса программы π_{3-i}, где $i \in \{1,2\}$, причем по определению опровергающего маршрута и лемме 3 трассу $pr_{3-i}(\Omega)$ можно продолжить до конечного вычисления программы π_{3-i}, совместного с вычислением $pr_i(\Omega)$.

Необходимость. Программы π_1, π_2 имеют либо совместные конечные вычисления с различными результатами, либо совместные конечное и бесконечное вычисления. Пусть это вычисления cp_1, cp_2. Рассмотрим корневой маршрут Ω графа Г, подходящий под условие леммы 2. Если маршрут Ω конечен, то он оканчивается в опровергающей вершине (иначе по лемме 1 результаты вычислений cp_1, cp_2 совпадают). Если маршрут Ω бесконечен, то, привлекая определение проекции корневого маршрута, получаем, что Ω — бесконечный опровергающий маршрут. Доказательство теоремы завершено.

5. Эффекты подавления

Все дальнейшие рассуждения строятся на основе существования функции $æ_s$, где $s \in S$, определяемой следующим образом: $æ_s(s_1) = s_2$, где $s_1 * s = s_2 * s$ и s_2 имеет наименьшую возможную норму. Функцию $æ_s$ будем называть *эффектом подавления*, индуцированным состоянием данных s, или, коротко, -подавлением. Существование s -подавления для рассматриваемой нами шкалы вытекает из следующего утверждения.

Лемма 4. Если уравнение $X * s = s'$ имеет решение относительно X, то ровно одно его решение имеет наименьшую среди всех решений норму.

Доказательство. Предположим, что s_1 и s_2 — два различных решения уравнения $X * s = s'$, имеющие наименьшую среди всех решений норму. Пусть h_i — минимальный представитель класса s_i, $i \in \{1,2\}$, и h — минимальный представитель класса s. По утверждению 3 термы $h_1 h$, $h_2 h$ являются минимальными представителями одного состояния данных. По утверждению 5 терм $h_2 h$ может быть получен из $h_1 h$ применением только соотношений перестановочности. Взяв зеркальные образы двух последних термов, применив утверждение 4 и применив повторно зеркальное

преобразование, получим вывод терма h_2 из h_1. Доказательство леммы завершено.

Множество всевозможных эффектов подавлений далее будем обозначать записью Æ. Введем порядок на множестве Æ: $æ_1 \le æ_2$ тогда и только тогда, когда существуют состояния данных s, s', такие что $s \le s'$, $æ_1 = æ_s$ и $æ_2 = æ_{s'}$.

Лемма 5. Отношение $\le$ на множестве Æ является нестрогим частичным порядком.

Доказательство. Предположим, что существуют два различных эффекта подавления $æ_1$, $æ_2$, такие что $æ_1 \le æ_2$ и $æ_2 \le æ_1$. Заметим, привлекая утверждение 3, что если $æ(s) = s'$ и h — минимальный представитель класса s, то можно получить минимального представителя h' класса s вычеркиванием некоторых символов из h, причем вычеркиваемые символы однозначно определяются эффектом подавления æ. Из определения порядка на множестве Æ следует: если $æ \le æ'$, $æ(s) = s_1$ и $æ'(s) = s_2$ то существуют минимальные представители h_1, h_2 классов s_1 и s_2 соответственно, такие что h_2 получается из h_1 вычеркиванием некоторых вхождений. Используя последний факт, получаем, что если $æ_1 \le æ_2$ и $æ_2 \le æ_1$, то $æ_1 = æ_2$.

Лемма 6. $æ_s(s_1) = æ_s(s_2)$ тогда и только тогда, когда $s_1 * s = s_2 * s$.

Лемма 7. Если $æ(s_1) = æ(s_2)$ и $æ \le æ'$, то $æ'(s_1) = æ'(s_2)$.

Лемма 6 следует из леммы 4 и определения эффекта подавления, лемма 7 следует из леммы 6.

Будем говорить, что автомат $\mathbb{A}$ с множеством меток Æ распознает эффекты подавления, если выполнены следующие два условия:

автомат $\mathbb{A}$ согласован с моноидом: для любой пары термов h_1, h_2 если $[h_1] = [h_2]$, то $\mathbb{A}(h_1) = \mathbb{A}(h_2)$;

автомат $\mathbb{A}$ согласован с эффектами подавления: $B_{\mathbb{A}}(h) = æ_h$ для любого терма h.

Для автомата $\mathbb{A}$, согласованного с моноидом, будем использовать записи $\mathbb{A}([h])$, $B_{\mathbb{A}}([h])$ наравне с $\mathbb{A}(h)$, $B_{\mathbb{A}}(h)$.

Лемма 8. Существует автомат, распознающий эффекты подавления.

