Московский государственный университет имени М.В. Ломоносова

На правах рукописи

Подымов Владислав Васильевич

Быстрые алгоритмы проверки эквивалентности программ в моделях с полугрупповой семантикой

01.01.09 – Дискретная математика и математическая кибернетика

ДИССЕРТАЦИЯ

на соискание учёной степени

кандидата физико-математических наук

Научный руководитель

д. ф.-м. н., проф.

Захаров Владимир Анатольевич

Москва – 2014

Оглавление

Глава 1

Введение

1.1. Проблема эквивалентности программ

Основная проблема, исследованию которой посвящена данная диссертация — проблема эквивалентности программ — может быть сформулирована так: выяснить, имеют ли заданные программы π_1, π_2 схожее (одинаковое, эквивалентное) поведение. Под программой может пониматься объект любой модели вычислений: машина Тьюринга, автомат, система переходов, описание на каком-либо языке программирования и так далее. Выбором модели вычислений обычно определяется и общий вид поведения программы. При этом выбор схожести поведений зависит от того, какова цель исследования проблемы эквивалентности.

Наиболее известным примером проблемы эквивалентности является проблема функциональной эквивалентности: поведение программы определяется как реализуемая ей функция преобразования входных данных в выходные данные, а эквивалентность программ — как совпадение реализуемых ими функций. Например, машиной Тьюринга реализуется вычислимая функции, а автоматом-распознавателем — функция, для каждого входного слова отвечающая, принадлежит ли оно распознаваемому автоматом языку.

Исследование функциональной эквивалентности затрудняется теоремой Райса-Успенского [94], из которой следует неразрешимость функциональной эквивалентности машин Тьюринга. Это означает, что функциональная эквивалентность в любой достаточно выразительной модели вычислений — совпадающей по возможностям с машинами Тьюринга — неразрешима. При этом в простых вычислительных моделях (например, в модели конечных автоматов-распознавателей) проблема функциональной эквивалентности может иметь до-

статочно простое решение. Но всё же большинство используемых на практике программ относятся к языкам, имеющим широкие выразительные возможности.

Другой пример известной и практически используемой эквивалентности — это бисимуляционная эквивалентность [16, 48, 56, 86]. С точки зрения бисимуляционной эквивалентности программа не вычисляет какую-либо функцию, а реагирует на сигналы, посылаемые внешним окружением. Бисимуляционная эквивалентность таких программ (реактивных, или реагирующих) означает, что программы имеют неотличимые с точки зрения внешнего наблюдателя возможности реагирования на внешние сигналы. Проблема бисимуляционной эквивалентности также может оказаться как разрешимой [16, 48, 56, 57, 86], так и неразрешимой [78, 99] в зависимости от выбора вычислительной модели.

Можно привести и много других примеров эквивалентности программ. Некоторые из них обсуждаются в разделе 1.2. Как видно из приведённых примеров, исследование проблемы эквивалентности предоставляет важную информацию о функционировании программ: как именно структура программ соотносится с их поведением. Более того, проблема эквивалентности лежит в основе многих актуальных задач исследования программ, как, например:

- оптимизация [46, 79–81, 93, 111]: преобразование программы с целью улучшения её характеристик (таких как время работы, размер описания, используемая память) — необходимо быть уверенным в том, что исходная и преобразованная программы имеют одинаковую функциональность;
- обфускация [58, 59, 70]: преобразование текста программы с целью усложнить получение информации о функциональности программы;
- рефакторинг [49, 65, 87, 98]: преобразование текста программы, направленное на улучшение понимания алгоритмов, лежащих в её основе;
- верификация [16, 48, 56, 72, 86]: проверка выполнения формальной специ-

фикации для программы с формализованной семантикой — спецификация может расцениваться как описание поведения “идеальной” программы, а верификация — как сравнение поведений идеальной и реальной программ;

- поиск вредоносного кода [54, 55, 91, 105, 108, 109]: один из популярных способов поиска — это проверка совпадения функциональности кода с сигнатурами (шаблонами, характерными особенностями) известных вредоносных программ;

Сложность решения проблемы эквивалентности позволяет судить и о сложности задач, в основе которых лежит эта проблема. Например, если проблема эквивалентности оказывается трудной, то и для основывающихся на этой проблеме задач, как правило, нельзя получить простого решения; и наоборот — если имеется простой алгоритм решения проблемы эквивалентности, то он может быть частично или целиком использован в решении таких задач. При этом для формулировки и исследования самой проблемы эквивалентности достаточно знать лишь базовую информацию о модели вычислений: каким образом строятся объекты этой модели (синтаксис программ) и каково поведение этих объектов (семантика программ).

1.2. Теория схем программ

Попытка приложить математические методы к исследованию свойств программ привела к созданию теории схем программ. Вообще говоря, согласно тезису Чёрча-Тьюринга исследование любых программ возможно в рамках модели машин Тьюринга, однако на практике это неверно: программы, написанные в рамках различных парадигм [42] и на различных языках, имеют совершенно непохожие структуру и смысл, и для их исследования разумно использовать подходы, учитывающие характерные особенности выбранных программ. Такие подходы и разрабатываются в рамках теории схем программ. Подробный обзор

развития теории схем программ до 1991 года можно найти в [17]. Далее приведён краткий обзор основных моделей, разработанных в теории схем программ, и результатов исследования проблемы эквивалентности в этих моделях.

Первая модель вычислений в теории схем программ была разработана несколько десятилетий назад и известна сейчас под названием “схемы Ляпунова-Янова” [23, 45]. В те годы появилась и начала активно развиваться область программирования вычислительных машин, и в схемах Ляпунова-Янова выделены и строго описаны общие принципы составления программ, существовавших в то время. Эти принципы позднее легли в основу парадигмы императивного программирования. В более удобной поздней формализации [4] схема Ляпунова-Янова — это граф, вершинами которого описываются инструкции реальной программы, а дугами — передача управления между инструкциями. Рассматриваются инструкции двух типов. Инструкция первого типа изменяет состояние данных программы и передаёт управление следующей инструкции — в графе ей соответствует вершина-преобразователь, помеченная операторным символом, описывающим действие данной инструкции на состояние данных. Инструкция второго типа проверяет записанное в ней условие и в зависимости от этого выбирает, какая инструкция будет выполнена следующей — в графе ей соответствует вершина-распознаватель, помеченная предикатным символом, описывающим условие инструкции. Выполнение схемы состоит в обходе графа этой схемы и накоплении последовательности операторных символов. Дуга, исходящей из распознавателя, выбирается в соответствии с выбранной перед обходом интерпретацией. Интерпретация наделяет операторные и предикатные символы смыслом, а именно определяет значение каждого предикатного символа после выполнения (то есть накопления) каждой последовательности операторных символов. Схемы Ляпунова-Янова эквивалентны, если в любой интерпретации либо обход обеих схем бесконечен, либо накопленные схемами последовательности операторных символов совпадают. Для схем Ляпунова-Янова была показана разрешимость проблемы эквивалентности [45].

После схем Ляпунова-Янова было разработано и исследовано множество других моделей и видов эквивалентности программ. Ниже приведено несколько наиболее известных примеров.

Программа в теории дискретных преобразователей [2, 3, 18–20] описывается двумя взаимодействующими автоматами: один определяет синтаксис программы, а другой — её семантику. Выбором последнего автомата определяется класс интерпретаций, для которых исследуется эквивалентность программ. Проблема эквивалентности дискретных преобразователей оказалась неразрешимой в общем случае, однако для широкого спектра семантик были разработаны алгоритмы проверки эквивалентности программ.

В модели стандартных схем программ [5, 14, 15, 83, 89] вместо операторных символов используются символы операций, и тем самым ослеживаются зависимости изменения значений переменных между инструкциями программы. Проблема эквивалентности стандартных схем также оказалась неразрешимой [83, 89] (также следует из [18, 19]).

Рекурсивные схемы [47, 69, 90, 104] подобно стандартным схемам описывают структуру и зависимости значений переменных для рекурсивных (функциональных) программ. Рекурсивные схемы более выразительны, чем стандартные схемы [85, 90], а значит, их эквивалентность также неразрешима. С другой стороны, существует подкласс рекурсивных схем, равный по выразительным возможностям схемам Ляпунова-Янова [69] — эквивалетность таких схем разрешима. Существуют и более выразительные классы рекурсивных схем с разрешимой проблемой эквивалентности [47, 69].

Для преодоления описанных результатов неразрешимости схем для них вводились и исследовались иные виды эквивалентности. Например, разрешимыми оказались проблемы: логико-термальной эквивалентности стандартных схем [1, 14, 43], сравнивающей синтаксические возможности получения значений предикатов в процессе выполнения программ; древесной эквивалентности рекурсивных схем [95], сравнивающей возможности синтаксического “разворa-

чивания" вычислений этих схем; эквивалентности схем с засылками констант [20–22, 69] — некоторые примитивы таких схем (операторы, функции) присваивают текущему состоянию данных заранее заданное значение. Были разработаны модели программ, позволяющие учитывать разнообразные семантические свойства примитивов этих программ при исследовании проблемы эквивалентности: алгебраические модели программ [25–27, 29, 34], пропозициональные модели последовательных [7] и рекурсивных [9] программ.

Теория схем программ оказалась тесно связанной с теорией автоматов. Например, в было показано, что: схемы Ляпунова-Янова моделируются конечными автоматами [96], что позволило применить к схемам техники исследования автоматов; логико-термальная эквивалентность стандартных схем программ сводится к эквивалентности двухленточных детерминированных автоматов [1]; эквивалентность детерминированных магазинных автоматов взаимно сводима с древесной эквивалентностью рекурсивных программ [61–63, 67] и эквивалентностью унарных рекурсивных схем [66]; эквивалентность многоленточных автоматов сводима к эквивалентности программ в алгебраических моделях [31], а для тривиальных алгебраических моделей верна и обратная сводимость [28]. Но наличие такого рода сводимостей не означает, что исследование схем программ можно прекращать: результаты, полученные для эквивалентности автоматов [50, 73, 76, 100, 101, 110], хотя и позволяют улучшить некоторые результаты, полученные для схем программ, но не дают исчерпывающих ответов об устройстве схем и сложности проблемы эквивалентности.

Синтаксис моделей вычислений в теории схем программ, как правило, задаётся таким образом, чтобы в точности описать исследуемые структурные особенности "реальных" программ. Тогда можно говорить об аппроксимации эквивалентности программ эквивалентностью схем: если схемы эквивалентны, то и программы, структуру которых описывают эти схемы, также эквивалентны. Все упомянутые в данном разделе виды эквивалентности схем так или иначе аппроксимируют функциональную эквивалентность программ: некоторые без-

условно (эквивалентность схем Ляпунова-Янова, стандартных и рекурсивных схем, логико-термальная эквивалентность, древесная эквивалентность), некоторые — в разумных допущениях о семантике программ (эквивалентность схем в модели, учитывающей какие-либо семантические свойства программ, аппроксимирует функциональную эквивалентность программ, обладающих этими свойствами). Таким образом, исследование эквивалентности схем программ позволяет в обход теоремы Райса-Успенского описывать достаточные условия функциональной эквивалентности программ.

1.3. Быстрые алгоритмы проверки эквивалентности

С момента своего создания область программирования претерпела значительные изменения. По мере развития принципов написания программ создаются новые языки и парадигмы программирования. С расширением области применения программ для решения всевозможных вычислительных задач возникают новые вопросы исследования программ и меняется актуальность уже существующих вопросов. В начале развития программирования активно проводилось разделение свойств программ на разрешимые и неразрешимые. Затем теоретические вопросы и практические потребности программирования всё более сближались: теоретические результаты применялись в решении важных практических задач (например, в оптимизирующих компиляторах [46]), на основе теоретических исследований возникали новые практические задачи (так, например, возникла верификация программ на моделях, или model checking [16, 48, 56]).

Одной лишь разрешимости теоретической проблемы недостаточно для того, чтобы получить на её основе решение практической задачи: попытка реализовать решающий алгоритм, имеющий высокую сложность, неизбежно приведёт к нехватке вычислительных ресурсов или времени. На практике применяются только алгоритмы, решающие задачу за “разумное” время и расходующие

“разумный” объём ресурсов. Такие алгоритмы часто называют “эффективными” или “быстрыми”.

Является ли конкретный алгоритм быстрым, зависит в числе прочего от решаемой задачи, развития вычислительной техники и теоретических оценок сложности, полученных при исследовании задачи. Так, например, алгоритмы, используемые в SAT-солверах [74, 82, 84, 88], не завершают свою работу даже за полиномиальное время, но при этом используются на практике и считаются быстрыми. С другой стороны, время работы алгоритмов, применяемых в оптимизирующих компиляторах, как правило есть полином невысокой степени. Третья, наиболее популярная и используемая в данной диссертации трактовка понятия “быстрый” означает полиномиальность времени работы алгоритма [68].

Нахождение быстрых алгоритмов проверки эквивалентности программ затрудняется следующим результатом: проблема эквивалентности схем Ляпунова-Янова с бесконечной сигнатурой (то есть возможностью использования неограниченного числа различных операторных и предикатных символов) co-NP-полна [77]. Это означает, что разработка быстрых алгоритмов проверки эквивалентности в моделях вычислений, превосходящих схемы Ляпунова-Янова по выразительным возможностям, крайне затруднительна. С другой стороны, из возможности моделирования схем Ляпунова-Янова автоматами [96] и известных результатов теории автоматов [76] следует полиномиальная разрешимость проблемы эквивалентности схем Ляпунова-Янова с конечной сигнатурой. Следуя работе [7], назовём вклад сигнатуры в сложность проблемы эквивалентности комбинаторной сложностью, а вклад, вносимый структурой схем, — структурной сложностью. Тогда co-NP-трудность эквивалентности схем Ляпунова-Янова с бесконечной сигнатурой обеспечивается только комбинаторной сложностью, тогда как структурная сложность проблемы оказывается низкой. В данной диссертации исследуется структурная сложность проблемы эквивалентности в моделях программ. Формально это выражается в том, что все рассматриваемые далее модели имеют конечную сигнатуру.

Актуальность нахождения быстрых алгоритмов проверки эквивалентности программ подтверждается и большим числом работ, посвящённых описанию таких алгоритмов для разных вычислительных моделей, например: конечных автоматов-распознавателей [76]; размеченных систем переходов [16, 48, 56, 71]; трансдьюсеров [64, 107]; односчётчиковых автоматов [52] (в том числе с реальным временем [51, 53]); магазинных автоматов [103, 106]; стандартных схем в контексте логико-термальной эквивалентности ([43, 97], затем улучшено в [11]); полугрупповых моделей программ [6–10, 30, 32, 33, 35, 36, 44, 112–115].

1.4. Результаты исследования

Основная цель исследования, проводимого в рамках данной диссертационной работы, — описание быстрых алгоритмов проверки эквивалентности программ в пропозициональных моделях последовательных и рекурсивных программ. Далее в разделе приводится содержательное описание понятий, необходимых для формулировки основных результатов работы, и с использованием этих понятий формулируются положения, выносимые на защиту. Строгие формулировки понятий приведены в главе 3.

Последовательная программа может быть записана как набор инструкций двух видов: присваивания и ветвления. Инструкция присваивания изменяет текущее состояние данных программы и передаёт управление следующей инструкции. Инструкция ветвления проверяет записанный в ней предикат и в зависимости от результата проверки выбирает, какой инструкции будет передано управление; при этом состояние данных остаётся неизменным. Последовательная программа в пропозициональной модели представляет собой ориентированный граф, описывающий инструкции присваивания (вершины) и передачу управления между ними (дуги). Вершинам графа приписаны операторы. Одним оператором описываются инструкции, одинаковым образом изменяющие состояние данных. Дуги графа помечены логическими условиями, имеющими

следующий смысл: логическим условием однозначно определено значение предикатов во всех инструкциях ветвления. Например, если в программе с переменными x и y используются только предикаты $x \leq y$ и $x = y$, то в модели могут использоваться логические условия "$x < y$", "$x = y$" и "$x > y$".

Пропозициональная последовательная программа производит вычисления над (абстрактными) состояниями данных. Процесс вычисления состоит в обходе графа программы, начиная с её выделенной вершины — входа. Для детерминизации обхода необходимо задать начальное состояние данных, придать каждому оператору значение функции преобразования данных и определить, какое логическое условие выполнено в каждом состоянии данных. Все эти составляющие, детерминизирующие обход, задаются интерпретацией. Если задана интерпретация, то при обходе графа текущее состояние данных изменяется согласно значению оператора, приписанного текущей вершине, а исходящая дуга выбирается согласно логическому условию, которому удовлетворяет текущее состояние данных. Если достигнута вершина графа, помеченная как выход, то обход завершается, и получившееся состояние данных выдаётся в качестве результата вычисления. Программы эквивалентны в интерпретации, если результаты их вычислений в ней совпадают. Интерпретация может быть подобрана так, чтобы в точности описывать семантику программ в каком-либо выразительном языке программирования, и тогда сформулированная эквивалентность обязательно окажется неразрешимой. Для исследования проблемы эквивалентности в обход такой неразрешимости в пропозициональной модели производится абстракция значений операторов и логических условий. Абстракция логических условий состоит в том, что вместо одной интерпретации $\mathcal{I}$ рассматривается класс интерпретацией, получаемых из $\mathcal{I}$ приданием всевозможных значений логическим условиям. Такой класс интерпретаций образует шкалу: шкалой определяются значения операторов, при этом логические условия могут иметь любое значение. Эквивалентностью программ на шкале $\mathcal{F}$ (коротко, $\mathcal{F}$-эквивалентностью) объявляется их эквивалентность в каждой интерпретации, основанной на этой

шкале. Абстракция значений операторов состоит в переходе от исходной шкалы $\mathcal{F}$ к новой (аппроксимирующей) шкале $\mathcal{F}'$ так, чтобы из $\mathcal{F}'$-эквивалентности программ обязательно следовала их $\mathcal{F}$-эквивалентность. Тривиальный пример такой шкалы $\mathcal{F}'$ — это шкала, согласно которой выполнение различных последовательностей операторов приводит к различным состояниям данных. Такая шкала называется свободной.

Приведём менее тривиальный пример выбора аппроксимирующей шкалы. Рассмотрим инструкцию присваивания вида $x = f(y_1, \ldots, y_k)$: вычислить функцию f на значениях переменных $y_1, \ldots, y_k$ и записать результат в переменную x. Объединив несколько таких инструкций в одну обобщённую инструкцию a: записать в переменные $x_1, \ldots, x_m$ значения, полученные с использованием переменных $y_1, \ldots, y_k$. Выделим для инструкции a множество используемых переменных $Use(a) = \{y_1, \ldots, y_k\}$ и множество изменяемых переменных $Mod(a) = \{x_1, \ldots, x_m\}$. Если для инструкций a, b справедливы равенства $Use(a) \cap Mod(b) = Mod(a) \cap Use(b) = Mod(a) \cap Mod(b) = \emptyset$, то порядок выполнения этих инструкций никак не влияет на результат. Этот факт запишем в виде соотношения коммутативности: $ab = ba$. Если для инструкций a, b справедливы соотношения $Mod(a) \subseteq Mod(b)$ и $Mod(a) \cap Use(b) = \emptyset$, то выполнение инструкции a не влияет на результат, если сразу после неё выполняется инструкция b. Этот факт запишем в виде соотношения подавления: $ab = b$. Если выбранная аппроксимирующая шкала учитывает только коммутативность некоторых пар операторов, то состояния данных этой шкалы являются частично коммутативным моноидом, образованным операторами и определяемый учитываемыми соотношениями коммутативности. Шкалу, состояния данных которой являются моноидом, образованным операторами, будем называть полугрупповой. Если добавить к рассмотрению соотношения подавления, то нами будет получена более сложная полугрупповая шкала. Такую шкалу (определяемую произвольным набором соотношений коммутативности и подавления) назовём шкалой с коммутативностью и подавлением. Можно легко убедиться, что если соотноше-

ния коммутативности и подавления получены в результате описанного анализа множеств Use и Mod, то элементы полученного моноида обязательно будут частично упорядочены относительно операции моноида. Полугрупповую шкалу с таким свойством частичной упорядоченности назовём упорядоченной. Один из результатов диссертации относится к эквивалентности пропозициональных последовательных программ на произвольных упорядоченных шкалах с коммутативностью и подавлением.

Рекурсивная программа представляет собой совокупность определений вида $F(x_1, \ldots, x_k) = t(x_1, \ldots, x_k)$. Здесь F — определяемый программой функциональный символ (коротко — заголовок), а t — (функциональный) терм, строящийся над заголовками и предопределёнными функциями. Среди предопределённых функций, как правило, особо выделяется функция ветвления $if\ x\ then\ y\ else\ z$: в зависимости от значения предиката x вернуть одно из значений y, z. Пропозициональная модель строится для унарных рекурсивных программ: все функции таких программ, кроме функции ветвления, зависят ровно от одного аргумента. Формализация произвольных рекурсивных программ в рамках пропозициональной модели также возможна: для этого достаточно записать каждую функцию $f(x_1, \ldots, x_k)$ в унарном виде $f(\overline{x})$ и соответствующим образом переопределить действие функции f. Унарная рекурсивная программа может быть легко переформулирована в терминах операторов и логических условий. Для этого достаточно переписать все термы определений программы так, чтобы они образовывали деревья ветвлений, в листьях которых записаны композиции унарных функций (проще говоря, достаточно вынести наружу проверки всех условий). Тогда дерево ветвлений может быть заменено на проверку логического условия с последующим выбором одной из композиций унарных функций. Перейти от функций к операторам можно следующим образом. Рассмотрим композицию функций вида $f_1(f_2(\ldots f_k(x)\ldots))$. Каждую предопределённую функцию $f_i(x)$ можно расценивать как оператор f_i, применяемый к текущему состоянию данных, представленному переменной x.

Каждый заголовок $f_j(x)$ заменяется на символ заголовка f_j. Тогда композиция $f_1(f_2(\dots f_k(x)\dots))$ принимает вид последовательности операторов и символов заголовков $f_k, \dots, f_2, f_1$, и каждое определение программы переписывается в виде набора записей вида $F \xrightarrow{\sigma} t$, где F — заголовок, σ — логическое условие и t — последовательность операторов и заголовков (для простоты далее будем называть её унарным термом). Совокупность таких записей и используется для описания унарной рекурсивной программы в пропозициональной модели. Чтобы определить вычисление программы, необходимо дополнить её описание запросом. Запрос в модели также представляет собой произвольный унарный терм. Вычисление унарной рекурсивной программы (записанной в операторном виде) начинается с запроса и состоит в следующем. Символы текущего терма просматриваются слева направо и применяются к текущему состоянию данных, пока они являются операторами. Если встречен заголовок программы, то он заменяется на терм в соответствии с текущим логическим условием. Так продолжается до тех пор, пока не будут устранены все заголовки и выполнены все операторы. Если вычисление завершилось, то его результатом объявляется полученное состояние данных. Эквивалентность унарных рекурсивных программ на шкале определяется так же, как и для последовательных программ.

В диссертации рассматриваются классы унарных рекурсивных программ, удовлетворяющих определённым ограничениям, а именно: программа является линейной, если в каждом терме в описании программы (в том числе в запросе) все символы, кроме, быть может, одного, являются операторами; программа является металинейной, если последнее требование применяется ко всем термам, кроме запроса.

На защиту выносятся следующие положения.

1. Разработаны полиномиальные алгоритмы проверки эквивалентности пропозициональных последовательных программ на произвольных упорядоченных шкалах с коммутативностью и подавлением.

2. Установлены достаточные условия полиномиальной разрешимости проблемы эквивалентности линейных унарных рекурсивных программ на упорядоченных полугрупповых шкалах; разработаны полиномиальные алгоритмы, проверяющие указанную эквивалентность при выполнении установленных достаточных условий.

3. Разработаны полиномиальные алгоритмы проверки эквивалентности металинейных унарных рекурсивных программ на свободных шкалах.

Полученные результаты представлены на конференции “Интеллектуальные системы и компьютерные науки” (2011 год), семинаре “Дискретная математика и её приложения” (2012 год), конференции “Проблемы теоретической кибернетики” (2014 год) и опубликованы в работах [12, 13, 37–41], четыре из них — в журналах из перечня ВАК [12, 38, 39, 41]. В работе [12] В.В. Подымову принадлежат формулировка и техника обоснования основного результата статьи. В работе [41] В.В. Подымову принадлежат формулировка и полное обоснование основного результата статьи.

1.5. Новизна полученных результатов

В данном разделе приводится обзор известных результатов, относящихся к исследованию полиномиальной разрешимости проблемы эквивалентности в рассматриваемых в диссертации моделях программ, показывающий новизну результатов, полученных в данной диссертации.

1.5.1. Последовательные программы

Первый результат, относящийся к проблеме эквивалентности в моделях последовательных программ, есть разрешимость проблемы эквивалентности схем Ляпунова-Янова [45]. Из моделирования схем автоматами [96] и известных результатов теории автоматов [76] следует полиномиальная разрешимость этой

проблемы. При этом схема Ляпунова-Янова легко (заменой предикатов на логические условия) может быть переформулирована как пропозициональная последовательная программа, и тогда эквивалентность схем в точности совпадёт с эквивалентностью программ на свободной шкале. Более близкой (как синтаксически, так и семантически) к пропозициональной модели последовательных программ является модель дискретных преобразователей. Однако из-за общности подходов, разработанных для исследования дискретных преобразователей, основанные на этих подходах решающие алгоритмы имеют высокую сложность [3].

Эквивалентность последовательных программ, учитывающая алгебраические (полугрупповые) свойства операторов, активно исследовалась в рамках алгебраических моделей программ. Синтаксис таких программ схож с синтаксисом схем Ляпунова-Янова, а семантика определяется в терминах полугрупп. Первый полиномиальный алгоритм проверки эквивалентности программ в алгебраических моделях был получен в работе [35] и учитывает только свойство коммутативности операторов. Затем в работах [7, 112] был разработан более общий подход к построению полиномиальных алгоритмов проверки эквивалентности, однако этот подход применим только к уравновешенным полугрупповым шкалам, то есть обладающим следующим свойством: если выполнение последовательностей операторов приводит в одно состояние данных, то эти последовательности имеют одинаковую длину. В частности, этот подход применим к произвольным шкалам с коммутативностью, но не позволяет учитывать подавление операторов.

В работе [10] разработан подход к построению полиномиальных алгоритмов проверки эквивалентности, применимый к ряду упорядоченных шкал, в том числе к произвольным упорядоченным шкалам с подавлением (но без коммутативности). Отдельно отметим работу [44], в которой были получены полиномиальные алгоритмы проверки эквивалентности на упорядоченных шкалах с коммутативностью и подавлением, удовлетворяющих следующим ограничени-

ям: операторы разбиваются на две непересекающиеся группы — $\mathfrak{O}_0$, $\mathfrak{O}_1$; произвольным образом выбираются пары операторов a'_0, a''_0 группы $\mathfrak{O}_0$, и для этих пар задаются соотношения коммутативности $a'_0 a''_0 = a''_0 a'_0$; для каждой пары операторов a_0, a_1, где $a_0 \in \mathfrak{O}_0$ и $a_1 \in \mathfrak{O}_1$, задаётся соотношение подавления $a_0 a_1 = a_1$; больше никаких определяющих соотношений не задано.

Результат, сформулированный в первом положении, выносимом на защиту: улучшает разрешимость проблемы эквивалентности на произвольных упорядоченных шкалах с коммутативностью и подавлением [3] до полиномиальной разрешимости; обобщает полиномиальную разрешимость проблемы эквивалентности на шкалах с коммутативностью [7, 35, 112], упорядоченных шкалах с подавлением [10] и упорядоченных шкалах с особенным образом сочетающимися коммутативностью и подавлением [44] до полиномиальной разрешимости на произвольных упорядоченных шкалах с коммутативностью и подавлением.

1.5.2. Унарные рекурсивные программы

Первое упоминание модели для рекурсивных программ — рекурсивных схем — принято относить к семинару IBM, проходившему в 1959 году; заметки, содержащие это упоминание, не были опубликованы, поэтому можно сослаться на одну из ранних работ, отсылающих к этому семинару, например, [69]. В работе [69], в частности, поставлены вопросы эквивалентности унарных рекурсивных схем и показана разрешимость проблемы эквивалентности линейных унарных рекурсивных схем (в терминах пропозициональной модели — на свободных шкалах). В том же году была показана разрешимость проблемы эквивалентности для другого широкого класса унарных рекурсивных схем — свободных рекурсивных схем [47]. В работе [66] была показана взаимная сводимость проблем эквивалентности унарных рекурсивных схем и детерминированных магазинных автоматов (линейных схем и линейных автоматов, металинейных схем и металинейных автоматов). Такая сводимость позволяет перенести результаты, полученные для детерминированных магазинных автоматов [100], на слу-

чай унарных рекурсивных схем, однако, насколько известно автору данной диссертации, вопрос о сложности проблемы эквивалентности детерминированных магазинных автоматов остаётся открытым.

В работе [9] был разработан подход к построению полиномиальных алгоримов проверки эквивалентности линейных унарных рекурсивных программ на уравновешенных полугрупповых шкалах. Результат, сформулированный во втором положении, выносимом на защиту, обобщает этот подход с уравновешенных шкал на существенно более широкий класс упорядоченных шкал.

