Применение алгоритмов проверки эквивалентности для оптимизации программ

[1, 2, 3, 4]*В.А.Захаров <zakh@cs.msu.su>*
[2, 4]*В.В. Подымов <valdus@yandex.ru>*
[1] *Институт системного программирования РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, дом 25
[2] *Московский государственный университет имени М.В. Ломоносова*
119991 ГСП-1 Москва, Ленинские горы, МГУ имени М.В. Ломоносова, 2-й учебный корпус, факультет ВМК
[3]*Московский физико-технический институт (государственный университет),*
141700, Московская область, г. Долгопрудный, Институтский пер., 9
[4]*НИУ Высшая школа экономики, Россия, Москва, 101000, ул. Мясницкая, д. 20*

Аннотация. На примере двух моделей программ показано, что задача оптимизации размера программ может быть эффективно решена при помощи процедур проверки эквивалентности программ в рассматриваемых моделях. Основной результат работы – полиномиальные по времени алгоритмы минимизации конечных детерминированных автоматов-преобразователей над конечно порожденными разрешимыми группами и схем последовательных программ, семантика которых определяется конечно порожденными разрешимыми упорядоченными левосократимыми полугруппами. Предложенные алгоритмы можно использовать в качестве теоретической основы для построения эффективных процедур глобальной оптимизации императивных и реагирующих программ.

Ключевые слова: эквивалентность программ; оптимизирующие преобразования; реагирующая программа; схема программ; разрешающий алгоритм.

DOI: 10.15514/ISPRAS-2015-27(4)-8

Для цитирования: Захаров В.А., Подымов В.В. Применение алгоритмов проверки эквивалентности для оптимизации программ. Труды ИСП РАН, том 27, вып. 4, 2015 г., стр. 145-174. DOI: 10.15514/ISPRAS-2015-27(4)-8.

1. Введение

В программировании задача проверки эквивалентности программ – для двух заданных программ выяснить, имеют ли эти программы одинаковое

поведение, – играет двоякую роль. С одной стороны, эта задача служит индикатором сложности модели вычислений, в которой формализовано понятие программы: оценивая сложность проблемы эквивалентности, можно узнать, насколько выбранная формальная модель программ пригодна для практического применения при решении других задач анализа поведения программ. Именно с этой целью проблема эквивалентности изучается в теории схем программ (см. [41,50,51]). С другой стороны, к задаче проверке эквивалентности программ сводятся многие задачи верификации и оптимизации программ. Если проблема эквивалентности имеет эффективный алгоритм решения, то его можно использовать как универсальную вспомогательную процедуру для решения многих других задач системного программирования. Цель данной статьи – продемонстрировать, как можно использовать быстрые алгоритмы проверки эквивалентности в разных формальных моделях программ для эффективной оптимизации программ.

Алгоритмы проверки эквивалентности программ имеют довольно широкую область применения в системном программировании. С их помощью удается решать некоторые задачи

- рефакторинга программ, связанные с выявлением фрагментов программного кода, реализующих известные алгоритмы [3,38];
- регрессионного анализа кода [5,11,16,21] для обнаружения ошибок, которые могут возникать в процессе модернизации программ;
- верификации и валидации оптимизирующих процедур в компиляторах [13,17,18,20,22,28,30,33,43-45];
- композиции программ [39] для корректного объединения нескольких фрагментов кода, реализующих разные функциональности, в одну программу;
- проверки безопасности потоков данных [7,36], чтобы удостовериться в отсутствии уязвимостей;
- обнаружения вредоносных фрагментов кода (программных «вирусов») [12,42].

Но наиболее важной областью применения алгоритмов проверки эквивалентности является разработка оптимизирующих преобразований программ. Исследования в этом направлении начались на рубеже 60-70 гг XX века (см. [1,6,10,14,19,46,50]) и привели к значительному прогрессу в развитии средств компиляции программ и повышении качества программного кода. Оптимизирующие преобразования программ нуждаются в проверке их

корректности, и это можно осуществить лишь при помощи средств проверки эквивалентности программ. Методы, используемые на практике для этой цели, ограничиваются анализом ациклических фрагментов программ (линейных участков) [5,15,24], конечных разверток циклов [21,31,33], или программ, выполняющих специальные преобразования, для которых удается вычислять инварианты циклов [17,23,32].

Однако алгоритмы проверки эквивалентности программ можно использовать не только как средства контроля корректности оптимизирующих преобразований, но и как средства построения самих оптимизирующих преобразований. Для пояснения этого тезиса обратимся к следующему примеру.

Одна из наиболее простых моделей вычислений – это детерминированные конечные автоматы. Задача оптимизации автоматов состоит в том, чтобы для заданного автомата A построить эквивалентный автомат B (т.е. автомат, распознающий тот же самый язык $L(A) = L(B)$), имеющий наименьшее число состояний. Решение этой задачи можно провести в три этапа.

1. Разбить состояния автомата на классы эквивалентности. Два состояния q' и q'' считаются эквивалентными, если эквивалентны автоматы $A[q']$ и $A[q'']$, имеющие эти состояния в качестве начальных. Именно для выделения классов эквивалентности состояний автомата A используется алгоритм проверки эквивалентности автоматов.
2. Склеить эквивалентные состояния. В каждом классе эквивалентности выбирается произвольное состояние, и в это состояние направляются все переходы автомата, которые ранее вели в другие состояния того же класса эквивалентности.
3. Удалить все бесполезные состояния вместе с входящими в них и исходящими из них переходами. Бесполезным считается всякое состояние, которое недостижимо из начального состояния или из которого недостижимо никакое финальное состояние. Для выявления бесполезных состояний достаточно воспользоваться общеизвестными алгоритмами теории графов.

В результате применения указанных преобразований к произвольному детерминированному конечному автомату A будет получен минимальный (по числу состояний) эквивалентный автомат B, т.е. решена задача оптимизации конечных автоматов.

В этой статье мы покажем, что та же самая стратегия решения задачи оптимизации может быть применена и к более сложным вычислительным системам, моделирующим программы. Мы ограничимся рассмотрением двух таких систем – конечных детерминированных автоматов-преобразователей и алгебраических моделей программ над полугруппами.

Автоматы-преобразователи (машины с конечным числом состояний, трансдьюсеры) возникли как обобщение конечных автоматов Рабина-Скотта. В отличие от конечных автоматов преобразователи вычисляют отношения на множествах конечных слов. Они находят применение в системном программировании для построения простейших компиляторов [2], драйверов, осуществляющих фильтрацию и преобразования строк, изображений, потоков данных [4,37], в системах автоматизированного проектирования управляющих систем для разработки контроллеров [29], в компьютерной лингвистике для создания программ распознавания речи [25] и др. Проблема эквивалентности для конечных детерминированных автоматов-преобразователей над строками была исследована в статьях [9,34,49]. В статье [26] для этого класса преобразователей был предложен алгебраический метод их минимизации.

Преобразователи также могут служить простой моделью реагирующих программ, функция которых состоит в том, чтобы вырабатывать правильные отклики в ответ на внешние воздействия. К числу программ такого рода относятся операционные системы, сетевые протоколы, драйверы, контроллеры. В этих программах разные последовательности действий могут приводить к одному и тому же результату, поэтому элементарные (базовые) действия реагирующей программы можно рассматривать как порождающие элементы некоторой полугруппы. Для конечных детерминированных автоматов-преобразователей над полугруппами в статье [49] был разработан полиномиальный по времени алгоритм проверки эквивалентности. Именно с его помощью для этой разновидности автоматов-преобразователей в данной статье будет предложен алгоритм минимизации.

Алгебраические модели программ были впервые введены в статье [53] как результат синтеза двух моделей вычислений – схем программ Ляпунова-Янова [52] (синтаксическая компонента) и дискретных преобразователей Глушкова-Летичевского [47] (семантическая компонента). Эта простая модель последовательных программ может быть использована для решения широкого спектра задач анализа поведения императивных программ, включая задачи оптимизации. В частности, в статье [54] для одного класса алгебраических моделей программ был предложен метод минимизации схем программ, основанный на построении полной системы эквивалентных преобразований.

В статье [47] было замечено, что задача минимизации схем программ тесно взаимосвязана с задачей проверки эквивалентности. В этой работе была предложена рекурсивная процедура проверки эквивалентности схем программ, в которой преобразования, направленные на минимизацию числа ветвлений переходов в состояниях схемы программы, чередуются с проверкой

эквивалентности схем программ, полученных в результате этих преобразований. В общих чертах этот метод подпадает под описанную выше схему минимизации моделей программ. Однако при отсутствии эффективного алгоритма проверки эквивалентности использовать этот метод для минимизации схем программ невыгодно – его сложность по времени экспоненциально зависит от размера оптимизируемой программы.

В данной статье мы описываем прямой (нерекурсивный) алгоритм минимизации схем программ, семантика операторов которых определяется полугруппами специального вида – левосократимыми полугруппами с неразложимой единицей. В основу этого алгоритма положены идеи, высказанные в работе [47]. Высокая эффективность нашего алгоритма обусловлена явным использованием известных эффективных алгоритмов проверки эквивалентности схем программ в полугрупповых моделях программ [40,48,55].

Статья организована следующим образом. В разделе 2 приведено описание модели вычислений конечных автоматов-преобразователей над полугруппами. В разделе 3 описан алгоритм минимизации для преобразователей, работающих над разрешимыми группами, доказана его корректность. В разделе 4 приведено описание синтаксиса и семантики схем программ в алгебраической модели программ. В разделе 5 описан алгоритм минимизации схем программ над упорядоченными полугруппами и доказана его корректность. В заключении проводится обсуждение используемого в статье подхода к решению задачи оптимизации моделей программ и оцениваются перспективы развития этого подхода и его применения для оптимизации других моделей программ.