Доказательство. Приведем явное описание автомата $\mathbb{A}$, распознающего эффекты подавления. Начнем построение с автомата $\mathbb{A}_0$, распознающего подавление операторов (утверждение 6). Состояния q_1, q_2, этого автомата объявим эквивалентными, если равенство $B_{\mathbb{A}_0}(q_1, h) = B_{\mathbb{A}_0}(q_2, h)$ выполнено для всех термов $h \in \mathfrak{U}^*$. Переберем классы эквивалентности состояний автомата $\mathfrak{U}_0$ и назначим i-му классу новую метку $æ_i$. В результате такой замены получим автомат $\mathbb{A}_1$. Заметим (утверждение 3), что $\mathbb{A}_1(h_1) = \mathbb{A}_2(h_2)$ тогда и только тогда, когда $æ_{g_1} = æ_{g_2}$, где g_i есть терм h_i, записанное наоборот, $i \in \{1,2\}$. Таким образом, можно считать, что $L_{\mathbb{A}_1} = Æ$. Отбросив

метки вершин и назначив вершины с меткой $æ_i$, получим автомат-распознаватель $\mathbb{A}_2^i$. Используя известные результаты теории автоматов, можно построить по автомату $\mathbb{A}_2^i$ автомат $\mathbb{A}_3^i$. Можно легко убедиться, что автомат $\mathbb{A}_3^i$ принимает язык $\{h \mid æ_{[h]} = æ_i\}$. Рассмотрим декартово произведение автоматов $\mathbb{A}_3^i$, назначив метку $æ_i$ состоянию $(q_1, \dots, q_k)$, где q_i — финальное состояние автомата $\mathbb{A}_3^i$ (заметим, что по опредлению эффекта подавления индекс i однозначно определен для любого состояния декартова произведения). Описанное декартово произведение и будет искомым автоматом $\mathbb{A}$. Доказательство леммы завершено.

В дальнейших рассуждениях считаем заданным автомат $\mathbb{A} = (Q, q^0, T_{\mathbb{A}}, L_{\mathbb{A}}, B_{\mathbb{A}})$, распознающий эффекты подавления.

Пусть $s \in S$ и $h = a_1 \dots a_k \in \mathfrak{U}^*$. Тогда *развитием s-подавления вдоль терма h* (коротко, (s,h)-*подавлением*) назовем последовательность $ȧæ_s^h$, получаемую из последовательности $(æ_s, æ_{s*[a_1]}, \dots, æ_{s*[h]})$ удалением соседних повторяющихся элементов. (s,pt)-подавлением, где $s \in S$ и pt – конечный путь в произвольной программе, будем называть $(s, B(pt))$-подавлением. Если путь pt бесконечен, то (s,pt)-*подавлением* назовем наибольшее по длине (s,pt')-подавление, где pt' — конечный префикс пути pt. Существование (s,pt)-подавления для бесконечного пути pt следует из очевидного факта: любое (s,pt)-подавление является цепью в конечном (лемма 8) частично упорядоченном (лемма 5) множестве Æ.

Конечный путь pt в программе назовем $\dot{s}$-*приведенным*, где $s \in S$, если для любых его различных префиксов pt_1, pt_2 они оканчиваются в различных состояниях управления или $\mathbb{A}(s * [pt_1]) \ne \mathbb{A}(s * [pt_2])$. Бесконечный путь pt будем называть $\dot{s}$-*приведенным*, если он представим в виде $pt_1(\to pt_2)^\omega$, где $pt_1 \to pt_2$ — конечный $\dot{s}$-приведенный путь.

Лемма 9. Пусть l — состояние управления программы π, pt — путь в программе π, начинающийся в состоянии l, и $s \in S$. Тогда существует $\dot{s}$-приведенный путь pt' в программе π, начинающийся в состоянии управления l и такой что: $ȧæ_s^{pt} = ȧæ_s^{pt'}$. При этом путь pt' конечен тогда и только тогда, когда путь pt конечен; если путь pt' конечен, то он завершается в том же состоянии управления, что и pt.

Доказательство. Опишем алгоритм построения пути pt' по pt. Пусть путь pt конечен. Если он не является $\dot{s}$-приведенным, то существуют два различных префикса pt_1 и $pt_1 \to pt_2$ этого пути, оканчивающиеся в одном состоянии управления и такие что $\mathbb{A}(s * [pt_1]) = \mathbb{A}(s * [pt_1 \to pt_2])$. Удалим из пути pt подпуть pt_2, и если результат не является $\dot{s}$-приведенным, то повторим процедуру. Путь pt конечен, а следовательно, за конечное число удалений будет получен искомый $\dot{s}$-приведенный путь pt'.