Попытка систематического исследования проблемы эквивалентности металинейных унарных рекурсивных схем была предпринята Л.П. Лисовиком [22]. Из полученных им результатов следует разрешимость проблемы эквивалентности металинейных унарных рекурсивных программ для некоторого класса (вообще говоря неупорядоченных) шкал, и в частности, для свободной шкалы. Однако из-за общности подхода предложенные им решающие алгоритмы имеют высокую сложность. При этом полиномиальных алгоритмов проверки эквивалентности металинейных унарных рекурсивных программ ранее не существовало. Таким образом, результат, сформулированный в третьем положении, выносимом на защиту, улучшает вытекающую из многих упомянутых работ разрешимость проблемы эквивалентности металинейных унарных рекурсивных программ [22, 47, 66, 100] до полиномиальной разрешимости.

1.6. Структура работы

Данная диссертационная работа состоит из введения (глава 1), шести глав основной части (главы 2–7), заключения (глава 8) и следующего за ним списка литературы.

В главе 2 приводятся понятия и обозначения, не связанные напрямую с исследуемыми моделями программ. Это делается, в числе прочего, для выбора системы обозначений, используемых для общеизвестных математических поня-

тий, а также для введения широко используемых в тексте диссертации общих математических конструкций и для полноты описания. В главе 3 приводится строгое описание пропозициональных моделей последовательных и рекурсивных программ, рассматриваемых в данной диссертации, и формулируется понятие эквивалентности программ в этих моделях. В главе 4 приводится более подробная, чем во введении, формулировка полученных в диссертации результатов, а также обсуждается подход, использованный для получения этих результатов, — метод совместных вычислений. Обоснованию полученных результатов посвящены главы 5–7.

Глава 2

Общие понятия и обозначения

В данном разделе приведены основные математические понятия, не относящиеся к моделям программ. Это сделано по крайней мере по двум причинам: во-первых, потому что обычно существует множество способов определения таких понятий и множество систем обозначений для них, и явное описание позволяет выбрать конкретную систему определений и обозначений; во-вторых, потому что некоторые из этих понятий могут недостаточно общеизвестны, так что их строгое определение не может быть опущено. Такие понятия и обозначения вводятся для последовательностей и слов (раздел 2.1), порядков (раздел 2.2), автоматов (раздел 2.3) и моноидов (раздел 2.4) — в том числе моноидов с коммутативностью и подавлением (раздел 2.5).

2.1. Последовательности, слова

Считаем, что нет необходимости определять понятие последовательности (конечной и бесконечной). Рассмотрим последовательность $S = a_1, a_2, \ldots$. Будем использовать следующие обозначения и понятия:

- $S[i]$ — *i-й элемент* последовательности S: $S[i] = a_i$;
- $|S|$ — *длина* последовательности S: если $S = a_1, \ldots, a_k$, то $|S| = k$; если S — бесконечная последовательность, то $|S| = \infty$;
- $\lceil S \rceil$ — *прореживание* конечной последовательности S: пока в S содержится пара соседних одинаковых элементов, будем удалять один из них; $\lceil S \rceil$ — последовательность, полученная после всех таких удалений;
- S^- — *зеркальный образ* конечной последовательности S: $S^- = a_{|S|}, \ldots, a_1$;

- set_S — множество элементов, входящих в последовательность S: $\mathsf{set}_S = \{a_1, a_2, \ldots\}$;
- $\mathsf{pref}(S, i)$ — *префикс* длины i последовательности S: если $|S| \leq i$, то $\mathsf{pref}(S, i) = S$; иначе $\mathsf{pref}(S, i) = a_1, \ldots, a_i$;
- $\overline{\mathsf{pref}}(S, i)$ — последовательность, определяемая соотношением $S = \mathsf{pref}(S, i)\,\overline{\mathsf{pref}}(S, i)$;
- $\mathsf{suf}(S, i)$ — *суффикс* длины i конечной последовательности S: если $|S| \leq i$, то $\mathsf{suf}(S, i) = S$; иначе $\mathsf{suf}(S, i) = a_{|S|-i+1}, \ldots, a_{|S|}$;
- $\overline{\mathsf{suf}}(S, i)$ — последовательность, определяемая соотношением $S = \overline{\mathsf{suf}}(S, i)\,\mathsf{suf}(S, i)$.
- $\mathsf{hd}(S)$ — первый элемент последовательности S: $\mathsf{hd}(S) = \mathsf{pref}(S, 1)$;
- $\overline{\mathsf{hd}}(S)$ — последний элемент конечной последовательности S: $\overline{\mathsf{hd}}(S) = \mathsf{suf}(S, 1)$;
- $\mathsf{tl}(S)$ — последовательность S без первого элемента: $\mathsf{tl}(S) = \overline{\mathsf{pref}}(S, 1)$;
- $\overline{\mathsf{tl}}(S)$ — конечная последовательность S без последнего элемента: $\overline{\mathsf{tl}}(S) = \overline{\mathsf{suf}}(S, 1)$.

В числе прочего в дальнейшем будет использоваться следующая конструкция. Рассмотрим последовательность последовательностей $S_1, S_2, \ldots$, такую что каждая предыдущая последовательность S_i является префиксом следующей последовательности S_{i+1}, и кроме того, длина последовательностей S_i неограниченно возрастает. Тогда *пределом* последовательности $S_1, S_2, \ldots$ назовём бесконечную последовательность S, префиксами которой являются все последовательности S_i.

Если последовательность S конечна состоит из элементов множества A, обозначенного как *алфавит*, то такую последовательность будем называть *(конечным) словом* в алфавите A. Элементы алфавита будем также называть *символами* этого алфавита. Множество всех слов в алфавите A будем обозначать записью A^*. Среди всех слов выделяется *пустое слово* λ. Множество всех непустых слов в алфавите A будем обозначать записью A^+. В записи слова между его символами не ставится запятая.

2.2. Порядки

В тексте диссертации будет неоднократно встречаться следующая конструкция. Рассмотрим отображение $f : S_1 \times S_2 \to S_1$, где S_1, S_2 — произвольные множества. Этим отображением индуцируется порядок $\leq$ на множестве S_1 следующего вида: $s_1 \leq s_1'$ тогда и только тогда, когда существует элемент s_2 множества S_2, для которого верно $f(s_1, s_2) = s_1'$. Чтобы опустить многократное описание этой конструкции, определенный таким образом порядок обозначим записью $\leq_f$. Вместе с этим порядком определяются и порядки $<_f$, $>_f$, $\geq_f$, $\|_f$:

- $s_1 <_f s_2$ тогда и только тогда, когда $s_1 \leq_f s_2$ и $s_1 \neq s_2$;
- $s_1 \geq_f s_2$ тогда и только тогда, когда $s_2 \leq_f s_1$;
- $s_1 >_f s_2$ тогда и только тогда, когда $s_2 <_f s_1$;
- $s_1 \|_f s_2$ тогда и только тогда, когда ни одно из соотношений $s_1 \leq_f s_2$ и $s_1 \geq_f s_2$ не справедливо.

Отметим, что в дальнейших построениях отношение $\leq_f$ будет, как правило, являться отношением нестрогого частичного порядка. В этом случае $<_f$, $>_f$, $\geq_f$, $\|_f$ — соответственно строгий частичный порядок, строгий обратный частичный порядок, нестрогий обратный частичный порядок и отношение, описывающее все пары несравнимых элементов.

2.3. Автоматы

Термин “автомат” описывает огромное разнообразие вычислительных моделей. В данной работе этот термин будет использоваться в следующем значении. *Автомат* $\mathfrak{A}$ (конечный, размеченный, детерминированный, инициальный) включает в себя:

- входной алфавит $\mathfrak{A}.A$;
- множество меток $\mathfrak{A}.L$;
- конечное множество состояний $\mathfrak{A}.Q$;
- инициальное состояние $\mathfrak{A}.q \in \mathfrak{A}.Q$;
- функцию переходов $\mathfrak{A}.T : \mathfrak{A}.Q \times \mathfrak{A}.A \to \mathfrak{A}.Q$;
- разметку состояний $\mathfrak{A}.M : \mathfrak{A}.Q \to \mathfrak{A}.L$.

Пусть $\mathfrak{A}$ — автомат, $q \in \mathfrak{A}.Q$, $h \in \mathfrak{A}.A^*$. Тогда:

- $\mathfrak{A}(q, h)$ — состояние автомата, в которое автомат $\mathfrak{A}$ переводится словом h из состояния q: $\mathfrak{A}(q, \lambda) = q$; $\mathfrak{A}(q, ag) = \mathfrak{A}(\mathfrak{A}.T(q, a)\,, g)$;
- $\mathfrak{A}(h) = \mathfrak{A}(\mathfrak{A}.q, h)$;
- $\mathfrak{A}.M(q, h)$ — метка состояния, в которое автомат $\mathfrak{A}$ приводится словом h из состояния q: $\mathfrak{A}.M(q, h) = \mathfrak{A}.M(\mathfrak{A}(q, h))$.
- $\mathfrak{A}.M(h) = \mathfrak{A}.M(\mathfrak{A}.q, h)$.

Содержательно, работа автомата состоит в следующем. Он начинает работу в инициальном состоянии, прочитывает слова над входным алфавитом и стандартным образом изменяет состояния в соответствии с прочитанными символами. В автомате отсутствуют финальные (заключительные, принимающие)

состояния, однако вместо них есть более общая конструкция — разметка состояний. Результатом работы автомата на заданном слове служит метка состояния, в которое приводит это слово.

Частным случаем автомата в приведённом выше определении является конечный автомат-распознаватель. Чтобы обосновать этот факт, достаточно заметить, что выделить финальные (заключительные, принимающие) состояния можно с использованием всего двух меток: “входит” и “не входит”. Тогда вхождение слова в распознаваемый автоматом язык равносильно тому, что результатом работы автомата на этом слове является метка “входит”.

2.4. Моноиды

Рассмотрим произвольное множество E и бинарную операцию $\circ : E \times E \to E$ на этом множестве. Операция $\circ$ *ассоциативна*, если для любых элементов e_1, e_2, e_3 множества E верно равенство $(e_1 \circ e_2) \circ e_3 = e_1 \circ (e_2 \circ e_3)$. Множество E вместе с ассоциативной операцией $\circ$ образуют *полугруппу* $(E, \circ)$. *Элементами* такой полугруппы являются элементы множества E. *Композицией*, или, по-другому, *произведением*, элементов $e_1, \ldots, e_k$ будем называть элемент $e_1 \circ \ldots \circ e_k$.

Нейтральный элемент ϵ полугруппы $(E, \circ)$ определяется так: для любого элемента e справедливы равенства $e \circ \epsilon = \epsilon \circ e = e$. Если полугруппа содержит нейтральный элемент, то она является *моноидом*.

Элемент e^{-1} моноида $(E, \circ)$, обратный к элементу e, определяется равенствами $e \circ e^{-1} = e^{-1} \circ e = \epsilon$. Если для любого элемента моноида существует обратный к нему элемент, то такой моноид является *группой*.

Полугруппа $(E, \circ)$ *(частично) упорядочена*, если $\leq_\circ$ — отношение нестрогого частичного порядка.

Моноид является *конечнопорождённым*, если можно выделить конечное множество G его элементов (*образующих*), такое что любой элемент этого моноида представим в виде композиции элементов G. Более строго, любой элемент e

конечнопорождённого моноида $(E, \circ)$ с множеством образующих G представим в виде $e = e_1 \circ \ldots \circ e_k$, где $e_1, \ldots, e_k \in G$. Нейтральный элемент такого моноида считается представимым в виде тривиальной пустой композиции образующих.

Один из способов описания конечнопорождённого моноида — это описание с использованием определяющих соотношений. Рассмотрим алфавит A. Конечнопорождённый моноид, элементами которого являются множества слов в алфавите A, будем называть *моноидом над алфавитом A*. *Определяющим соотношением* в алфавите A назовем равенство вида $h_1 = h_2$, где $h_1, h_2 \in A^*$. Содержательно, определяющие соотношения описывают слова и подслова, неотличимые в моноиде. Более строго, слова h, g принадлежат одному элементу моноида в том и только в том случае, если одно из них может быть получено из другого конечным числом замен подслов согласно определяющим соотношениям (то есть подслова, совпадающего с левой частью соотношения, на слово правой части, и наоборот).

Дадим формальное определение конечнопорождённого моноида $\mathcal{M} = (E, \circ)$ над алфавитом A, определяемого соотношениями $\mathcal{T}$ (в этом алфавите). Рассмотрим слова h, g в алфавите A. Если они представимы в виде $h = uh_1w$, $g = uh_2w$, то будем говорить, что они находятся в отношении $h \overset{|u|,h_1,h_2}{\rightleftharpoons} g$ или, если длина подслова u не имеет значения, $h \overset{h_1,h_2}{\rightleftharpoons} g$. Если при этом $(h_1 = h_2) \in \mathcal{T}$ или $(h_2 = h_1) \in \mathcal{T}$, то будем писать $h \overset{|u|}{\rightleftharpoons} g$ и $h \rightleftharpoons g$ и говорить, что слова h, g образуют *шаг вывода*. *Выводом* назовем непустую конечную последовательность слов $h_1 \rightleftharpoons h_2 \rightleftharpoons \ldots$. Слово g назовем *выводимым* из h, если существует вывод $h \rightleftharpoons \ldots \rightleftharpoons g$. Факт такой выводимости обозначим записью $h \rightleftharpoons^* g$. Элемент $\langle h \rangle$ моноида $\mathcal{M}$, порождаемый словом h, есть множество всех слов, выводимых из h. Таким образом, $E = \{\langle h \rangle \mid h \in A^*\}$. Операция монода определяется так: $\langle h \rangle \circ \langle g \rangle = \langle hg \rangle$. Отметим, что нейтральным элементом описанного моноида является элемент $\langle \lambda \rangle$, а в качестве образующих можно взять множество $\{\langle a \rangle \mid a \in A\}$.

Среди всех конечнопорождённых моноидов над алфавитом A особо вы-

делим моноид, определяемый пустым множеством соотношений. Такой моноид назовём *свободным*. Множество элементов такого моноида имеет вид $\{\{h\} \mid h \in A^*\}$. Чтобы упростить представление элементов свободного моноида, будем считать, что элементом $\langle h \rangle$ является не множество $\{h\}$, а само слово h. С учётом такого упрощения операцией свободного моноида является *конкатенация* слов: $h \circ g = hg$.

2.5. Коммутативность и подавление

В одном из результатов данной диссертации рассматриваются определяющие соотношения особого вида — соотношения коммутативности и подавления. Рассмотрим алфавит A. Определяющее соотношение будем называть:

- соотношением *коммутативности*, если оно имеет вид $ab = ba$, где $a, b \in A$;
- соотношением *подавления*, если оно имеет вид $ab = b$, где $a, b \in A$.

Моноидом *с коммутативностью и подавлением* будем называть конечнопорождённый моноид, определяемый множеством соотношений, состоящим только из соотношений коммутативности и подавления. Для компактного и удобного описания такого моноида заметим, что каждое из соотношений $ab = ba$, $ab = b$ однозначно описывается парой символов (a, b). Заменив каждое определяющее соотношение коммутативности $ab = ba$ на пару (a, b), получим множество $\mathcal{C} \subseteq A^* \times A^*$. Сделав то же самое для соотношений подавления, получим множество $\mathcal{A} \subseteq A^* \times A^*$. Моноид с коммутативностью и подавлением, для которого таким образом получены множества $\mathcal{C}$ и $\mathcal{A}$, обозначим записью $\mathcal{M}(\mathcal{C}, \mathcal{A})$. При дальнейшем использовании записи $\mathcal{M}(\mathcal{C}, \mathcal{A})$ алфавит будет однозначно определяться контекстом и потому опущен в записи.

Шаг вывода $h \overset{i,ab,ba}{\rightleftharpoons} g$ будем называть *коммутацией* i-го и $(i+1)$-го символов; шаг вывода $h \overset{i,ab,b}{\rightleftharpoons} g$ — *порождением* i-го символа; шаг вывода $h \overset{i,b,ab}{\rightleftharpoons} g$

— *поглощением* i-го символа. Коммутацию i-го и $(i+1)$-го символов будем обозначать записями $\overset{i|}{\rightleftharpoons}$ и $\overset{|}{\rightleftharpoons}$; порождение i-го символа — записями $\overset{i^+}{\rightleftharpoons}$ и $\overset{+}{\rightleftharpoons}$; поглощение i-го символа — записями $\overset{i^-}{\rightleftharpoons}$ и $\overset{-}{\rightleftharpoons}$. Если контекстом заданы множества $\mathcal{C}$, $\mathcal{A}$, то: запись $a \leftrightsquigarrow b$ будем для удобства использовать вместо $(a,b) \in \mathcal{C}$; запись $a \not\leftrightsquigarrow b$ — вместо $(a,b) \notin \mathcal{C}$; запись $a \hookleftarrow b$ — вместо $(a,b) \in \mathcal{A}$; запись $a \not\hookleftarrow b$ — вместо $(a,b) \notin \mathcal{A}$. Если $A' \subseteq A$ то запись $a \leftrightsquigarrow A'$ будет означать $a \leftrightsquigarrow b$ для всех символов $b \in A'$.

Глава 3

Модели программ

В данной главе приводятся строгие определения, относящиеся к рассматриваемым в данной диссертации моделям программ: пропозициональным последовательным программам (раздел 3.1) и (пропозициональным) унарным рекурсивным программам (раздел 3.2). Семнатика обеих моделей описывается на основе структур динамической логики: детерминированной динамической шкалы и детерминированной динамической модели [72]. Для краткости и во избежание использования слова “модель” они будут называться шкалой и интерпретацией соответственно. Описание этих структур приводится только в разделе 3.1.2 в контексте последовательных программ и без изменений переносится на случай унарных рекурсивных программ.

Чтобы выделить слова, составленные из операторов, среди всех других слов, будем в этой и последующих главах называть их *цепочками (операторов)*.

3.1. Пропозициональные последовательные программы

В данном разделе приводятся основные определения, относящиеся к пропозициональной модели последовательных программ. Для краткости в рамках данного раздела будем называть пропозициональные последовательные программы просто программами.

В разделе 3.1.1 приведён синтаксис программ. В этом же разделе вводятся (как синтаксические конструкции) понятия пути, трассы и вычисления программы. В разделе 3.1.2 вводятся понятия шкалы и интерпретации. На основе понятия интерпретации в разделе 3.1.3 описываются семантические аспекты трасс и вычислений: реализуемость трассы и вычисления в интерпретации, результат вычисления. В разделе 3.2.3 вводятся понятия: эквивалентности программ в интерпретации; эквивалентности программ на шкале; совместности трасс про-

грамм. Понятие совместности трасс программ предоставляет альтернативную формулировку эквивалентности программ на шкале и активно используется в дальнейших рассуждениях. В разделе 3.1.5 приведены утверждения, касающиеся пропозициональных последовательных программ и не являющиеся заслугой автора данной диссертации. Эти утверждения:

- позволяют не проводить анализ реализуемости произвольных трасс программ в рассматриваемых в исследовании интерпретациях (утверждение 3.1.1);
- предоставляют простой способ проверки совместности трасс программ (утверждение 3.1.2).

3.1.1. Синтаксис

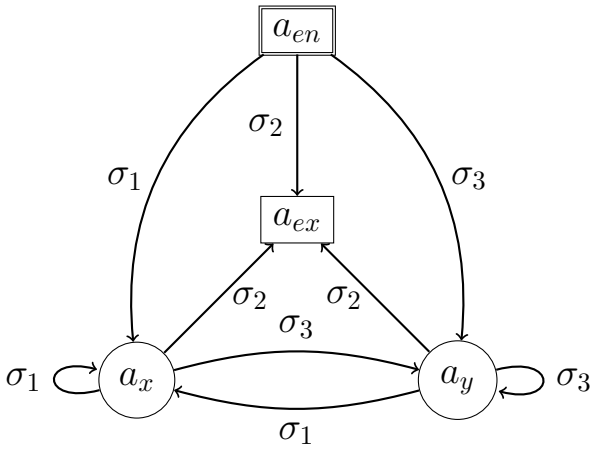
Рассмотрим конечные алфавиты операторов $\mathfrak{O}$ и логических условий $\mathfrak{L}$. Программа π, построенная над этими двумя алфавитами, состоит из:

- конечного множества *состояний управления* $\pi.L$;
- *входа* $\pi.en \in \pi.L$;
- *выхода* $\pi.ex \in \pi.L$;
- *функции переходов* $\pi.T : (\pi.L \setminus \{\pi.ex\}) \times \mathfrak{L} \to \pi.L$;
- *функции привязки* операторов к состояниям управления $\pi.B : \pi.L \to \mathfrak{O}$.

Введённое определение можно пояснить следующим примером. На рисунке 3.1(a) приведён фрагмент программы на языке Си, вычисляющий наиболее общий делитель переменных x, y и записывающий результат в эти переменные. На рисунке 3.1(b) приведена пропозициональная последовательная программа, описывающая структуру этого фрагмента. Кругами и прямоугольниками обозначены состояния управления программы. Состояния, обозначенные кругами,

```
while(x != y) {
  if(x > y) x = x - y;
  else      y = y - x;
}
```

(a) Алгоритм Эвклида



(b) Пропозициональная последовательная программа

Рис. 3.1. Пример построения пропозициональной последовательной программы

отвечают выполнению инструкций "$x = x - y$" и "$y = y - x$" исходного фрагмента. Функцией привязки этим состояниям приписаны операторы a_x и a_y соответственно. Кроме того, в программу добавлены вход (двойной прямоугольник) и выход (одинарный прямоугольник) — они соответствуют входу во фрагмент и выходу из него. Дугами на рисунке изображена функция переходов. Логические условия σ_1, σ_2, σ_3, которыми помечены дуги, отвечают разбиению пространства состояния данных (пар целых чисел) согласно соотношениям "$x > y$", "$x = y$", "$x < y$".

Под *размером* $|\pi|$ программы π будем понимать количество ее состояний управления: $|\pi| = |\pi.L|$. Такого определения размера программы будет достаточно для обоснования полиномиальности времени работы алгоритмов, решающих проблему эквивалентности.

Для удобства вместо $\pi.T(l, \sigma) = l'$ будем писать $l \xrightarrow{\sigma}_\pi l'$ или, если логическое условие не имеет значения, $l \to_\pi l'$. *Путь* pt программы π — это любая непустая последовательность ее состояний управления, в которой все соседние состояния l, l' связаны отношением $l \to_\pi l'$:

$$pt = pt\,[1] \to_\pi pt\,[2] \to_\pi \ldots\ .$$

Путь, который нельзя продолжить (т.е. бесконечный либо оканчивающийся в выходе), будем называть *полным*. Чтобы выделить состояние управления, в котором начинается путь pt, будем его называть $\mathsf{hd}(pt)$-путем. *Трассой* программы будем называть ее $\pi.en$-путь. Полную трассу будем называть *вычислением*. Если путь pt программы π конечен, то записью $\pi.B(pt)$ будем обозначать цепочку $\pi.B(pt\,[1])\,\pi.B(pt\,[2])\ldots\pi.B(pt\,[|pt|])$. Чтобы еще более сократить нотацию, состояние данных $[\![\pi.B(pt)]\!]$ будем обозначать записью $[\![pt]\!]$. Состояние управления l будем называть *завершаемым* в программе π, если существует конечный полный l-путь этой программы.

Замечание. *Термин "вычисление" принято относить к семантике программ, а не к синтаксису, однако общая структура вычисления может быть описана на синтаксическом уровне и потому приведена в данном разделе. Семантический аспект вычисления — реализуемость его в интерпретации — обсуждается далее, в разделе 3.1.3.*

3.1.2. Интерпретации

Рассмотрим конечный алфавит операторов $\mathfrak{O}$. Шкала $\mathcal{F}$ над алфавитом $\mathfrak{O}$ включает в себя:

- множество *состояний данных* $\mathcal{F}.S$;
- *начальное* состояние данных $\mathcal{F}.s \in \mathcal{F}.S$;
- *функцию преобразования* данных $\mathcal{F}.R : \mathcal{F}.S \times \mathfrak{O} \to \mathcal{F}.S$.

Добавим к рассмотрению конечный алфавит логических условий $\mathfrak{L}$. Интерпретация $\mathcal{I}$ над алфавитами $\mathfrak{O}$, $\mathfrak{L}$ состоит из:

- шкалы $\mathcal{I}.\mathcal{F}$ над алфавитом $\mathfrak{O}$;
- *оценки логических условий* $\mathcal{I}.\xi : \mathcal{I}.\mathcal{F}.S \to \mathfrak{L}$.

Таким образом, шкала $\mathcal{F}$ придаёт каждому оператору a значение функции преобразования состояний данных, а интерпретация $\mathcal{I}$, помимо этого, оценивает истинность логических условий в каждом состоянии данных:

$$\overline{a}(s) = \mathcal{F}.R(s, a)\,, \qquad \overline{p}(s) = (p \in \mathcal{I}.\xi(s))\,.$$

Далее будут рассматриваться только шкалы над алфавитом операторов $\mathfrak{O}$ и интерпретации над алфавитами операторов $\mathfrak{O}$ и логических условий $\mathfrak{L}$, поэтому упоминание этих алфавитов при задании шкал и интерпретаций будет опускаться. Интерпретации, включающие в себя шкалу $\mathcal{F}$, будем называть *основанными на шкале* $\mathcal{F}$, а также $\mathcal{F}$*-интерпретациями.*

Рассмотрим шкалу $\mathcal{F}$. Обобщим функцию $\mathcal{F}.R$ с множества $\mathfrak{O}$ на множество $\mathfrak{O}^*$:

$$\begin{aligned} \mathcal{F}.R^*(s, \lambda) &= s, \\ \mathcal{F}.R^*(s, ha) &= \mathcal{F}.R(\mathcal{F}.R^*(s, h)\,, a)\,, \text{ где } a \in \mathfrak{O} \text{ и } h \in \mathfrak{O}^*. \end{aligned}$$

Содержательно, значением $\mathcal{F}.R^*(s, g)$ является состояние данных шкалы $\mathcal{F}$, в которое приводит выполнение цепочки g из состояния s. Если контекстом однозначно определяется шкала $\mathcal{F}$, то вместо $\mathcal{F}.R^*(\mathcal{F}.s, g)$ будем писать $[\![g]\!]$.

Существует множество ограничений на структуру шкал. В рамках диссертации представляют интерес следующие ограничения:

- шкала $\mathcal{F}$ является *полугрупповой*, если существует конечнопорожденный моноид $\mathcal{M} = (E, \circ)$ над алфавитом $\mathfrak{O}$ (далее называемый *определяющим* моноидом), такой что $\mathcal{F}.S = E$, $\mathcal{F}.s = \langle\lambda\rangle$ и $\mathcal{F}.R(s, a) = s \circ \langle a\rangle$;
- шкала $\mathcal{F}$ является *упорядоченной*, если для любых цепочек $h \in \mathfrak{O}^*$, $g \in \mathfrak{O}^+$ верно: $[\![hg]\!] \neq [\![h]\!]$.

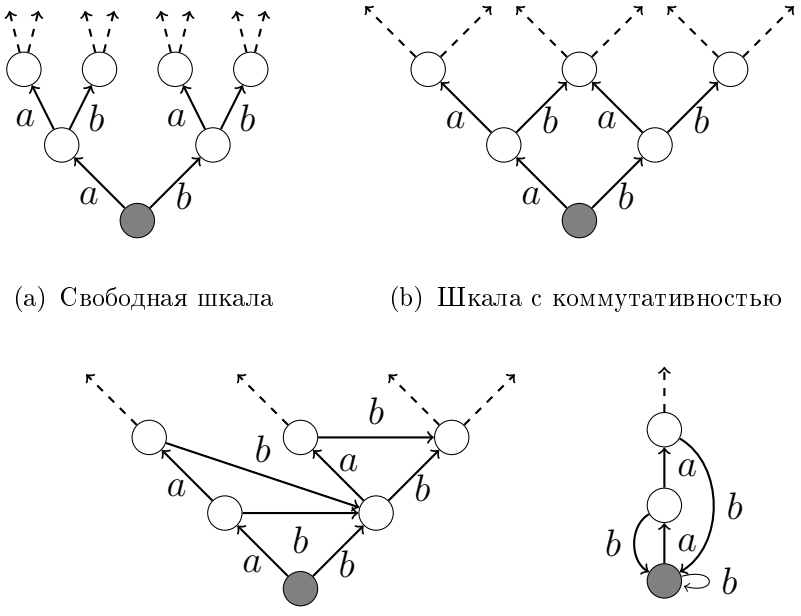
Содержательно, состояния данных полугрупповой шкалы образуют полугруппу, а состояния данных упорядоченной шкалы, достижимые из начального, частично упорядочены относительно функции преобразования данных.

Результаты данной диссертации касаются только полугрупповых упорядоченных шкал. Символ $\circ$ в контексте полугрупповой шкалы будет использоваться для обозначения операции её определяющего моноида. Можно легко убедиться в справедливости следующего факта: полугрупповая шкала упорядочена тогда и только тогда, когда её определяющий моноид упорядочен. Иными словами, отношение $\leq_{\mathcal{F}.R^*}$ для упорядоченной полугрупповой шкалы $\mathcal{F}$ является отношением нестрогого частичного порядка. Чтобы выделить порядки, индуцируемые функцией $\mathcal{F}.R^*$, среди других порядков, встречающихся в рассуждениях, обозначим символами $\preceq$, $\prec$, $\succeq$, $\succ$, $\|$ порядки $\leq_{\mathcal{F}.R^*}$, $<_{\mathcal{F}.R^*}$, $\geq_{\mathcal{F}.R^*}$, $>_{\mathcal{F}.R^*}$ и $\|_{\mathcal{F}.R^*}$ соответственно (предполагая, что шкала $\mathcal{F}$ однозначно определяется контекстом). Состояние данных s, удовлетворяющее отношению $s' \preceq s$, будем называть *достижимым* из состояния s'.