2. Автоматы-преобразователи: основные понятия

Пусть задан конечный алфавит Σ. Записью Σ^* обозначим множество всех конечных слов над алфавитом Σ, включая пустое слово λ. Буквы алфавита Σ могут быть истолкованы как элементарные события, происходящие во внешней среде и оказывающие влияние на информационную систему, взаимодействующую с этой средой. В этом случае слова можно понимать как возможный сценарий поведения внешней среды по отношению к информационной системе.

Рассмотрим полугруппу S с бинарной операцией $\cdot$, порожденную множеством элементов B и имеющую единичный (нейтральный) элемент e. Порождающие элементы полугруппы можно рассматривать как элементарные операторы (действия), которые выполняет информационная система в ответ на внешние воздействия. Бинарная операция полугруппы в этом случае соответствует последовательной композиции этих операторов. Если две композиции операторов отвечают одному и тому же элементу полугруппы S, то это означает, что эти последовательности операторов вычисляют одинаковый результат. При этом любой элемент полугруппы может быть представлен в

виде *полугруппового выражения* – последовательной композиции базовых элементов множества B.

Конечным автоматом-преобразователем (далее просто преобразователем) над полугруппой S называется система $\pi = \langle \Sigma, S, Q, q_0, F, T \rangle$, в которой

- Q – конечное множество состояний;
- q_0 – начальное состояние, $q_0 \in Q$;
- F – подмножество финальных состояний, $F \subseteq Q$;
- $T \subseteq Q \times \Sigma \times S \times Q$ – конечное отношение переходов.

Четверки (q, a, s, q') из множества T называются *переходами*; для их обозначения используем запись $q \xrightarrow{a/s} q'$. Для каждого состояния $q, q \in Q$, запись $\pi[q]$ будет обозначать преобразователь $\langle \Sigma, S, Q, q, F, T \rangle$, в котором состояние q играет роль начального состояния. *Размером* преобразователя π называется число, равное сумме числа состояний $|Q|$ и длин полугрупповых выражений, задающих элементы s, которые приписаны переходам преобразователя.

Вычислением преобразователя π на входном слове $w = a_1 a_2 \dots a_n$ называется всякая последовательность переходов

$$run = q_i \xrightarrow{a_1/s_1} q_{i+1} \xrightarrow{a_2/s_2} \cdots \xrightarrow{a_{n-1}/s_{n-1}} q_{i+n-1} \xrightarrow{a_n/s_n} q_{i+n} .$$

Конечный преобразователь может служить моделью последовательной реагирующей программы. Каждый переход $q \xrightarrow{a/s} q'$ означает, что при получении внешнего воздействия a в состоянии управления q эта программа совершает переход в состояние управления q' и выполняет последовательность операторов s. Последовательность таких переходов образует вычисление реагирующей программы. Элемент $s = s_1 \cdot s_2 \cdots s_n$ полугруппы S называется *образом* входного слова w, а пара (w, s) называется *меткой* вычисления run. Запись $q_i \xrightarrow{w/s} q_{i+n}$,будет использоваться для сокращенного обозначения вычисления преобразователя на входном слове w. Если $q_i = q_0$, то вычисление run называется *начальным вычислением*. Если $q_{i+n} \in F$, то вычисление run называется *финальным вычислением*. *Полным вычислением* называется всякое вычисление преобразователя, которое является как начальным, так и финальным. Записью $Lab(\pi)$ обозначим множество меток (w, s) всех полных вычислений преобразователя π; это множество является отношением, которое реализует преобразователь π. Запись $Lang(\pi)$ будем использовать для обозначения множества всех слов, на которых преобразователь π имеет полное вычисление, т.е. $Lang(\pi) = \{w : (w, \alpha) \in Lab(\pi)\}$. Очевидно, для любого конечного автомата-преобразователя π множество $Lang(\pi)$ является регулярным языком.

Преобразователи π' и π'' считаются *эквивалентными* (и обозначаются записью $\pi' \sim \pi''$), если $Lab(\pi') = Lab(\pi'')$. Образ входного слова представляет собой результат вычисления реагирующей программы в ответ на последовательность воздействий среды. Внешний наблюдатель регистрирует результаты лишь тех вычислений, которые достигают финальных состояний управления. Поэтому наблюдаемое поведение реагирующей программы π полностью определяется множеством меток $Lab(\pi)$. Таким образом, эквивалентность преобразователей означает, что соответствующие реагирующие программы имеют одинаковое наблюдаемое поведение. Преобразователь считается *минимальным* (по числу состояний), если не существует эквивалентных ему преобразователей с меньшим числом состояний.

Состояние q преобразователя π считается *полезным*, если хотя бы одно полное вычисление проходит через это состояние. В противном случае состояние будем называть *бесполезным*. Преобразователь считается *детерминированным*, если для любой буквы a и для любого состояния q множество T содержит не более одного перехода вида $q \xrightarrow{a/s} q'$. Для детерминированного преобразователя π введем обозначение

$$\pi(w) = \begin{cases} s, \text{если } (w,s) \in Lab(\pi) \\ \text{неопределено иначе} \end{cases}.$$

Таким образом, два детерминированных преобразователя π' и π'' эквивалентны в том и только том случае, если равенство $\pi'(w) = \pi''(w)$ выполняется для любого слова w.

В статье [49] предложен общий метод построения разрешающих процедур для задач проверки эквивалентности детерминированных преобразователей на полугруппами S , вложимыми в конечно порожденные разрешимые группы. Полугруппа S называется *вложимой* в группу G, если эта группа содержит подполугруппу S' , изоморфную полугруппе S. Полугруппа называется разрешимой, если в ней разрешима *проблема равенства слов*, т.е. существует алгоритм, который для любой пары выражений, составленных из порождающих элементов полугруппы, может определить, представляют ли эти выражения один и тот же элемент полугруппы. В статье [49] также показано, что если проблема равенства слов в группе G разрешима за полиномиальное время, то и проблема эквивалентности конечных детерминированных преобразователей над полугруппой S разрешима за полиномиальное время. Опираясь на этот результат, мы покажем, как можно эффективно решить задачу минимизации конечных детерминированных преобразователей, применяя ту же самую стратегию, какую используют для минимизации детерминированных конечных автоматов. В этой статье при решении этой задачи мы будем полагать, что полугруппа S является группой,

т.е. $S = G$. Для всякого элемента s группы G запись s^- будет обозначать элемент группы G, обратный элементу s.

3. Минимизация детерминированных автоматов-преобразователей над группами

В предложенном методе минимизации автоматов преобразователей используется процедура проверки эквивалентности, однако, конкретный вид этой процедуры не имеет существенного значения. Фактически, эта процедура используется как «черный ящик» или оракул.

Пусть задан конечный детерминированный автомат-преобразователь $\pi = \langle \Sigma, G, Q, q_0, F, T \rangle$. Два состояния q' и q'' назовем *подобными*, если существует такой элемент s группы G, что для любого слова w выполняется равенство $s \cdot \pi[q'](w) = \pi[q''](w)$. Этот элемент s будем называть *балансиром* для пары подобных состояний q' и q''. Состояния q' и q'' назовем *сбалансированными*, если они подобны с балансиром e. Алгоритм минимизации конечных детерминированных автоматов-преобразователей над группой определяется на основании следующих утверждений.

Утверждение 1. Отношение подобия и отношение сбалансированности на множестве состояний автомата-преобразователя π являются отношениями эквивалентности.

Доказательство. Рефлексивность отношений подобия и сбалансированности очевидна. Симметричность отношения подобия имеет место ввиду того, что $s \cdot \pi[q'](w) = \pi[q''](w) \Rightarrow \pi[q'](w) = s^- \cdot \pi[q''](w)$. Поскольку $e^- = e$, отсюда следует симметричность отношения сбалансированности. Транзитивность отношения подобия следует из соотношения

$$s_1 \cdot \pi[q'](w) = \pi[q''](w) \ \wedge \ s_2 \cdot \pi[q''](w) = \pi[q'''](w) \Rightarrow s_2 s_1 \cdot \pi[q'](w) = \pi[q'''](w).$$

Поскольку $e \cdot e = e$, отсюда следует транзитивность отношения сбалансированности. QED

Утверждение 2. Если q' и q'' – пара подобных состояний с балансиром s, и при этом состояние q' является финальным, то состояние q'' также является финальным, а s – нейтральный элемент e группы G.

Доказательство. Следует из того факта, что всякое состояние q является финальным тогда и только тогда, когда значение $\pi[q](\lambda)$ определено и при этом равно e. QED

Утверждение 3. Если группа G является разрешимой, то отношение подобия состояний автомата-преобразователя также разрешимо. Более того, если проблема равенства слов в группе G разрешима за полиномиальное время, то

и подобие состояний конечного детерминированного автомата-преобразователя можно проверить за полиномиальное время.