Допустим, что путь pt бесконечен. Рассмотрим произвольный конечный префикс $pt'' \to l$ пути pt, такой что $ä_s^{pt''} \neq ä_s^{pt}$ и $ä_s^{pt''\to l} = ä_s^{pt}$. Применим к нему процедуру удаления подпути. Добавим к результирующему пути бесконечный «хвост», отброшенный перед удалением. Найдем в этом хвосте первое повторение состояния управления и заменим хвост на бесконечное повторение найденного цикла. Результатом будет искомый путь pt'. Доказательство завершено.

Лемма 10. Пусть $s \in S$ и pt — $\dot{s}$-приведенный путь в программе π, представимый в виде $pt = pt'(\to pt'')^\omega$ (здесь путь pt'' считается пустым в том и только в том случае, если путь pt конечен). Тогда $|pt'| + |pt''| \le |\pi| \cdot |Q|$.

Справедливость леммы 10 объясняется тем, что существует ровно $|\pi| \cdot |Q|$ различных пар (l, q), где l – состояние управления программы π и q – состояние автомата $\mathbb{A}$.

Пусть $s \in S$ и $h = a_1 \dots a_k \in \mathfrak{A}^*$. Тогда *обобщенным развитием s-подавления вдоль терма h*, или, коротко, $[\![s,h]\!]$-подавлением, будем называть последовательность $ä_s^h$, получаемую из последовательности $\left((æ_{s^0}, æ_s), (æ_{[a_1]}, æ_{s*[a_1]}), \dots, (æ_{[h]}, æ_{s*[h]})\right)$ удалением соседних повторяющихся элементов. $[\![s,pt]\!]$-подавлением, где $s \in S$ и pt — конечный путь в произвольной программе, будем называть $[\![s,[pt]]\!]$-подавление. Если путь pt бесконечен, то $[\![s,pt]\!]$-подавлением будем называть наибольшее по длине $[\![s,pt']\!]$-подавление, где pt' — конечный префикс пути pt. Существование $[\![s,pt]\!]$-подавления для бесконечного пути pt следует из очевидного факта: любое $[\![s,pt]\!]$-подавление является цепью в декартовом квадрате конечного частично упорядоченного множества *Æ*.

Конечный путь pt (в какой-либо программе) будем называть *$\ddot{s}$-приведенным*, где $s \in S$, если для любых его различных префиксов pt_1, pt_2, оканчивающихся в состояниях управления l_1, l_2 соответственно, выполнено: $(l_1, \mathbb{A}([pt_1]), \mathbb{A}(s * pt1 \neq l2, \mathbb{A}pt2, \mathbb{A}s*pt2$. Бесконечный путь pt будем называть *s-приведенным*, если он представим в виде $pt_1(\to pt_2)^\omega$, где $pt_1 \to pt_2$ — конечный $\ddot{s}$-приведенный путь. $\ddot{s}$-приведенный путь будем также называть $\mathbb{A}(s)$-приведенным.

Лемма 11. Пусть l — состояние управления программы π, pt — путь в программе π, начинающийся в состоянии l, и $s \in S$. Тогда существует $\ddot{s}$-приведенный путь pt' в программе π, начинающийся в состоянии управления l и такой что: $ä_s^{pt} = ä_s^{pt'}$. При этом путь pt' конечен тогда и только тогда, когда путь pt конечен; если путь pt' конечен, то он оканчивается в том же состоянии управления, что и путь pt.

Лемма 12. Пусть $s \in S$ и pt — $\ddot{s}$-приведенный путь в программе π, представимый в виде $pt = pt_1(\to pt_2)^\omega$ (здесь путь pt_2 считается пустым в

том и только в том случае, если путь pt конечен). Тогда $|pt_1| + |pt_2| \le |\pi| \cdot |Q|^2$.

Обоснования лемм 11, 12, по существу, повторяют обоснования лемм 9, 10 с незначительными изменениями и потому опущены.

6. Анализ графа совместных вычислений

В данном разделе приводятся утверждения, обеспечивающие полиномиальный обход графа Г. Существование такого обхода совместно с упомянутой теоремой 1 обеспечивает полиномиальную разрешимость проблемы эквивалентности программ в рассматриваемой модели вычислений.