Особое место среди упорядоченных полугрупповых шкал занимает *свободная* шкала — шкала, определяемая свободным моноидом. Состояниями данных такой шкалы являются цепочки операторов, начальным состоянием данных является пустое слово, а функцией переходов — операция конкатенации цепочек.

Один из результатов данной диссертации относится к шкалам, определяемым моноидами с коммутативностью и подавлением. Такие шкалы будем называть *шкалами с коммутативностью и подавлением*. Шкалу, определяемую моноидом $\mathcal{M}(\mathcal{C}, \mathcal{A})$, будем обозначать записью $\mathcal{F}(\mathcal{C}, \mathcal{A})$.

Рисунок 3.2 наглядно иллюстрирует понятие шкалы. На рисунке изображены фрагменты шкал над алфавитом $\mathfrak{O} = \{a, b\}$. Кругами обозначены состояния данных. Начальное состояние данных выделено серым цветом. Равенство $\mathcal{F}.R(s, c) = s'$ изображается дугой $s \xrightarrow{c} s'$. Рисунок 3.2(a) иллюстрирует свободную шкалу; рисунок 3.2(b) — упорядоченную полугрупповую шкалу, задаваемую соотношением коммутативности $ab = ba$; рисунок 3.2(c) — упорядоченную полугрупповую шкалу, задаваемую соотношением подавления $ab = b$; рисунок 3.2(d) — шкалу, в которой оператор b сбрасывает текущее состояние данных в начальное. Последняя шкала не является ни упорядоченной, ни полугрупповой.



(a) Свободная шкала

(b) Шкала с коммутативностью

(c) Шкала с подавлением

(d) Шкала с оператором сброса

Рис. 3.2. Примеры шкал

3.1.3. Реализуемость трасс в интерпретации

Будем говорить, что трасса tr программы π *реализуется в интерпретации* $\mathcal{I}$, если для всех i, $1 \leq i < |tr|$, справедливо соотношение $tr\,[i] \xrightarrow{\mathcal{I}.\xi([\![\mathsf{pref}(tr,i)]\!])}_{\pi} tr\,[i+1]$. Такую трассу также будем называть *трассой в интерпретации* $\mathcal{I}$. Иными словами, в трассе tr, реализующейся в интерпретации $\mathcal{I}$, при переходе к следующему состоянию управления: рассматривается накопленная к этому переходу цепочка операторов; в соответствии со шкалой $\mathcal{I}.\mathcal{F}$ выбирается состояние данных, в которое приводит выполнение этой цепочки; в соответствии с оценкой $\mathcal{I}.\xi$ выбирается логическое условие, по которому делается переход в программе. Отдельно отметим, что в любой интерпретации реализуется ровно одно вычисление программы.

Результатом конечного вычисления cp на шкале $\mathcal{F}$ и в любой $\mathcal{F}$-интерпретации объявляется состояние данных $[\![cp]\!]$. Бесконечные вычисления *безрезультатны* (то есть имеют результат $\perp$, не являющийся состоянием

данных). При рассуждениях о результате вычисления программы π в интерпретации $\mathcal{I}$ без явного указания этого вычисления всегда будет подразумеваться вычисление *ср* программы π в интерпретации $\mathcal{I}$.

3.1.4. Эквивалентность и совместность

Программы π_1, π_2 *эквивалентны в интерпретации*, если результаты их вычислений в этой интерпретации совпадают. Программы *эквивалентны в классе интерпретаций*, если они эквивалентны в каждой интерпретации этого класса. В данной диссертационной работе рассматриваются классы интерпретаций особого вида, а именно: пусть $\mathcal{F}$ — шкала; тогда $\overline{\mathcal{I}}_{\mathcal{F}}$ — класс всевозможных $\mathcal{F}$-интерпретаций. *Эквивалентностью программ на шкале* $\mathcal{F}$ будем называть их эквивалентность в классе интерпретаций $\overline{\mathcal{I}}_{\mathcal{F}}$. Для эквивалентности на шкале $\mathcal{F}$ будем использовать и более короткое название — $\mathcal{F}$-эквивалентность.

В дальнейших рассуждениях и построениях будет активно использоваться понятие совместности трасс. Пусть $\mathcal{F}$ — шкала. Трассы tr_1, tr_2 будем называть $\mathcal{F}$-совместными, если существует $\mathcal{F}$-интерпретация, в которой реализуются обе эти трассы. Введенное понятие совместности трасс позволяет очевидным образом переформулировать определение эквивалентности программ: программы $\mathcal{F}$-эквивалентны тогда и только тогда, когда любые их $\mathcal{F}$-совместные вычисления имеют одинаковый результат.

3.1.5. Утверждения

В данном подразделе приведены два утверждения, которые облегчают анализ трасс программ. Утверждение 3.1.1 позволяет рассматривать произвольные трассы, опуская рассуждения о реализуемости их в какой-либо интерпретации. Утверждение 3.1.2 предоставляет простой способ определения $\mathcal{F}$-совместности трасс, который будет использоваться далее наряду с основным определением $\mathcal{F}$-совместности.

Утверждение 3.1.1 ([10]). *Пусть $\mathcal{F}$ — упорядоченная шкала. Тогда для любой трассы любой программы существует $\mathcal{F}$-интерпретация, в которой реализуется эта трасса.*

Утверждение 3.1.2 ([10]). *Пусть $\mathcal{F}$ — произвольная упорядоченная шкала и tr_1, tr_2 — трассы программ π_1, π_2. Тогда трассы tr_1, tr_2 $\mathcal{F}$-совместны в том и только в том случае, если для любых i_1, i_2, $1 \leq i_1 < |tr_1|$, $1 \leq i_2 < |tr_2|$, верно: если $[\![\mathsf{pref}(tr_1, i_1)]\!] = [\![\mathsf{pref}(tr_2, i_2)]\!]$, то существует логическое условие σ, такое что $tr_1[i_1] \xrightarrow{\sigma}_{\pi_1} tr_1[i_1 + 1]$ и $tr_2[i_2] \xrightarrow{\sigma}_{\pi_2} tr_2[i_2 + 2]$.*

3.2. Унарные рекурсивные программы

В данном разделе приводятся основные определения, относящиеся к пропозициональной модели рекурсивных программ, за исключениям способа задания интерпретаций, описанного ранее в разделе 3.1.2 в контексте модели последовательных программ и использующегося без изменений для унарных рекурсивных программ. Для краткости в рамках данного раздела будем называть унарные рекурсивные программы просто программами.

В разделе 3.2.1 приведён синтаксис программ. В этом же разделе вводятся (как синтаксические конструкции) понятия пути, трассы и вычисления программы. В разделе 3.2.2 обсуждаются семантические аспекты трасс и вычислений: реализуемость трассы и вычисления в интерпретации, результат вычисления. В разделе 3.2.3 вводятся понятия: эквивалентности программ в интерпретации; эквивалентности программ на шкале; совместности трасс программ. Понятие совместности трасс программ предоставляет альтернативную формулировку эквивалентности программ на шкале и активно используется в дальнейших рассуждениях. Раздел 3.2.4 описывает ограничения на структуру программ: линейность и металинейность. Результаты данной диссертации относятся к линейным и металинейным унарным рекурсивным программам. В разделе 3.2.5 состояния управления (то есть определяемые программой функциональные символы

и символ λ, обозначающий завершение вычисления) классифицируются по возможности получить из них конечные и бесконечные вычисления. В разделе 3.2.6 вводятся ограничения на структуру линейных и металинейных программ, облегчающие анализ поведения этих программ. Совокупность этих ограничений образует понятие нормальной формы программы. В разделе 3.2.7 приведены утверждения, касающиеся унарных рекурсивных программ и не являющиеся заслугой автора данной диссертации. Эти утверждения:

- позволяют без ограничения общности рассматривать только программы, удовлетворяющие ограничениям нормальной формы (утверждения 3.2.1, 3.2.2);
- позволяют не проводить анализ реализуемости произвольных трасс программ в рассматриваемых в исследовании интерпретациях (утверждение 3.2.3);
- предоставляют простой способ проверки совместности трасс программ (утверждение 3.2.4).

3.2.1. Синтаксис

Как и в случае пропозициональных последовательных программ, перед описанием унарных рекурсивных программ положим заданными конечные алфавиты операторов $\mathfrak{O}$ и логических условий $\mathfrak{L}$. В контексте унарных рекурсивных программ будем использовать следующие понятия и обозначения:

- $\mathfrak{F}$ — счётно-бесконечный алфавит *функциональных символов*, не пересекающийся с алфавитом $\mathfrak{O}$;
- *терм* — слово в алфавите $\mathfrak{O} \cup \mathfrak{F}$;
- Terms — множество всех термов;

- *базовый терм* — синоним понятия “цепочка операторов” в контексте унарных рекурсивных программ;
- BTerms — множество всех базовых термов;

В дальнейшем будут использоваться разложения терма t следующего вида: $t = t^{\downarrow}t^{\downarrow}t^{\downarrow} = t^{\downarrow}t^{\Downarrow}$, где $t^{\downarrow} \in \mathsf{BTerms}$ и либо $t^{\downarrow} = t^{\downarrow} = \lambda$, либо $t^{\downarrow} \in \mathfrak{F}$. Цепочку $t^{\downarrow}$ в таком разложении будем называть *выполненной*, символ $t^{\downarrow}$ — *обрабатываемым*, терм $t^{\downarrow}$ — *отложенной частью*, а терм $t^{\Downarrow}$ — *остатком*.

Программа π состоит из:

- *запроса* $\pi.q \in \mathsf{Terms}$;
- конечного множества *заголовков функций* $\pi.H \subset \mathfrak{F}$;
- *описания тел функций* $\pi.D : \pi.H \times \mathfrak{L} \to \mathsf{Terms}$.

Введённое определение можно пояснить следующим примером. На рисунке 3.3(a) приведён фрагмент программы в синтаксисе языка Haskell, вычисляющий степень числа 2, а на рисунке 3.3(b) — описание тел функций унарной рекурсивной программы, построенное по этому фрагменту. Множество заголоков программы имеет вид $\{Pow\}$. Логические условия σ_1, σ_2, σ_3 отвечают разбиению множества состояний данных (то есть целых чисел) на классы, описываемые соотношениями “x < 0”, “x $= 0$” и “x > 0”. Оператор *one* соответствует функции “1”, оператор *decrease* — функции “$x-1$”, оператор *duplicate* — функции “$2*x$”. Если аргумент функции отрицателен, то результатом будет ошибочное значение. Это значение получается применением оператора *error*. Чтобы описать вычисление программы, в неё добавляется запрос. Предположим, например, что функция вычисления степени двойки используется для определения значения 2^{10}. Степень 10 может быть расценена двояко: либо как начальное состояние данных, либо как оператор *ten*. В первом случае запрос к программе в модели будет иметь вид Pow, во втором — $ten\ Pow$.

```
Pow :: Integer -> Integer
Pow 0 = 1
Pow x | x>0 = 2*Pow(x-1)
```

(a) Функция вычисления степени двойки

$$D(Pow, \sigma_1) = error$$
$$D(Pow, \sigma_2) = one$$
$$D(Pow, \sigma_3) = decrease\ Pow\ duplicate$$

(b) Описание тел функций унарной рекурсивной программы

Рис. 3.3. Пример построения унарной рекурсивной программы

Размер $|\pi|$ программы π включает в себя две составляющие: длину запроса $|\pi|.q$ и размер описания тел функций $|\pi|.D$: $|\pi|.q = |\pi.q|$, $|\pi|.D = \sum_{f \in \pi.H, \sigma \in \mathfrak{L}} |\pi.D(f, \sigma)|$. Такого определения размера программы достаточно для обоснования полиномиальности времени работы алгоритмов, решающих проблему эквивалентности.

Рассмотрим программу π, логическое условие σ и терм t, для которого верно $t^{\downarrow} \in \pi.H$. Будем называть терм $\left(t^{\downarrow}\ \pi.D\left(t^{\downarrow}, \sigma\right)\ t^{\downarrow}\right)$ *σ-преемником* терма t в программе π. Если логическое условие σ не имеет значения, то будем называть его просто *преемником*. Записью $t \xrightarrow{\sigma}_{\pi} t'$ будем обозначать тот факт, что терм t' является σ-преемником терма t в программе π, а записью $t \to_{\pi} t'$ — что терм t' является преемником терма t в программе π.

Рассмотрим программу π. Последовательность термов вида

$$t_1 \to_{\pi} t_2 \to_{\pi} \ldots \to_{\pi} t_k \to_{\pi} \ldots$$

будем называть t_1-*путем* программы π. *Трассой* программы будем называть ее $\pi.q$-путь. Если трасса бесконечна или оканчивается термом, не имеющим преемников в программе, то такую трассу будем называть *вычислением*.

Замечание. *Термин "вычисление" принято относить к семантике программ, а не к синтаксису, однако общая структура вычисления может быть описана на синтаксическом уровне и потому приведена в данном разделе. Семантический аспект вычисления — реализуемость его в интерпретации — обсуждается далее, в разделе 3.2.2.*

3.2.2. Реализуемость трасс в интерпретации

В данном разделе используются без изменений понятия шкалы и интерпретации, описанные в разделе 3.1.2.

Будем говорить, что трасса tr программы π *реализуется в интерпретации* $\mathcal{I}$, если для всех i, $1 \leq i < |tr|$, справедливо соотношение $tr\,[i] \xrightarrow{\mathcal{I}.\xi([\![t^{\downarrow}]\!])}_{\pi} tr\,[i+1]$. Такую трассу будем также называть *трассой в интерпретации* $\mathcal{I}$. Иными словами, в трассе tr, реализующейся в интерпретации $\mathcal{I}$, при переходе к следующему терму: рассматривается выполненная цепочка операторов текущего терма t; в соответствии со шкалой $\mathcal{I}.\mathcal{F}$ выбирается состояние данных, в которое приводит выполнение этой цепочки; в соответствии с оценкой $\mathcal{I}.\xi$ выбирается логическое условие, по которому делается переход в программе.

Рассмотрим вычисление cp, оканчивающееся базовым термом h. *Результатом* этого вычисления на шкале $\mathcal{F}$ и в любой $\mathcal{F}$-интерпретации объявляется состояние данных $[\![h]\!]$. Остальные вычисления *безрезультатны* (имеют результат $\perp$, не являющийся состоянием данных). При рассуждениях о результате вычисления программы π в интерпретации $\mathcal{I}$ без явного указания этого вычисления всегда будет подразумеваться вычисление cp программы π в интерпретации $\mathcal{I}$.

3.2.3. Эквивалентность и совместность

Понятия эквивалентности программ и совместности трасс программ дословно переносятся со случая пропозициональных последовательных программ на случай унарных рекурсивных программ:

- программы *эквивалентны в интерпретации*, если результаты их вычислений в этой интерпретации совпадают;
- программы *эквивалентны на шкале* $\mathcal{F}$, или $\mathcal{F}$*-эквивалентны*, если они эквивалентны во всех $\mathcal{F}$-интерпретациях;

- трассы tr_1, tr_2 *$\mathcal{F}$-совместны*, где $\mathcal{F}$ — шкала, если существует $\mathcal{F}$-интерпретация, в которой реализуются обе эти трассы.

Особое положение занимает эквивалентность программ на свободной шкале, то есть шкале, определяемой свободным моноидом. Такую эквивалентность будет назвать *сильной*.

3.2.4. Линейность и металинейность

Результаты данной диссертации относятся к эквивалентности унарных рекурсивных программ, структура которых удовлетворяет требованиям линейности и металинеености. Требования линейности и металинейности программ, в свою очередь, основываются на требовании линейности термов. Терм t является *линейным*, если его отложенная часть является базовым термом: $t^{\downarrow} \in \mathsf{BTerms}$. Таким образом, линейный терм — это терм, содержащий не более одного вхождения функциональных символов. Множество всех линейных термов обозначим записью LTerms.

Программу π назовем *металинейной*, если область значений функции $\pi.D$ состоит только из линейных термов:

$$\forall\ f \in \pi.H,\ \sigma \in \mathfrak{L}.\quad \pi.D(f,\sigma) \in \mathsf{LTerms}.$$

Программу π назовем *линейной*, если выполнены следующие условия:

- она является металинейной;
- запрос программы является линейным термом: $\pi.q \in \mathsf{LTerms}$.

В качестве пояснения отметим, что приведённая в разделе 3.2.1 унарная рекурсивная программа, описывающая вычисление степени двойки, является линейной, а значит, и металинейной.

3.2.5. Классификация заголовков

В дальнейших рассуждениях важную роль играют возможности построения конечных и бесконечных путей, начинающихся в терме с заданным обрабатываемым символом. Чтобы выразить эти возможности, проведем классификацию заголовков программ. В эту классификацию также попадает пустое слово λ как обрабатываемый символ любого базового терма. Пусть символ ϕ — либо заголовок схемы π, либо λ. Тогда будем называть символ ϕ:

- *завершаемым* символом программы π, если существует ϕ-путь этой программы, оканчивающийся базовым термом;
- *граничным* символом программы π, если все ϕ-пути этой программы конечны;
- *существенно рекурсивным* символом программы π, если существует бесконечный ϕ-путь этой программы;
- *крайним* символом программы π, если в этой программе существуют преемники t_1, t_2 терма ϕ, такие что:
  - $t_1{}^{\downarrow}$ — существенно рекурсивный символ программы π;
  - $t_2{}^{\downarrow}$ — граничный символ программы π.

Способы быстрого определения существенно рекурсивных, завершаемых и граничных заголовков программы приведены в работе [9]. Символ λ очевидным образом классифицируется по этим трём признакам: λ является граничным и завершаемым символом и не является существенно рекурсивным символом любой программы. Классификация символов по остальным признакам может быть легко получена после определения существенно рекурсивных, завершаемых и граничных символов.

3.2.6. Нормальная форма

Одно из основных преимуществ упорядоченных шкал перед неупорядоченными в контексте пропозициональных последовательных программ сформулировано в утверждении 3.2.3: нет необходимости обосновывать реализуемость трассы в какой-либо $\mathcal{F}$-интерпретации, если $\mathcal{F}$ — упорядоченная шкала. Это утверждение основывается на том, что при продолжении трассы хотя бы на один шаг обязательно получается состояние данных, отличное от всех предыдущих, а значит, для этого состояния данных в интерпретации допустимо любое логическое условие. Это свойство не выполнено для унарных рекурсивных программ произвольного вида, даже если они удовлетворяют требованиям линейности и металинейности: достаточно поместить в описание тел функций программы терм t, для которого выполнено равенство $t^{\downarrow} = \lambda$. Чтобы сформулировать утверждение, аналогичное утверждению 3.2.3, для унарных рекурсивных программ, для них вводится понятие нормальной формы.

Программа π находится *в нормальной форме*, или *нормализована*, если для нее выполнены следующие условия:

- запрос программы состоит только из её заголовков: $\pi.q \in (\pi.H)^*$;
- для любого терма t из области значений описания тел функций верно: либо t — непустой базовый терм, либо $t^{\downarrow} \in \mathfrak{O}$ и $t^{\Downarrow} \in (\mathfrak{O} \cup \pi.H)^*$;
- множество заголовков программы содержит специальный функциональный символ f_∞ — будем называть его *тупиковым символом*;
- все заголовки программы, кроме f_∞, являются завершаемыми;
- для любого логического условия σ верно: $\pi.D(f_\infty, \sigma)^{\downarrow} = f_\infty$.

Замечание. *Нормализованная программа не содержит конечных безрезультатных вычислений.*

3.2.7. Утверждения

В [9] был предложен алгоритм, позволяющий быстро приводить программы к нормальной форме, при этом сохраняя их линейность и металинейность. Этот факт отражен в следующих двух утверждениях.

Утверждение 3.2.1 ([9]). *По любой программе π можно за время $\mathsf{O}\big(|\pi|.q + |\pi|.D^4\big)$ построить сильно эквивалентную ей нормализованную программу π', такую что $|\pi'|.q \leq |\pi|.q$ и $|\pi'|.D = \mathsf{O}\big(|\pi|.q + |\pi|.D^2\big)$. При этом если программа π линейна (металинейна), то программа π' также линейна (металинейна).*

Утверждение 3.2.2 ([9]). *Сильно эквивалентные унарные рекурсивные программы эквивалентны на любой упорядоченной шкале.*

Следующее утверждение позволяет рассматривать произвольные трассы нормализованных программ, опуская рассуждения, обосновывающие их реализуемость в какой-либо из рассматриваемых интерпретаций.

Утверждение 3.2.3 ([9]). *Пусть $\mathcal{F}$ — упорядоченная шкала. Тогда для любой трассы нормализованной программы существует $\mathcal{F}$-интерпретация, в которой реализуется эта трасса.*

Следующее утверждение предоставляет простой способ проверки совместности трасс нормализованных программ, который будет использоваться в дальнейших рассуждениях наряду с основным определением совместности трасс.

Утверждение 3.2.4 ([9]). *Пусть $\mathcal{F}$ — упорядоченная шкала и tr_1, tr_2 — трассы нормализованных программ π_1, π_2. Тогда трассы tr_1, tr_2 $\mathcal{F}$-совместны в том и только том случае, если для любых i_1, i_2, где $1 \leq i_1 < |tr_1|$, $1 \leq i_2 < |tr_2|$, верно: если $\Big[\!\Big[tr_1\,[i_1]^{\downarrow}\Big]\!\Big] = \Big[\!\Big[tr_2\,[i_2]^{\downarrow}\Big]\!\Big]$, то существует логическое условие σ, такое что $tr_1\,[i_1] \xrightarrow{\sigma}_{\pi_1} tr_1\,[i_1+1]$ и $tr_2\,[i_2] \xrightarrow{\sigma}_{\pi_2} tr_2\,[i_2+1]$.*

Глава 4

Формулировка результатов и методология исследования программ

В данной главе приведены строгая и более подробная, чем во введении, формулировка результатов, полученных в данной диссертации, и общее описание подхода, использованного для получения этих результатов, — метода совместных вычислений в приложении к исследованию пропозициональных моделей последоваетельных и рекурсивных программ.

4.1. Формулировка результатов

В данной диссертации выделены три основных результата.

Первый результат: доказана полиномиальная разрешимость проблемы эквивалентности пропозициональных последовательных программ на произвольной упорядоченной шкале с коммутативностью и подавлением. Обоснование этого результата приведено в главе 5 и заключается в следующем. Разработан алгоритм, решающий рассматриваемую проблему эквивалентности. Алгоритм принимает на вход описание шкалы с коммутативностью и подавлением $\mathcal{F}(\mathcal{C}, \mathcal{A})$ (более точно, на вход подаются множества $\mathcal{C}$ и $\mathcal{A}$) и описания программ и определяет, являются ли эти программы $\mathcal{F}(\mathcal{C}, \mathcal{A})$-эквивалентными. Строго обоснованы корректность алгоритма и полиномиальность времени его работы относительно размеров подаваемых на вход программ.

Второй результат: установлены достаточные условия, которым должна удовлетворять упорядоченная полугрупповая шкала, чтобы проблема эквивалентности линейных унарных рекурсивных программ на этой шкале оказалась полиномиально разрешимой. Достаточность строго обоснована. Обоснование приведено в главе 6 и заключается в следующем. Разработан алгоритм, решаю-

щий проблему эквивалентности линейных унарных рекурсивных программ на упорядоченной полугрупповой шкале. Описание алгоритма имеет смысл только при выполнении описанных в результате достаточных условий и зависит от выбора шкалы и структур в описании достаточных условий. Строго обоснованы корректность алгоритма и полиномиальность времени его работы относительно размеров подаваемых на вход программ в предположении о том, что выполняются установленные достаточные условия.

Третий результат: доказана полиномиальная разрешимость проблемы сильной эквивалентности металинейных унарных рекурсивных программ. Обоснование этого результата приведено в главе 7 и заключается в следующем. Разработан алгоритм, решающий рассматриваемую проблему эквивалентности. Строго обоснованы корректность алгоритма и полиномиальность времени его работы относительно размеров подаваемых на вход программ.

4.2. Метод совместных вычислений

В данном разделе приводится общее описание метода исследования проблемы эквивалентности программ (пропозициональных последовательных и унарных рекурсивных) — метода совместных вычислений. Основная идея этого метода состоит в следующем.

Основная идея этого метода — анализ пар вычислительных конфигураций, которые могут быть получены при одновременном построении вычислений программ в интерпретациях, рассматриваемых при описании эквивалентности. Эта идея не нова [7–10, 76, 102, 112–114], однако сама по себе не даёт решения проблемы эквивалентности, а тем более описания быстрых алгоритмов проверки эквивалентности программ. В частности, методы, предложенные во всех упомянутых работах, не могут быть использованы напрямую для получения основных результатов данной диссертации.

Применительно к рассматриваемым моделям программ, конкретизация

основной идеи метода совместных вычислений — это анализ всевозможных $\mathcal{F}$-совместных трасс программ π_1, π_2, эквивалентность которых требуется проверить. Для анализа описывается графовая структура $\Gamma = \Gamma(\pi_1, \pi_2, \mathcal{F})$, называемая далее *графом совместных вычислений*. Для получения каждого из основных результатов формально описывается и анализируется понятие графа совместных вычислений с целью предложить конечный и, более того, полиномиальный по времени обход, позволяющий заключить, эквивалентны ли программы π_1, π_2. Основная цель данного раздела — выделить общие черты используемых для получения результатов понятий и техник и тем самым пояснить назначение используемых в главах 5, 6, 7 формальных конструкций.

Вычисление программы в любой интерпретации $\mathcal{I}$ может быть формализовано как последовательность вычислительных конфигураций. Вычислительной конфигурацией унарной рекурсивной программы является терм, а пропозициональной последовательной программы — цепочка операторов, приписанная текущей трассе, в совокупности с текущим состояние управления. Программой определяется способ переписывания вычислительных конфигураций. Переписывание основывается на том, что по текущей вычислительной конфигурации можно определить выполненную к текущему шагу цепочку операторов (для терма t это цепочка $t^{\downarrow}$) и текущее состояние управления (для терма t это символ $t^{\downarrow}$). По выполненной цепочке операторов h можно определить текущее состояние данных $[\![h]\!]$ и текущее логическое условие $\mathcal{I}.\xi(h)$. По описанию программы и текущему логическому условию определяется следующая вычислительная конфигурация. Преобразование вычислительной конфигурации происходит, пока не будет достигнута завершающая вычислительная конфигурация: для последовательных программ — содержащая выход; для рекурсивных программ — базовый терм.

Согласно методу совместных вычислений следует перейти к одновременному построению пар вычислительных конфигураций рассматриваемых программ π_1, π_2 в интерпретации $\mathcal{I}$, то есть к *совместному выполнению* этих про-

грамм. Таким образом, совместное вычисление программ π_1, π_2 есть последовательность пар $(\mathscr{c}_1, \mathscr{c}_2)$, где $\mathscr{c}_i$ — вычислительная конфигурация программы π_i. Первая пара этой последовательности состоит из инициальных конфигураций программ (для рекурсивных программ — запросов, для последовательных программ — операторов, приписанных входам, вместе с самими входами). При переходе к следующей паре конфигураций выбираются *активные* программы (первая, вторая или обе), и конфигурации активных программ преобразуются согласно тому, как строятся вычисления программ в интерпретации $\mathcal{I}$. Для устранения недетерминизма выбирается *стратегия совместного выполнения*, по паре конфигураций однозначно определяющая активность программ π_1, π_2.

Каждая пара конфигураций в совместном выполнении заменяется на *различие* этих конфигураций. Структура различий подбирается таким образом, чтобы можно было по различию пары конфигураций:

- определить текущие состояния управления конфигураций;
- проверить равенство текущих состояний данных конфигураций;
- определить активность программ в контексте выбранной стратегии выполнения;
- определить различие следующей в совместном выполнении пары конфигураций, используя описание программ.

Если подобран вид различий пар конфигураций, удовлетворяющий всем описанным свойствам, то совместное выполнение программ может быть переформулировано в терминах различий (а не пар конфигураций). Более того, по различию можно проверить, построены ли программами в процессе совместного выполнения конечные вычисления, и если обе программы построили конечные вычисления, то совпадают ли результаты этих вычислений.

Граф совместных вычислений $\Gamma(\pi_1, \pi_2, \mathcal{F})$ может быть описан следующим образом. Отталкиваясь только от шкалы $\mathcal{F}$, выберем стратегию совместного

выполнения. Рассмотрим совместное выполнение программ π_1, π_2 в каждой $\mathcal{F}$-интерпретации $\mathcal{I}$ с выбранной стратегией выполнения. Перейдем от совместного выполнения программ к последовательности различий. Объединим все полученные последовательности в ориентированный граф: вершинами графа являются всевозможные различия пар конфигураций; дугами в графе соединены различия, являющиеся соседними хотя бы в одном из совместных выполнений. Корнем графа объявим различие инициальных конфигураций программ.