Доказательство. Пусть q' и q'' – пара состояний конечного детерминированного автомата-преобразователя $\pi = \langle \Sigma, G, Q, q_0, F, T\rangle$. Если $Lang(\pi[q']) \neq Lang(\pi[q''])$, то состояния q' и q'', очевидно, не являются подобными. Если $Lang(\pi[q']) = Lang(\pi[q'']) = \emptyset$, то состояния q' и q'', очевидно, являются подобными, и любой элемент группы G (в т.ч. нейтральный) может служить балансиром пары q', q''. В заключение рассмотрим случай, когда $Lang(\pi[q']) = Lang(\pi[q'']) = L \neq \emptyset$. Возьмем произвольное слово $w \in L$. Пусть $\pi[q'](w) = \alpha, \pi[q''](w) = \beta$. Тогда состояния q' и q'' могут быть подобными в том и только том случае, когда равенство $\beta \cdot \alpha^- \cdot \pi[q'](u) = \pi[q''](u)$ выполняется для любого слова u. Чтобы проверить это равенство, обратимся к следующей паре преобразователей $\pi' = \langle \Sigma, S, Q \cup \{q\}, q, F, T'\rangle$ и $\pi'' = \langle \Sigma, S, Q \cup \{q\}, q, F, T''\rangle$, где $q \notin Q$, $T' = T \cup \{(q, a, \beta \cdot \alpha^-, q')\}$, $T'' = T \cup \{(q, a, e, q'')\}$, и a – произвольная буква алфавита Σ. Из определения этих преобразователей видно, что $Lang(\pi') = Lang(\pi'') = aL$, и для любого слова u справедливы равенства $\pi'(au) = \beta \cdot \alpha^- \cdot \pi[q'](u)$ и $\pi''(au) = \pi[q''](u)$. Таким образом, для проверки подобия состояний q' и q'' достаточно проверить эквивалентность преобразователей π' и π''. Проверку эквивалентности преобразователей, работающих над разрешимой группой G, можно провести, используя метод, описанный в статье [49].

QED

Пусть имеются преобразователь π, состояние q этого преобразователя и элемент s группы G. Обозначим записью $Tr_1(\pi, q, s)$ преобразователь, получающийся из π следующим образом:

- каждый переход вида $p \xrightarrow{x/g} q$, где $p \neq q$, заменяется на переход $p \xrightarrow{x/g \cdot s^-} q$;
- каждый переход вида $q \xrightarrow{y/h} p$, где $p \neq q$, заменяется на переход $q \xrightarrow{y/s \cdot h} p$;
- каждый переход вида $q \xrightarrow{x/h} q$ заменяется на переход $q \xrightarrow{y/s \cdot h \cdot s^-} q$.

Утверждение 4. Каковы бы ни были преобразователь $\pi = \langle \Sigma, G, Q, q_0, F, T\rangle$, состояние q, не являющееся ни начальным, ни финальным (т.е. $q \notin F \cup \{q_0\}$), и элемент $s, s \in G$, для преобразователя $\pi_1 = Tr_1(\pi, q, s)$ справедливы соотношения

$Lab(\pi) = Lab(\pi_1)$, т.е. Tr_1 – эквивалентное преобразование трансдьюсеров;

$\pi_1[q](w) = s \cdot \pi[q](w)$ для любого слова w;

$\pi_1[p](w) = \pi[p](w)$ для любого слова w и состояния p, отличного от q.

Доказательство. Поскольку состояние q не является ни начальным, ни финальным, каждое финальное вычисление преобразователя π_1, проходящее через состояние q, имеет вид $p \xrightarrow{a_1/s_1} \cdots \xrightarrow{x/g \cdot s^-} q \xrightarrow{y/s \cdot h} \cdots$. Поэтому для любого слова w и состояния p, отличного от q, добавленные элементы s^- и s на переходах, инцидентных состоянию q, взаимно поглощаются, и верно равенство $\pi_1[p](w) = \pi[p](w)$. Поскольку состояние q не является начальным, отсюда следует равенство $Lab(\pi) = Lab(\pi_1)$. Однако для финального вычисления, начинающегося из состояния q, добавленный элемент s в первом переходе не будет компенсирован, и поэтому имеет место равенство $\pi_1[q](w) = s \cdot \pi[q](w)$. QED

Следствие. Предположим, что q' и q'' – пара подобных состояний с балансиром s в преобразователе π, и при этом состояние q' не является ни начальным, ни финальным. Тогда q' и q'' сбалансированы в преобразователе $Tr_1(\pi, q', s)$.

Пусть имеются преобразователь π и пара состояний q', q'' этого преобразователя. Обозначим записью $Tr_2(\pi, q', q'')$ преобразователь, получающийся из π заменой каждого перехода $p \xrightarrow{x/g} q'$, ведущего в состояние q', на переход $p \xrightarrow{x/g} q''$.

Утверждение 5. Если состояния q' и q'' сбалансированы в преобразователе π, то $\pi \sim Tr_2(\pi, q', q'')$.

Доказательство. Пусть $\pi_1 = Tr_2(\pi, q', q'')$. Рассмотрим произвольное состояние q и покажем, что для любого слова w выполняется равенство $\pi[q](w) = \pi_1[q](w)$. Для этого воспользуемся индукцией по длине слова w. Базис индукции (случай $w = \lambda$) очевидным образом следует из утверждения 2. Рассмотрим случай $w = xu$. Если в преобразователе π имеется переход $q \xrightarrow{x/g} p$, где $p \neq q'$, то этот переход сохраняется в преобразователе π_1. Поэтому $\pi[q](xu) = g \cdot \pi[p](u) = g \cdot \pi_1[p](u) = \pi_1[q](xu)$. Если в преобразователе π имеется переход $q \xrightarrow{x/g} q'$, то в преобразователе π_2 вместо этого перехода используется переход $q \xrightarrow{x/g} q''$. Учитывая, что $\pi[q'](u) = \pi[q''](u)$, можем получить следующую цепочку равенств $\pi[q](xu) = g \cdot \pi[q'](u) = g \cdot \pi[q''](u) = g \cdot \pi_1[q''](u) = \pi_1[q](xu)$. QED

Пусть имеются преобразователь $\pi = \langle \Sigma, G, Q, q_0, F, T\rangle$ и состояние q, $q \in Q \setminus \{q_0\}$. Обозначим записью $Tr_3(\pi, q)$ преобразователь, получающийся из π удалением состояния q и всех переходов, входящих в это состояние и исходящих из него. Очевидно следующее

Утверждение 6. Если преобразователь π имеет бесполезное состояние q, отличное от начального, то $\pi \sim Tr_3(\pi, q)$.

Назовем автомат-преобразователь *приведенным*, если он не содержит бесполезных состояний, отличных от начального, а также различных подобных состояний.

Утверждение 7. Автомат-преобразователь минимален по числу состояний тогда и только тогда, когда он приведен.

Доказательство. Необходимость. Если преобразователь π содержит бесполезное состояние q, отличное от начального, то согласно утверждению 6 эквивалентный ему преобразователь $Tr_3(\pi, q)$ имеет меньшее число состояний. Предположим, что преобразователь π содержит различные подобные состояния q' и q'' с балансиром s. Тогда хотя бы одно из этих состояний (например, q') не является начальным. Если хотя бы одно из состояний q' и q'' является финальным, то согласно утверждению 2 оба состояния являются финальными и при этом сбалансированными (т.е. $s = e$). Тогда согласно утверждениям 5 и 6 преобразователь $Tr_3(Tr_2(\pi, q', q''), q')$ эквивалентен преобразователю π и имеет меньшее число состояний. Если оба состояния q' и q'' не являются финальными, то согласно утверждениям 4-6 преобразователь $Tr_3(Tr_2(Tr_1(\pi, q', s), q', q''), q')$ эквивалентен преобразователю π и имеет меньшее число состояний.

Достаточность. Предположим противное: существуют два таких эквивалентных преобразователя $\pi = \langle \Sigma, G, Q, q_0, F, T\rangle$ и $\pi' = \langle \Sigma, G, Q', q'_0, F', T'\rangle$, что π является приведенным, π' является минимальным и при этом $|Q'| < |Q|$. Коль скоро преобразователь π' минимальный, у него нет бесполезных состояний. Так как $|Q'| < |Q|$, существует такая пара слов w_1, w_2 и три состояния $q' \in Q'$ и $q_1, q_2 \in Q$, для которых имеют место начальные вычисления

$$q'_0 \xrightarrow{w_1/g_1} q', q'_0 \xrightarrow{w_2/g_2} q', q_0 \xrightarrow{w_1/h_1} q_1, q_0 \xrightarrow{w_2/h_2} q_2.$$

Поскольку $\pi \sim \pi'$, для любого слова w верны равенства $g_1\pi'[q'](w) = h_1\pi[q_1](w)$ и $g_2\pi'[q'](w) = h_2\pi[q_2](w)$. Значит, $g_1^- h_1\pi[q_1](w) = g_2^- h_2\pi[q_2](w)$ и, следовательно, состояния q_1, q_2 преобразователя π подобны вопреки условию утверждения. Полученное противоречие свидетельствует о справедливости утверждения. QED

Итеративный алгоритм минимизации конечных детерминированных автоматов-преобразователей работает следующим образом. Для заданного преобразователя $\pi = \langle \Sigma, G, Q, q_0, F, T\rangle$ до тех пор, пока это возможно, в произвольном порядке выполняются следующие действия.

1. Выделить бесполезное состояние q, $q \neq q_0$, и применить преобразование $Tr_3(\pi, q)$.

2. Выделить пару различных сбалансированных состояний q', q'', где $q' \neq q_0$, и применить преобразование $Tr_3(Tr_2(\pi, q', q''), q')$.
3. Выделить пару различных подобных состояний q', q'' с балансиром s, где $q' \neq q_0$ и $s \neq e$, и применить преобразование $Tr_3(Tr_2(Tr_1(\pi, q', s), q', q''), q')$.

Каждое из применяемых преобразований сокращает число состояний в преобразователе π. Поэтому алгоритм обязательно завершает работу спустя не более $|Q|$ шагов. Утверждения 2, 4-6 гарантируют, что преобразователь, полученный в результате работы алгоритма, будет эквивалентен исходному, а утверждение 1 – приведенность полученного преобразователя. Поэтому, согласно утверждению 7, по окончании работы описанного алгоритма будет построен минимальный по числу состояний автомат-преобразователь, эквивалентный исходному. Для сведения проверки подобия состояний преобразователя к проверке равенства слов в группе G, а также для вычисления балансира подобных состояний достаточно использовать утверждение 3. Таким образом, верна

Теорема 1. Если проблема равенства слов в группе G разрешима за полиномиальное время, то существует полиномиальный по времени алгоритм минимизации конечных детерминированных преобразователей над группой G.