Лемма 13. Пусть $q \in Q$, $l_1 \in L_1$, $l_2 \in L_2$, $s_2 \in S$, $m = |Q|^3 \cdot |\pi_1| \cdot |\pi_2|^2 + 2$ и:

- $l_1 \to pt_1$ и $l_2 \to pt_2$ — полные пути программ π_1, π_2, и хотя бы один из этих путей конечен;
- $ä_q^{pt_1} = \left((æ_1^1, æ_1^2), \dots, (æ_k^1, æ_k^2)\right)$;
- $ä_{s_2}^{pt_2} = (æ_1, \dots, æ_s)$;
- $1 \le k' \le k$;
- из корня графа Г достижимы -вершины $v_i = \left(l_1, l_2, s_1^i, s_2\right)$, где $i \in \{1, \dots, m\}$;
- состояния данных $æ_{k'}^1(s_1^i)$, $i \in \{1, \dots, m\}$ попарно различны;
- $\{æ_1, \dots, æ_s\} \cap \{æ_{k'+1}^2, \dots, æ_k^2\} = \emptyset$.

Тогда $\pi_1 \nsim \pi_2$.

Доказательство. Пусть Ω_i — корневой маршрут, оканчивающийся в вершине v_i, $i \in \{1,2\}$, и $pr(\Omega_i) = \left(tr_1^i \to l_1, tr_2^i \to l_2\right)$. Из леммы 1 получим представления $\left[tr_1^i \to l_1\right] = s^i * s_1^i$ и $\left[tr_2^i \to l_2\right] = s^i * s_2$. Приведем подробное обоснование только для одного случая: состояние управления l_2 завершаемо, $k' = k$, путь pt_1 бесконечен. Остальные случаи обосновываются схожим образом, дополнительно привлекая леммы 10, 11, и потому опущены. Без ограничения общности полагаем, что путь pt_1 q-приведен (лемма 9). Пусть $l_2 \to pt$ – кратчайший полный путь в программе π_2. Достаточно показать, что хотя бы одна пара вычислений $cp_1^i = tr\_1\text{^}i \to pt_1$, $cp_2^i = tr_2^i \to pt$ совместна. Предположим, что это не так. Тогда по утверждению 1 для любого индекса $j \in \{1, \dots, m\}$ найдутся собственные префиксы $\widehat{tr}_1^j \xrightarrow{\delta_1} l_1'$, $\widehat{tr}_2^j \xrightarrow{\delta_2} l_2'$ вычислений cp_1^j, cp_2^j, такие что $\left[\widehat{tr}_1^j\right] = \left[\widehat{tr}_2^j\right]$ и $\delta_1 \neq \delta_2$. Привлекая утверждение 1 и лемму 3, получим неравенства $\left|\widehat{tr}_1^j\right| > \left|tr_1^j\right|$, $\left|\widehat{tr}_2^j\right| > \left|tr_2^j\right|$. Тогда по утверждению 4 для некоторых префиксов $\widehat{pt}_1^j$, $\widehat{pt}_2^j$ путей pt_1, pt верно равенство $s_1^j * \left[\widehat{pt}_1^j\right] = s_2 * \left[\widehat{pt}_2^j\right]$. Из неравенства $|pt| \le |\pi_2|$ получаем следующие соотношения:

$\left\|s_1^j * \left[\widehat{pt}_1^j\right]\right\| = \left\|s_2 * \left[\widehat{pt}_2^j\right]\right\| \le |\pi_2|$. По лемме 12 получим неравенство $\left|\widehat{pt}_1^j\right| \le |Q|^2 \cdot |\pi_1| \cdot |\pi_2|$. Следовательно, пара длин $\left(\left|\widehat{pt}_1^j\right|, \left|\widehat{pt}_2^j\right|\right)$ может принимать лишь $(m-2)$ различных значений, а значит, найдутся различные индексы p, q, такие что $\left(\widehat{pt}_1^p, \widehat{pt}_2^p\right) = \left(\widehat{pt}_1^q, \widehat{pt}_2^q\right)$. Привлекая леммы 6, 7, получим равенство $æ_k\left(s_1^p\right) = æ_k\left(s_1^q\right)$, что противоречит условию леммы. Полученное противоречие завершает доказательство леммы.

Лемма 14. Пусть $q \in Q$, $l_1 \in L_1$, $l_2 \in L_2$, $s_2 \in S$, $m = |Q|^3 \cdot |\pi_1| \cdot |\pi_2|^2 + 2$ и:

- $l_1 \to pt_1$ и $l_2 \to pt_2$ — полные пути программ π_1, π_2;
- $ä_q^{pt_1} = \left((æ_1^1, æ_1^2), \dots, (æ_k^1, æ_k^2)\right)$;
- $ȧ_{s_2}^{pt_2} = (æ_1, \dots, æ_s)$;
- из корня графа Г достижимы -вершины $v_i = \left(l_1, l_2, s_1^i, s_2\right)$, где $i \in \{1, \dots, m\}$;
- $1 \le k' < k$, $1 \le s' < s$;
- состояния данных $æ_{k'}^1\left(s_1^i\right)$, $i \in \{1, \dots, m\}$, попарно различны;
- состояния данных $æ_{k'+1}^1\left(s_1^i\right)$, $i \in \{1, \dots, m\}$, совпадают;
- $\{æ_1, \dots, æ_{s'}\} \cap \{æ_{k'+1}^2, \dots, æ_k^2\} = \emptyset$;
- $æ_{s'+1} \in \{æ_{k'+1}^2, \dots, æ_k^2\}$;
- Ω — корневой маршрут графа Г, оканчивающийся в вершине v_m;
- $pr_1(\Omega) \to pt_1$ и $pr_2(\Omega) \to pt_2$ — совместные вычисления программ π_1, π_2, имеющие различные результаты.