Проверка $\mathcal{F}$-эквивалентности программ π_1, π_2 сводится к исследованию графа $\Gamma(\pi_1, \pi_2, \mathcal{F})$. Для этого среди маршрутов графа $\Gamma(\pi_1, \pi_2, \mathcal{F})$ выделяются *опровергающие (эквивалентность программ)*, то есть отвечающие построению совместных вычислений программ π_1, π_2 с различными результатами. Таким образом, проверка $\mathcal{F}$-эквивалентности программ сводится к поиску опровергающих маршрутов в графе совместных вычислений. Поиск осложняется тем, что граф $\Gamma(\pi_1, \pi_2, \mathcal{F})$, вообще говоря, бесконечен, однако в каждом из полученных в данной диссертации результатов описан полиномиальный обход графа совместных вычислений, позволяющий определить, являются ли программы π_1, π_2 $\mathcal{F}$-эквивалентными.

Подводя итог общему описанию подхода к исследованию проверки $\mathcal{F}$-эквивалентности программ, использованного для получения результатов, сформулированных в разделе 4.1, обозначим основные этапы применения этого подхода.

1. Выбор стратегии совместного выполнения, зависящий только от шкалы $\mathcal{F}$, но не от программ π_1, π_2.

2. Описание различия пар конфигураций, и на основании различий и выбранной стратегии — описание графа совместных вычислений.

3. Описание структуры опровергающих маршрутов графа совместных вычислений.

4. Описание полиномиального обхода графа совместных вычислений, по завершении которого можно определить, являются ли программы π_1, π_2 $\mathcal{F}$-эквивалентными.

5. Получение полиномиальной разрешимости проблемы $\mathcal{F}$-эквивалентности программ в качестве следствия наличия описанного обхода.

Глава 5

Эквивалентность пропозициональных последовательных программ на шкалах с коммутативностью и подавлением

Данная глава посвящена:

- обоснованию полиномиальной разрешимости проблемы эквивалентности пропозициональных последовательных программ на произвольных упорядоченных шкалах с коммутативностью и подавлением;
- описанию быстрого (полиномиального по времени относительно размеров программ) алгоритма, решающего проблему эквивалентности, с обоснованием корректности и сложности этого алгоритма.

Результаты данной главы опубликованы в работах [12, 41] в журналах из перечня ВАК.

В данной главе используются следующие понятия и утверждения:

- общие понятия, описанные в главе 2;
- понятия и утверждения, относящиеся к пропозициональным последовательным программам (раздел 3.1);
- понятия, введённые при описании подхода к исследованию проблемы эквивалентности (раздел 4.2).

Для краткости в данной главе будем называть пропозициональные последовательные программы просто программами.

В разделе 5.1 приводится критерий упорядоченности моноидов с коммутативностью и подавлением, позволяющий легко установить упорядоченность

моноида $\mathcal{M}(\mathcal{C}, \mathcal{A})$ по множествам $\mathcal{C}$ и $\mathcal{A}$. Этот критерий полезен по двум причинам: во-первых, он добавляет "практичности" в построение алгоритмов, решающих проблему эквивалентности программ на упорядоченных шкалах с коммутативностью и подавлением, позволяя перед началом работы алгоритма определить, является ли исследуемая шкала упорядоченной; во-вторых, он будет использоваться для обоснования основных свойств моноидов с коммутативностью и подавлением (раздел 5.2) и для построения автомата, распознающего подавление операторов (раздел 5.3). Свойства моноидов с коммутативностью и подавлением используются:

- в описании и обосновании свойств графа совместных вычислений (раздел 5.4), позволяющего свести проверку эквивалентности программ на упорядоченной шкале с коммутативностью и подавлением к проверке наличия особых маршрутов в графе;
- при введении аппарата "эффектов подавлений" (раздел 5.5), позволяющего ограничить число вершин графа совместных вычислений, которые достаточно исследовать для определения эквивалентности программ (раздел 5.6).

Наконец, в разделе 5.7 описывается полиномиальный обход графа совместных вычислений, по завершении которого можно определить, эквивалентны ли рассматриваемые программы, и формулируется основная теорема — полиномиальная разрешимость проблемы эквивалентности программ на упорядоченных шкалах с коммутативностью и подавлением (теорема 5.7.1). Корректность обхода основывается на утверждениях раздела 5.6.

5.1. Упорядоченность моноидов с коммутативностью и подавлением

В данном разделе приведен критерий упорядоченности моноидов с коммутативностью и подавлением над алфавитом операторов $\mathfrak{O}$, позволяющий легко установить упорядоченность по множествам $\mathcal{C}$ и $\mathcal{A}$. Критерий сформулирован в теореме 5.1.1 и предварен несколькими техническими леммами. Во всех рассуждениях данного раздела запись $\mathcal{C}^*$ обозначает рефлексивно-симметричное замыкание множества $\mathcal{C}$, а запись $\mathcal{A}^*$ — транзитивное замыкание множества $\mathcal{A}$. Запись $\leftrightsquigarrow^*$ будет обозначать рефлексивно-симметричное замыкание отношения коммутации $\leftrightsquigarrow$; запись $\not\leftrightsquigarrow^*$ — дополнение отношения $\leftrightsquigarrow^*$; запись $\hookleftarrow^*$ — транзитивное замыкание отношения $\hookleftarrow$; запись $\not\hookleftarrow^*$ — дополнение отношения $\hookleftarrow^*$.

Лемма 5.1.1. *Моноиды $\mathcal{M}(\mathcal{C}, \mathcal{A})$ и $\mathcal{M}(\mathcal{C}^*, \mathcal{A}^*)$ совпадают.*

Доказательство. **Этап 1**: покажем, что справедливость всех соотношений коммутативности, порождаемых множеством $\mathcal{C}^*$, следует из соотношений коммутативности, порождаемых множеством $\mathcal{C}$. Иными словами, достаточно показать, что если $a_1 \leftrightsquigarrow a_2$, то цепочка a_1a_2 выводима из a_2a_1. Если $a_1 = a_2$, то выводимость очевидна. Иначе выводимость следует из симметрии в определении шага вывода.

Этап 2: покажем, что справедливость всех соотношений подавления, порождаемых множеством $\mathcal{A}^*$, следует из соотношений коммутативности, порождаемых множеством $\mathcal{A}$. Иными словами, достаточно показать, что если $a_1 \hookleftarrow a_2 \hookleftarrow a_3$, то цепочка a_3 выводима из a_1a_3. Для обоснования такой выводимости приведем в явном виде вывод: $a_1a_3 \overset{+}{\rightleftharpoons} a_1a_2a_3 \overset{-}{\rightleftharpoons} a_2a_3 \overset{-}{\rightleftharpoons} a_3$. □

Критерий упорядоченности формулируется довольно просто: моноид $\mathcal{M}(\mathcal{C}, \mathcal{A})$ упорядочен тогда и только тогда, когда множества $\mathcal{C}^*$ и $\mathcal{A}^*$ не пересекаются. Для обоснования этого критерия производится подсчёт операто-

ров в цепочках вывода, и на основании этого подсчёта показывается, что если $\langle hg \rangle = \langle h \rangle$, то цепочка g не содержит никаких операторов. Для подсчёта операторов введем следующие функции:

- $\mathsf{count}(a, h)$ — число вхождений оператора a в цепочку h;
- $\mathsf{pos_r}(a, h)$ — позиция самого правого вхождения оператора, способного подавить a, в цепочку h: $\mathsf{pos_r}(a, h) = \max\{i \mid a \hookleftarrow^* h[i]\}$ (частично определённая функция);
- $\mathsf{count_r}(a, h)$ — число вхождений оператора a в цепочку $\overline{\mathsf{pref}}(h, \mathsf{pos_r}(a, h))$ (определена для тех же значений, что и функция $\mathsf{pos_r}$).

Лемма 5.1.2. *Если $\mathcal{C}^* \cap \mathcal{A}^* = \emptyset$, $h \rightleftharpoons h'$ — шаг вывода в моноиде $\mathcal{M}(\mathcal{C}, \mathcal{A})$ и значение $\mathsf{pos_r}(a, h)$ определено, то $\mathsf{count_r}(a, h) = \mathsf{count_r}(a, h')$.*

Доказательство. **Случай 1**: $h \overset{i|}{\rightleftharpoons} h'$. Тогда $h[i] \rightsquigarrow^* h[i+1] \rightsquigarrow^* h[i]$, а значит, $h[i] \not\hookleftarrow^* h[i+1] \not\hookleftarrow^* h[i]$. Осталось заметить, что цепочки h, h' отличаются только перестановкой i-го и $(i+1)$-го операторов.

Случай 2: $h \overset{i^-}{\rightleftharpoons} h'$. В силу транзитивной замкнутости множества $\mathcal{A}^*$, если $b \hookleftarrow^* h[i]$, то $b \hookleftarrow^* h[i+1]$. Следовательно, поглощённое вхождение никак не влияет на функции $\mathsf{pos_r}$, $\mathsf{count_r}$.

Случай 3: $h \overset{i^+}{\rightleftharpoons} h'$ Рассуждения аналогичны случаю (**2**) с заменой h на h' и поглощения на порождение. □

Теорема 5.1.1. *Моноид $\mathcal{M}(\mathcal{C}, \mathcal{A})$ упорядочен тогда и только тогда, когда $\mathcal{C}^* \cap \mathcal{A}^* = \emptyset$.*

Доказательство. ($\Rightarrow$): Предположим, что существует пара операторов $(a_1, a_2) \in \mathcal{C}^* \cap \mathcal{A}^*$. Используя лемму 5.1.1, получим вывод $a_2 \overset{+}{\rightleftharpoons} a_1 a_2 \overset{|}{\rightleftharpoons} a_2 a_1$. Следовательно, $\langle a_2 \rangle = \langle a_2 a_1 \rangle$, что противоречит упорядоченности шкалы.

($\Leftarrow$): Рассмотрим цепочки h, g, для которых верно $\langle hg \rangle = \langle h \rangle$. Для завершения обоснования достаточно показать, что для любого оператора a верно

равенство $\mathsf{count}(a, g) = 0$. По условию теоремы множество $\mathcal{A}^*$ не содержит ни одной пары вида (a, a), $a \in \mathfrak{O}$. Следовательно, $\mathcal{A}^*$ — отношение строгого частичного порядка. Тогда множество set_{hg} операторов, входящих в цепочку hg, может быть разбито на конечное число слоев $\mathcal{L}_0, \ldots, \mathcal{L}_k$ следующим образом:

- $\mathcal{L}_0 \subseteq \mathfrak{O}$ — все максимальные относительно порядка $\mathcal{A}^*$ элементы множества set_{hg};
- $\mathcal{L}_i \subseteq \mathfrak{O}$, где $1 \leq i \leq k$ — все максимальные относительно порядка $\mathcal{A}^*$ элементы множества $\mathsf{set}_{hg} \setminus \bigcup_{s=0}^{i-1} \mathcal{L}_s$.

Из определения слоёв следует, что операторы слоя $\mathcal{L}_i$ могут быть поглощены и порождены только операторами слоёв $\mathcal{L}_0, \ldots, \mathcal{L}_{i-1}$. Покажем, используя индукцию по номеру i слоя $\mathcal{L}_i$, что цепочка g не содержит операторов ни одного слоя $\mathcal{L}_i$.

База индукции: $i = 0$. Операторы слоя $\mathcal{L}_0$ не могут быть ни поглощены, ни порождены ни в одном выводе, начинающемся с цепочки h. Следовательно, $\mathsf{count}(a, h) = \mathsf{count}(a, hg)$ для всех операторов $a \in \mathcal{L}_0$. Так как $\mathsf{count}(a, hg) = \mathsf{count}(a, h) + \mathsf{count}(a, g)$, немедленно получаем равенство $\mathsf{count}(a, g) = 0$ для $a \in \mathcal{L}_0$.

Индуктивный шаг: пусть цепочка g не содержит операторов слоёв $\mathcal{L}_0, \ldots, \mathcal{L}_{i-1}$; покажем, что оно не содержит операторов слоя $\mathcal{L}_i$. Рассмотрим произвольный оператор a слоя $\mathcal{L}_i$. По построению слоёв в цепочке hg существет оператор, способный поглотить a. По предположению индукции все такие операторы содержатся в цепочке h. Следовательно, значение функции $\mathsf{pos}_\mathsf{r}(a, h)$ определено. По определению конечнопорожденного моноида существует вывод цепочки hg из h. Используя существование такого вывода, наличие значения $\mathsf{pos}_\mathsf{r}(a, h)$ и лемму 5.1.2, немедленно получаем равенство $\mathsf{count}_\mathsf{r}(a, h) = \mathsf{count}_\mathsf{r}(a, hg)$. Так как все операторы, способные поглотить a, находятся в цепочке h, получаем, что цепочка g не содержит оператора a. □

5.2. Основные свойства упорядоченных моноидов с коммутативностью и подавлением

В данном разделе приводятся утверждения, относящиеся к моноидам $\mathcal{M}(\mathcal{C}, \mathcal{A})$ над алфавитом $\mathfrak{O}$ — определяющим моноидам шкал $\mathcal{F}(\mathcal{C}, \mathcal{A})$. Чтобы коротко сформулировать эти утверждения, введём ряд понятий, относящихся к таким моноидам. Перед этим, основываясь на лемме 5.1.1, положим, что множество $\mathcal{C}$ симметрично замкнуто и не содержит пар вида (a, a), $a \in \mathfrak{O}$, а множество $\mathcal{A}$ транзитивно замкнуто.

Рассмотрим моноид $\mathcal{M}(\mathcal{C}, \mathcal{A})$ и вывод D в этом моноиде (точнее, вывод в контексте соотношений коммутативности и подавления, определяющих этот моноид). По аналогии со вхождением оператора в цепочку введём формализм, описывающий вхождение оператора в вывод в следующем смысле:

- если $D[i] \overset{j|}{\rightleftharpoons} D[i+1]$, то j-е и $(j+1)$-е вхождения меняются местами, а остальные вхождения остаются на своих местах;
- если $D[i] \overset{j^-}{\rightleftharpoons} D[i+1]$, то все вхождения слева от j-го остаются на своих местах, j-е вхождение исчезает, а все вхождения справа от j-го сдвигаются на единицу влево;
- если $D[i] \overset{j^+}{\rightleftharpoons} D[i+1]$, то все вхождения слева от j-го остаются на своих местах, все вхождения, начиная с j-го, сдвигаются на одну позицию вправо, и появляется новое j-е вхождение.

Опишем понятие вхождения оператора в вывод формально. *Координатами* оператора в выводе будем называть пару чисел (i, j): содержательно, такая пара указывает на j-й оператор i-й цепочки вывода. *Вхождение* — это всякое минимальное по включению непустое множество координат θ, для которого верно:

- если $D[i] \overset{j|}{\rightleftharpoons} D[i+1]$, то:

  - если $1 \leq s < j$ или $j + 1 < s \leq |D[i]|$, то $(i, s) \in \theta$ тогда и только тогда, когда $(i + 1, s) \in \theta$;
  - $(i, j) \in \theta$ тогда и только тогда, когда $(i + 1, j + 1) \in \theta$;
  - $(i, j + 1) \in \theta$ тогда и только тогда, когда $(i + 1, j) \in \theta$;

- если $D[i] \overset{j^-}{\rightleftharpoons} D[i + 1]$, то:
  - если $1 \leq s < j$, то $(i, s) \in \theta$ тогда и только тогда, когда $(i + 1, s) \in \theta$;
  - если $j < s \leq |D[i]|$, то $(i, s) \in \theta$ тогда и только тогда, когда $(i + 1, s - 1)$;

- если $D[i] \overset{j^+}{\rightleftharpoons} D[i + 1]$, то:
  - если $1 \leq s < j$, то $(i, s) \in \theta$ тогда и только тогда, когда $(i + 1, s) \in \theta$;
  - если $j \leq s \leq |D[i]|$, то $(i, s) \in \theta$ тогда и только тогда, когда $(i + 1, s + 1)$.

Можно легко убедиться, что операторы, на которые указывают все координаты (произвольно взятого) вхождения, одинаковы. Основываясь на этом факте, можно говорить о *вхождении оператора* в вывод. Вхождение назовем *главным*, если в него входят координаты $(1, j)$ для какого-либо j. Для краткости записью $\theta(i, j)$ будем обозначать вхождение, содержащее координаты (i, j). Отметим, что всякое вхождение $\theta(i, j)$ определено однозначно.

Для каждого вхождения θ определены:

- время порождения: $start(\theta) = \min\{i \mid \theta = \theta(i, j)\}$;
- время поглощения: $finish(\theta) = \max\{i \mid \theta = \theta(i, j)\}$;
- проекция на i-ю цепочку вывода: $\theta{\downarrow_i} = j$, если $\theta = \theta(i, j)$ (частично определено).

Рассмотрим вхождения θ, θ' в вывод D. Будем говорить, что вхождение θ' *поглощается* вхождением θ, если $\theta{\downarrow}_{finish(\theta')+1} = \theta'{\downarrow}_{finish(\theta')}$. Будем говорить, что вхождение θ' *порождается* вхождением θ, если $\theta{\downarrow}_{start(\theta')-1} = \theta'{\downarrow}_{start(\theta')}$. Отношение *косвенного порождения* — рефлексивно-транзитивное замыкание отношения порождения.

Рассмотрим множество вхождений Θ. Назовем это множество *замкнутым по подавлению* (в выводе D), если для каждого вхождения множества Θ все поглощаемые и порождаемые им вхождения также входят в Θ. *Замыканием* (по подавлению) вхождения θ назовем минимальное по включению замкнутое по подавлению множество, содержащее θ. Замыкание (по подавлению) множества вхождений есть объединение замыканий его элементов.

Пусть Θ — множество вхождений в вывод D. Тогда $D^{-\Theta}$ — последовательность, получаемая в результате удаления всех вхождений $\theta \in \Theta$ из вывода D, то есть в результате удаления всех операторов с координатами $(i, j) \in \theta$, $\theta \in \Theta$.

Следующая лемма носит технический характер и представляет собой основной приём, используемый при обосновании свойств моноидов $\mathcal{M}(\mathcal{C}, \mathcal{A})$.

Лемма 5.2.1. *Пусть D — вывод в моноиде $\mathcal{M}(\mathcal{C}, \mathcal{A})$ и Θ — замкнутое по подавлению множество вхождений в этот вывод. Тогда последовательность $\left\lceil D^{-\Theta} \right\rceil$ является выводом.*

Доказательство. Достаточно показать, что если $D\,[i] \overset{|}{\rightleftharpoons} D\,[i+1]$ ($D\,[i] \overset{-}{\rightleftharpoons} D\,[i+1]$, $D\,[i] \overset{+}{\rightleftharpoons} D\,[i+1]$), то либо $D^{-\Theta}\,[i] = D^{-\Theta}\,[i]$, либо $D^{-\Theta}\,[i] \overset{|}{\rightleftharpoons} D^{-\Theta}\,[i]$ ($D^{-\Theta}\,[i] \overset{-}{\rightleftharpoons} D^{-\Theta}\,[i]$, $D^{-\Theta}\,[i] \overset{+}{\rightleftharpoons} D^{-\Theta}\,[i]$ соответственно). Для шага коммутации всё очевидно: если ни один из коммутирующих операторов не удаляется, то в после удаления получается шаг коммутации, иначе — равенство. Такие рассуждения применимы и к шагам поглощения и порождения, за исключением двух случаев.

Случай 1: $hag \overset{|h|+1^+}{\rightleftharpoons} hbag$, при этом выделенное вхождение оператора a удаляется, а выделенное вхождение оператора b не удаляется.

Случай 2: $hbag \overset{|h|+1^-}{\rightleftharpoons} hag$, при этом выделенное вхождение оператора a удаляется, а выделенное вхождение оператора b не удаляется.

Оба случая невозможны по причине замкнутости множества Θ. □

Следующая лемма содержательно формулируется довольно просто: если операторы упорядоченного моноида $\mathcal{M}(\mathcal{C}, \mathcal{A})$ не коммутируют, то относительный порядок вхождений этих операторов сохраняется во всех цепочках любого вывода.

Лемма 5.2.2. *Пусть:*

- θ_1, θ_2 — *вхождения операторов* a_1, a_2 *соответственно в вывод* D *в упорядоченном моноиде* $\mathcal{M}(\mathcal{C}, \mathcal{A})$*;*
- $a_1 \not\rightsquigarrow a_2$*;*
- $(i, j_1), (i+1, j_1') \in \theta_1$, $(i, j_2), (i+1, j_2') \in \theta_2$.

Тогда $j_1 < j_2$ *в том и только в том случае, если* $j_1' < j_2'$.

Доказательство. На каждом шаге вывода происходит либо коммутация, либо порождение, либо поглощение. Можно легко проверить, что относительный порядок вхождений операторов a_1, a_2 изменяется только при коммутации этих операторов. При этом по условию леммы операторы a_1, a_2 не могут коммутировать. □

Лемма 5.2.3. *Пусть:*

- θ_1, θ_2 — *вхождения в вывод в упорядоченном моноиде* $\mathcal{M}(\mathcal{C}, \mathcal{A})$*;*
- *вхождение* θ_1 *косвенно порождается вхождением* θ_2*;*
- $(s, k_1) \in \theta_1$, $(s, k_2) \in \theta_2$.

Тогда $k_1 < k_2$.

Лемма 5.2.3 является прямым следствием теоремы 5.1.1 и леммы 5.2.2.

Лемма 5.2.4. *Пусть существует вывод в упорядоченном моноиде $\mathcal{M}(\mathcal{C}, \mathcal{A})$, начинающийся с цепочки h, в котором поглощается какое-либо главное вхождение. Тогда существует цепочка h', получающаяся из h вычеркиванием одного оператора, такая что $\langle h \rangle = \langle h' \rangle$.*

Доказательство. Рассмотрим кратчайший вывод D, в котором поглощается главное вхождение (для определенности — $\theta(1, j)$). В таком выводе D главные вхождения на всех шагах, кроме последнего, участвуют только в коммутации, а последний шаг состоит в поглощении вхождения $\theta(1, j)$. Рассмотрим последовательность вхождений $\theta_0, \ldots, \theta_k$ следующего вида:

- θ_0 — главное вхождение оператора a_0;
- вхождение θ_i оператора a_i порождается вхождением θ_{i-1}, $1 \leq i \leq k$;
- вхождение $\theta(1, j)$ поглощается вхождением θ_k.

Пройдемся по цепочкам вывода D и, удаляя из них некоторые операторы и анализируя результат, пошагово выстроим требуемый вывод D'. Операторы, удаляемые из цепочки $D\,[s]$, определяются следующим множеством вхождений Θ: выберем вхождение θ_{p_s} с наибольшим индексом p_s, $1 \leq p_s \leq k$, такое что проекция $\theta_{p_s}{\downarrow_s}$ определена; тогда Θ_s — все неглавные вхождения в вывод D, отличные от θ_{p_s}.

На s-м шаге построения вывода D', $1 \leq s \leq |D|$, рассматривается цепочка $h_s = D^{-\Theta_s}\,[s]$. Цепочку h_1 запишем в вывод D'. Если $s > 1$, то проверим отношения $h_{s-1} = h_s$ и $h_{s-1} \rightleftharpoons h_s$. Если $h_{s-1} = h_s$, то оставим вывод D' без изменений. Если $h_{s-1} \rightleftharpoons h_s$, то допишем цепочку h_s в конец вывода D'.

Остаётся рассмотреть следующий случай: ни одно из условий $h_{s-1} = h_s$, $h_{s-1} \rightleftharpoons h_s$ не выполнено. Можно легко убедиться, что этот случай в точности соответствует шагам, на которых порождаются вхождения θ_i, $3 \leq i \leq k$. Пусть,

для определённости, вхождение θ_i порождается на шаге $D\,[i] \overset{+}{\rightleftharpoons} D\,[i+1]$. Для завершения обоснования достаточно показать справедливость равенства $\langle h_{q_i} \rangle = \langle h_{q_i+1} \rangle$ — тогда вывод D' можно будет продолжить с цепочки h_{q_1} до цепочки h_{q_i+1}. Заметим, что цепочки h_{q_i} и $h_{q_i} + 1$ представимы в виде $h_{q_i} = ua_{i-1}w$, $h_{q_i+1} = ua_iw$. Чтобы обосновать равенство $\langle h_{q_i} \rangle = \langle h_{q_i+1} \rangle$, достаточно показать, что $\langle a_{i-1}w \rangle = \langle w \rangle$. Докажем это индукцией по индексу i.

База индукции: $i = 3$. Рассмотрим множество вхождений $\Theta = \Theta_{q_3} \cup \{\theta(q_3, m) \mid m < \theta_2{\downarrow_{q_3}}\}$. По леммам 5.2.1, 5.2.3 получаем два факта:

- последовательность $\left[D^{-\Theta}\,[q_2], \ldots, D^{-\Theta}\,[q_3]\right]$ есть вывод цепочки a_2w из некоторой цепочки w';
- последовательность $\left[D^{-\Theta\cup\{\theta_2\}}\,[q_2], \ldots, D^{-\Theta\cup\{\theta_2\}}\,[q_3]\right]$ есть вывод цепочки w из той же цепочки w'.

Из двух последних выводов можно легко получить вывод цепочки w из a_2w.

Индуктивный шаг. Пусть после q_{i-1}-го шага построения вывод D' имел длину r_1, а на текущем шаге построения имеет длину r_2. Пусть также θ'_{i-2} и θ'_{i-1} — вхождения в вывод D', в которые перешли вхождения θ_{i-2}, θ_{i-1}. Рассмотрим цепочки вывода D' с r_1-й по r_2-ю (здесь выделенный оператор a_{i-2} отвечает вхождению θ'_{i-2}, а выделенный оператор a_{i-1} — вхождению θ'_{i-1}):

$$ga_{i-2}g' \overset{+}{\rightleftharpoons} ga_{i-1}a_{i-2}g' \rightleftharpoons^* ga_{i-1}g' \overset{|\ *}{\rightleftharpoons} ua_{i-1}w.$$

Индуктивное предположение формулируется так: $\langle a_{i-2}g' \rangle = \langle g' \rangle$. Требуется показать, что $\langle a_{i-1}w \rangle = \langle w \rangle$.

Рассмотрим множество вхождений, отвечающее выделенной цепочке u: $\Theta = \left\{\theta(r_2, m) \mid m < \theta'_{i-1}{\downarrow_{r_2}}\right\}$ По леммам 5.2.1, 5.2.3 получаем два факта:

- последовательность $\left[D'^{-\Theta}\,[r_1], \ldots, D'^{-\Theta}\,[r_2]\right]$ есть вывод цепочки $a_{i-1}w$ из некоторой цепочки w';

- последовательность $\left[D'^{-\Theta\cup\left\{\theta'_{i-1}\right\}}[r_1], \ldots, D'^{-\Theta\cup\left\{\theta'_{i-1}\right\}}[r_2]\right]$ есть вывод цепочки w из той же цепочки w'.

Из двух последних выводов можно легко получить вывод цепочки w из $a_{i-1}w$. Обоснование индуктивного шага завершено.

В конце построения вывод D' будет иметь вид

$$D' = h \rightleftharpoons \ldots \rightleftharpoons gaa_k g' \overline{\rightleftharpoons} ga_k g'.$$

Здесь выделены оператор a, отвечающий поглощаемому главному вхождению θ' в вывод D', и оператор a_k, отвечающий вхождению θ'_k, поглощающему θ'. По лемме 5.2.3 выделенное вхождение θ'_k не входит в замыкание вхождения θ'. Более того, по построению вывода D' вхождение θ' не порождает и не поглощает никаких других вхождений. Удалим из вывода D' вхождение θ'. По лемме 5.2.1 прореживание результата будет выводом, имеющим следующий вид:

$$h' \rightleftharpoons \ldots \rightleftharpoons ga_k g'.$$

Из вывода D' и последнего вывода можно легко получить вывод цепочки h' из h. □

Цепочку h назовём *представителем* элемента $\langle h\rangle$. Если для всех представителей g элемента $\langle h\rangle$ верно $|g| \geq |h|$, то цепочку h будем называть *минимальным представителем*. Множество всех минимальных представителей элемента e обозначим записью $\lfloor e \rfloor$.

Лемма 5.2.5. *Пусть моноид $\mathcal{M}(\mathcal{C}, \mathcal{A})$ упорядочен. Тогда ни в каком выводе, начинающемся с минимального представителя, не поглощается ни одно главное вхождение.*

Лемма 5.2.5 является прямым следствием леммы 5.2.4 с учетом определения минимального представителя.

Лемма 5.2.6. *Пусть моноид $\mathcal{M}(\mathcal{C},\mathcal{A})$ упорядочен, $h \in \lfloor h \rfloor$, $a \in \mathfrak{O}$ и $\langle ah\rangle \neq \langle h\rangle$. Тогда $ah \in \lfloor ah \rfloor$.*

Доказательство. Достаточно показать, что ни в одном выводе, начинающемся с цепочки ah, не поглощается ни одно главное вхождение. Предположим, что это не так: существует вывод D, начинающийся с цепочки ah, в котором поглощается главное вхождение. Так как $\langle ah\rangle \neq \langle h\rangle$, из цепочки ah нельзя вывести цепочку h. Значит, по утверждению 5.2.4 из цепочки ah выводима цепочка ah', где h' получается из h вычеркиванием одного оператора. Удалим из вывода D замыкание вхождения $\theta(1,1)$. По лемме 5.2.1 прореживание D' результата удаления является выводом. По теореме 5.1.1 и лемме 5.2.2 ни одно главное вхождение, кроме вхождения $\theta(1,1)$, не входит в замыкание вхождения $\theta(1,1)$. Следовательно, вывод D' начинается с цепочки h.