4. Схемы последовательных программ: основные понятия

Схемы последовательных программ, являющиеся основным объектом исследований теории схем программ, допускают разные формы представления; для их описания можно использовать алгебраические формулы [52], размеченные графы или системы переходов специального вида [47,51,53]. Схемы программ также можно рассматривать как детерминированные автоматы-преобразователи с конечным числом состояний, имеющие особую семантику. Как раз в таком виде схемы программ были представлены в работах [46,47]. Для поддержания синтаксического единообразия анализируемых моделей вычислений мы воспользуемся в данной статье именно этой формой представления схем программ.

Детерминированный автомат-преобразователь $\pi = \langle \Sigma, S, Q, q_0, F, T\rangle$ с входным алфавитом Σ, работающий над полугруппой S, может расцениваться как *схема программ* (далее просто схема), если полагать, что

- множество состояний Q – это множество *точек ветвления* (или просто точек) программы, в которых проводится проверка логических условий;

- состояние q_0 – это *точка входа* в программу, или просто *вход* программы;
- в схеме имеется не более одного финального состояния f – это *точка выхода* из программы, или просто *выход* программы; из этой точки не исходит ни одного перехода, и в ней завершаются все результативные вычисления программы;
- алфавит Σ – это множество *значений логических условий*, в зависимости от которых осуществляется передача управления в ту или иную точку программы;
- полугруппа S – это множество *состояний данных*, с которыми работает программа;
- образующие полугруппы S – это *элементарные операторы* программы, осуществляющие преобразования данных: если $s \in S$ и a – это образующий элемент полугруппы S, то элемент $s' = s \cdot a$ нужно рассматривать как результат применения элементарного оператора a к состоянию данных s;
- полугрупповые выражения $g = a_1 \cdot a_2 \cdot \cdots \cdot a_k$ – это последовательные композиции элементарных операторов программы, которые мы будем называть *операторными цепочками*;
- каждый переход $p \xrightarrow{x/g} q$ преобразователя помечен непустой операторной цепочкой g и описывает *линейный участок* программы, в котором цепочка g выполняется в том случае, когда в точке ветвления p логические условия принимают значение x, и после выполнения этой цепочки операторов управление передается в точку ветвления q.

Вычисление схемы – это чередующаяся последовательность проверок логических условий в точках ветвления, сопровождающихся выбором соответствующих линейных участков, и выполнений операторных цепочек, приписанных этим участкам. Успешные вычисления схемы начинаются в ее входе и завершаются в ее выходе. Состояние данных, которое достигается по завершении успешного вычисления, считается результатом вычисления.

Приведенные соображения, определяющие, по сути дела, основные принципы устройства и функционирования императивных программ, могут быть более строго формализованы следующим образом. Пусть заданы алфавит (множество значений логических условий) Σ и полугруппа (множество состояний данных) S. *Интерпретацией* называется всякая функция $I: S \to \Sigma$, которая ставит в соответствие каждому состоянию данных некоторое значение логических условий. *Вычислением* схемы π в интерпретации I называется последовательность переходов этой схемы

$$run(\pi, I) = q_0 \xrightarrow{a_1/g_1} q_1 \xrightarrow{a_2/g_2} \cdots \xrightarrow{a_{n-1}/g_{n-1}} q_{n-1} \xrightarrow{a_n/g_n} \cdots,$$

удовлетворяющая двум условиям:

1. q_0 – вход схемы π;
2. для любого $i, i \geq 1$, выполняется равенство $a_i = I(e \cdot g_1 \cdot g_2 \cdot \cdots \cdot g_{i-1})$.

Вычисление, оканчивающееся в выходе $q_n = f$, называется *терминальным вычислением*; состояние данных $s_n = g_1 \cdot g_2 \cdot \cdots \cdot g_n$, с которым завершается терминальное вычисление $run(\pi, I)$, считается *результатом вычисления схемы π в интерпретации I* и обозначается записью $\pi(I)$. Если схема π не имеет терминальных вычислений в интерпретации I, то результат вычисления $\pi(I)$ считается неопределенным. Схемы π_1 и π_2 считаются *эквивалентными* (и обозначаются записью $\pi_1 \approx \pi_2$), если равенство $\pi_1(I) = \pi_2(I)$ выполняется для любой интерпретации I. Схема считается *минимальной* (по числу точек ветвления), если не существует эквивалентной ей схемы с меньшим числом точек ветвления.

Некоторые понятия, введенные для автоматов-преобразователей, сохраняют тот же смысл и для схем программ. Особо отметим лишь понятия полезности и бесполезности точки ветвления, которые для схем программ имеют несколько иное значение. Точка ветвления схемы π является *полезной*, если через нее проходит вычисление этой схемы хотя бы в одной интерпретации I, и *бесполезной* иначе. Полезность состояния автомата-преобразователя равносильна тому, что оно лежит на каком-либо пути из начального состояния в финальное. Для схем программ это в общем случае неверно.

Полугруппа S называется *упорядоченной*, если для любой пары ее элементов h, g выполнено соотношение $g \cdot h = g \Rightarrow h = e$, и *левосократимой* (или обладающей свойством левого сокращения), если для любой тройки g, h_1, h_2 ее элементов выполнено соотношение $g \cdot h_1 = g \cdot h_2 \Rightarrow h_1 = h_2$. В следующем разделе приводятся описание и анализ алгоритма минимизации схем программ над разрешимыми упорядоченными левосократимыми полугруппами. Некоторые конструкции и технические приемы этого алгоритма были предложены в работе [47], однако авторы этой работы осуществляли минимизацию схем программ по другому параметру –

быстродействию схемы. Кроме того, предложенный ими перекрестно-рекурсивный метод проводит совместное решение двух задач – минимизации и проверки эквивалентности схем программ. Процедура минимизации обращается к процедуре проверки эквивалентности при выполнении упрощающих преобразований, а для проверки эквивалентности нужно, в свою очередь, осуществить минимизацию схем программ. Такой рекурсивный алгоритм имеет экспоненциальную сложность и не представляет практической ценности. Вместе с тем, в целом ряде статей (см., например [40-42,48,55]) были предложены полиномиальные по времени алгоритмы проверки эквивалентности схем программ, работающих над упорядоченными полугруппами. Опираясь на них, мы предлагаем более простое и эффективное решение задачи минимизации схем программ. Фактически, для этого нам пришлось вычленить процедуру минимизации из решения, описанного в статье [47], совместить ее с классической процедурой минимизации конечных автоматов-распознавателей и адаптировать к явному использованию алгоритмов проверки эквивалентности схем программ.

5. Минимизация схем программ над упорядоченными левосократимыми полугруппами

В этом разделе мы рассматриваем схемы программ, работающие над разрешимой упорядоченной левосократимой полугруппой S. Как и в случае автоматов-преобразователей, процедура проверки эквивалентности схем программ используется в виде «черного ящика». Ранее мы отметили, что принадлежность точки ветвления какому-либо пути из входа в выход, вообще говоря, еще не свидетельствует о ее полезности. Однако следующее утверждение показывает, что для рассматриваемых нами полугрупп эти понятия оказываются равносильными.

Утверждение 8 [38]. Пусть S – упорядоченная левосократимая полугруппа, и в схеме $\pi = \langle \Sigma, G, Q, q_0, F, T \rangle$ имеется последовательность переходов $q_0 \xrightarrow{a_1/g_1} q_1 \xrightarrow{a_2/g_2} \cdots \xrightarrow{a_{n-1}/g_{n-1}} q_{n-1} \xrightarrow{a_n/g_n} \cdots$. Тогда существует интерпретация, в которой эта последовательность переходов является вычислением.

Следствие. Точка q схемы π полезна тогда и только тогда, когда она лежит на каком-либо пути из входа в выход.

Пусть имеются схема π и точка q этой схемы. Обозначим записью $Mod_1(\pi, q)$ схему, получающуюся из π удалением точки q и всех переходов, начинающихся или оканчивающихся в этой точке. Тогда очевидно

Утверждение 9. Если q – бесполезная точка схемы π, отличная от входа, то $\pi \approx Mod_1(\pi, q)$.

Точка q называется *существенной* в схеме π, если существуют такие интерпретации I_1, I_2, отличающиеся только значением $I_1(e) \neq I_2(e)$ в нейтральном элементе полугруппы S (начальном состоянии данных), что

результаты вычислений схемы $\pi[q]$ в этих интерпретациях различны: $\pi[q](I_1) \neq \pi[q](I_2)$. В противном случае точка q называется *фиктивной*. Пусть имеются схема π, точка q этой схемы и логическое условие a. Обозначим записью $Mod_2(\pi, q, a)$ схему, получающуюся из π следующим образом:

- если в схеме π содержится переход вида $q \xrightarrow{a/h} q'$, то каждый переход $q \xrightarrow{b/g} q''$, начинающийся в точке q, заменяется на переход $q \xrightarrow{b/h} q'$;
- в противном случае из схемы удаляются все переходы, начинающиеся в точке q.

Утверждение 10 [45]. Пусть a – произвольное логическое условие. Тогда точка q схемы π фиктивна в том и только том случае, если $\pi \approx Mod_2(\pi, q, a)$.

Пусть имеются схема π, состояние q этой схемы и логическое условие a. Обозначим записью $Mod_3(\pi, q, a)$ схему, получающуюся из π следующим образом:

- если в схеме π содержится переход вида $q \xrightarrow{a/h} q'$, то каждый переход $p \xrightarrow{b/g} q$, оканчивающийся в точке q, заменяется на переход $p \xrightarrow{b/gh} q'$;
- в противном случае из схемы удаляются все переходы, оканчивающиеся в точке q.