Тогда для некоторого $j \in \{1, \dots, m\}$ из вершины v_j исходит опровергающий маршрут.

Доказательство. Пусть Ω_i — корневой маршрут графа Г, оканчивающийся в вершине v_i, $i \in \{1,2\}$, и $pr(\Omega_i) = \left(tr_1^i \to l_1, tr_2^i \to l_2\right)$. По лемме 1 получим представления $\left[tr_1^i \to l_1\right] = s^i * s_1^i$ и $\left[tr_2^i \to l_2\right] = s^i * s_2$. Рассмотрим альтернативные случаи.

Случай 1: одно из состояний управления l_i незавершаемо. Без ограничения общности полагаем $i = 1$. Тогда вычисление $pr_1(\Omega) \to pt_1$ бесконечно, и из условия леммы получаем, что путь pt_2 конечен. По лемме 3 трасса $tr_1^1 \to l_1$ и вычисление $cp_2 = tr_2^1 \to l_2 \to pt_2$ совместны. Используя утверждение 1, можно продолжить трассу $tr_1^1 \to l_1$ до бесконечного вычисления, совместного с конечным вычислением cp_2.

Случай 2: состояние управления l_1 завершаемо и существует бесконечный путь $l_2 \to pt_2'$, такой что $ȧ_{s_2}^{pt_2'}$ есть префикс $ȧ_{s_2}^{pt_2}$ длины менее s'. Рассуждая так же, как и в доказательстве леммы 13, можно получить совместные вычисления $tr_1^j \to l_1 \to \overline{\overline{pt}}_1$, $tr_2^j \to l_2 \to \overline{\overline{pt}}_2$ программ π_1, π_2. Достроим

маршрут Ω_j наидлиннейшим образом до маршрута Ω' так же, как и в обосновании леммы 2. Используя лемму 1, можно легко показать, что маршрут Ω' является опровергающим.

Случай 3: состояние управления l_2 завершаемо и существует бесконечный путь $l_1 \to pt_1'$, такой что $ä_q^{pt_1'}$ есть префикс $ä_q^{pt_1}$ длины не более k'. Рассуждения данного случая аналогичны случаю 2.

Случай 4: состояния управления l_1, l_2 завершаемы и не существует бесконечных путей, удовлетворяющих условиям случаев 2, 3. Пусть $pt_1' \to l_1'$, $pt_2' \to l_2'$ — префиксы путей pt_1, pt_2, такие что $\left|ä_q^{pt_1'}\right| = k' \ne \left|ä_q^{pt_1' \to l_1'}\right|$ и $\left|ȧ_{s_2}^{pt_2'}\right| = s' \ne \left|ȧ_{s_2}^{pt_2' \to l_2'}\right|$. По леммам 10, 12 имеем неравенства $|pt_1'| \le |Q|^2 \cdot |\pi_1|$, $|pt_2'| \le |Q| \cdot |\pi_2|$. Рассуждая так же, как в доказательстве леммы 13, получим индекс $j \in \{1, \dots, m-1\}$, такой что трассы $tr_1^j \to l_1 \to pt_1' \to l_1'$ и $tr_2^j \to l_2 \to pt_2' \to l_2'$ совместны. Достаточно показать, что вычисления $cp_1^j = tr_1^j \to l_1 \to pt_1$ и $cp_2^j = tr_2^j \to l_2 \to pt_2$ совместны и имеют различные результаты. Различие результатов может быть легко получено с использованием утверждения 4, леммы 7, представлений $\left[tr_1^i \to l_1\right] = s^i * s_1^i$ и $\left[tr_2^i \to l_2\right] = s^i * s_2$ и того факта, что результаты вычислений $cp_1^m = tr_1^m \to l_1 \to pt_1$, $cp_2^m = tr_2^m \to l_2 \to pt_2$ различны. Совместность вычислений cp_1^j, cp_2^j может быть легко получена из совместности вычислений cp_1^m, cp_2^m с использованием утверждений 1, 6 и лемм 3, 8.