Случай 1: вывод D' оканчивается цепочкой ah'. Тогда из выводов D и D' можно легко получить вывод цепочки h из ah, что противоречит условию леммы.

Случай 2: вывод D' оканчивается цепочкой g, отличной от ah'. Значит, цепочка g получена из ah' вычеркиванием хотя бы одного оператора. Тогда $|g| < |ah'| = |h|$, а значит, $h \notin \lfloor h \rfloor$, что противоречит условию леммы. □

Лемма 5.2.7. *Пусть моноид $\mathcal{M}(\mathcal{C},\mathcal{A})$ упорядочен и для его элементов e, e_1, e_2 верно равенство $e \circ e_1 = e \circ e_2$. Тогда $e_1 = e_2$.*

Доказательство. Предположим противное: существуют элементы e, e_1, e_2 моноида $\mathcal{M}(\mathcal{C},\mathcal{A})$, такие что $e \circ e_1 = e \circ e_2$ и $e_1 \neq e_2$.

Пусть $e = \langle h\rangle$. Тогда существует индекс j, $1 \leq j < |h|$, такой что $\langle \mathsf{suf}(h,j)\rangle \circ e_1 \neq \langle \mathsf{suf}(h,j)\rangle \circ e_2$ и $\langle \mathsf{suf}(h,j+1)\rangle \circ e_1 = \langle \mathsf{suf}(h,j+1)\rangle \circ e_2$. Можно повторить рассуждения для элементов $\langle \mathsf{suf}(h,j)\rangle \circ e_1$, $\langle \mathsf{suf}(h,j)\rangle \circ e_2$, $\langle h\,[|h|-j]\rangle$ вместо e_1, e_2, e соответственно, и таким образом без ограничения общности считать, что $e = \langle a\rangle$, где $a \in \mathfrak{O}$.

Пусть $h_1 \in \lfloor e_1 \rfloor$ и $h_2 \in \lfloor e_2 \rfloor$. Тогда по предположению цепочка ah_2 выводима из ah_1, а цепочка h_2 не выводима из h_1.

Случай 1: ни одна из цепочек ah_1, ah_2 не является минимальным представителем. Тогда по лемме 5.2.6 верны соотношения $h_1 \rightleftharpoons^* ah_1 \rightleftharpoons^* ah_2 \rightleftharpoons^* h_2$.

Случай 2: хотя бы одна из цепочек ah_1, ah_2 является минимальным представителем. Для определённости положим, что цепочка ah_1 является минимальным представителем. Рассмотрим вывод $D = ah_1 \rightleftharpoons \ldots \rightleftharpoons ah_2$. По лемме 5.2.5 в выводе D не поглощается ни одно главное вхождение. По теореме 5.1.1 и лемме 5.2.2 вхождения $\theta(1,1)$ и $\theta(|D|,1)$ совпадают, а вхождения, отвечающие цепочке h_2, не входят в замыкание вхождения $\theta(1,1)$. Удалим из вывода D замыкание вхождения $\theta(1,1)$. По лемме 5.2.1 прореживание результата является выводом цепочки h_2 из h_1. □

Лемма 5.2.8. *Пусть $\mathcal{A} = \emptyset$, e, e_1, e_2 — элементы моноида $\mathcal{M}(\mathcal{C}, \mathcal{A})$ и $e_1 \circ e = e_2 \circ e$. Тогда $e_1 = e_2$.*

Доказательство. По теореме 5.1.1 моноид $\mathcal{M}(\mathcal{C}, \mathcal{A})$ упорядочен. Используя симметрию соотношений коммутативности и отсутствие соотношений подавления, можно легко показать следующее: цепочка h_1 выводима из h_2 тогда и только тогда, когда цепочка ${h_1}^-$ выводима из ${h_2}^-$. Тогда справедливость леммы следует из леммы 5.2.7. □

Лемма 5.2.9. *Пусть цепочки h_1, h_2 являются минимальными представителями элемента e упорядоченного моноида $\mathcal{M}(\mathcal{C}, \mathcal{A})$. Тогда $h_1 \overset{|}{\rightleftharpoons}{}^* h_2$.*

Доказательство. Достаточно рассмотреть произвольный вывод цепочки h_2 из h_1, удалить в этом выводе все неглавные вхождения и применить леммы 5.2.1, 5.2.5. □

Весом $\|e\|$ элемента e моноида $\mathcal{M}(\mathcal{C}, \mathcal{A})$ назовём длину минимальных представителей этого элемента.

Лемма 5.2.10. *Пусть e_1, e_2 — элементы упорядоченного моноида $\mathcal{M}(\mathcal{C}, \mathcal{A})$, $h \in \mathfrak{O}^*$, $g \in \mathfrak{O}^+$, $n \in \mathbb{N}_0$ и $k \in \mathbb{N}$. Тогда справедливы следующие утверждения:*

1. $\|\langle h \rangle\| \leq |h|$;
2. $\|e_1 \circ e_2\| \leq \|e_1\| + \|e_2\|$;
3. $\|e_1 \circ e_2\| \geq \|e_2\|$.
4. $\|g^n \mathsf{pref}(g, k)\| \geq n + 1$;

Доказательство. Первые два утверждения напрямую вытекают из определения веса. Справедливость третьего утверждения следует из того, что минимальный представитель элемента e_2 является суффиксом минимального представителя элемента $e_1 \circ e_2$ (лемма 5.2.6).

Осталось обосновать последнее утверждение. Заметим, что $\mathcal{A}$ — строгий частичный порядок на множестве $\mathfrak{O}$ (теорема 5.1.1). Пусть a и a' — максимальные относительно порядка $\mathcal{A}$ операторы, содержащийся в цепочках g, $\mathsf{pref}(g, k)$ соответственно. Тогда по теореме 5.1.1 и лемме 5.2.2 ни в каком выводе, начинающемся с цепочки $g^n \mathsf{pref}(g, k)$ не могут быть поглощены хотя бы n главных вхождений оператора a (по одному на каждую копию цепочки g) и хотя бы одно главное вхождение оператора a'. □

5.3. Распознавание подавления операторов

Данный раздел посвящен следующей задаче: в предположении об упорядоченности моноида $\mathcal{M}(\mathcal{C}, \mathcal{A})$ описать автомат, который, принимая на вход цепочку h^-, приходит в состояние, по метке которого можно однозначно заключить, для каких операторов a выполнено равенство $\langle ah \rangle = \langle h \rangle$. Теорема 5.3.1 гласит, что такой автомат существует для любого упорядоченного моноида $\mathcal{M}(\mathcal{C}, \mathcal{A})$; в доказательстве теоремы приводится явное описание такого автомата.

Автомат $\mathfrak{A}$, *распознающий подавление операторов* в моноиде $\mathcal{M}(\mathcal{C}, \mathcal{A})$ над алфавитом $\mathfrak{O}$, обладает следующими свойствами:

- $\mathfrak{A}.A = \mathfrak{O}$, $\mathfrak{A}.L = 2^{\mathfrak{O}}$;
- согласованность с моноидом: если $\langle h \rangle = \langle g \rangle$, то $\mathfrak{A}(h^-) = \mathfrak{A}(g^-)$;
- согласованность с подавлением: $\mathfrak{A}.M(h^-) = \{a \mid \langle ah \rangle = \langle h \rangle\}$.

Чтобы описать автомат, распознающий подавление операторов, введём понятие *канонического вывода*. Он может быть определён индуктивно следующим образом (здесь $a \in \mathfrak{O}$, $h \in \mathfrak{O}^*$):

1. $h \overset{1^+}{\rightleftharpoons} ah$ — канонический вывод *глубины* 1;
2. пусть D — канонический вывод глубины i; тогда:
   а. $a\,\mathsf{hd}(D) \rightleftharpoons \ldots \rightleftharpoons a\,\overline{\mathsf{hd}}(D) \overset{1^|}{\rightleftharpoons} h$ — канонический вывод *глубины* i;
   б. $\mathsf{hd}(D) \rightleftharpoons \ldots \rightleftharpoons \overline{\mathsf{hd}}(D) \overset{1^+}{\rightleftharpoons} a\,\overline{\mathsf{hd}}(D) \rightleftharpoons \ldots \rightleftharpoons a\,\mathsf{hd}(D)$ — канонический вывод *глубины* $(i+1)$.

Содержательно, канонический вывод начинается с цепочки h, оканчивается цепочкой ah и имеет следующую структуру: на первом шаге порождается некоторый оператор a_1; последний порождённый оператор a_i продвигается влево коммутацией, порождает оператор a_{i+1}, затем продвигается вправо до позиции, на которой был порождён, и поглощается теми же действиями, которыми до этого был порождён; так происходит, пока требуемый оператор a не породится и не окажется в самой левой позиции цепочки, после чего вывод прекращается. Как видно из содержательного описания, канонические выводы имеют довольно простую по сравнению с произвольными выводами структуру. Несмотря на такую простоту, рассмотрение канонических выводов оказывается достаточным для анализа того, какие операторы могут быть порождены в самой левой позиции цепочки.

Лемма 5.3.1. *Пусть e — элемент упорядоченного моноида $\mathcal{M}(\mathcal{C}, \mathcal{A})$, $a \in \mathfrak{O}$ и $\langle a \rangle \circ e = e$. Тогда существует канонический вывод $h \rightleftharpoons \ldots \rightleftharpoons ah$, где $h \in \lfloor e \rfloor$.*

Доказательство. Рассуждения в данном обосновании схожи с рассуждениями в обосновании леммы 5.2.4 для поглощаемого главного вхождения $\theta(1,1)$. Далее в обосновании знак "✓" будет обозначать "объект значит то же, что и одноимённый объект в обосновании леммы 5.2.4" и "в доказательстве леммы 5.2.4 приведены рассуждения, достаточные для обоснования факта".

Рассмотрим произвольный вывод D минимального представителя g из ag. Пошагово перестроим вывод D в другой вывод D', на основании которого будет легко обосновано существование требуемого канонического вывода. На каждом шаге будем либо не изменять вывод D', либо добавлять в конец D' цепочку, либо изменять цепочки, добавленные в D' к текущему шагу. Для каждого (s-го, $1 \leq s \leq |D|$) шага будет определено множество вхождений Θ'_s, удаляемых из s-й и $(s-1)$-й цепочек вывода D. Таким образом, на s-м шаге будут рассматриваться цепочки $h_s = D^{-\Theta'_s}[s]$ и $g_s = D^{-\Theta'_s}[s-1]$.

Первый шаг построения: $\Theta'_1 = \emptyset$, в вывод D' записывается цепочка $\mathsf{hd}(D)$. Рассмотрим s-й шаг построения, где $s > 1$. Имеем множество вхождений Θ_s, индекс p_{s-1} и вхождение $\theta_{p_{s-1}}$ (✓). Тогда $\Theta'_s = \Theta'_{s-1} \cup \Theta_s \cup \left\{\theta(s-1, m) \mid m > \theta_{p_{s-1}}{\downarrow_{s-1}}\right\}$. Особо подчеркнём, что множество Θ'_s подобрано так, чтобы в цепочке g_s оказались удалены все вхождения справа от $\theta_{p_{s-1}}$ и все неглавные вхождения, отличные от $\theta_{p_{s-1}}$. Тогда возможны только следующие четыре случая.

Случай 1: $g_s = h_s$. Оставляем вывод D' без изменений.

Случай 2 (продвижение $\theta_{p_{s-1}}$ на позицию влево): $g_s \overset{|g_s|-1|}{\rightleftharpoons} h_s$. Дополняем вывод D' до вывода $D' \overset{|g_s|-1|}{\rightleftharpoons} u$.

Случай 3 (коммутация вхождений, отличных от $\theta_{p_{s-1}}$): $g_s \overset{i|}{\rightleftharpoons} h_s$, где $i < |g_s|-1$. Меняем каждую цепочку $D'[j]$ вывода на цепочку u_j, для которой верно $D'[j] \overset{i|}{\rightleftharpoons} u_j$.

Случай 4 (порождение вхождения $\theta_{p_{s-1}+1}$): $g_s \neq h_s$, $\overline{\mathsf{tl}}(g_s) = \overline{\mathsf{tl}}(h_s)$. Продолжим вывод D' с его последней цепочки $ua_{p_{s-1}}w$ до цепочки $ua_{p_{s-1}+1}w$ (✓).

По окончании построения получим вывод D' вида $g'ag'' \rightleftharpoons \ldots \rightleftharpoons g'aa_kg'' \overset{-}{\rightleftharpoons} g'a_kg''$, где $g'g''$ — минимальный представитель и все операторы из g' коммутируют с a. Удалив вхождение, отвечающее выделенному оператору a и проредив результат, получим вывод D'' вида $g'g'' \rightleftharpoons \ldots \rightleftharpoons g'a_kg''$ (✓). Проанализировав структуру вывода D', можно легко убедиться, что $D'' \rightleftharpoons \mathsf{tl}(D') \overset{|g'|}{\rightleftharpoons} h^1 \overset{|g'|-1|}{\rightleftharpoons} \ldots \overset{1|}{\rightleftharpoons} ag'g''$ — требуемый канонический вывод. □

Чтобы описать автомат, распознающий подавление операторов, достаточно привести конечное множество, в которое можно отобразить любое семейство канонических выводов так, чтобы выполнялось следующее:

1. по образу множества всех канонических выводов, начинающихся в минимальных представителях элемента e, можно определить, для каких операторов a верно равенство $\langle a \rangle \circ e = e$;

2. по образу множества всех канонических выводов, начинающихся в минимальных представителях элемента e, и по оператору a можно определить образ множества всех канонических выводов, начинающихся в минимальных представителях элемента $\langle a \rangle \circ e$.

Имея конечное множество с такими свойствами, можно легко описать требуемый автомат: достаточно объявить это множество состояниями автомата, разметить состояния автомата согласно первому свойству и соединить их дугами согласно второму свойству. Опишем такое множество.

Рассмотрим канонический вывод D цепочки ah из h. Пройдемся по его цепочкам $D[i]$, $2 \leq i \leq |D|$, и для каждой из них выберем самое левое присутствующее неглавное вхождение θ'_i, то есть неглавное вхождение, для которого значение $\theta'_i{\downarrow_i}$ определено и принимает наименьшее возможное значение. Проредив последовательность $\theta'_2, \ldots, \theta'_{|D|}$, получим из неё последовательность

вхождений $\theta_1, \ldots, \theta_k$. Пусть $a_1, \ldots, a_k$ — операторы, приписанные вхождениям $\theta_1, \ldots, \theta_k$ соответственно. Отметим, что в содержательном описании канонического вывода, приведённом сразу его формального определения, одноимённые операторы a_i имеют то же значение. Последовательность $a_1, \ldots, a_k$ назовём *цепью канонического вывода* D. Пусть $\mathfrak{O}_i$ — множество операторов, приписанных вхождениям, с которыми коммутирует вхождение θ_i, $1 \leq i \leq k$. Последовательность $\mathfrak{O}_1, \ldots, \mathfrak{O}_k$ назовём *характеристикой канонического вывода* D. Наконец, рассмотрим оператор a, приписанный вхождению θ, порождающему вхождение θ_1. Назовём этот оператор *главным*.

Кодом (канонического) вывода назовём систему cd, включающую в себя главный оператор $cd.a$, цепь $cd.ch$ и характеристику $cd.S$:

- $cd.a \in \mathfrak{O}$;
- $cd.ch$ — последовательность операторов;
- $cd.S$ — последовательность множеств операторов, $|cd.S| = |cd.ch|$;
- $\mathsf{hd}(cd.ch) \leftarrow cd.a$;
- $cd.ch\,[i] \leftarrow cd.ch\,[i-1]$, $1 < i \leq |cd.ch|$.

Обозначим записью $\mathcal{CD}$ множество всевозможных кодов выводов. Если рассматриваемый моноид $\mathcal{M}(\mathcal{C}, \mathcal{A})$ упорядочен, то по теореме 5.1.1 для всякого кода вывода cd верно $|cd.ch| < |\mathfrak{O}|$. Это означает, что множество $\mathcal{CD}$ конечно (если моноид $\mathcal{M}(\mathcal{C}, \mathcal{A})$ упорядочен). Однако одного только множества $\mathcal{CD}$ недостаточно для описания требуемого автомата.

Пусть e — элемент моноида $\mathcal{M}(\mathcal{C}, \mathcal{A})$ и a — оператор. Опишем то, как соотносятся между собой канонические выводы, начинающиеся с минимальных представителей элементов e и $\langle a \rangle \circ e$. Формальное описание и обоснование этого соотношения будут приведены в леммах 5.3.2, 5.3.3. Если $\langle a \rangle \circ e = e$, то элементы $\langle a \rangle \circ e$ и e, очевидно, имеют одниаковые множества канонических выводов.

Пусть теперь $\langle a \rangle \circ e \neq e$. Тогда любой минимальный представитель элемента $\langle a \rangle \circ e$ представим в виде hag, где hg — минимальный представитель элемента e и все символы цепочки h коммутируют с оператором a. Коды канонических выводов, начинающихся в цепочке hag, главные операторы которых отличны от a, могут быть легко восстановлены по кодам канонических выводов, начинающихся в цепочке hg: достаточно проверить, что символ a коммутирует со всеми символами set_h, и вставить его в нужное место цепочки, и все эти действия легко выражаются в терминах кодов. Если же a — главный оператор канонического вывода, начинающегося с цепочки hag, то код этого вывода, вообще говоря, не может быть восстановлен по кодам выводов, начинающихся в цепочке hg. Информацию о том, какими будут коды канонических выводов с главным символом a для элемента $\langle a \rangle \circ e$, $a \in \mathfrak{O}$, назовём *прогнозом (канонических выводов)* для элемента e.

Код ecd элемента моноида $\mathcal{M}(\mathcal{C}, \mathcal{A})$ включает в себя множество кодов выводов $ecd.cd$ и прогноз $cd.fc$. С точки зрения синтаксиса, прогноз также является множеством кодов выводов; семантика прогноза пояснена выше. Обозначим записью $\mathcal{ECD}$ множество всевозможных кодов элементов упорядоченного моноида $\mathcal{M}(\mathcal{C}, \mathcal{A})$. Так как множество $\mathcal{CD}$ конечно, конечным является и множество $\mathcal{ECD}$.

Опишем *функцию преобразования кодов* $\mathsf{Upd} : \mathcal{ECD} \times \mathfrak{O} \to \mathcal{ECD}$, по коду элемента e и оператору a предоставляющую код элемента $\langle a \rangle \circ e$. Для этого введём ряд вспомогательных функций.

- Функция $\mathsf{push} : \mathcal{CD} \times \mathfrak{O} \times \mathbb{N} \to \mathcal{CD}$. Значением $cd' = \mathsf{push}(cd, a, i)$ является код канонического вывода, получаемого из вывода с кодом cd вставкой оператора a так, чтобы в выводе с ним коммутировал оператор $cd.ch\,[i]$:

$$\begin{aligned} cd'.a &= cd.a; \\ cd'.ch &= cd.ch; \\ cd'.S\,[i] &= cd.S\,[i] \cup \{a\}\,; \\ cd'.S\,[j] &= cd.S\,[j]\,, \text{ если } j \neq i. \end{aligned}$$

- Функция $\mathsf{push}_c : \mathcal{CD} \times \mathfrak{O} \times \mathbb{N} \to 2^{\mathcal{CD}}$ делает то же, что и функция push, с дополнительной проверкой того, можно ли вставить оператор a так, как это пояснено в содержательном описании преобразования канонических выводов.

$$\mathsf{push}_c(cd, a, i) = \begin{cases} \{\mathsf{push}(cd, a, i)\}\,, & \text{если } a \leftrightsquigarrow cd.ch\,[i] \\ & \text{и } a \leftrightsquigarrow cd.S\,[i+1] \cup \cdots \cup \overline{\mathsf{hd}}(cd.S)\,; \\ \emptyset, & \text{иначе.} \end{cases}$$

- Функция $\mathsf{push}_f : \mathcal{CD} \times \mathfrak{O} \times \mathbb{N} \to 2^{\mathcal{CD}}$ делает то же, что и push_c, для прогноза, то есть в предположении о том, что главный оператор будет добавлен позднее.

$$\mathsf{push}_f(cd, a, i) = \begin{cases} \mathsf{push}_c(cd, a, i)\,, & \text{если } a \leftrightsquigarrow cd.a; \\ \emptyset, & \text{иначе.} \end{cases}$$

- $\mathsf{push}_c(cd, a) = \bigcup_{1 \leq i \leq |cd.ch|} \mathsf{push}_c(cd, a, i)$;

- $\mathsf{push}_f(cd, a) = \bigcup_{1 \leq i \leq |cd.ch|} \mathsf{push}_f(cd, a, i)$.

- Функция $\mathsf{id}_c : \mathcal{CD} \times a \to 2^{\mathcal{CD}}$ проверяет, можно ли вставить оператор a правее вхождения, отвечающего главному оператору, соблюдая при этом требования коммутации в содержательном описании преобразования канонических выводов.

$$\mathsf{id}_c(cd, a) = \begin{cases} \{cd\}, & \text{если } a \leftrightsquigarrow cd.a \\ & \text{и } a \leftrightsquigarrow \mathsf{hd}(cd.S) \cup \cdots \cup \overline{\mathsf{hd}}(cd.S)\,; \\ \emptyset, & \text{иначе.} \end{cases}$$

- Функция $\mathsf{id}_f : \mathcal{CD} \times a \to 2^{\mathcal{CD}}$ делает то же, что и id_c, для прогноза, то есть в предположении о том, что главный оператор будет добавлен позднее.

$$\mathsf{id}_c(cd, a) = \begin{cases} \{cd\}, & \text{если } a \leftrightsquigarrow \mathsf{hd}(cd.S) \cup \cdots \cup \overline{\mathsf{hd}}(cd.S)\,; \\ \emptyset, & \text{иначе.} \end{cases}$$

Функция преобразования кодов Upd может быть описана следующим образом. Пусть ecd — код элемента моноида $\mathcal{M}(\mathcal{C}, \mathcal{A})$ и $a \in \mathfrak{D}$. Если множество $ecd.cd$ содержит код cd, для которого верно $\overline{\mathsf{hd}}(cd.ch) = a$, то $\mathsf{Upd}(ecd, a) = ecd$. Иначе значение $ecd' = \mathsf{Upd}(ecd, a)$ определяется так:

$$\begin{aligned} ecd'.cd &= \bigcup_{cd \in ecd.cd} (\mathsf{push}_c(cd, a) \cup \mathsf{id}_c(cd, a)) \cup \{cd \mid cd \in ecd.fc, cd.a = a\}\,; \\ ecd'.fc &= \bigcup_{cd \in ecd.fc} \left(\mathsf{push}_f(cd, a) \cup \mathsf{id}_f(cd, a)\right). \end{aligned}$$

Перейдём к обоснованию свойств кодов элементов моноида $\mathcal{M}(\mathcal{C}, \mathcal{A})$.

Лемма 5.3.2. *Пусть e — элемент упорядоченного моноида $\mathcal{M}(\mathcal{C}, \mathcal{A})$, $a \in \mathfrak{D}$ и $\langle a \rangle \circ e \neq e$. Тогда $\lfloor \langle a \rangle \circ e \rfloor = \{hag \mid hg \in \lfloor e \rfloor,\ a \leftrightsquigarrow \mathsf{set}_h\}$.*

Доказательство. Пусть $hg \in \lfloor e \rfloor$ и $a \leftrightsquigarrow \mathsf{set}_h$. Покажем включение $hag \in \lfloor \langle a \rangle \circ e \rfloor$. Цепочка ahg выводима из hag: $hag \overset{|h|}{\rightleftharpoons} \ldots \overset{1}{\rightleftharpoons} ahg$. При этом по лемме 5.2.6 справедливо соотношение $ahg \in \lfloor \langle a \rangle \circ e \rfloor$.

Пусть $h \in \lfloor \langle a \rangle \circ e \rfloor$. Покажем, что цепочка h представима в виде $h = g_1 a g_2$, где $g_1 g_2 \in \lfloor e \rfloor$ и $a \leftrightsquigarrow \mathsf{set}_{g_1}$. Пусть $g \in \lfloor e \rfloor$. По лемме 5.2.6 имеем соотношение $ag \in \lfloor \langle a \rangle \circ e \rfloor$. По лемме 5.2.9 существует вывод $D = h \overset{|}{\rightleftharpoons} \ldots \overset{|}{\rightleftharpoons} ag$. По лемме 5.2.5 выделенное вхождение оператора a в вывод D является главным. Пусть, для определенности, это вхождение $\theta(1, i)$. Тогда цепочка h представима в виде

$h = h_1 a h_2$, где $|h_1| = i - 1$. По лемме 5.2.2 получаем соотношение $a \leftrightsquigarrow \mathsf{set}_{h_1}$. Удалим из вывода D вхождение $\theta(1, i)$ и проредим результат. По лемме 5.2.1 получим вывод $h_1 h_2 \overset{|}{\rightleftharpoons} \ldots \overset{|}{\rightleftharpoons} g$. Следовательно, $h_1 h_2 \in \lfloor e \rfloor$. □

Лемма 5.3.3. *Пусть:*

- e *— элемент упорядоченного моноида* $\mathcal{M}(\mathcal{C}, \mathcal{A})$ *и* $a, b \in \mathfrak{D}$.
- ecd *— код элемента моноида* $\mathcal{M}(\mathcal{C}, \mathcal{A})$;
- $ecd.cd$ *есть множество кодов всех канонических выводов, начинающихся с минимальных представителей элемента* e;
- $\left\{cd \mid cd \in ecd.fc,\ \overline{\mathsf{hd}}(cd.ch) = b\right\}$ *есть множество кодов всех канонических выводов, начинающихся с минимальных представителей элемента* $\langle b \rangle \circ e$ *и имеющих оператор* b *в качестве главного.*
- $ecd' = \mathsf{Upd}(ecd, a)$.

Тогда:

- $ecd'.cd$ *есть множество кодов всех канонических выводов, начинающихся в минимальных представителях элемента* $\langle a \rangle \circ e$;
- $\left\{cd \mid cd \in ecd'.fc,\ \overline{\mathsf{hd}}(cd.ch) = b\right\}$ *есть множество кодов всех канонических выводов, начинающихся с минимальных представителей элемента* $\langle ba \rangle \circ e$.

Достаточным обоснованием леммы 5.3.3 являются леммы 5.3.1 5.3.2 и содержательные пояснения, относящиеся к кодам канонических выводов, прогнозу, кодам элементов моноида и функции преобразования кодов.

Теорема 5.3.1. *Если моноид* $\mathcal{M}(\mathcal{C}, \mathcal{A})$ *над алфавитом* $\mathfrak{D}$ *упорядочен, то существует автомат, распознающий подавление операторов в этом моноиде.*

Доказательство. Опишем явно автомат $\mathfrak{A}$, распознающий подавление операторов в моноиде $\mathcal{M}(\mathcal{C}, \mathcal{A})$. Прежде всего, автомат прочитывает операторы: $\mathfrak{A}.A = \mathfrak{O}$. Состояниями автомата являются всевозможные коды элементов моноида: $\mathfrak{A}.Q = \mathcal{ECD}$. Инициальное состояние $\mathfrak{A}.q$ имеет следующий вид: $\mathfrak{A}.q.cd = \emptyset$; $\mathfrak{A}.q.fc$ состоит из всевозможных кодов выводов cd, таких что характеристика $cd.S$ состоит только из пустых множеств. Состояния автомата размечены множествами операторов: $\mathfrak{A}.L = 2^{\mathfrak{O}}$. Разметка состояний автомата имеет следующий вид: $a \in \mathfrak{A}.M(q)$ тогда и только тогда, когда множество $q.cd$ содержит элемент cd, такой что $\overline{\mathsf{hd}}(cd.ch) = a$. Функция переходов автомата есть функция преобразования кодов: $\mathfrak{A}.T = \mathsf{Upd}$.

Чтобы обосновать, что автомат $\mathfrak{A}$ распознаёт подавление операторов в моноиде $\mathcal{M}(\mathcal{C}, \mathcal{A})$, достаточно (согласно определению) показать его согласованность с моноидом и с подавлением. Можно легко убедиться, что инициальное состояние автомата $\mathfrak{A}$ является кодом элемента $\langle\lambda\rangle$. Из этого и леммы 5.3.3 следует, что состояние $\mathfrak{A}(h)$ является кодом элемента $\langle h\rangle$, $h \in \mathfrak{O}^*$. Согласованность автомата $\mathfrak{A}$ с моноидом $\mathcal{M}(\mathcal{C}, \mathcal{A})$ следует из того, что код каждого элемента этого моноида определён однозначно. Согласованность автомата $\mathfrak{A}$ с подавлением следует из леммы 5.3.1 и выбора разметки состояний автомата. □

5.4. Граф совместных вычислений

В данном разделе и в разделах 5.5, 5.6 считаем заданными:

- конечные алфавиты операторов $\mathfrak{O}$ и логических условий $\mathfrak{L}$;
- программы π_1, π_2;
- упорядоченную шкалу $\mathcal{F} = \mathcal{F}(\mathcal{C}, \mathcal{A})$.

Чтобы сделать описание графа совместных вычислений понятнее, предварим его описанием стратегии совместного выполнения и различия вычисли-

тельных конфигураций. Напомним, что вычислительной конфигурацией программы π является пара (h, l), состоящая из цепочки h и состояния управления l программы π.