Утверждение 11. Пусть q – фиктивная точка схемы π, отличная от выхода, а a – логическое условие. Тогда $\pi \approx Mod_3(\pi, q, a)$.

Доказательство. Согласно утверждению 10, достаточно показать эквивалентность схем $\pi' = Mod_2(\pi, q, a)$ и $\pi'' = Mod_3(\pi, q, a)$. Рассмотрим произвольную интерпретацию I. Если схема π не содержит перехода, начинающегося в точке q и помеченного буквой a, то ни одно из вычислений $run(\pi', I)$, $run(\pi'', I)$ не достигает выхода. В противном случае вычисление $run(\pi'', I)$ получается из вычисления $run(\pi', I)$ заменой каждой пары соседних переходов $q' \xrightarrow{b/h} q \xrightarrow{c/g} q''$, проходящих через точку q, на переход $q' \xrightarrow{b/hg} q''$. Легко видеть, что в обоих случаях $\pi'(I) = \pi''(I)$. QED

Точки q_1, q_2 схем π_1, π_2 назовем *эквивалентными* (и обозначим записью $q_1 \approx_{(\pi_1,\pi_2)} q_2$), если эквивалентны схемы $\pi_1[q_1]$ и $\pi_2[q_2]$. Если $\pi_1 = \pi_2 = \pi$, то для краткости вместо $\approx_{(\pi_1,\pi_2)}$ будем писать $\approx_\pi$.

Утверждение 12. Пусть точки q, q' эквивалентны в схеме π. Тогда точка q существенна в том и только в том случае, если существенна точка q'.

Доказательство. Допустим, что точка q существенна, а точка q' - нет. Тогда существуют интерпретации I_1, I_2, отличающиеся только значением в начальном состоянии данных ($I_1(e) \neq I_2(e)$), для которых верно соотношение $\pi[q](I_1) \neq \pi[q](I_2)$. При этом $\pi[q'](I_1) = \pi[q'](I_2)$ вопреки условию эквивалентности точек q, q'. QED

Схему назовем *безызбыточной*, если:

- она не содержит бесполезных точек, кроме, быть может, входа;
- она не содержит фиктивных точек, кроме, быть может, входа и выхода;
- если ее вход фиктивен, то в него не ведет ни одного перехода.

Для безызбыточной схемы удается построить разбиение точек на классы эквивалентности, настолько же удобное для анализа, как и разбиение состояний обычного конечного автомата-распознавателя.

Утверждение 13. Пусть π – безызбыточная схема, $q \xrightarrow{a/h} p$ – переход этой схемы, и $q \approx_\pi q'$. Тогда в схеме π имеется также переход $q' \xrightarrow{a/h} p'$, причем $p \approx_\pi p'$.

Доказательство. Если вход схемы π бесполезен, то схема не содержит других точек, и справедливость утверждения очевидна. Далее полагаем, что вход схемы полезен. Из полезности точки q и утверждения 8 следует, что существует такая интерпретация I, что вычисление $run(\pi[q], I)$ результативно и начинается с перехода $q \xrightarrow{a/h} p$. Так как $q \approx_\pi q'$, вычисление $run(\pi[q'], I)$ также результативно. Пусть вычисление $run(\pi[q'], I)$ начинается с перехода $q' \xrightarrow{b/g} p'$. Тогда $b = I(e) = a$. Осталось обосновать два соотношения: $h = g$ и $p \approx_\pi p'$.

Предположим, что $h \neq g$. Возможен один из трех случаев: 1) существует такое состояние данных u, что $hu = g$; 2) существует такое состояние данных u, что $h = gu$; 3) для любого состояния данных u верны соотношения $hu \neq g$ и $h \neq gu$. Рассмотрим подробно только первый случай: $h \neq g$, $hu = g$; те же самые рассуждения можно провести и в двух других случаях. Если p – выход, то в силу упорядоченности полугруппы S вычисления $run(\pi[q], I)$, $run(\pi[q'], I)$ не могут иметь одинаковый результат. Значит, точка p не является ни входом, ни выходом и существенна. Тогда существуют две такие интерпретации I_1, I_2, различающиеся только значением в начальном состоянии данных ($I_1(e) \neq I_2(e)$), что $\pi[p](I_1) \neq \pi[p](I_2)$. Рассмотрим интерпретации I_1', I_2' со следующими свойствами: для любого состояния данных s верно $I_i'(hs) = I_i(s)$; $I_1'(e) = I_2'(e) = a$. Такие интерпретации можно корректно определить для любой упорядоченной левосократимой полугруппы S. Тогда $\pi[q](I_1') = h \cdot \pi[p](I_1) \neq h \cdot \pi[p](I_2) = \pi[q](I_2')$. Заметим, что

вычисления $run(\pi[q'], I_1')$ и $run(\pi[q'], I_2')$ начинаются с перехода $q' \xrightarrow{a/g} p'$, и для рассматриваемого случая верно равенство $run(\pi[q'], I_1') = run(\pi[q'], I_2')$. Поэтому $\pi[q'](I_1') = \pi[q'](I_2')$. Но коль скоро $q \approx_\pi q'$, справедливы также равенства $\pi[q](I_1') = \pi[q'](I_1') = \pi[q'](I_2') = \pi[q](I_2')$ вопреки полученному выше соотношению $\pi[q](I_1') \neq \pi[q](I_2')$. Этим противоречием обосновывается равенство $h = g$.

Теперь обоснуем соотношение $p \approx_\pi p'$. Предположим противное: точки p, p' не эквивалентны в схеме π. Тогда существует такая интерпретация I, что $\pi[p](I) \neq \pi[p'](I)$. Основываясь на этой интерпретации, построим интерпретацию I', обладающую следующими свойствами: для любого состояния данных s верно равенство $I'(hs) = I(s)$; $I'(e) = a$. Из эквивалентности точек q, q' следует цепочка равенств $h \cdot \pi[p](I) = \pi[q](I') = \pi[q'](I') = g \cdot \pi[p'](I)$. Ввиду того что $h = g$ и полугруппа S является левосократимой, должно быть справедливо равенство $\pi[p](I) = \pi[p'](I)$, которое противоречит установленному ранее соотношению $\pi[p](I) \neq \pi[p'](I)$. QED

Пусть имеется схема π, содержащая точки q и q'. Обозначим записью $Mod_4(\pi, q, q')$ схему, получаемую из π заменой каждого перехода $p \xrightarrow{a/h} q$, оканчивающегося в точке q, на переход $p \xrightarrow{a/h} q'$.

Утверждение 14. Пусть π – безызбыточная схема, и q, q' – ее различные эквивалентные существенные точки. Тогда $\pi \approx Mod_4(\pi, q, q')$.

Доказательство. Пусть $\pi' = Mod_4(\pi, q, q')$. Рассмотрим произвольную интерпретацию I и вычисления

$$run(\pi, I) = q_0 \xrightarrow{a_1/h_1} q_1 \xrightarrow{a_2/h_2} q_2 \xrightarrow{a_3/h_3} \dots,$$

$$run(\pi', I) = p_0 \xrightarrow{b_1/g_1} p_1 \xrightarrow{b_2/g_2} p_2 \xrightarrow{b_3/g_3} \dots.$$

Для обоснования утверждения достаточно показать, что эти вычисления имеют одинаковую длину и при этом для любого $i, i \geq 1$, справедливы соотношения $h_i = g_i$ и $q_i \approx_\pi p_i$. Воспользуемся индукцией по индексу i, предварительно заметив, что ввиду того, что схема π безызбыточна, точки $q_1, q_2, \dots, p_1, p_2, \dots$ существенны в этой схеме.

Базис индукции. Если в схеме π не содержится переход, начинающийся в точке $q_0 = p_0$ и помеченный буквой $I(e)$, то оба вычисления $run(\pi, I)$, $run(\pi', I)$ имеют длину 0. Иначе оба этих вычисления непусты, $a_1 = I(e) = b_1$, и согласно определению преобразования Mod_4 верно равенство $h_1 = g_1$. Если переход $q_0 \xrightarrow{a_1/h_1} q_1$ содержится в схеме π', то $q_1 = p_1$. Иначе $q_1 = q$ и $p_1 = q'$. В любом случае справедливо соотношение $q_1 \approx_\pi p_1$.

Индуктивный переход. Предположим, что $q_i \approx_\pi p_i$, $h_1 = g_1$, …, $h_i = g_i$, и длина хотя бы одного из вычислений $run(\pi, I)$, $run(\pi', I)$ превышает i. Для

определенности будем считать, что таковым является вычисление $run(\pi, I)$, содержащее переход $q_i \xrightarrow{a_{i+1}/h_{i+1}} q_{i+1}$ (аналогичные рассуждения применимы, если предполагать, что длина вычисления $run(\pi', I)$ превышает i). Согласно утверждению 13 схема π содержит переход $p_i \xrightarrow{a_{i+1}/h_{i+1}} p$, где $p \approx_\pi q_{i+1}$. Из равенств $a_{i+1} = h_1 \cdot \ldots \cdot h_i = g_1 \cdot \ldots \cdot g_i$ следует, что если $p \neq q$, то $p_i \xrightarrow{a_{i+1}/h_{i+1}} p$ – $(i+1)$-й по порядку переход вычисления $run(\pi', I)$, а иначе таковым является переход $p_i \xrightarrow{a_{i+1}/h_{i+1}} q'$. В любом случае длина вычисления $run(\pi', I)$ превосходит i и справедливы соотношения $g_{i+1} = h_{i+1}$, $q_{i+1} \approx_\pi p_{i+1}$. QED

Безызбыточную схему назовем *приведенной*, если она не содержит различных эквивалентных точек.