7. Разрешающий алгоритм

Далее полагаем $n = \max(|\pi_1|, |\pi_2|)$. Опишем алгоритм α, разрешающий проблему эквивалентности последовательных программ на упорядоченной шкале $\mathcal{F}(C, A)$. Рассмотрим произвольный обход графа Г (например, обход в глубину или в ширину). Модифицируем этот обход следующим образом. Разобьем рассмотренные при обходе вершины на $O(n^2)$ групп, определяя в одну группу вершины (l_1, l_2, s_1, s_2), (l_1', l_2', s_1', s_2'), если $l_1 = l_1'$, $l_2 = l_2'$, $\mathbb{A}(s_1) = \mathbb{A}(s_1')$ и $s_2 = s_2'$. При рассмотрении очередной вершины v производятся следующие действия. Если v – опровергающая вершина, то алгоритм останавливается и констатирует неэквивалентность программ. Если через вершину v и остальные рассмотренные вершины проходит цикл, являющийся «хвостом» бесконечного опровергающего маршрута, то алгоритм останавливается и констатирует неэквивалентность программ. Если при добавлении вершины v в свою группу число вершин в ее группе оказывается $(|Q|^3 \cdot n^3 + 2)$ вершин, то согласно леммам 13, 14 либо программы признаются неэквивалентными, либо дуги, исходящие из вершины v, игнорируются в дальнейшем обходе. Если обход завершен и программы не признаны неэквивалентными, то они признаются эквивалентными.

Лемма 15. Равенство состояний $[h_1]$, $[h_2]$ шкалы $\mathcal{F}(C,A)$ может быть проверено за время $O(m^2)$, где $m = \max(|h_1|, |h_2|)$.

Доказательство. По утверждениям 3, 6 минимальные представители g_1, g_2 классов $[h_1]$, $[h_2]$ могут быть построены за время $O(m)$. По утверждению 5 достаточно проверить, можно ли получить g_2 из g_1, применяя только соотношения перестановочности. Если $|g_1| \neq |g_2|$, то этого сделать нельзя. Пусть теперь $|g_1| = |g_2|$. Если $g_1 = \lambda$, то достаточно проверить равенство $g_2 = \lambda$. Иначе рассмотрим самый левый символ a терма g_1 и самое левое вхождение того же символа в терм g_2. Если такое вхождение не найдено, то $[h_1] \neq [h_2]$. Если это вхождение найдено, но слева от него есть вхождение символа b, такое что a и b не перестановочны, то $[h_1] \neq [h_2]$. Иначе вычеркнем левый символ терма g_1 и рассматриваемое вхождение в терм g_2 и повторим описанную процедуру проверки для полученных более коротких термов. Всего производится не более m процедур проверки, каждая из которых завершается за время $O(m)$.

Лемма 16. Алгоритм α завершает свою работу за время $O(n^{18})$.

Доказательство. Алгоритм α исследует $O(n^5)$ вершин графа Г (леммы 13, 14). Для хранения состояний данных в каждой вершине достаточно термов длины $O(n^5)$. При исследовании вершины производится не более $O(n^3)$ сравнений состояний данных (леммы 13, 14). Сравнение двух состояний данных может быть произведено за время $O(n^{10})$ (лемма 15).

Лемма 17. $\pi_1 \sim \pi_2$ тогда и только тогда, когда алгоритм α признает программы эквивалентными.

Справедливость леммы 17 следует из лемм 16, 13, 14 и теоремы 1. Леммы 16, 17 подводят итог исследованию, проведенному в данной работе, и позволяют считать обоснованной следующую теорему.

Теорема 2. Пусть шкала $\mathcal{F} = \mathcal{F}(C,A)$ упорядочена. Тогда задача проверки $\mathcal{F}$-эквивалентности программ π_1, π_2 разрешима за полиномиальное относительно размера программ время.

8. Заключение

Основной результат статьи – теорему 2 – можно оценивать двояко. В работе [9] показано, каким образом, используя алгоритм проверки эквивалентности, можно построить эффективный алгоритм минимизации программ, операторы которых являются образующими элементами упорядоченного левосократимого моноида. Таким образом, полиномиальный алгоритм проверки эквивалентности программ с перестановочными и подавляемыми операторами, предложенный в разделе 8, свидетельствует о том, что оптимизировать программы можно столь же эффективно и просто, как и конечные автоматы. Однако высокая степень полинома, оценивающего сложность этого алгоритма, пока не дает оснований говорить о том, что этот метод оптимизации программ является практически пригодным. Поэтому цель

дальнейших исследований – совершенствование предложенного алгоритма в стремлении понизить его сложность до величины порядка $n^2 \div n^4$. Результаты работ [13-15] позволяют полагать, что такие ожидания небеспочвенны.