Стратегия совместного выполнения может быть описана так. Пусть (h_1, l_1), (h_2, l_2) — текущие конфигурации программ π_1, π_2 соответственно.

- Если $l_1 \neq \pi_1.ex$, $l_2 \neq \pi_2.ex$ и $[\![h_1]\!] = [\![h_2]\!]$, то активны обе программы.
- Если $l_1 \neq \pi_1.ex$, $l_2 \neq \pi_2.ex$ и $[\![h_1]\!] \parallel [\![h_2]\!]$, то активна программа π_1.
- Если $l_1 \neq \pi_1.ex$ и $[\![h_1]\!] \prec [\![h_2]\!]$, то активна программа π_1.
- Если $l_2 \neq \pi_2.ex$ и $[\![h_1]\!] \succ [\![h_2]\!]$, то активна программа π_2.
- Иначе обе программы неактивны.

Различие конфигураций (h_1, l_1), (h_2, l_2) определяется так. В различии без изменений сохраняются состояния управления l_1, l_2. Цепочки h_1, h_2 преобразуются в состояния данных $s_1 = [\![h_1]\!]$, $s_2 = [\![h_2]\!]$. Затем определяются состояния данных s, такие что $s \prec s_1$, $s \prec s_2$. Среди них выбирается состояние данных s' с наибольшим весом. В различии сохраняются состояния данных s'_1, s'_2, для которых верно $s' \circ s'_1 = s_1$, $s' \circ s'_2 = s_2$. Заметим, что с учетом выбора стратегии совместного выполнения вес состояния s'_2 в любом различии, получаемом в процессе совместного выполнения программ, не превышает единицы. Обоснование этого факта приведено в дальнейших рассуждениях.

Опишем граф совместных вычислений $\Gamma = \Gamma(\pi_1, \pi_2, \mathcal{F})$. Вершина v графа Γ включает в себя компоненты $v.l_1$, $v.l_2$, $v.s_1$, $v.s_2$:

- $v.l_i \in \pi_i.L$ — текущие состояния управления программ;
- $v.s_1, v.s_2 \in \mathcal{F}.S$ — состояния данных, сохранённые в различии.

Корнем графа объявляется вершина v_{root}: $v_{root}.l_1 = \pi_1.en$; $v_{root}.l_2 = \pi_2.en$; если $[\![\pi_1.en]\!] = [\![\pi_2.en]\!]$, то $v_{root}.s_1 = v_{root}.s_2 = [\![\lambda]\!]$; если $[\![\pi_1.en]\!] \parallel [\![\pi_2.en]\!]$, то

$v_{root}.s_1 = [\![\pi_1.en]\!]$ и $v_{root}.s_2 = [\![\pi_2.en]\!]$; если $[\![\pi_1.en]\!] \prec [\![\pi_2.en]\!]$, то $v_{root}.s_1 = [\![\lambda]\!]$ и $v_{root}.s_2 = [\![\pi_2.en]\!]$; если $[\![\pi_1.en]\!] \succ [\![\pi_2.en]\!]$, то $v_{root}.s_1 = [\![\pi_1.en]\!]$ и $v_{root}.s_2 = [\![\pi_2.en]\!]$.

Среди вершин графа Γ выделяются *терминальные* вершины v — удовлетворяющие одному из следующих условий:

1. $v.l_1 = \pi_1.en$, $v.l_2 \neq \pi_2.en$, $v.s_2 \neq [\![\lambda]\!]$;
2. $v.l_1 \neq \pi_1.en$, $v.l_2 = \pi_2.en$, $v.s_1 \neq [\![\lambda]\!]$;
3. $v.l_1 = \pi_1.en$, $v.l_2 = \pi_2.en$, $(v.s_1, v.s_2) \neq ([\![\lambda]\!], [\![\lambda]\!])$;
4. $v.l_1 = \pi_2.en$, $v.l_2 = \pi_2.en$, $v.s_1 = v.s_2 = [\![\lambda]\!]$.

При выполнении условия $i \in \{1, 2, 3\}$ будем называть терминальную вершину v *опровергающей вершиной типа* i. Метки дуг графа Γ выбираются из множества $(\mathfrak{L} \cup \{\varepsilon\})^2$. Содержательно, составляющая $\ell[i]$ метки ℓ есть логическое условие, по которому программа π_i совершает переход согласно стратегии совместного выполнения; значение ε означает неактивность программы (и отсутствие перехода). Из терминальной вершины не исходит ни одной дуги. Рассмотрим нетерминальну вершину v графа Γ.

- Если $v.l_1 \neq \pi_1.ex$, $v.l_2 \neq \pi_2.ex$ и $v.s_1 = v.s_2 = [\![\lambda]\!]$, то из вершины v исходит ровно по одной дуге с меткой (σ, σ) для каждого логического условия σ.
- Если $v.l_2 = \pi_2.ex$ или $v.s_2 \neq [\![\lambda]\!]$, то из вершины v исходит ровно по одной дуге с меткой (σ, ε) для каждого логического условия σ.
- Если либо $v.l_1 = \pi_1.ex$, либо $v.s_1 \neq [\![\lambda]\!]$ и $v.s_2 = [\![\lambda]\!]$, то из вершины v исходит ровно по одной дуге с меткой (ε, σ) для каждого логического условия σ.

Для завершения описания графа Γ определим вершину v', в которую ведет дуга с меткой ℓ из вершины v:

- если $\ell[1] = \varepsilon$, то $v'.l_1 = v.l_1$ и $s_1 = v.s_1$;
- если $\ell[1] = \sigma \in \mathfrak{L}$, то $v.l_1 \xrightarrow{\sigma}_{\pi_1} v'.l_1$ и $s_1 = v.s_1 \circ [\![v'.l_1]\!]$;
- если $\ell[2] = \varepsilon$, то $v'.l_2 = v.l_2$ и $s_2 = v.s_2$;
- если $\ell[2] = \sigma \in \mathfrak{L}$, то $v.l_2 \xrightarrow{\sigma}_{\pi_2} v'.l_2$ и $s_2 = [\![v'.l_1]\!]$ (здесь же отметим, что в этом случае $v.s_2 = [\![\lambda]\!]$);
- если $s_1 = s_2$, то $v'.s_1 = v'.s_2 = [\![\lambda]\!]$;
- если $s_1 \| s_2$, то $v'.s_1 = s_1$ и $v'.s_2 = s_2$;
- если $s_i \prec s_{3-i}$, то $v'.s_i = [\![\lambda]\!]$ и $v'.s_{3-i} = s$, где $s_{3-i} = s_i \circ s$.

По лемме 5.2.7 состояние s в последнем пункте определения вершины v' определено однозначно, а значит, определение вершины v' корректно.

Далее словом "маршрут" будем обозначать (как конечные, так и бесконечные) маршруты графа Г, начинающиеся в его корне. Запись $|\Omega|$, где Ω — маршрут, будет использоваться для обозначения длины маршрута, то есть числа вершин, входящих в этот маршрут. Чтобы поставить соответствие между маршрутами и парами трасс программ π_1, π_2, введем понятие проекции маршрута. *Проекция* $\mathsf{pr}(\Omega)$ маршрута Ω может быть определена индуктивно следующим образом:

- $\mathsf{pr}(\Omega) = (\mathsf{pr}_1(\Omega), \mathsf{pr}_2(\Omega))$;
- $\mathsf{pr}_i(v_{root}) = \pi_i.en$;
- если $\ell[i] = \varepsilon$, то $\mathsf{pr}_i\left(\Omega \xrightarrow{\ell} v\right) = \mathsf{pr}_i(\Omega)$;
- если $\ell[i] = \sigma \in \mathfrak{L}$, то $\mathsf{pr}_i\left(\Omega \xrightarrow{\ell} v\right) = \mathsf{pr}_i(\Omega) \xrightarrow{\sigma}_{\pi_i} v.l_i$;
- если Ω — бесконечный маршрут, то $\mathsf{pr}_i(\Omega)$ есть предел последовательности $\mathsf{pr}_i(\mathsf{pref}(\Omega, 1)), \mathsf{pr}_i(\mathsf{pref}(\Omega, 2)), \ldots$.

Составляющую $\mathsf{pr}_i(\Omega)$ проекции $\mathsf{pr}(\Omega)$ будем называть i-проекцией маршрута Ω. В следующих далее леммах 5.4.1–5.4.3 сформулированы основные свойства маршрутов, показывающие их связь с трассами программ π_1, π_2.

Лемма 5.4.1. *1-проекция и 2-проекция маршрута Ω суть $\mathcal{F}$-совместные трассы программ π_1, π_2. Более того, если маршрут Ω оканчивается в вершине v, то:*

- *существует состояние данных s, для которого верно $[\![\mathsf{pr}_1(\Omega)]\!] = s \circ v.s_1$ и $[\![\mathsf{pr}_2(\Omega)]\!] = s \circ v.s_2$;*
- *либо $v.s_2 = [\![\lambda]\!]$, либо $v.s_2 \in \{[\![a]\!] \mid a \in \mathfrak{O}\}$;*
- *если $v.s_1 \neq [\![\lambda]\!]$ и $v.s_2 \neq [\![\lambda]\!]$, то $v.s_1 \| v.s_2$.*

Лемма 5.4.1 легко обосновывается индукцией по длине маршрута Ω с привлечением утверждения 3.1.2. Далее тот факт, что проекциями маршрута являются трассы программ π_1, π_2, будет использоваться без ссылки на лемму 5.4.1.

Лемма 5.4.2. *Пусть cp_1, cp_2 — $\mathcal{F}$-совместные вычисления программ π_1, π_2 соответственно. Тогда существует маршрут Ω, бесконечный либо оканчивающийся в терминальной вершине, такой что $\mathsf{pr}_1(\Omega)$, $\mathsf{pr}_2(\Omega)$ суть префиксы вычислений cp_1, cp_2 соответственно.*

Лемма 5.4.2 легко обосновывается индуктивным построением маршрута Ω согласно приведённой в начале раздела стратегии совместного выполнения. Метки дуг при построении выбираются согласно вычислениям cp_1, cp_2. Существование выбираемой метки обосновывается утверждением 3.1.2 и леммой 5.4.1.

Лемма 5.4.3. *Пусть Ω — маршрут, $i \in \{1, 2\}$ и трасса $\mathsf{pr}_i(\Omega)$ является префиксом трассы tr программы π_i. Тогда трассы tr и tr_{3-i} $\mathcal{F}$-совместны.*

Доказательство. Без ограничения общности положим $i = 1$. Предположим, что трассы tr и $\mathsf{pr}_2(\Omega)$ не являются $\mathcal{F}$-совместными. По лемме 5.4.1 трассы $\mathsf{pr}_1(\Omega)$ и $\mathsf{pr}_2(\Omega)$ $\mathcal{F}$-совместны. Тогда по утверждению 3.1.2 получим следующее: существуют собственный префикс tr'_1 трассы tr и собственный префикс tr'_2 трассы $\mathsf{pr}_2(\Omega)$, такие что $|tr'_1| \geq |\mathsf{pr}_1(\Omega)|$ и $[\![tr'_1]\!] = [\![tr'_2]\!]$.

Рассмотрим кратчайший префикс Ω' маршрута Ω, для которого верно $\mathsf{pr}_2(\Omega) = tr'_2$. По выбору трассы tr'_2 и лемме 5.4.1 маршрут Ω' продолжается до маршрута Ω по меткам вида ℓ, где $\ell\,[2] = \varepsilon$, пока 1-проекция не станет равной $\mathsf{pr}_1(\Omega)$. Таким образом нами получен маршрут Ω'' с проекциями $\mathsf{pr}_1(\Omega'') = \mathsf{pr}_1(\Omega)$ и $\mathsf{pr}_2(\Omega'') = tr'_2$. По выбору трассы tr'_2 верно неравенство $[\![tr'_2]\!] \succeq [\![\mathsf{pr}_1(\Omega)]\!]$, а значит, при попытке достроить маршрут Ω'' до Ω будет выбрана дуга с меткой ℓ, такой что $\ell\,[1] \in \mathfrak{L}$. В результате получим префикс маршрута Ω, 1-проекция которого превосходит по длине 1-проекцию маршрута Ω, чего быть не может по определению проекции. Полученное противоречие обосновывает $\mathcal{F}$-совместность трасс tr, $\mathsf{pr}_2(\Omega)$. □

Маршрут будем называть *опровергающим (эквивалентность программ)* в следующих случаях:

- он оканчивается в опровергающей вершине;
- он бесконечен, для всех его меток ℓ, кроме, быть может, конечного числа, верно $\ell\,[1] = \varepsilon$, и составляющие $v.l_1$ всех его вершин v, кроме, быть может, конечного числа, завершаемы (опровергающий маршрут типа 1);
- он бесконечен, для всех его меток ℓ, кроме, быть может, конечного числа, верно $\ell\,[2] = \varepsilon$, и составляющие $v.l_2$ всех его вершин v, кроме, быть может, конечного числа, завершаемы (опровергающий маршрут типа 2).

Теорема 5.4.1. *Программы π_1, π_2 $\mathcal{F}$-эквивалентны тогда и только тогда, когда граф $\Gamma(\pi_1, \pi_2, \mathcal{F})$ не содержит опровергающих маршрутов.*

Доказательство. ($\Leftarrow$): Пусть программы π_1, π_2 не являются $\mathcal{F}$-эквивалентными. Это означает, что существуют $\mathcal{F}$-совместные вычисления cp_1, cp_2 этих программ, имеющие различные результаты. Рассмотрим маршрут Ω, определяемый леммой 5.4.2.

Случай 1: маршрут Ω конечен. Тогда он оканчивается в терминальной вершине v. Если эта вершина не является опровергающей, то по лемме 5.4.1 получим, что результаты вычислений cp_1, cp_2 совпадают.

Случай 2: маршрут Ω бесконечен. Тогда одно из вычислений (скажем, cp_i) конечно, тогда как другое (cp_{3-i}) бесконечно. Рассмотрев все конечные префиксы маршрута Ω и привлекая лемму 5.4.1, можно легко убедиться, что Ω — опровергающий маршрут типа i.

($\Rightarrow$): Пусть Ω — опровергающий маршрут.

Случай 1: маршрут Ω оканчивается в опровергающей вершине типа 1. По лемме 5.4.1 проекцией маршрута Ω будут $\mathcal{F}$-совместные конечное вычисление $cp = \mathsf{pr}_1(\Omega)$ программы π_1 и конечная трасса $tr = \mathsf{pr}_2(\Omega)$ программы π_2, такие что либо $[\![cp]\!] \prec [\![tr]\!]$, либо $[\![cp]\!] \parallel [\![tr]\!]$. Продолжим трассу tr произвольным образом до вычисления cp'. По лемме 5.4.3 вычисление cp' $\mathcal{F}$-совместно с вычислением cp. В любом из случаев: $[\![cp]\!] \prec [\![tr]\!]$, $[\![cp]\!] \parallel [\![tr]\!]$ — результаты вычислений cp, cp' не могут совпадать.

Случай 2: маршрут Ω оканчивается в опровергающей вершине типа 2. Рассуждения аналогичны случаю (**1**).

Случай 3: маршрут Ω оканчивается в опровергающей вершине типа 3. Тогда по лемме 5.4.1 проекцией маршрута Ω будут конечные вычисления, имеющие различные результаты.

Случай 4: Ω — бесконечный опровергающий маршрут типа 1. По лемме 5.4.1 проекцией маршрута Ω будут $\mathcal{F}$-совместные бесконечное вычисление $cp = \mathsf{pr}_2(\Omega)$ программы π_2 и конечная трасса $tr = \mathsf{pr}_1(\Omega)$ программы π_1, оканчивающаяся завершаемым состоянием управления. Продолжим трассу tr произвольным образом до конечного вычисления cp' — это можно сделать по завер-

шаемости последнего состояния управления трассы. По лемме 5.4.1 конечное вычисление cp' $\mathcal{F}$-совместно с бесконечным вычислением cp.

Случай 5: Ω — бесконечный опровергающий маршрут типа 2. Рассуждения аналогичны случаю (**4**). □

5.5. Эффекты подавления

В данном разделе вводятся понятия, с помощью которых в утверждениях раздела 5.6 будет ограничено полиномом число вершин графа совместных вычислений, которые достаточно исследовать для ответа на вопрос о $\mathcal{F}$-эквивалентности программ. Основным понятием, вводимым в данном разделе, является понятие эффекта подавления.

Эффект подавления æ_s, индуцированный состоянием данных s шкалы $\mathcal{F}$, представляет собой отображение $\text{æ}_s : S \to S$, определяемое так: $\text{æ}_s(s') = s''$, где:

- $s' \circ s = s'' \circ s$;
- если $s''' \circ s = s'' \circ s$, то $\|s'''\| \geq \|s''\|$.

Эффект подавления æ_s будем также называть s-подавлением. Корректность определения s-подавления обеспечивается следующей леммой.

Лемма 5.5.1. *Пусть s, s' — состояния данных и S — множество состояний данных, являющихся решениями уравнения $X \circ s = s'$ относительно X. Тогда ровно один элемент S имеет наименьший (среди всех элементов S) вес.*

Доказательство. Предположим, что существует два различных элемента s_1, s_2 множества S, имеющие наименьший вес. Пусть $h_1 \in \lfloor s_1 \rfloor$ и $h_2 \in \lfloor s_2 \rfloor$. Покажем, что цепочка h_2 выводима из h_1 — из такой выводимости немедленно будет следовать равенство $s_1 = s_2$, противоречащее исходному предположению и обосновывающее лемму.

Пусть $h \in \lfloor s \rfloor$. Так как $s_1 \circ s = s_2 \circ s$, получаем выводимость цепочки $h_2 h$ из $h_1 h$. По лемме 5.2.6 и минимальности веса элементов s_1, s_2 цепочки $h_1 h$ и $h_2 h$ являются минимальными представителями элемента $s_1 \circ s$. По лемме 5.2.9 существует вывод цепочки $h_1 h \overset{|}{\rightleftharpoons} \ldots \overset{|}{\rightleftharpoons} h_2 h$. Тогда выводимость цепочки h_2 из h_1 следует из леммы 5.2.8. □

Множество всех эффектов подавления обозначим записью Æ. Введем порядок $\lhd$ на множесте Æ: $\text{æ}_1 \lhd \text{æ}_2$ тогда и только тогда, когда существуют состояния данных s_1, s_2, такие что $\text{æ}_1 = \text{æ}_{s_1}$, $\text{æ}_2 = \text{æ}_{s_2}$ и $s_1 \preceq s_2$.

Лемма 5.5.2. *Отношение $\lhd$ есть нестрогий частичный порядок на множестве Æ.*

Доказательство. Предположим, что отношение $\lhd$ не является нестрогим частичным порядком. Тогда существуют различные эффекты подавления æ_1, æ_2, такие что $\text{æ}_1 \lhd \text{æ}_2$ и $\text{æ}_2 \lhd \text{æ}_1$.

Из леммы 5.2.6 следует, что если $s \in \mathcal{F}.S$ и æ — эффект подавления, то: если $h \in \lfloor s \rfloor$, то существует цепочка $g \in \lfloor \text{æ}(s) \rfloor$, получаемая из h удалением вхождений операторов. Из импликации $[\![a]\!] \circ s = s \Rightarrow [\![a]\!] \circ s \circ s' = s \circ s'$ следует: если $\text{æ}_i \lhd \text{æ}_{3-i}$ и $h_i \in \lfloor \text{æ}_i(s) \rfloor$, то существует цепочка $h_{3-i} \in \lfloor \text{æ}_{3-i}(s) \rfloor$, получающаяся из h_i удалением вхождений операторов.

Пусть $s \in \mathcal{F}.S$ и $h_1 \in \lfloor \text{æ}_1(s) \rfloor$. По неравенству $\text{æ}_1 \lhd \text{æ}_2$ существует цепочка $h_2 \in \lfloor \text{æ}_2(s) \rfloor$, получаемая из h_1 удалением вхождений операторов. По неравенству $\text{æ}_2 \lhd \text{æ}_1$ существует цепочка $h_1' \in \lfloor \text{æ}_1(s) \rfloor$, получаемая из h_2 удалением вхождений операторов. При определению минимальных представителей цепочки h_1 и h_1' имеют одинаковую длину. Следовательно, $h_1 = h_2 = h_1'$, а значит, $\text{æ}_1(s) = \text{æ}_2(s)$. □

Используя лемму 5.5.2, будем далее говорить о *частично упорядоченном множестве эффектов подавлений*, имея в виду множество Æ, на котором введён порядок $\lhd$. Ссылка на лемму 5.5.2 будет неявно подразумеваться везде, где используется частичная упорядоченность множества Æ.

Лемма 5.5.3. *$\text{æ}_s(s_1) = \text{æ}_s(s_2)$ тогда и только тогда, когда $s_1 \circ s = s_2 \circ s$.*

Справедливость леммы 5.5.3 следует из леммы 5.5.1 и равенств $s_i \circ s = \text{æ}_s(s_i) \circ s$, $i \in \{1, 2\}$.

Лемма 5.5.4. *Если $\text{æ}_1(s_1) = \text{æ}_1(s_2)$ и $\text{æ}_1 \lhd \text{æ}_2$, то $\text{æ}_2(s_1) = \text{æ}_2(s_2)$.*

Доказательство. Для определённости положим $\text{æ}_1 = \text{æ}_s$ и $\text{æ}_2 = \text{æ}_{s \circ s'}$. По лемме 5.5.3 верно равенство $s_1 \circ s = s_2 \circ s$. Тогда верно и равенство $s_1 \circ s \circ s' = s_2 \circ s \circ s'$. Используя еще раз лемму 5.5.3, получим равенство $\text{æ}_2(s_1) = \text{æ}_2(s_2)$. □

Существует бесконечное число функций, преобразующих состояния данных, при этом каждый эффект подавления является такой функцией. Однако оказывается, что множество всевозможных эффектов подавления конечно (лемма 5.5.6), и более того, распознаваемо автоматом (лемма 5.5.5). Автомат $\mathfrak{A}$, *распознающий эффекты подавления*, обладает следующими свойствами:

- $\mathfrak{A}.A = \mathfrak{O}$, $\mathfrak{A}.L = \text{Æ}$;
- согласованность с моноидом: если $\langle h \rangle = \langle g \rangle$, то $\mathfrak{A}(h) = \mathfrak{A}(g)$;
- согласованность с эффектами подавления: $\mathfrak{A}.M(h) = \text{æ}_{[\![h]\!]}$.

Лемма 5.5.5. *Существует автомат, распознающий эффекты подавления.*

Доказательство. Построим явным образом автомат $\mathfrak{A}$, распознающий эффекты подавления. Начнём построение с автомата $\mathfrak{A}_0$, распознающего подавление операторов (теорема 5.3.1). Состояния q_1, q_2 этого автомата объявим эквивалентными, если для любой цепочки h верно равенство $\mathfrak{A}_0.M(q_1, h) = \mathfrak{A}_0.M(q_2, h)$. Рассмотрим счётно-бесконечное множество меток $\{\text{æ}_1, \text{æ}_2, \ldots\}$. Переберём классы эквивалентности состояний автомата $\mathfrak{A}_0$, и всем состояниям i-го рассматриваемого класса назначим метку æ_i. В результате такого переназначения меток в автомате $\mathfrak{A}_0$ получим автомат $\mathfrak{A}_1$. Используя лемму 5.2.6, можно легко показать справедливость следующего факта: $\mathfrak{A}_1.M(h_1) = \mathfrak{A}_1.M(h_2)$ тогда

и только тогда, когда $æ_{[\![h_1^-]\!]} = æ_{[\![h_2^-]\!]}$. Значит, существует взаимно однозначное отображение множества использованных в автомате $\mathfrak{A}_1$ меток $\{æ_1, \ldots, æ_k\}$ в множество эффектов подавления Æ, отождествляющее метку $\mathfrak{A}_1.M(h)$ с $[\![h^-]\!]$-подавлением. Существование такого отображения означает, что можно считать метки $æ_1, \ldots, æ_k$ различными эффектами подавления, в совокупности образующими множество Æ.

Заметим, что для требуемого автомата $\mathfrak{A}$ должно быть выполнено равенство $\mathfrak{A}.M(h) = æ_{[\![h]\!]}$, тогда как для автомата $\mathfrak{A}_1$ выполнено равенство $\mathfrak{A}.M(h) = æ_{[\![h^-]\!]}$. Это означает, что для получения требуемого автомата $\mathfrak{A}$ достаточно "развернуть" автомат $\mathfrak{A}_1$. Если состояния автомата помечены всего двумя метками, то он может расцениваться как автомат-распознаватель, и для него известен способ требуемого разворачивания [24]. Сведём разворачивание произвольного автомата к разворачиванию автомата-распознавателя. Рассмотрим совокупность автоматов $\mathfrak{A}_2^1, \ldots, \mathfrak{A}_2^k$: автомат $\mathfrak{A}_2^i$ получается из автомата $\mathfrak{A}_1$ удалением всех меток и объявлением всех вершин, помеченных эффектом подавления $æ_i$, финальными. Таким образом, автомат $\mathfrak{A}_2^i$ распознаёт язык $\left\{h \mid æ_{[\![h^-]\!]}\right\} = æ_i$. Развернув автомат $\mathfrak{A}_2^i$, получим автомат $\mathfrak{A}_3^i$, распознающий язык $\left\{h \mid æ_{[\![h]\!]} = æ_i\right\}$. Построим декартово произведение автоматов $\mathfrak{A}_3^1, \ldots, \mathfrak{A}_3^k$, на обращая внимание на финальность вершин результата. Рассмотрим состояние $(q_1, \ldots, q_k)$ декартова произведения. По определению эффектов подавлений ровно одно из состояний q_i является финальным в автомате $\mathfrak{A}_3^i$ — иначе существовала бы цепочка h, такая что состояние данных $[\![s]\!]$ имеет два различных эффекта подавления. Зная этот индекс i, пометим состояние $(q_1, \ldots, q_k)$ декартова произведения эффектом подавления $æ_i$. Размеченное таким образом декартово произведение и является требуемым автоматом $\mathfrak{A}$. Согласованность построенного автомата $\mathfrak{A}$ с эффектами подавления следует из рассуждений о метках и распознаваемых языках, приведённых в доказательстве. Согласованность автомата $\mathfrak{A}$ с моноидом следует из согласованности с моноидом автомата $\mathfrak{A}_0$ (с точностью до зеркального образа входных цепочек) и того, что декар-

тово произведение автоматов, согласованных с моноидом, также согласовано с моноидом. □

Лемма 5.5.6. *Множество Æ конечно.*

Справедливость леммы 5.5.6 следует из существования (конечного) автомата, распознающего эффекты подавления (лемма 5.5.5). Множество меток такого автомата конечно и совпадает с множеством Æ.

До конца главы считаем заданным автомат $\mathfrak{A}$, распознающий эффекты подавления. Основываясь на согласованности автомата $\mathfrak{A}$ с моноидом, будем использовать запись $\mathfrak{A}(\llbracket h \rrbracket)$ наряду с $\mathfrak{A}(h)$ и запись $\mathfrak{A}.M(\llbracket h \rrbracket)$ наряду с $\mathfrak{A}.M(h)$. Вершину v, для которой верно $\mathfrak{A}(v.s_1) = q$, будем также называть *q-вершиной*.

Обобщим понятие эффекта подавления с состояний данных на конечные последовательности состояний данных, получаемые в результате выполнения трасс программ. Такими обобщениями являются приведённые далее развитие эффекта подавления и обобщённое развитие эффекта подавления.

Содержательно, *развитие s-подавления вдоль цепочки h*, или просто *(s,h)-развитие*, отслеживает изменение эффекта подавления æ_s при изменении состояния данных последовательным применением к нему операторов цепочки h. Формально, (s,h)-развитие $\bar{\text{æ}}^h_s$ определяется так: $\bar{\text{æ}}^h_s = \left\lceil \text{æ}_s, \text{æ}_{s\circ\llbracket \mathsf{pref}(h,1) \rrbracket}, \text{æ}_{s\circ\llbracket \mathsf{pref}(h,2) \rrbracket}, \ldots, \text{æ}_{s\circ\llbracket h \rrbracket} \right\rceil$. Запись $\bar{\text{Æ}}$ будем использовать для обозначения множества всевозможных (s,h)-подавлений.

Лемма 5.5.7. *Любое (s,h)-развитие является цепью в частично упорядоченном множестве эффектов подавлений.*

Доказательство. По определению частичного порядка на множестве эффектов подавлений выполнены неравенства $\mathsf{hd}\left(\bar{\text{æ}}^h_s\right) \lhd \ldots \lhd \overline{\mathsf{hd}}\left(\bar{\text{æ}}^h_s\right)$. По определению (s,h)-развития выполнены неравенства $\mathsf{hd}\left(\bar{\text{æ}}^h_s\right) \neq \cdots \neq \overline{\mathsf{hd}}\left(\bar{\text{æ}}^h_s\right)$. □

Лемма 5.5.8. *Для любого состояния данных s и любых цепочек h, g верно: (s,h)-развитие является префиксом (s,hg)-развития.*

Справедливость леммы 5.5.8 очевидным образом следует из определения развития эффекта подавления.