Утверждение 15. Схема программ минимальна тогда и только тогда, когда она является приведенной.

Доказательство. Необходимость. Если схема π содержит бесполезную точку q, отличную от входа, то согласно утверждению 9 схема $Mod_1(\pi, q)$ эквивалентна π и содержит меньшее число точек. Если схема π содержит фиктивную полезную точку q, отличную от входа и выхода, то согласно утверждению 8 в схеме π содержится переход $q \xrightarrow{a/h} q'$, где $q \neq q'$. Тогда согласно утверждениям 8, 9, 11 схема $Mod_1(Mod_3(\pi, q, a), q)$ эквивалентна π и содержит меньшее число точек. Если схема π безызбыточна и содержит различные эквивалентные точки q', q'', то по утверждению 12 обе эти точки существенны, и хотя бы одна из них (скажем, q') отлична от входа. Тогда согласно утверждениям 8, 9, 14 схема $Mod_1(Mod_4(\pi, q', q''), q')$ эквивалентна π и содержит меньшее число точек. Осталось рассмотреть следующий случай. Схема π: не содержит бесполезных точек, отличных от входа q_0; не содержит фиктивных точек, отличных от входа и выхода; содержит различные эквивалентные точки q', q''; не является безызбыточной. Тогда по утверждению 12 схема π содержит переход $q_0 \xrightarrow{a/h} q$, где $q \neq q_0$, и используя утверждение 11, получим безызбыточную схему $Mod_3(\pi, q_0, a)$, эквивалентную π, имеющую столько же точек и содержащую различные эквивалентные точки q', q''. Способ получения из нее эквивалентной схемы с меньшим числом состояний описан ранее.

Достаточность. Пусть $\pi = \langle \Sigma, G, Q, q_0, F, T \rangle$ – произвольная приведенная схема. Рассмотрим произвольную эквивалентную ей минимальную схему $\pi' = \langle \Sigma, G, Q', q_0', F', T' \rangle$. Как было показано выше, схема π' является приведенной. Так как $\pi \approx \pi'$, справедливо соотношение $q_0 \approx_{(\pi,\pi')} q_0'$. Тогда, используя утверждение 13, нетрудно показать, что для каждой точки схемы π существует эквивалентная ей точка схемы π', и для каждой точки схемы π' существует эквивалентная ей точка схемы π. Если схема π' содержит точку q', эквивалентную двум различным точкам q_1, q_2 схемы π, то $q_1 \approx_\pi q_2$, что

противоречит приведенности схемы π. Значит, отношение эквивалентности между точками схемы π и точками схемы π' является взаимно-однозначным соответствием. Таким образом, $|Q| = |Q'|$. QED

Итеративный алгоритм минимизации схем программ работает в два этапа: вначале для исходной схемы строится эквивалентная ей безызбыточная схема, а затем построенная безызбыточная схема преобразуется к приведенному виду.

На первом этапе, пока это возможно, для заданной схемы $\pi = \langle \Sigma, S, Q, q_0, F, T \rangle$ выполняются следующие действия.

1. Выделить бесполезную точку q, $q \neq q_0$, и применить преобразование $Mod_1(\pi, q)$.
2. Выделить полезную фиктивную точку q, $q \notin F \cup \{q_0\}$, выбрать произвольный переход $q \xrightarrow{a/h} q'$, где $q' \neq q$, и применить преобразование $Mod_1(Mod_3(\pi, q, a), q)$.
3. Если вход схемы полезен и фиктивен, то выбрать переход $q_0 \xrightarrow{a/h} q$, где $q \neq q_0$, и применить преобразование $Mod_3(\pi, q_0, a)$.
4. Если вход схемы бесполезен, то удалить все переходы, начинающиеся во входе.

Действия 3, 4 выполняются не более одного раза. Каждое из действий 1, 2 сокращает число точек схемы. Поэтому первый этап минимизации завершается спустя не более $|Q|$ шагов. Согласно утверждениям 8, 9 и 11 в результате выполнения действий первого этапа будет построена безызбыточная схема, эквивалентная исходной схеме. На основании утверждения 8 проверка полезности точки сводится к проверке достижимости вершин в графе переходов схемы. Также утверждением 8 гарантируется наличие переходов, требуемых в действиях 2 и 3. Из утверждения 10 следует, что проверка фиктивности точки q в схеме π сводится к проверке эквивалентности схем π и $Mod_2(\pi, q, a)$.

На втором этапе для безызбыточной схемы $\pi = \langle \Sigma, S, Q, q_0, F, T \rangle$ выполняется одно и то же действие, пока оно может быть выполнено: выделить пару различных эквивалентных точек q', q'', где $q' \neq q_0$, и применить преобразование $Mod_1(Mod_4(\pi, q', q''), q')$. Согласно определению безызбыточности и утверждению 12 обе точки q', q'' отличны от входа и выхода и существенны. Утверждения 9 и 14 гарантируют эквивалентность исходной и преобразованной схем. Применяемое преобразование сокращает число состояний в схеме. Поэтому второй этап минимизации схемы завершается спустя не более $|Q|$ шагов. По завершении второго этапа

получается приведенная схема. Ее минимальность гарантируется утверждением 15. Таким образом, справедлива

Теорема 2. Если проблема эквивалентности схем программ над упорядоченной левосократимой полугруппой S разрешима за полиномиальное время, то существует полиномиальный по времени алгоритм минимизации схем программ над полугруппой S.

6. Заключение

Мы показали, что для некоторых моделей программ эффективные алгоритмы решения задачи минимизации размеров программ могут быть легко построены на основе алгоритмов проверки эквивалентности программ. Это свидетельствует о важной роли, которую могут сыграть алгоритмы проверки эквивалентности программ в решении задач глобальной оптимизации программ. Полученные результаты открывают новое направление исследований проблемы эквивалентности в теории моделей программ – разработка методов применения процедур проверки эквивалентности схем программ для построения оптимизирующих преобразований программ. На следующем этапе исследований можно предпринять попытку усилить утверждения теорем 1 и 2 настоящей статьи. В разделе 3 описан алгоритм минимизации конечных детерминированных автоматов-преобразователей, работающих над разрешимыми группами. Вероятно, этот алгоритм можно распространить и на класс автоматов-преобразователей, работающих над полугруппами, вложимыми в разрешимые группы. Именно для этого класса преобразователей в статье [49] была доказана разрешимость проблемы эквивалентности, и поэтому здесь для решения задачи минимизации также можно воспользоваться процедурами проверки эквивалентности. В разделе 5 предложен алгоритм минимизации схем программ, работающие над разрешимой упорядоченными левосократимыми полугруппами. Однако в статьях [40,42,48] были предложены полиномиальные по времени алгоритмы проверки эквивалентности схем программ, работающих над другими классами полугрупп; эти алгоритмы также можно было бы использовать в качестве вспомогательного средства для разработки эффективных процедур минимизации схем программ.

Список литературы

[1]. Abel N.E., Bell J.R. Global optimization in compilers. Proceedings of the First USA-Japan Computer Conference. 1972.
[2]. Aho A.V., Sethi R., Ullman J.D. Compilers: principles, techniques, and tools. Addison-Wesley, Reading, MA, 1986.
[3]. Alias, C., Barthou, D. On the recognition of algorithm templates. Proceedings of the 2-nd International Workshop on Compiler Optimization Meets Compiler Verification, Electronic Notes in Theoretical Computer Science, 2004, v. 82, N 2, p. 395–409.

[4]. Alur R., Cerny P. Streaming transducers for algorithmic verification of single-pass list-processing programs. Proceedings of 38th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, 2011, p. 599-610.
[5]. Arons T., Elster E., Fix L., Mador-Haim S., Mishaeli M., Shalev J., Singerman E., Tiemeyer A., Vardi M. Y., Zuck L. D. Formal verification of backward compatibility of microcode. CAV, 2005, p. 185–198.
[6]. Allen F.E., Cocke J.L. A catalogue of optimizing transformations. Design and Optimization of Compilers, Prentice-Hall, Englewood Cliffs, N.J., 1972.
[7]. Barthe, G., D'Argenio, P.R., Rezk, T. Secure information flow by self-composition. Proceedings of the 17th IEEE Workshop on Computer Security Foundations, 2004, p. 100.
[8]. Benton, N. Simple relational correctness proofs for static analyses and program transformations. Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2004, ACM, New York.
[9]. Blattner M, Head T. The decidability of equivalence for deterministic finite transducers. Journal of Computer and System Sciences, 1979, v. 19, p. 45-49.
[10]. Busam V.A., England D.E. Optimization of expressions in Fortran. Communications of the Association for Computing Machinary. 1969, v. 12, N 12, p. 666-674.
[11]. Chaki, S., Gurfinkel, A., Strichman, O. Regression verification for multi-threaded programs. Proceedings of the 13th International Conference on Verification, Model Checking, and Abstract Interpretation, 2012, p. 119-135.
[12]. Christodorescu M., Jha S., Seshia S. A., Song D., Bryant R. E. Semantic-aware malware detection. Proceedings of the 2005 IEEE Symposium on Security and Privacy, Oakland, 2005.
[13]. Dissegna, S., Logozzo, F., Ranzato, F. Tracing compilation by abstract interpretation. Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2014, p. 47-59.
[14]. 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.
[15]. Feng X., Hu A. J. Cutpoints for formal equivalence verification of embedded software. Proceedings of the 5-th ACM International Conference on Embedded Software, 2005, p. 307–316.
[16]. Godlin, B., Strichman, O. Regression verification. Proceedings of the 46th Annual Design Automation Conference, 2009, IEEE Computer Society, Washington.
[17]. Goldberg, B., Zuck, L., Barrett, C. Into the loops: practical issues in translation validation for optimizing compilers. Theoretical Computer Science, 2005, v. 132, N 1, p. 53–71.
[18]. Guo, S.-Y., Palsberg, J. The essence of compiling with traces. Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2011, p. 563-574.
[19]. Kildall G. A unified approach to global program optimization. Conference Record of the Annual ACM SIGPLAN-SIGACT Simposium on Principles of Programming Languages, 1973, p. 194-206.
[20]. Kundu, S., Tatlock, Z., Lerner, S. Proving optimizations correct using parameterized program equivalence. Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation, 2009, p. 327-337.
[21]. Lahiri S. K., Hawblitzel C., Kawaguchi M., Rebelo H. SYMDIFF: A language-agnostic semantic diff tool for imperative programs. Proceedings of the 24th International Conference on Computer Aided Verification, 2012, p. 712–717.