Список литературы

[1]. Ахо А., Лам М., Сети Р., Ульман Д. Компиляторы: принципы, технологии и инструментарий. — 2-е издание. — М.: «Вильямс», 2008. — 1184 с.

[2]. Hoare C.A.R. An axiomatic basis for computer programming // Communications of the ACM. – 1969. – v. 12, N 10, p. 576-580.

[3]. Ершов А.П. Сведение задачи экономии памяти при составлении программ к задаче раскраски вершин графа // Доклады АН СССР. – 1962. – т. 142, N 4. – с. 785–787.

[4]. Глушков В.М. Теория автоматов и формальные преобразования микропрограмм // Кибернетика. – 1965. – N 5. – с. 1–9.

[5]. Ляпунов А.А. О логических схемах программ // Проблемы кибернетики, вып. 1. – М.:Физматгиз, 1958. – с. 46-74.

[6]. Янов Ю.И. О логических схемах алгоритмов // Проблемы кибернетики, вып. 1. – М.:Физматгиз, 1958. – с. 75-127.

[7]. Ершов А.П. Операторные схемы (Об операторных схемах Янова) // Проблемы кибернетики, вып. 20. – М.:Физматгиз, 1967. – с. 181-200.

[8]. Ershov A.P. Alpha – an automatic programming system of high efficiency // Journal of the Association for Computing Machinary. – 1966. – v. 13, N 1. – p. 17-24.

[9]. Глушков В.М., Летичевский А.А. Теория дискретных преобразователей. // Избранные вопросы алгебры и логики: сб.статей. – Новосибирск: Наука, 1973. – с. 5-39.

[10]. Ершов А.П. Современное состояние теории схем программ // Проблемы кибернетики, вып. 27. – М.:Наука, 1973. – с. 87-110.

[11]. Подловченко Р.И., Захаров В.А. Полиномиальный по сложности алгоритм, распознающий коммутативную эквивалентность схем программ // Доклады РАН, серия Информатика. – 1998. – т. 362, N 6. – с. 27-31.

[12]. Zakharov V.A. An efficient and unified approach to the decidability of equivalence of propositional program schemes // Lecture Notes in Computer Science. – 1998. – v. 1443. – p. 247-258.

[13]. Захаров В.А. Быстрые алгоритмы разрешения эквивалентности операторных программ на уравновешенных шкалах // Математические вопросы кибернетики, вып.7. – М.:Физматлит, 1998. – с. 303-324.

[14]. Захаров В.А. Проверка эквивалентности программ при помощи двухленточных автоматов // Кибернетика и системный анализ. – 2010. – N 4. – с. 39-48.

[15]. Захаров В.А., Щербина В.Л. Об эквивалентности программ с операторами, обладающими свойствами коммутативности и подавления // Материалы 9-го Международного семинара «Дискретная математика и ее приложения», Москва, 2007. – 2007. – с. 191-194.

[16]. Harel D., Kozen D., Tiuryn J. Dynamic logic. MIT Press, Cambridge, MA, USA. – 2000. – 450 p.

[17]. Подымов В.В., Захаров В.А. Об одной полугрупповой модели программ, определяемой при помощи двухленточных автоматов // Научные ведомости Белгородского государственного университета. Серия История, экономика, политология, информатика, том 14. – № 7. – с. 94-101.

A polynomial algorithm for checking the equivalence in models of programs with commutation and vast operators

[1] *V.V. Podymov*
<*valdus@yandex.ru*>
[2] *V.A. Zakharov*
<zakh@cs.msu.su>
[1] *Lomonosov Moscow State University, Faculty CMC,*
2nd Education Building, GSP-1, Leninskie Gory, Moscow
[2] *ISP RAS, 25 Alexander Solzhenitsyn Str., Moscow, 109004, Russian Federation*

Abstract. In this paper we study the equivalence problem in the model of sequential programs which assumes that some instructions are commutative and absorbing. Two instructions are commutative if the result of their executions does not depend on an order of their execution. An instruction b absorbs an instruction a if the sequential composition $a; b$ yields the same result as the single instruction b. A.A. Letichevskij in 1971 proved the decidability of equivalence checking problem in this model of programs. Nevertheless a possibility of building polynomial time equivalence checking procedures remains an open problem till nowadays. The main result of this paper is the description of a polynomial time algorithm for checking the equivalence of sequential programs with commutative and absorbing instructions. The paper includes 9 sections. In Section 1 we introduce informally the model of programs under consideration, the equivalence checking problem, and give a brief overview of the preceding results in the study of equivalence checking problem for this model. In Sections 2 and 3 the syntax and a semigroup-based semantics of propositional model of sequential programs are formally defined. The algebraic properties of semigroups of commutative and absorbing program instructions are studied in Section 4. In Section 5 we introduce a graph of joined computations which is the key structure in designing our equivalence checking techniques. In Section 6 and 7 we show that the cumulative absorbing effect of finite sequences of commutative instructions can be specified by means of finite automata; this is another key step in building the decision procedure. In Section 8 we show that equivalence checking of two programs is reducible to a traversal of a bounded fragment of the graph of joint computations of these programs; the latter can be performed in time polynomial of the size of programs to be checked.