Лемма 5.5.9. *Множество* $\bar{\text{Æ}}$ *конечно.*

Доказательство. Любое развитие эффекта подавления является цепью в частично упорядоченном множестве Æ (лемма 5.5.7). Множество Æ конечно (лемма 5.5.6). При этом число цепей в конечном частично упорядоченном множестве конечно. □

Лемма 5.5.10. *Пусть* $s_1, s_2 \in \mathcal{F}.S$, $\mathfrak{A}(s_1) = \mathfrak{A}(s_2)$ *и* $h \in \mathfrak{O}^*$. *Тогда* $\bar{æ}^h_{s_1} = \bar{æ}^h_{s_2}$.

Справедливость леммы 5.5.10 следует из очевидных равенств: $\bar{æ}^h_{s_1} = \lceil \mathfrak{A}.M(s_1), \mathfrak{A}.M(s_1 \circ \langle \mathsf{pref}(h,1) \rangle), \ldots, \mathfrak{A}.M(s_1 \circ \langle h \rangle) \rceil = \lceil \mathfrak{A}.M(s_2), \mathfrak{A}.M(s_2 \circ \langle \mathsf{pref}(h,1) \rangle), \ldots, \mathfrak{A}.M(s_2 \circ \langle h \rangle) \rceil = \bar{æ}^h_{s_2}$.

Перейдём от развития вдоль цепочек к развитию вдоль путей. Развитием эффекта подавления вдоль конечного пути pt программы π назовем развитие этого эффекта вдоль цепочки $\pi.B(pt)$. Развитие эффекта подавления вдоль бесконечного пути pt — это предел развитий вдоль путей $\mathsf{pref}(pt,1), \mathsf{pref}(pt,2), \ldots$. При ограничении числа вершин графа совместных вычислений будут рассматриваться пути программ, "кратчайшие" с точки зрения развития эффекта подавления. Такие пути будем называть *s-приведёнными*, где $s \in \mathcal{F}.S$. Конечный путь программы π назовём s-приведённым, где $s \in \mathcal{F}.S$, если для любых его различных префиксов pt_1, pt_2 верно хотя бы одно из неравенств $\overline{\mathsf{hd}}(pt_1) \neq \overline{\mathsf{hd}}(pt_2)$, $\mathfrak{A}(s \circ [\![pt_1]\!]) \neq \mathfrak{A}(s \circ [\![pt_2]\!])$. Бесконечный путь программы π будем называть s-приведённым, если он представим в виде $pt_1 \to_\pi pt_2 \to_\pi pt_2 \to_\pi \ldots \to_\pi pt_2 \to_\pi \ldots$, где $pt_1 \to_\pi pt_2$ — конечный s-приведённый путь. Такое представление для бесконечного s-приведённого пути будем называть *s-приведённым разложением* и коротко записывать как $pt_1 (\to_\pi pt_2)^\omega$.

Лемма 5.5.11. *Пусть* pt *— l-путь программы* π *и* $s \in \mathcal{F}.S$. *Тогда существует s-приведённый l-путь* pt' *программы* π, *такой что:*

- $\bar{\text{æ}}_s^{pt} = \bar{\text{æ}}_s^{pt'}$;
- *путь* pt' *конечен тогда и только тогда, когда путь* pt *конечен;*
- *если путь* pt *конечен, то* $\overline{\mathsf{hd}}(pt) = \overline{\mathsf{hd}}(pt')$.

Доказательство. **Случай 1**: путь pt конечен. Рассмотрим кратчайший l-путь pt', такой что $\overline{\mathsf{hd}}(pt) = \overline{\mathsf{hd}}(pt')$ и $\bar{\text{æ}}_s^{pt} = \bar{\text{æ}}_s^{pt'}$. Покажем, что он является s-приведённым. Предположим, что это не так. Тогда существуют два его различных префикса $pt_1 = \mathsf{pref}(pt', k_1)$, $pt_2 = \mathsf{pref}(pt', k_2)$, такие что $\overline{\mathsf{hd}}(pt_1) = \overline{\mathsf{hd}}(pt_2)$ и $\mathfrak{A}(s \circ [\![pt_1]\!]) = \mathfrak{A}(s \circ [\![pt_2]\!])$. Привлекая лемму 5.5.8, можно легко убедиться, что $pt'' = pt_1 \to_\pi \overline{\mathsf{pref}}(pt, k_2)$ — l-путь, для которого верно $\overline{\mathsf{hd}}(pt) = \overline{\mathsf{hd}}(pt')$ и $\bar{\text{æ}}_s^{pt} = \bar{\text{æ}}_s^{pt''}$. При этом $|pt''| < |pt'|$, что противоречит выбору пути pt'.

Случай 2: путь pt бесконечен. Рассмотрим длиннейший префикс pt_1 пути pt, для которого верно $\bar{\text{æ}}_s^{pt_1} = \overline{\mathsf{tl}}(\bar{\text{æ}}_s^{pt})$. По определению развития эффекта подавления путь pt_1 конечен. Применив к нему рассуждения случая (**1**), получим s-приведённый l-путь pt_1', такой что $\overline{\mathsf{hd}}(pt_1') = \overline{\mathsf{hd}}(pt_1)$ и $\bar{\text{æ}}_s^{pt_1'} = \bar{\text{æ}}_s^{pt_1}$. Рассмотрим бесконечный путь $pt_2 = \overline{\mathsf{pref}}(pt, |pt_1|)$. Существуют два его конечных префикса $pt_2^1 = \mathsf{pref}(pt_2, k_1)$, $pt_2^2 = \mathsf{pref}(pt_2, k_2)$, для которых верно $\overline{\mathsf{hd}}\big(pt_2^1\big) = \overline{\mathsf{hd}}\big(pt_2^2\big)$ и $\mathfrak{A}\big([\![pt_1 \to_\pi pt_2^1]\!]\big) = \mathfrak{A}\big([\![pt_1 \to_\pi pt_2^2]\!]\big)$. Рассмотрим такие префиксы pt_2^1, pt_2^2, для которых $k_2 > k_1$ и k_2 имеет наименьшее возможное значение. Пусть $pt_3^1 = \overline{\mathsf{tl}}\big(pt_2^1\big)$ и $pt_3^2 = \overline{\mathsf{pref}}\big(\overline{\mathsf{tl}}\big(pt_2^2\big), |pt_3^1|\big)$. Можно легко убедиться, что путь $pt_1' \to_\pi pt_3^1 \big(\to_\pi pt_3^2\big)^\omega$ — s-приведённое разложение требуемого пути pt'. □

Лемма 5.5.12. *Пусть* pt *—* s*-приведённый путь программы* π*, где* $s \in \mathcal{F}.S$*. Если он конечен, то* $|pt| \le |\pi| \cdot |\mathfrak{A}.Q|$*. Если он бесконечен и* $pt_1 \big(\to_\pi pt_2\big)^\omega$ *— его* s*-приведённое разложение, то* $|pt_1| + |pt_2| \le |\pi| \cdot |\mathfrak{A}.Q|$.

Справедливость леммы 5.5.12 следует из того, что существует всего $|\pi| \cdot |\mathfrak{A}.Q|$ различных пар (l, q), где $l \in \pi.L$ и $q \in \mathfrak{A}.Q$.

Лемма 5.5.13. *Пусть pt — бесконечный s-приведённый путь программы π, где $s \in \mathcal{F}.S$, и пусть $k > |\pi| \cdot |\mathfrak{A}.Q|$. Тогда* $\overline{\mathsf{hd}}(\bar{\text{æ}}_s^{pt}) = \text{æ}_{s \circ [\![\mathsf{pref}(pt,k)]\!]}$.

Справедливость леммы 5.5.13 следует из леммы 5.5.12 и свойств автомата $\mathfrak{A}$.

Введём второе обобщение эффектов подавления — обобщённое развитие эффекта подавления. Содержательно, *обобщённое развитие s-подавления вдоль цепочки h*, или просто *обобщённое (s,h)-развитие*, отслеживает изменение s-подавления и $[\![\lambda]\!]$-подавления при последовательном применении к этим состояниям данных операторов цепочки h. Формально, обобщённое (s,h)-развитие $\bar{\bar{\text{æ}}}_s^h$ определяется так: $\bar{\bar{\text{æ}}}_s^h = \left\lceil \left(\text{æ}_{[\![\lambda]\!]}, \text{æ}_s\right), \left(\text{æ}_{[\![\mathsf{hd}(h)]\!]}, \text{æ}_{s \circ [\![\mathsf{hd}(h)]\!]}\right), \ldots, \left(\text{æ}_{[\![h]\!]}, \text{æ}_{s \circ [\![h]\!]}\right) \right\rceil$. Если $\bar{\bar{\text{æ}}}$ — обобщённое развитие, то записью $\mathsf{pr}_i(\bar{\bar{\text{æ}}})$, где $i \in \{1,2\}$, обозначим последовательность $\left\lceil \mathsf{hd}(\bar{\bar{\text{æ}}})\,[i], \ldots, \overline{\mathsf{hd}}(\bar{\bar{\text{æ}}})\,[i] \right\rceil$. Свойства обобщённого развития аналогичны свойствам (необобщённого) развития, и доказательство этих свойств тоже аналогично, поэтому обоснование свойств обобщённого различия опущено. Исключением является свойство, сформулированное в лемме 5.5.14, показывающее взаимосвязь развитий и обобщённых развитий, однако для обоснования этой леммы достаточно выписать определения развития и обобщённого развития и тут же убедиться в справедливости леммы.

Лемма 5.5.14. $\mathsf{pr}_1\left(\bar{\bar{\text{æ}}}_s^h\right) = \bar{\text{æ}}_{[\![\lambda]\!]}^h$, $\mathsf{pr}_2\left(\bar{\bar{\text{æ}}}_s^h\right) = \bar{\text{æ}}_s^h$.

Записью $\bar{\bar{\text{Æ}}}$ обозначим множество всех обощённых развитий.

Лемма 5.5.15. *Множество $\bar{\bar{\text{Æ}}}$ конечно.*

Лемма 5.5.16. *Пусть $s_1, s_2 \in \mathcal{F}.S$, $\mathfrak{A}(s_1) = \mathfrak{A}(s_2)$ и $h \in \mathfrak{O}^*$. Тогда* $\bar{\bar{\text{æ}}}_{s_1}^h = \bar{\bar{\text{æ}}}_{s_2}^h$.

Основываясь на лемме 5.5.16, будем наряду с обобщёнными (s,h)-развитиями говорить про обобщённые (q,h)-развития, где $q = \mathfrak{A}(s)$. Обобщённым развитием эффекта подавления вдоль конечного пути pt программы π назовём обобщённое развитие эффекта подавления вдоль цепочки

$\pi.B(pt)$. Обобщённым развитием эффекта подавления вдоль бесконечного пути pt назовём предел последовательности обобщённых развитий эффекта подавления вдоль путей $\mathsf{pref}(pt,1), \mathsf{pref}(pt,2), \ldots$. Конечный путь pt программы π будем называть *q-приведённым*, где $q \in \mathfrak{A}.Q$, если для любых его различных префиксов pt_1, pt_2 верно хотя бы одно из неравенств $\overline{\mathsf{hd}}(pt_1) \neq \overline{\mathsf{hd}}(pt_2)$, $\mathfrak{A}([\![pt_1]\!]) \neq \mathfrak{A}([\![pt_2]\!])$, $\mathfrak{A}(q, \pi.B(pt_1)) \neq \mathfrak{A}(q, \pi.B(pt_2))$. Бесконечный путь pt будем называть q-приведённым, если он представим в виде $pt = pt_1\,(\to_\pi pt_2)^\omega$, где $pt_1 \to_\pi pt_2$ — конечный q-приведённый путь. Последнее представление будем называть *q-приведённым разложением* пути pt.

Лемма 5.5.17. *Пусть pt — l-путь программы π и $q \in \mathfrak{A}.Q$. Тогда существует q-приведённый l-путь pt' программы π, такой что:*

- $\bar{\bar{\mathfrak{a}}}_q^{pt} = \bar{\bar{\mathfrak{a}}}_q^{pt'}$;
- *путь pt' конечен тогда и только тогда, когда путь pt конечен;*
- *если путь pt конечен, то $\overline{\mathsf{hd}}(pt) = \overline{\mathsf{hd}}(pt')$.*

Лемма 5.5.18. *Пусть pt — q-приведённый путь программы π, где $q \in \mathfrak{A}.Q$. Если он конечен, то $|pt| \leq |\pi| \cdot |\mathfrak{A}.Q|^2$. Если он бесконечен и $pt_1\,(\to_\pi pt_2)^\omega$ — его q-приведённое разложение, то $|pt_1| + |pt_2| \leq |\pi| \cdot |\mathfrak{A}.Q|^2$.*

Лемма 5.5.19. *Пусть pt — бесконечный q-приведённый путь программы π, где $q \in \mathfrak{A}.Q$, и пусть $k > |\pi| \cdot |\mathfrak{A}.Q|^2$. Тогда $\overline{\mathsf{hd}}\big(\bar{\bar{\mathfrak{a}}}_q^{pt}\big) = \big(\mathfrak{a}_{[\![\mathsf{pref}(pt,k)]\!]}, \mathfrak{a}_{s\circ[\![\mathsf{pref}(pt,k)]\!]}\big)$.*

5.6. Ограничение числа вершин графа совместных вычислений

Леммы 5.6.1, 5.6.2, приведённые в данном разделе, позволяют ограничить полиномом (относительно размеров программ) число вершин графа совместных вычислений, которые достаточно исследовать для ответа на вопрос об

$\mathcal{F}$-эквивалентности программ. Перед формулировкой лемм приведено содержательное описание принципов, используемых в обосновании этих лемм, то есть, в конечном итоге, принципы ограничения числа вершин.

Рассмотрим вершину v графа Г. По лемме 5.4.1 компоненты $v.l_1$, $v.l_2$, $v.s_2$ могут принимать лишь конечное число значений (для заданных шкалы $\mathcal{F}$ и программ π_1, π_2). Таким образом, причина, по которой граф Г содержит бесконечное число вершин, — это бесконечное разнообразие значений $v.s_1$. Таким образом, основное назначение лемм 5.6.1, 5.6.2 — ограничить это разнообразие значений. При накоплении достаточно большого числа значений $v.s_1$ для заданных значений $v.l_1$, $v.l_2$, $v.s_2$ лемма 5.6.1 позволяет констатировать неэквивалентность программ, а лемма 5.6.2 — игнорировать в дальнейшем обходе дуги, исходящие из исследуемой вершины v.

Для констатации неэквивалентности программ в лемме 5.6.1 проекция одной из накопленных вершин графа Г явным образом продолжается до вычислений программ π_1, π_2, имеющих различные результаты. В большинстве случаев достаточно рассмотреть кратчайшие вычисления с заданными свойствами — конечность, s-приведённость, q-приведённость — до которых продолжаются проекции маршрутов, оканчивающихся в накопленных вершинах, и с использованием утверждения 3.1.2 показать, что вычисления хотя бы одной из полученных пар $\mathcal{F}$-совместны. В более сложных случаях проекции накопленных маршрутов продолжаются до $\mathcal{F}$-совместных вычислений в несколько этапов — на каждом этапе решение о том, по каким логическим условиям продолжить трассы с сохранением их $\mathcal{F}$-совместности, зависит от свойств полученных трасс.

Для возможности сократить обход графа Г, игнорируя дуги, исходящие из вершины v, в лемме 5.6.2 показывается, что если через вершину v проходит опровергающий маршрут, то опровергающий маршрут проходит и через одну из уже накопленных вершин v', где $v.l_1 = v'.l_1$, $v.l_2 = v'.l_2$ и $v.s_2 = v'.s_2$. В ряде случаев для этого показывается справедливость леммы 5.6.1, а значит, безусловное наличие такого опровергающего маршрута. Для остальных случаев

показывается, что маршрут до хотя бы одной вершины v' можно продолжить с использованием тех же логических условий, которые позволяли продолжить маршрут до вершины v до опровергающего.

Лемма 5.6.1. *Пусть* $q \in \mathfrak{A}.Q$, $l_1 \in \pi_1.L$, $l_2 \in \pi_2.L$, $s_2 \in \mathcal{F}.S$, $m = |\mathfrak{A}.Q|^3 \cdot |\pi_1| \cdot |\pi_2|^2 + 2$ *и:*

- *из корня графа* Γ *достижимы* q*-вершины* v_i, $i \in \{1, \ldots, m\}$, *такие что* $v_i.l_1 = l_1$, $v_i.l_2 = l_2$ *и* $v_i.s_2 = s_2$*;*
- pt_1, pt_2 *— полные* l_1*-путь и* l_2*-путь программ* π_1, π_2 *соответственно;*
- *хотя бы один из путей* pt_1, pt_2 *конечен;*
- $\bar{\bar{æ}} = \bar{\bar{æ}}_q^{\mathsf{tl}(pt_1)}$, $\bar{æ} = \bar{æ}_{s_2}^{\mathsf{tl}(pt_2)}$*;*
- $1 \leq k \leq |\bar{\bar{æ}}|$*;*
- *состояния данных* $\bar{\bar{æ}}[k][1](v_i.Fs_1)$, $i \in \{1, \ldots, m\}$, *попарно различны;*
- $\mathsf{set}_{\bar{æ}} \cap \{\bar{\bar{æ}}[i][2] \mid k < i \leq |\bar{\bar{æ}}|\} = \emptyset$.

Тогда программы π_1, π_2 *не являются* $\mathcal{F}$*-эквивалентными.*

Доказательство. Пусть $\dot{pt}_1 = \mathsf{tl}(pt_1)$ и $\dot{pt}_2 = \mathsf{tl}(pt_2)$. Без ограничения общности считаем, что путь $\dot{pt}_1$ q-приведён (лемма 5.5.17), а путь $\dot{pt}_2$ s_2-приведён (лемма 5.5.11). Пусть Ω_i — маршрут, оканчивающийся в вершине v_i, и $tr_j^i = \mathsf{pr}_j(\Omega_i)$, $i \in \{1, \ldots, m\}$, $j \in \{1, 2\}$. По лемме 5.4.1 имеем $[\![tr_1^i]\!] = s^i \circ v_i.s_1$, $[\![tr_2^i]\!] = s^i \circ s_2$, где $s^i \in \mathcal{F}.S$, $i \in \{1, \ldots, m\}$.

Случай 1: $k = |\bar{\bar{æ}}|$, состояние управления l_2 завершаемо в программе π_2. Рассмотрим кратчайший полный l_2-путь pt_2' программы π_2. Тогда $|pt_2'| \leq |\pi_2|$. Пусть $\dot{pt}_2' = \mathsf{tl}(pt_2')$, $cp_1^i = tr_1^i \to_{\pi_1} \dot{pt}_1$ и $cp_2^i = tr_2^i \to_{\pi_2} \dot{pt}_2'$. Для заверешения обоснования достаточно показать, что существует индекс $j \in \{1, \ldots, m\}$, такой что вычисления cp_1^j, cp_2^j $\mathcal{F}$-совместны и имеют различные результаты.

Подслучай 1.а: путь pt_1 бесконечен. Если вычисления cp_1^i, cp_2^i не являются $\mathcal{F}$-совместными, то по утверждению 3.1.2 существуют их собственные префиксы $\dot{tr}_1^i$, $\dot{tr}_2^i$, такие что: $\left[\!\left[\dot{tr}_1^i\right]\!\right] = \left[\!\left[\dot{tr}_2^i\right]\!\right]$; ни для какого логического условия σ не выполняются оба равенства $\overline{\mathsf{hd}}\left(\dot{tr}_1^i\right) \xrightarrow{\sigma}_{\pi_1} cp_1^i\left[|\dot{tr}_1^i|+1\right]$, $\overline{\mathsf{hd}}\left(\dot{tr}_2^i\right) \xrightarrow{\sigma}_{\pi_2} cp_2^i\left[|\dot{tr}_2^i|+1\right]$. По утверждению 3.1.2 и лемме 5.4.1 справедливо хотя бы одно из неравенств $|\dot{tr}_1^i| \geq |tr_1^i|$, $|\dot{tr}_2^i| \geq |tr_2^i|$. По лемме 5.4.3 справедливы оба последних неравенства. По лемме 5.2.7 существуют собственные префиксы $\ddot{pt}_1^i$, $\ddot{pt}_2^i$ путей $\dot{pt}_1$, $\dot{pt}_2'$, для которых верно $v_i.s_1 \circ \left[\!\left[\ddot{pt}_1^i\right]\!\right] = s_2 \circ \left[\!\left[\ddot{pt}_2^i\right]\!\right]$. По лемме 5.2.10 и выбору пути pt_2' справедливы соотношения $\|v_i.s_1 \circ \left[\!\left[\ddot{pt}_1^i\right]\!\right]\| = \|s_2 \circ \left[\!\left[\ddot{pt}_2^i\right]\!\right]\| < |\pi_2|$. По леммам 5.2.10, 5.5.12 верно неравенство $|\ddot{pt}_1^i| < |\mathfrak{A}.Q|^2 \cdot |\pi_1| \cdot |\pi_2|$. Запишем все полученные неравенства на длины префиксов:

$$0 \leq |\ddot{pt}_1^i| < |\mathfrak{A}.Q|^2 \cdot |\pi_1| \cdot |\pi_2|, \quad 0 \leq |\ddot{pt}_2^i| < |\pi_2|.$$

Отсюда следует, что пара $\left(|\ddot{pt}_1^i|, |\ddot{pt}_2^i|\right)$ может принимать не более $|\mathfrak{A}.Q|^2 \cdot |\pi_1| \cdot |\pi_2|^2$ значений.

Предположим, что ни для какого i, $1 \leq i \leq m$, вычисления cp_1^i, cp_2^i не являются $\mathcal{F}$-эквивалентными. Тогда найдутся индексы p, q, $1 \leq p < q \leq m$, такие что $\left(|\ddot{pt}_1^p|, |\ddot{pt}_2^p|\right) = \left(|\ddot{pt}_1^q|, |\ddot{pt}_2^q|\right)$. Значит, $v_p.s_1 \circ \left[\!\left[\ddot{pt}_1^p\right]\!\right] = s_2 \circ \left[\!\left[\ddot{pt}_2^p\right]\!\right] = s_2 a \left[\!\left[\ddot{pt}_2^q\right]\!\right] = v_q.s_1 \circ \left[\!\left[\ddot{pt}_1^q\right]\!\right]$. По лемме 5.5.3 получим равенство $\text{æ}_{\left[\!\left[\ddot{pt}_1^p\right]\!\right]}(v_p.s_1) = \text{æ}_{\left[\!\left[\ddot{pt}_1^q\right]\!\right]}(v_q.s_1)$. Тогда по лемме 5.5.4 верно равенство $\overline{\mathsf{hd}}(\bar{\bar{\text{æ}}})\,[1](v_p.s_1) = \overline{\mathsf{hd}}(\bar{\bar{\text{æ}}})\,[2](v_q.s_2)$, что противоречит условию леммы.

Подслучай 1.б: путь pt_1 конечен. Теми же рассуждениями, что и в случае (**1.а**), и применяя лемму 5.5.18, получим неравенства

$$0 \leq |\ddot{pt}_1^i| < |\mathfrak{A}.Q|^2 \cdot |\pi_1|, \quad 0 \leq |\ddot{pt}_2^i| < |\pi_2|.$$

Тогда существуют индексы p, q, $1 \leq p < q \leq m$, такие что вычисления cp_1^p, cp_2^p $\mathcal{F}$-совместны и вычисления cp_1^q, cp_2^q $\mathcal{F}$-совместны. Предположим, что в каждой паре результаты вычислений совпадают. По лемме 5.2.7 получим равен-

ства $v_p.s_1 \circ [\![\dot{pt}_1]\!] = s_2 \circ [\![\dot{pt}'_2]\!] = v_q.s_1 \circ [\![\dot{pt}_1]\!]$. По лемме 5.5.3 верно равенство $\overline{\mathsf{hd}}(\bar{\bar{\mathfrak{ae}}})\,[1](v_p.s_1) = \overline{\mathsf{hd}}(\bar{\bar{\mathfrak{ae}}})\,[1](v_q.s_1)$, что противоречит условию леммы. Полученное противоречие обосновывает то, что либо ($\mathcal{F}$-совместные) вычисления cp_1^p, cp_2^p, либо ($\mathcal{F}$-совместные) вычисления cp_1^q, cp_2^q имеют различные результаты.

Случай 2: $k = |\bar{\bar{\mathfrak{ae}}}|$, состояние управления l_2 незавершаемо в программе π_2. По лемме 5.4.3 вычисление $cp_1 = tr_1^1 \to_{\pi_1} \dot{pt}_1$ и трасса tr_2^1 совместны. Используя утверждение 3.1.2, достроим трассу tr_2^1 до вычисления cp_2, $\mathcal{F}$-совместного с вычислением cp_1. По рассматриваемому случаю вычисление cp_1 конечно, а вычисление cp_2 бесконечно.

Случай 3: $k < |\bar{\bar{\mathfrak{ae}}}|$, путь pt_2 конечен. Теми же рассуждениями, что и в случае (**1.a**), заменив путь pt'_2 на pt_2, получим неравенства

$$0 \le |\ddot{pt}_1^i| < |\mathfrak{A}.Q|^2 \cdot |\pi_1|, \quad 0 \le |\ddot{pt}_2^i| < |\mathfrak{A}.Q| \cdot |\pi_2|.$$

Верхняя граница для $|\ddot{pt}_2^i|$ следует из конечности и приведённости пути pt_2 (лемма 5.5.12). Верхняя граница для $|\ddot{pt}_1^i|$ может быть получена так. Предположим, что $|\ddot{pt}_1^i| > |\mathfrak{A}.Q|^2 \cdot |\pi_1|$ и $v_i.s_1 \circ [\![\ddot{pt}_1^i]\!] = s_2 \circ [\![\ddot{pt}_2^i]\!]$. По леммам 5.5.18, 5.5.19 имеем равенство $\mathfrak{ae}_{v_i.s_1 \circ [\![\ddot{pt}_1^i]\!]} = \overline{\mathsf{hd}}(\bar{\bar{\mathfrak{ae}}})\,[2]$. Из равенства $v_i.s_1 \circ [\![\ddot{pt}_1^i]\!] = s_2 \circ [\![\ddot{pt}_2^i]\!]$ получим $\mathfrak{ae}_{s_2 \circ [\![\ddot{pt}_2^i]\!]} = \overline{\mathsf{hd}}(\bar{\bar{\mathfrak{ae}}})\,[2]$. Это значит, что $\overline{\mathsf{hd}}(\bar{\bar{\mathfrak{ae}}})\,[2] \in \mathsf{set}_{\bar{\mathfrak{ae}}}$, что противоречит условию леммы.

Как и в случае (**1.a**), получим $\mathcal{F}$-совместные вычисления $cp_1 = tr_1^p \to_{\pi_1} \dot{pt}_1$, $cp_2 = tr_2^p \to_{\pi_2} \dot{pt}_2$. По рассматриваемому случаю вычисление cp_2 конечно. Если вычисление cp_1 бесконечно, то, очевидно, $[\![cp_1]\!] \neq [\![cp_2]\!]$. Если вычисление cp_1 конечно, то несовпадение результатов следует из неравенства $\overline{\mathsf{hd}}(\bar{\mathfrak{ae}}) \neq \overline{\mathsf{hd}}(\bar{\bar{\mathfrak{ae}}})\,[2]$.

Случай 4: $k < |\bar{\bar{\mathfrak{ae}}}|$, путь pt_2 бесконечен, $\overline{\mathsf{hd}}(\bar{\mathfrak{ae}}) \notin \{\bar{\bar{\mathfrak{ae}}}\,[j]\,[2] \mid 1 \le j \le k\}$. Из условия леммы следует, что путь pt_1 конечен. Рассуждения в этом случае повторяют случай (**3**) с поправкой на обоснование верхних границ для r_1^i, r_2^i (сами границы имеют те же значения). Верхняя граница для $|\ddot{pt}_1^i|$ следует из леммы 5.5.18. Верхняя граница для $|\ddot{pt}_2^i|$ получается так. Предположим, что

$|\ddot{pt}_2^i| > |\mathfrak{A}.Q| \cdot |\pi_2|$ и $v_i.s_1 \circ \left[\!\left[\ddot{pt}_1^i\right]\!\right] = s_2 \circ \left[\!\left[\ddot{pt}_2^i\right]\!\right]$. По приведённости пути $\dot{pt}_2$ и лемме 5.5.19 имеем равенство $\text{æ}_{s_2 \circ \left[\!\left[\ddot{pt}_2^i\right]\!\right]} = \overline{\mathsf{hd}}(\bar{\text{æ}})$. Из равенства $\text{æ}_{s_2 \circ \left[\!\left[\ddot{pt}_2^i\right]\!\right]} = \text{æ}_{v_i.s_1 \circ \left[\!\left[\ddot{pt}_1^i\right]\!\right]}$ следует включение $\overline{\mathsf{hd}}(\bar{\text{æ}}) \in \mathsf{set}_{\mathsf{pr}_2(\bar{\bar{\text{æ}}})}$, что противоречит либо условию леммы, либо рассматриваемому случаю.

Случай 5: $k < |\bar{\bar{\text{æ}}}|$, путь pt_2 бесконечен и k' — наибольший индекс, такой что $k' \leq k$ и $\overline{\mathsf{hd}}(\bar{\text{æ}}) = \bar{\bar{\text{æ}}}\,[k']$ [2]. Из условия леммы следует, что путь pt_1 конечен.