[22]. Le V., Afshari, M., Su, Z. Compiler validation via equivalence modulo inputs. Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2014, ACM, New York.
[23]. Lopes N. P., · Monteiro J. Automatic equivalence checking of programs with uninterpreted functions and integer arithmetic. International Journal on Software Tools for Technology Transfer, 2015, N 1, p. 1-16.
[24]. Matsumoto, T., Saito, H., Fujita, M. Equivalence checking of C programs by locally performing symbolic simulation on dependence graphs. Proceedings of the 7-th International Symposiumon Quality Electronic Design, 2006, p. 370-375.
[25]. Mohri M. Finite state transducers in language and speech processing. Computer Linguistics, 1997, v. 23, N 2.
[26]. Mohri M. Minimization algorithms for sequential transducers. Theoretical Computer Science, 2000, v. 234, p. 177-201.
[27]. Namjoshi, K.S., Zuck, L.D. Witnessing program transformations. Proceedings of the 20th International Conference on Static Analysis, 2013, Springer, Berlin, Heidelberg.
[28]. Necula, G.C. Translation validation for an optimizing compiler. Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation. 2000, p. 83-94.
[29]. Nerode A., Kohn W. Models for hybrid systems: automata, topology, controllability, observability. Cornell University, Technical Report 93-28, 1993, MIT Press, Cambridge.
[30]. Pnueli, A., Siegel, M., Singerman, E. Translation validation. Proceedings of the 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems, 1998, p. 151-166.
[31]. Ramos D. A., Engler D. R. Practical, low-effort equivalence verification of real code. CAV, 2011, p. 669–685.
[32]. Sharma R., Schkufza E., B/ Churchill, A. Aiken. Data-driven equivalence checking. Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications, 2013, p. 391-406.
[33]. Shashidhar, K.C., Bruynooghe,M., Catthoor, F., Janssens, G. Verification of source code transformations by program equivalence checking. Proceedings of the 14th International Conference on Compiler Construction. Springer, 2005, Berlin, Heidelberg.
[34]. de Souza R. On the decidability of the equivalence for k-valued transducers. Proceedings of 12th International Conference on Developments in Language Theory, 2008, p. 252-263.
[35]. Stepp, M., Tate, R., Lerner, S. Equality-based translation validator for LLVM. Proceedings of the 23rd International Conference on Computer Aided Verification, 2011, p. 737-742.
[36]. Terauchi, T.: Aiken, A. Secure information flow as a safety problem. Proceedings of the 12th International Conference on Static Analysis. 2005, p. 352-367.
[37]. Veanes M., Hooimeijer P., Livshits B., et al. Symbolic finite state transducers: algorithms and applications. Proceedings of the 39th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, 2012.
[38]. Verdoolaege, S., Janssens, G., Bruynooghe, M. Equivalence checking of static affine programs using widening to handle recurrences. Proceedings of the 21st International Conference on Computer Aided Verification, 2009, Springer, Berlin, Heidelberg.
[39]. Yang W., Horowitz S., Reps T. A program integration algorithm that accommodates semantics-preserving transformations. ACM Transactions of Software Engineering and Methodology, 1992, v. 1, N 3, p. 310–354.

[40]. 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.
[41]. Zakharov V.A.. The equivalence problem for computational models: decidable and undecidable cases. Lecture Notes in Computer Science, 2001, v. 2055, p. 133-153.
[42]. Zakharov V.A., Kuzurin N.N., Podlovchenko R.I., Shcherbina V.V. Using algebraic models of programs for detecting metamorphic malwares. Труды Института Системного программирования, Том 12, 2007. с. 77-94.
[43]. Zaks, A., Pnueli, A. CoVaC: Compiler validation by program analysis of the cross-product. In: Proceedings of the 15th International Symposium on Formal Methods. Springer, 2008, Berlin, Heidelberg.
[44]. Zhao, J., Nagarakatte, S., Martin, M.M.K., Zdancewic, S. Formal verification of SSA-based optimizations for LLVM. Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2013, ACM, New York.
[45]. Zuck, L., Pnueli, A., Goldberg, B., Barrett, C., Fang, Y., Hu, Y. Translation and run-time validation of loop transformations. Formal Methods for System Design, 2005, v. 27, N 3, p. 335–360.
[46]. Глушков В.М. К вопросу о минимизации микропрограмм и схем алгоритмов. Кибернетика. 1966, N 5, с. 1-3.
[47]. Глушков В.М., Летичевский А.А. Теория дискретных преобразователей. Избранные вопросы алгебры и логики: сб. статей. Новосибирск:Наука, 1973, с. 5-39.
[48]. Захаров В.А. Быстрые алгоритмы разрешения эквивалентности пропозициональных операторных программ на упорядоченных полугрупповых шкалах. Вестник Московского университета, сер. 15, Вычислительная математика и кибернетика. - 1999, N 3. с. 29-35.
[49]. Захаров В.А. Моделирование и анализ последовательных реагирующих программ. Труды Института системного программирования РАН, том 27, выпуск 2, 2015, стр. 221-250. DOI: 10.15514/ISPRAS-2015-27(2)-13
[50]. Касьянов В.Н. Оптимизирующие преобразования программ. М.: Наука, 1988. 336 с.
[51]. Котов В.Е., Сабельфельд В.К. Теория схем программ. М.:Наука, 1991. 348 с.
[52]. Ляпунов А.А. О логических схемах программ. Проблемы кибернетики, вып. 1, М.:Физматгиз, 1958, с. 46-74.
[53]. Подловченко Р.И. О минимизации схем программ с перестановочными блоками. Программирование, 2008, N 4, с. 72-77.
[54]. Подловченко Р.И. Полугрупповые модели программ. Программирование. 1981, N 4, с.3-13.
[55]. Подымов В.В., Захаров В.А. Полиномиальный алгоритм проверки эквивалентности в модели программ с перестановочными и подавляемыми операторами. Труды ИСП РАН, 2014, вып. 3, с. 145-166. DOI: 10.15514/ISPRAS-2014-26(3)-8

On the Application of Equivalence Checking Algorithms for Program Minimization

[1, 2, 3, 4]V.A.Zakharov <zakh@cs.msu.su>
[2, 4]V.V.Podymov <valdus@yandex.ru>
Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., 109004, Moscow, Russia
[2]*Lomonosov Moscow State University, 2nd Education Building, Faculty CMC, GSP-1, Leninskie Gory, Moscow, 119991, Russian Federation*
[3]*Moscow Institute of Physics and Technology, 9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia*
[4]*Higher School of Economics, National Research University, 20 Myasnitskaya Ulitsa, Moscow 101000, Russia*

Annotation. Equivalence checking algorithms found vast applications in system programming; they are used in software refactoring, security checking, malware detection, program integration, regression verification, compiler verification and validation. In this paper we show that equivalence checking procedures can be utilized for the development of global optimization transformation of programs. We consider minimization problem for two formal models of programs: deterministic finite state transducers over finitely generated decidable groups that are used as a model of sequential reactive programs, and deterministic program schemata that are used as a model of sequential imperative programs. Minimization problem for both models of programs can be solved following the same approach that is used for minimization of finite state automata by means of two basic optimizing transformations, namely, removing of useless states and joining equivalent states. The main results of the paper are Theorems 1 and 2.

Theorems 1. If $\boldsymbol{G}$ is a finitely generated group and the word problem in $\boldsymbol{G}$ is decidable in polynomial time then minimization problem for finite state deterministic transducers over $\boldsymbol{G}$ is decidable in polynomial time as well.

Theorem 2. If $\boldsymbol{S}$ is a decidable left-contracted ordered semigroup of basic program statements and the word problem in $\boldsymbol{S}$ is decidable in polynomial time then minimization problem for program schemata operating on the interpretation over $\boldsymbol{S}$ is decidable in polynomial time as well.

Keywords: program equivalence, optimizing transformations, reactive program, program scheme, decision procedure.

DOI: 10.15514/ISPRAS-2015-27(4)-8

For citation: Zakharov V.A., Podymov V.V. On the Application of Equivalence Checking Algorithms for Program Minimization. *Trudy ISP RAN/Proc. ISP RAS*, vol. 27, issue 4, 2015, pp. 145-174 (in Russian). DOI: 10.15514/ISPRAS-2015-27(4)-8.

References

[1]. Abel N.E., Bell J.R. Global optimization in compilers. Proceedings of the First USA-Japan Computer Conference. 1972.

[2]. Aho A.V., Sethi R., Ullman J.D. Compilers: principles, techniques, and tools. Addison-Wesley, Reading, MA, 1986.

[3]. Alias, C., Barthou, D. On the recognition of algorithm templates. Proceedings of the 2-nd International Workshop on Compiler Optimization Meets Compiler Verification, Electronic Notes in Theoretical Computer Science, 2004, v. 82, N 2, p. 395–409.