Keywords: program, commuting instructions, absorbing instructions, program equivalence, finite automata, decision procedure, polynomial time complexity.

References

[1]. Aho A., Lam M., Sethi R, Ullman J. D. Compilers: Principles, Techniques, and Tools. Pearson Education, Ltd., 2014, 940 p.
[2]. Hoare C.A.R. An axiomatic basis for computer programming. Communications of the ACM. 1969, v. 12, N 10, p. 576-580.
[3]. Ershov, A. P. Axiomatics for Memory Allocation, *Acta Inform.,* Vol. 6, 1976, pp. 61-75.
[4]. Glushkov V.M. Teoriya avtomatov i formalnyie preobrazovaniya mikroprogramm [Automata theory and formal transformations of microprograms]. Kybernetika [Cybernetics], 1965, N 5.
[5]. Lyapunov A.A. O logicheskih shemah programm [On the logical program schemata]. Problemy kibernetiki [Problems of cybernetics], 1958, vol. 1, p. 46-74.
[6]. Yanov Ju. I. O logicheskih shemah algoritmov [On the logical algorithm schemata]. Problemy kibernetiki [Problems of cybernetics], 1958, vol. 1, p. 75-127.
[7]. Ershov A.P. Operatornye skhemy (Ob operatornykh skhemakh Yanova) [Operating schemata (On Yanov operator shemata)]. Problemy kibernetiki [Problems of cybernetics], 1967, v. 20, p. 181-200.
[8]. Ershov A.P. Alpha – an automatic programming system of high efficiency // Journal of the Association for Computing Machinary. – 1966. – v. 13, N 1. – p. 17-24.
[9]. Glushkov V.M., Letichevsky A.A. Teoriya diskretnykh preobrazovateley [Theory of discrete transducers]. Izbrannye voprosy algebry i logiki: sb.statey.. Izbrannyje voprosy algebry I logiki [Selected problems in algebra and logics: Proceedings], 1973, Novosibirsk: Nauka, p. 5-39.
[10]. Ershov, A. P., Theory of Program Schemata. *Proc. IFIP 1971,* North-Holland, Amsterdam, pp. 144-163.
[11]. Podlovchenko R.I., Zakharov V.A. Polinomial'niy po slojnosti algoritm raspoznayushchiy ekvivalentost skhem program [Polynomial equivalence checking algorithm for program schemata], Doklady Mathematics (*Doklady Akademii Nauk)*, 1998, vol. 362, N 6, p. 27-31.
[12]. Zakharov V.A. An efficient and unified approach to the decidability of equivalence of propositional program schemes. Lecture Notes in Computer Science, 1998, v. 1443, p. 247-258.
[13]. Zakharov V.A. Byistryie algoritmyi razresheniya ekvivalentnosti operatornyih programm na uravnoveshennyih shkalah [Swift algorithms deciding equivalence of operator schemata on length-preserving frames]. Matematicheskie voprosyi kibernetiki [Mathematical Problems of Cybernetics], vol.7. – Moscow PhysMathLit, 1998, p. 303-324.
[14]. Zakharov V.A. Program equivalence checking by two-tape automata. Cybernetics and Systems Analysis. 46, № 4, p. 554-562.
[15]. Zakharov V.A., Shcherbina V.L. Ob ekvivalentnosti programm s operatorami, obladayushchimi svoystvami kommutativnosti i podavleniya [On the equivalence of programs with commutative and absorbing operators]. Materialy 9-go Mezhdunarodnogo seminara «Diskretnaya matematika i eye prilozheniya» [Proceedings of the 9-th International Workshop "Discrete mathematics and its application"], Mosocw, 2007, p. 191-194.
[16]. Harel D., Kozen D., Tiuryn J. Dynamic logic. MIT Press, Cambridge, MA, USA. – 2000. – 450 p.
[17]. Podymov V.V., Zakharov V.A. Ob odnoy polugruppovoy modeli programm, opredelyaemoy pri pomoshchi dvukhlentochnykh avtomatov [On a semigroup model of

programs specified by two-tape automata]. Nauchnye vedomosti Belgorodskogo gosudarstvennogo universiteta. Seriya Istoriya, ekonomika, politologiya, informatika [Scientific Bulletin of Belgorod State University: History, Economics, Politology, Informatics], vol 14. N 7, p. 94-101.