[4]. Alur R., Cerny P. Streaming transducers for algorithmic verification of single-pass list-processing programs. Proceedings of 38th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, 2011, p. 599-610.

[5]. Arons T., Elster E., Fix L., Mador-Haim S., Mishaeli M., Shalev J., Singerman E., Tiemeyer A., Vardi M. Y., Zuck L. D. Formal verification of backward compatibility of microcode. CAV, 2005, p. 185–198.

[6]. Allen F.E., Cocke J.L. A catalogue of optimizing transformations. Design and Optimization of Compilers, Prentice-Hall, Englewood Cliffs, N.J., 1972.

[7]. Barthe, G., D'Argenio, P.R., Rezk, T. Secure information flow by self-composition. Proceedings of the 17th IEEE Workshop on Computer Security Foundations, 2004, p. 100.

[8]. Benton, N. Simple relational correctness proofs for static analyses and program transformations. Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2004, ACM, New York.

[9]. Blattner M, Head T. The decidability of equivalence for deterministic finite transducers. Journal of Computer and System Sciences, 1979, v. 19, p. 45-49.

[10]. Busam V.A., England D.E. Optimization of expressions in Fortran. Communications of the Association for Computing Machinary. 1969, v. 12, N 12, p. 666-674.

[11]. Chaki, S., Gurfinkel, A., Strichman, O. Regression verification for multi-threaded programs. Proceedings of the 13th International Conference on Verification, Model Checking, and Abstract Interpretation, 2012, p. 119-135.

[12]. Christodorescu M., Jha S., Seshia S. A., Song D., Bryant R. E. Semantic-aware malware detection. Proceedings of the 2005 IEEE Symposium on Security and Privacy, Oakland, 2005.

[13]. Dissegna, S., Logozzo, F., Ranzato, F. Tracing compilation by abstract interpretation. Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2014, p. 47-59.

[14]. 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.

[15]. Feng X., Hu A. J. Cutpoints for formal equivalence verification of embedded software. Proceedings of the 5-th ACM International Conference on Embedded Software, 2005, p. 307–316.

[16]. Godlin, B., Strichman, O. Regression verification. Proceedings of the 46th Annual Design Automation Conference, 2009, IEEE Computer Society, Washington.

[17]. Goldberg, B., Zuck, L., Barrett, C. Into the loops: practical issues in translation validation for optimizing compilers. Theoretical Computer Science, 2005, v. 132, N 1, p. 53–71.

[18]. Guo, S.-Y., Palsberg, J. The essence of compiling with traces. Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2011, p. 563-574.

[19]. Kildall G. A unified approach to global program optimization. Conference Record of the Annual ACM SIGPLAN-SIGACT Simposium on Principles of Programming Languages, 1973, p. 194-206.
[20]. Kundu, S., Tatlock, Z., Lerner, S. Proving optimizations correct using parameterized program equivalence. Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation, 2009, p. 327-337.
[21]. Lahiri S. K., Hawblitzel C., Kawaguchi M., Rebelo H. SYMDIFF: A language-agnostic semantic diff tool for imperative programs. Proceedings of the 24th International Conference on Computer Aided Verification, 2012, p. 712–717.
[22]. Le V., Afshari, M., Su, Z. Compiler validation via equivalence modulo inputs. Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2014, ACM, New York.
[23]. Lopes N. P., · Monteiro J. Automatic equivalence checking of programs with uninterpreted functions and integer arithmetic. International Journal on Software Tools for Technology Transfer, 2015, N 1, p. 1-16.
[24]. Matsumoto, T., Saito, H., Fujita, M. Equivalence checking of C programs by locally performing symbolic simulation on dependence graphs. Proceedings of the 7-th International Symposiumon Quality Electronic Design, 2006, p. 370-375.
[25]. Mohri M. Finite state transducers in language and speech processing. Computer Linguistics, 1997, v. 23, N 2.
[26]. Mohri M. Minimization algorithms for sequential transducers. Theoretical Computer Science, 2000, v. 234, p. 177-201.
[27]. Namjoshi, K.S., Zuck, L.D. Witnessing program transformations. Proceedings of the 20th International Conference on Static Analysis, 2013, Springer, Berlin, Heidelberg.
[28]. Necula, G.C. Translation validation for an optimizing compiler. Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation. 2000, p. 83-94.
[29]. Nerode A., Kohn W. Models for hybrid systems: automata, topology, controllability, observability. Cornell University, Technical Report 93-28, 1993, MIT Press, Cambridge.
[30]. Pnueli, A., Siegel, M., Singerman, E. Translation validation. Proceedings of the 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems, 1998, p. 151-166.
[31]. Ramos D. A., Engler D. R. Practical, low-effort equivalence verification of real code. CAV, 2011, p. 669–685.
[32]. Sharma R., Schkufza E., B/ Churchill, A. Aiken. Data-driven equivalence checking. Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications, 2013, p. 391-406.
[33]. Shashidhar, K.C., Bruynooghe,M., Catthoor, F., Janssens, G. Verification of source code transformations by program equivalence checking. Proceedings of the 14th International Conference on Compiler Construction. Springer, 2005, Berlin, Heidelberg.
[34]. de Souza R. On the decidability of the equivalence for k-valued transducers. Proceedings of 12th International Conference on Developments in Language Theory, 2008, p. 252-263.
[35]. Stepp, M., Tate, R., Lerner, S. Equality-based translation validator for LLVM. Proceedings of the 23rd International Conference on Computer Aided Verification, 2011, p. 737-742.
[36]. Terauchi, T.: Aiken, A. Secure information flow as a safety problem. Proceedings of the 12th International Conference on Static Analysis. 2005, p. 352-367.

[37]. Veanes M., Hooimeijer P., Livshits B., et al. Symbolic finite state transducers: algorithms and applications. Proceedings of the 39th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, 2012.
[38]. Verdoolaege, S., Janssens, G., Bruynooghe, M. Equivalence checking of static affine programs using widening to handle recurrences. Proceedings of the 21st International Conference on Computer Aided Verification, 2009, Springer, Berlin, Heidelberg.
[39]. Yang W., Horowitz S., Reps T. A program integration algorithm that accommodates semantics-preserving transformations. ACM Transactions of Software Engineering and Methodology, 1992, v. 1, N 3, p. 310–354.
[40]. 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.
[41]. Zakharov V.A.. The equivalence problem for computational models: decidable and undecidable cases. Lecture Notes in Computer Science, 2001, v. 2055, p. 133-153.
[42]. Zakharov V.A., Kuzurin N.N., Podlovchenko R.I., Shcherbina V.V. Using algebraic models of programs for detecting metamorphic malwares. Труды Института Системного программирования, Том 12, 2007. с. 77-94.
[43]. Zaks, A., Pnueli, A. CoVaC: Compiler validation by program analysis of the cross-product. In: Proceedings of the 15th International Symposium on Formal Methods. Springer, 2008, Berlin, Heidelberg.
[44]. Zhao, J., Nagarakatte, S., Martin, M.M.K., Zdancewic, S. Formal verification of SSA-based optimizations for LLVM. Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2013, ACM, New York.
[45]. Zuck, L., Pnueli, A., Goldberg, B., Barrett, C., Fang, Y., Hu, Y. Translation and run-time validation of loop transformations. Formal Methods for System Design, 2005, v. 27, N 3, p. 335–360.
[46]. Glushkov V.M. K voprosu o minimizatcii microprogram i skhem algoritmov [On the minimization of microprograms and algorithm schemata]. Kibernetika [Cybernetics]. 1966, N 5, p. 1-3.
[47]. Glushkov V.M., Letichevskii A.A. Teoriya diskretnyh preobrazovateley [Theory of discrete transducers]. Izbranniye voprosy algebry i logiki [Selected issues in algebra and logics]. Nauka Publ., 1973, p. 5-39.
[48]. Zakharov V.A. Bystriye algoritmy razresheniya ekvivalentnosti propozicionalnyh operatornyh program na uporyadochennyh polugruppovyh shkalah [Fast equivalence checking algorithms for propositional sequential programs on the ordered semigroup frames]. Vestnik Moskovskogo Universiteta. Vychislitelnaya matematika i kibernetika [Moscow University Computational Mathematics and Cybernetics] . 1999, N 3. p. 29-35.
[49]. Zakharov V.A. Modelirovanije i analiz posledovatelnyh i reagirujuschih program [Modelling and analysis of sequential reactive programs]. Trudy ISP RAN [The Proceedings of ISP RAS], v. 27, issue 2, 2015, pp. 221-250. DOI: 10.15514/ISPRAS-2015-27(2)-13
[50]. Kasyanov V.N. Optimizirujuschije preobrazovanija program [Optimizing transformations of programs]. Nauka Publ., 1988. 336 p.
[51]. Kotov V.E., Sabelfeld V.K. Teorija shem program [Theory of program schemata]. Nauka Publ., 1991. 348 p.
[52]. Lyapunov A.A. O logicheskih shemah program [On logical program schemata]. Problemy kibernetiki [Problems of Cybernetics], 1958, v. 1, p. 46-74.

[53]. Podlovchenko R.I. Minimization problem for schemes of programs with commutative blocks. Programming and computer software, 2008, v. 34, N 4, c. 237-241.

[54]. Podlovchenko R.I. Polygruppovije modeli program [Semigroup models of programs]. Programmirovanije [Programming and computer software]1981, N 4, c.3-13.

[55]. Podymov V.V., Zakharov V.A. Polynomialnij algoritm proverki ekvivalentnosti program s perestanovochnimi I podavlyaemymi operatorami [A polynomial algorithm for checking the equivalence in models of programs with commutation and vast operators]. Trudy ISP RAN [The Proceedings of ISP RAS], v. 26, N 3, c. 145-166. DOI: 10.15514/ISPRAS-2014-26(3)-8