Подслучай 5.а: не существует конечного полного l_2-пути pt_2' программы π_2, такого что $\bar{\text{æ}}_{s_2}^{\dot{pt}_2}$ есть префикс развития $\bar{\text{æ}}_{s_2}^{\mathsf{tl}(pt_2')}$. Рассмотрим конечный l_2-путь pt_2', такой что $\bar{\text{æ}}_{s_2}^{\overline{\mathsf{tl}}(\mathsf{tl}(pt_2'))} \neq \bar{\text{æ}}_{s_2}^{\mathsf{tl}(pt_2')} = \bar{\text{æ}}$. Пусть $\dot{pt}_2' = \mathsf{tl}(pt_2')$. Без ограничения общности считаем путь $\dot{pt}_2'$ s_2-приведённым (лемма 5.5.11). Теми же рассуждениями, что и в случае (**1.б**), получим индекс p, для которого вычисление $cp_1 = tr_1^p \to_{\pi_1} \dot{pt}_1$ и трасса $tr = tr_2^p \to_{\pi_2} \dot{pt}_2'$ $\mathcal{F}$-совместны. Используя утверждение 3.1.2, продолжим трассу tr до вычисления cp_2, $\mathcal{F}$-совместного с вычислением cp_1. По лемме 5.5.8 и рассматриваемому случаю вычисление cp_2 бесконечно. При этом вычисление cp_1 конечно.

Подслучай 5.б:

- существует конечный полный l_2-путь pt_2' программы π_2, такой что $\bar{\text{æ}}_{s_2}^{\mathsf{tl}(pt_2)}$ есть префикс развития $\bar{\text{æ}}_{s_2}^{\mathsf{tl}(pt_2')}$;
- существует бесконечный l_1-путь pt_1' программы π_1, такой что $\bar{\bar{\text{æ}}}_q^{\mathsf{tl}(pt_1')} = \mathsf{pref}(\bar{\bar{\text{æ}}}, k')$.

Можно без ограничения общности считать, что путь $\mathsf{tl}(pt_1')$ $\mathfrak{A}$-приведён (лемма 5.5.17) и путь $\mathsf{tl}(pt_2')$ s_2-приведён (лемма 5.5.11). Далее повторяются рассуждения случая (**1.а**) с заменой путей pt_1, pt_2 на пути pt_1', pt_2'.

Подслучай 5.в:

- не существует бесконечного l_1-пути pt_1' программы π_1, такого что $\bar{\bar{\text{æ}}}_q^{pt_1'} = \mathsf{pref}(\bar{\bar{\text{æ}}}, k')$;

- для любого l_1-пути pt_1'' программы π_1, такого что $\mathsf{pref}(\bar{\bar{\text{æ}}}, k') = \bar{\bar{\text{æ}}}_q^{\overline{\mathsf{tl}}(\mathsf{tl}(pt_1''))} \neq \bar{\bar{\text{æ}}}_q^{\mathsf{tl}(pt_1'')}$, состояние управления $\overline{\mathsf{hd}}(pt_1'')$ завершаемо.

По лемме 5.4.3 для любого $i \in \{1, \ldots, m\}$ вычисление $cp_2^i = tr_2^i \to_{\pi_2} \dot{pt}_2$ совместно с трассой tr_1^i. Рассмотрим префикс $\ddot{pt}_1$ пути $\dot{pt}_1$, такой что $\bar{\bar{\text{æ}}}_q^{\overline{\mathsf{tl}}(\ddot{pt}_1)} \neq \bar{\bar{\text{æ}}}_q^{\ddot{pt}_1} = \mathsf{pref}(\bar{\bar{\text{æ}}}, k')$. Теми же рассуждениями, что и в случае (**4**), получим индекс p, для которого вычисление cp_2^p и трасса $tr_p = tr_1^p \to_{\pi_1} \ddot{pt}_1$ $\mathcal{F}$-совместны. Используя утверждение 3.1.2, продолжим трассу tr_p до трассы tr_p', такой что она $\mathcal{F}$-совместна с вычислением cp_2^p и $\mathsf{pref}(\bar{\bar{\text{æ}}}, k') = \bar{\bar{\text{æ}}}_q^{\overline{\mathsf{tl}}\left(\overline{\mathsf{pref}}\left(tr_p', |tr_1^p|\right)\right)} \neq \bar{\bar{\text{æ}}}_q^{\overline{\mathsf{pref}}\left(tr_p', |tr_1^p|\right)}$. По рассматриваемому случаю такая трасса tr_p' существует, и более того, существует конечный полный $\overline{\mathsf{hd}}\left(tr_p'\right)$-путь pt программы π_1. Пусть $\dot{pt} = \mathsf{tl}(pt)$. Для любого префикса $\ddot{pt}$ пути $\dot{pt}$ выполнено включение $\text{æ}_{\overline{\mathsf{pref}}\left(tr_p', |tr_1^p|\right) \to_{\pi_1} \ddot{pt}} \in \{\bar{\bar{\text{æ}}}\,[j]\,[2] \mid k' < j \leq |\bar{\bar{\text{æ}}}|\}$. Теми же рассуждениями, что и в конце случаев (**3**), (**4**), получаем $\mathcal{F}$-совместность бесконечного вычисления cp_2^p и конечного вычисления $tr_p' \to_{\pi_1} \dot{pt}$.

Случай 5г:

- существует конечный полный l_2-путь pt_2' программы π_2, такой что $\bar{\text{æ}}_{s_2}^{\mathsf{tl}(pt_2)}$ есть префикс развития $\bar{\text{æ}}_{s_2}^{\mathsf{tl}(pt_2')}$;

- не существует бесконечного l_1-пути pt_1'' программы π_1, такого что $\bar{\bar{\text{æ}}}_q^{\mathsf{tl}(pt_1')} = \mathsf{pref}(\bar{\bar{\text{æ}}}, k')$;

- существует конечный l_1-путь pt' программы π_1, такой что $\mathsf{pref}(\bar{\bar{\text{æ}}}, k') = \bar{\bar{\text{æ}}}_q^{\overline{\mathsf{tl}}(\mathsf{tl}(pt'))} \neq \bar{\bar{\text{æ}}}_q^{\mathsf{tl}(pt')}$.

Теми же рассуждениями, что и в случае (**1.б**), получим индекс p, для которого вычисление $cp_2^p = tr_2^p \to_{\pi_2} \mathsf{tl}(pt_2')$ и трасса $tr_p = tr_1^p \to_{\pi_1} \mathsf{tl}(pt_1')$ $\mathcal{F}$-совместны. Используя утверждение 3.1.2, продолжим трассу tr_p до вычисления cp_1^p, $\mathcal{F}$-совместного с вычислением cp_2^p. По рассматриваемому случаю вычисление cp_1^p бесконечно, а вычисление cp_2^p конечно. □

Лемма 5.6.2. *Пусть* $q \in \mathfrak{A}.Q$, $l_1 \in \pi_1.L$, $l_2 \in \pi_2.L$, $s_2 \in \mathcal{F}.S$, $m = |\mathfrak{A}.Q|^3 \cdot |\pi_1| \cdot |\pi_2|^2 + 2$ *и:*

- *из корня графа* Γ *достижимы* q*-вершины* v_i, $i \in \{1, \ldots, m\}$, *такие что* $v_i.l_1 = l_1$, $v_i.l_2 = l_2$ *и* $v_i.s_2 = s_2$;
- pt_1, pt_2 *— полные* l_1*-путь и* l_2*-путь программ* π_1, π_2 *соответственно;*
- $\bar{\bar{æ}} = \bar{\bar{æ}}_q^{\mathsf{tl}(pt_1)}$, $\bar{æ} = \bar{æ}_{s_2}^{\mathsf{tl}(pt_2)}$;
- $1 \leq k < |\bar{\bar{æ}}|$, $1 \leq s < |\bar{æ}|$;
- *состояния данных* $\bar{\bar{æ}}[k][1](v_i.Fs_1)$, $i \in \{1, \ldots, m\}$, *попарно различны;*
- *состояния данных* $\bar{\bar{æ}}[k+1][1](v_i.Fs_1)$, $i \in \{1, \ldots, m\}$, *совпадают;*
- $\mathsf{set}_{\mathsf{pref}(\bar{æ}, s)} \cap \{\bar{\bar{æ}}[i][2] \mid k < i \leq |\bar{\bar{æ}}|\} = \emptyset$.
- $\bar{æ}[s+1] \in \{\bar{\bar{æ}}[i][2] \mid k < i \leq |\bar{\bar{æ}}|\}$;
- Ω *— маршрут, оканчивающийся в вершине* v_m;
- *вычисления* $\mathsf{pr}_1(\Omega) \to_{\pi_1} \mathsf{tl}(pt_1)$, $\mathsf{pr}_2(\Omega) \to_{\pi_2} \mathsf{tl}(pt_2)$ $\mathcal{F}$*-совместны и имеют различные результаты.*

Тогда через одну из вершин $v_1, \ldots, v_{m-1}$ *проходит опровергающий маршрут.*

Доказательство. Пусть $\dot{pt}_1 = \mathsf{tl}(pt_1)$, $\dot{pt}_2 = \mathsf{tl}(pt_2)$, и пусть Ω_i — маршрут, оканчивающийся в вершине v_i, и $tr_j^i = \mathsf{pr}_j(\Omega_i)$, $i \in \{1, \ldots, m\}$, $j \in \{1, 2\}$. При этом считаем, что $\Omega_m = \Omega$. По лемме 5.4.1 имеем $[\![tr_1^i]\!] = s^i \circ v_i.s_1$, $[\![tr_2^i]\!] = s^i \circ s_2$, где $s^i \in \mathcal{F}.S$, $i \in \{1, \ldots, m\}$.

Случай 1: состояние управления l_1 незавершаемо в программе π_1. Тогда путь pt_1 бесконечен, а значит, путь pt_2 конечен. По лемме 5.4.3 трасса tr_1^1 и

вычисление $cp_2 = tr_2^1 \to_{\pi_2} \dot{pt}_2$ $\mathcal{F}$-совместны. Используя утверждение 3.1.2, продолжим трассу tr_1^1 до вычисления cp_1, $\mathcal{F}$-совместного с вычислением cp_1. По условию леммы и рассматриваемому случаю вычисление cp_1 бесконечно, а вычисление cp_2 конечно.

Случай 2: состояние управления l_2 незавершаемо в программе π_2. Рассуждения аналогичны случаю (**1**) и потому опущены.

Случай 3: состояние управления l_1 завершаемо в программе π_1, и существует бесконечный l_2-путь pt_2', такой что $\bar{\text{æ}}_{s_2}^{\mathsf{tl}(pt_2')}$ — префикс развития $\mathsf{pref}(\bar{\text{æ}}, s)$. Повторяя рассуждения случаев (**4**), (**5**) обоснования леммы 5.6.2, получим индекс $p \in \{1, \ldots, m-1\}$, l_1-путь pt_1'' программы π_1 и l_2-путь pt_2'' программы π_2, такие что вычисления $cp_1 = tr_1^p \to_{\pi_1} \mathsf{tl}(pt_1'')$ и $cp_2 = tr_2^p \to_{\pi_2} \mathsf{tl}(pt_2'')$ $\mathcal{F}$-совместны и имеют различные результаты. Затем, применив те же рассуждения, что и в случае (**2**) обоснования достаточности теоремы 5.4.1, получим опровергающий маршрут, проходящий через вершину v_p.

Случай 4: состояние управления l_2 завершаемо в программе π_2, и существует бесконечный l_1-путь pt_1', такой что $\bar{\bar{\text{æ}}}_q^{\mathsf{tl}(pt_1')}$ есть префикс обобщённого развития $\bar{\bar{\text{æ}}}$. Тогда повторим рассуждения случая (**3**), заменив упоминание случаев (**4**), (**5**) леммы 5.6.2 на случай (**1**) этой леммы.

Случай 5:

- точки l_1, l_2 завершаемы в своих программах;
- не существует бесконечного l_2-пути pt_2', такого что $\bar{\text{æ}}_{s_2}^{\mathsf{tl}(pt_2')}$ — префикс развития $\mathsf{pref}(\bar{\text{æ}}, s)$;
- не существует бесконечного l_1-пути pt_1', такого что $\bar{\bar{\text{æ}}}_q^{\mathsf{tl}(pt_1')}$ — префикс обобщённого развития $\mathsf{pref}(\bar{\bar{\text{æ}}}, k)$.

Рассмотрим префиксы $\ddot{pt}_1$, $\ddot{pt}_2$ путей $\dot{pt}_1$, $\dot{pt}_2$ соответственно, такие что $\mathsf{pref}(\bar{\bar{\text{æ}}}, k) = \bar{\bar{\text{æ}}}_q^{\overline{\mathsf{tl}}(\ddot{pt}_1)} \neq \bar{\bar{\text{æ}}}_q^{\ddot{pt}_1}$ и $\mathsf{pref}(\bar{\text{æ}}, s) = \bar{\text{æ}}_{s_2}^{\overline{\mathsf{tl}}(\ddot{pt}_2)} \neq \bar{\text{æ}}_{s_2}^{\ddot{pt}_2}$. По рассматриваемому случаю путь $\ddot{pt}_1$ является q-приведённым, а путь $\ddot{pt}_2$ — s_2-приведённым. По

леммам 5.5.12, 5.5.18 имеем неравенства $|\ddot{pt}_1| \le |\mathfrak{A}.Q|^2 \cdot |\pi_1|$, $|\ddot{pt}_2| \le |\mathfrak{A}.Q| \cdot |\pi_2|$. Повторяя рассуждения случая (**1.б**) обоснования леммы 5.6.1, получим индекс p, такой что трассы $tr_1^p \to_{\pi_1} \ddot{pt}_1$ и $tr_2^p \to_{\pi_2} \ddot{pt}_2$ $\mathcal{F}$-совместны.

Рассмотрим вычисления $cp_1 = tr_1^p \to_{\pi_1} \dot{pt}_1$, $cp_2 = tr_2^p \to_{\pi_2} \dot{pt}_2$. Для завершения обоснования осталось показать, что вычисления cp_1, cp_2 $\mathcal{F}$-совместны и имеют различные результаты.

Покажем различие результатов. Так как вычисления $\mathsf{pr}_1(\Omega) \to_{\pi_1} \dot{pt}_1$, $\mathsf{pr}_2(\Omega) \to_{\pi_2} \dot{pt}_2$ имеют различные результаты, одно из них конечно. Конечность этих вычислений равносильна конечности путей $\dot{pt}_1$, $\dot{pt}_2$ соответственно. Если ровно один из путей $\dot{pt}_1$, $\dot{pt}_2$ конечен, а значит, ровно одно из вычислений cp_1, cp_2 конечно, а значит, эти вычисления имеют различные результаты. Пусть теперь оба пути $\dot{pt}_1$, $\dot{pt}_2$ конечны. Тогда оба вычисления cp_1, cp_2 конечны.

Предположим, что $[\![cp_1]\!] = [\![cp_2]\!]$. По условию леммы верно $\bar{\bar{\mathsf{æ}}}\,[k+1]\,[1](v_p.s_1) = \bar{\bar{\mathsf{æ}}}\,[k+1]\,[1](v_m.s_1)$. По лемме 5.5.4 получаем $\overline{\mathsf{hd}}(\bar{\bar{\mathsf{æ}}})\,[1](v_p.s_1) = \overline{\mathsf{hd}}(\bar{\bar{\mathsf{æ}}})\,[1](v_m.s_1)$. При этом $\overline{\mathsf{hd}}(\bar{\bar{\mathsf{æ}}})\,[1] = \mathsf{æ}_{[\![\dot{pt}_1]\!]}$, а значит, $v_p.s_1 \circ [\![\dot{pt}_1]\!] = v_m.s_1 \circ [\![\dot{pt}_1]\!]$ (лемма 5.5.3). По предположению и лемме 5.2.7 верно равенство $v_p.s_1 \circ [\![\dot{pt}_1]\!] = s_2 \circ [\![\dot{pt}_2]\!]$, а значит, $v_m.s_1 \circ [\![\dot{pt}_1]\!] = s_2 \circ [\![\dot{pt}_2]\!]$, что означает совпадение результатов вычислений $\mathsf{pr}_1(\Omega) \to_{\pi_1} \dot{pt}_1$, $\mathsf{pr}_2(\Omega_2) \to_{\pi_2} \dot{pt}_2$, что противоречит условию леммы. Полученное противоречие позволяет считать обоснованным неравенство $[\![cp_1]\!] \ne [\![cp_2]\!]$.

Покажем совместность. Предположим, что вычисления cp_1, cp_2 несовместны. По утверждению 3.1.2 это означает, что существуют собственные префиксы tr_1, tr_2 вычислений cp_1, cp_2 соответственно, такие что $[\![tr_1]\!] = [\![tr_2]\!]$ и ни для какого логического условия σ не выполняются оба равенства $\overline{\mathsf{hd}}(tr_1) \xrightarrow{\sigma}_{\pi_1} cp_1\,[|tr_1|+1]$, $\overline{\mathsf{hd}}(tr_2) \xrightarrow{\sigma}_{\pi_2} cp_2\,[|tr_2|+1]$. По лемме 5.4.3 справедливы неравенства $|tr_1| \ge |tr_1^p|$, $|tr_2| \ge |tr_2^p|$. Значит, справедливо представление $tr_1 = tr_1^p \to_{\pi_1} pt_1'$, $tr_2 = tr_2^p \to_{\pi_2} pt_2'$, и тогда по лемме 5.2.7 имеем $v_p.s_1 \circ [\![pt_1']\!] = s_2 \circ [\![pt_2']\!]$. Напомним, что трассы $tr_1^p \to_{\pi_1} \ddot{pt}_1$ и $tr_2^p \to_{\pi_2} \ddot{pt}_2$ $\mathcal{F}$-совместны. Это означает, что справедливо хотя бы одно из неравенств $|pt_1'| \ge |\ddot{pt}_1|$, $|pt_2'| \ge |\ddot{pt}_2|$. Если верно

первое из этих неравенств, то по условию леммы и по лемме 5.5.3 получим: $v_m.s_1 \circ [\![pt'_1]\!] = \text{æ}_{[\![pt'_1]\!]}(v_m.s_1) \circ [\![pt'_1]\!] = \text{æ}_{[\![pt'_1]\!]}(v_p.s_1) \circ [\![pt'_1]\!] = v_p.s_1 \circ [\![pt'_1]\!] = s_2 \circ [\![pt'_2]\!]$. Полученные равенства с учётом изначальных ограничений на трассы tr_1, tr_2 приводит к несовместности вычислений $\mathsf{pr}_1(\Omega) \to_{\pi_1} \dot{pt}_1$, $\mathsf{pr}_2(\Omega) \to_{\pi_2} \dot{pt}_2$, что противоречит условию леммы. Следовательно, $|pt'_1| < |\ddot{pt}_1|$ и $|pt'_2| \geq \ddot{pt}_2$.

Существуют индексы q, r, такие что $\text{æ}_{v_p.s_1 \circ [\![pt'_1]\!]} = \bar{\bar{\text{æ}}}\,[q]\,[2]$ и $\text{æ}_{s_2 \circ [\![pt'_2]\!]} = \bar{\text{æ}}\,[r]$. Равенство $v_p.s_1 \circ [\![pt'_1]\!] = s_2 \circ [\![pt'_2]\!]$ влечёт за собой равенство $\bar{\bar{\text{æ}}}\,[q]\,[2] = \bar{\text{æ}}\,[r]$. Неравенство $|pt'_2| \geq \ddot{pt}_2$ означает $r \geq s+1$, а неравенство $|pt'_1| < |\ddot{pt}_1|$ — $q \leq k$. При этом по условию леммы $\bar{\text{æ}}\,[s+1] \in \{\bar{\bar{\text{æ}}}\,[j]\,[2] \mid k < j \leq |\bar{\bar{\text{æ}}}|\}$, а значит, $\bar{\bar{\text{æ}}}\,[q]\,[2] \lhd \bar{\bar{\text{æ}}}\,[k]\,[2] \lhd \bar{\text{æ}}\,[s+1] \lhd \bar{\text{æ}}\,[r]$ и $\bar{\bar{\text{æ}}}\,[k]\,[2] \neq \bar{\text{æ}}\,[s+1]$, то есть равенство $\bar{\bar{\text{æ}}}\,[q]\,[2] = \bar{\text{æ}}\,[r]$ невозможно. Полученное противоречие обосновывает совместность вычислений cp_1, cp_2. □

Из лемм 5.6.1, 5.6.2 вытекает следующий принцип ограничения числа вершин графа совместных вычислений. Рассмотрим состояния управления l_1, l_2 программ π_1, π_2 соответственно, состояние данных s_2 и состояние q автомата $\mathfrak{A}$. Определим в одну группу все вершины v, для которых верно $v.l_1 = l_1$, $v.l_2 = l_2$, $v.s_2 = s_2$ и $\mathfrak{A}(v.s_1) = q$. Пусть в процессе обхода исследовано $|\mathfrak{A}.Q|^3 \cdot |\pi_1| \cdot |\pi_2|^2 + 2$ вершин из заданной группы. Если существуют l_1- и l_2-пути, удовлетворяющие лемме 5.6.1, то можно прекратить обход графа и констатировать неэквивалентность программ. В противном случае в дальнейшем обходе можно игнорировать дуги, исходящие из последней исследованной вершины и всех неисследованных вершин группы. Для фиксированного множества операторов $\mathfrak{O}$ и фиксированной шкалы $\mathcal{F}$ множители в числе групп, добавляемые состояниями s_2 и q константны с учетом леммы 5.4.1. Таким образом, число групп вершин полиномиально относительно размеров программ, и в каждой группе в процессе обхода накапливается полиномиальное число вершин, так что леммы 5.6.1, 5.6.2 позволяют описать полиномиальный обход графа Γ, по завершении которого даётся ответ о $\mathcal{F}$-эквивалентности программ. Алгоритм, основывающийся на таком об-

ходе, приведён в разделе 5.7.

5.7. Полиномиальная разрешимость проблемы эквивалентности

Алгоритм проверки $\mathcal{F}$-эквивалентности программ π_1, π_2 для упорядоченной шкалы $\mathcal{F} = \mathcal{F}(\mathcal{C}, \mathcal{A})$ состоит в произвольном обходе графа $\Gamma = \Gamma(\pi_1, \pi_2, \mathcal{F})$. Для простоты записи оценок сложности введём сокращение: $N = \max\{|\pi_1|, |\pi_2|\}$. В процессе обхода все вершины разбиваются на группы. В группу $\mathsf{Group}\,(l_1, l_2, q, s)$ попадают вершины v, для которых верно: $v.l_1 = l_1$, $v.l_2 = l_2$, $\mathfrak{A}(v.s_1) = q$, $v.s_2 = s$. Обход начинается в корне графа Γ. При исследовании очередной q-вершины v последовательно производятся следующие действия.

- Если вершина v опровергающая, то обход прекращается, и констатируется неэквивалентность программ.
- Если добавление вершины v создаёт в исследованном фрагменте цикл, такой что для всех меток ℓ его дуг верно $\ell\,[i] \in \varepsilon$ и для всех его вершин v' верно: состояние $v'.l_i$ завершаемо в программе π_i — то обход прекращается, и констатируется неэквивалентность программ (здесь $i \in \{1, 2\}$).
- Вершина v добавляется в группу $\mathsf{Group}\,(v.l_1, v.l_2, \mathfrak{A}(v.s_1)\,, v.s_2)$.
- Если группа $\mathsf{Group}\,(v.l_1, v.l_2, \mathfrak{A}(v.s_1)\,, v.s_2)$ удовлетворяет лемме 5.6.1, то обход прекращается, и констатируется неэквивалентность программ.
- Если группа $\mathsf{Group}\,(v.l_1, v.l_2, \mathfrak{A}(v.s_1)\,, v.s_2)$ удовлетворяет лемме 5.6.2, то дуги, исходящие из вершины v, игнорируются в дальнейшем обходе.
- Если непосещённые вершины исчерпаны, то констатируется эквивалентность программ; иначе исследуется следующая вершина.

Чтобы обосновать полиномиальность предложенного обхода, достаточно показать, что каждый шаг в его описании может быть произведён за полиномиальное время. Для этого детально опишем все нетривиальные вычислительные действия, производимые алгоритмом.

Перед началом обхода производятся следующие действия:

- для всех состояний управления программ π_1, π_2 определяется, завершаемы ли они в своих программах;
- строится автомат $\mathfrak{A}$, распознающий эффекты подавления;
- определяются все развития и обобщённые развития эффектов подавлений;
- для каждого развития и каждого состояния управления l программы π_2 проверяется, существует ли в этой программе конечный полный l-путь заданным развитием и бесконечный l-путь с заданным развитием;
- для каждого обобщённого развития и каждого состояния управления l программы π_1 проверяется, существует ли в этой программе конечный полный l-путь с заданным обобщённым развитием и бесконечный l-путь с заданным обобщённым развитием.

Чтобы определить завершаемость состояний управления, достаточно проверить для каждого состояния, достижим ли из него выход программы. Автомат $\mathfrak{A}$ может быть построен за константное время, как это описано в обоснования леммы 5.5.5, и имеет константный (не зависящий от размеров программ) размер. По автомату $\mathfrak{A}$ и его декартову квадрату можно определить за константное время все развития и обобщённые развития — их число также не зависит от размеров программ. Для ответов на вопросы о наличии конечных и бесконечных путей достаточно рассмотреть декартово произведение программы π_2 с автоматом $\mathfrak{A}$ и программы π_1 с декартовым квадратом автомата $\mathfrak{A}$ (то есть “расклеить” каждое состояние управления по состояниям автомата и соединить

состояния управления так, чтобы сохранялись переходы как программы, так и автомата) и для каждого состояния управления проверить:

- достижим ли из этого состояния выход программы по пути с заданными метками (состояний автомата);
- достижим ли из этого состояния цикл, все вершины которого помечены заданной меткой, по пути с заданными метками.

Перед анализом остальных действий, производимых алгоритмом, приведём несколько утверждений о сложности вычислений в моноиде $\mathcal{M}(\mathcal{C}, \mathcal{A})$ и о размере графа Γ.

Лемма 5.7.1. *Обходом исследуется не более* $\mathsf{O}\left(N^5\right)$ *вершин графа* Γ.

Доказательство. Всего существует $\mathsf{O}\left(N^2\right)$ различных групп $\mathsf{Group}\,(l_1, l_2, q, s)$ (теорема 5.3.1, лемма 5.4.1) и константное число обобщённых развитий (лемма 5.5.15). Длины обобщённых развитий ограничены сверху общей константой (лемма 5.5.6, 5.5.7, 5.5.14). Таким образом, в каждой группе достаточно накопить $\mathsf{O}\left(N^3\right)$ вершин, до того как эта группа удовлетворит одной из лемм 5.6.1, 5.6.2. □

Лемма 5.7.2. *Минимальный представитель состояния данных* $[\![h]\!]$*, где* $h \in \mathfrak{O}^*$*, может быть вычислен за время* $\mathsf{O}(|h|)$.

Доказательство. Подадим цепочку h^- на вход автомату $\mathfrak{A}$, распознающему подавление операторов (теорема 5.3.1). Для состояния $\mathfrak{A}(g^-)$ определено множество операторов a, таких что $[\![ag]\!] = [\![g]\!]$. По лемме 5.2.6 требуемый минимальный представитель получается из h удалением таких операторов a. □

Лемма 5.7.3. *Для заданных цепочек* h*,* g *можно за время* $\mathsf{O}\Big(\left(|h| + |g|\right)^2\Big)$ *проверить соотношение* $[\![h]\!] \preceq [\![g]\!]$ *и, если оно выполнено, вычислить цепочку* h'*, для которой верно* $[\![g]\!] = [\![h]\!] \circ [\![h']\!]$.

Доказательство. Проверим требуемое соотношение пошагово, на каждом шаге удаляя символы из текущих цепочек и проверяя соотношение $[\![h_i]\!] \preceq [\![g_i]\!]$ для полученных цепочек h_i, g_i. На первом шаге рассмотрим минимальных представителей h_1, g_1 состояний $[\![h]\!]$, $[\![g]\!]$ соответственно. Пусть на i-м шаге получены цепочки h_i, g_i. Если $h_i = \lambda$, то соотношение $[\![h_i]\!] \preceq [\![g_i]\!]$ выполнено и $h' = g_i$. Пусть теперь $h_i \neq \lambda$. По леммам 5.2.6, 5.2.9 соотношение $[\![h_i]\!] \preceq [\![g_i]\!]$ выполнено тогда и только тогда, когда существует минимальный представитель $h'_i h''_i$ класса $[\![g_i]\!]$, которого можно получить из g_i, применяя только коммутацию, и такой что h'_i получено из h_i удалением операторов. Выделим в цепочке h_i самый левый оператор: $h_i = aw$. Если $[\![ag_i]\!] = [\![g_i]\!]$, то в цепочке $h'_i h''_i$ оператор a удалён (последнее равенство может быть проверено с помощью автомата, распознающего подавление операторов — теорема 5.3.1). Тогда перейдём к следующему шагу: $h_{i+1} = \mathsf{tl}(h_i)$, $g_{i+1} = g_i$. Иначе оператор a не удалён в цепочке $h'_i g'_i$. Если цепочка g_i не содержит вхождений оператора a, то этот оператор нельзя получить из g_i, используя только коммутацию, а значит, соотношение $[\![h_i]\!] \preceq [\![g_i]\!]$ не выполнено. Пусть теперь $g_i\,[j]$ — самое левое вхождение оператора a в цепочку g_i. Если $\mathsf{hd}(g_i) \leftrightsquigarrow a, \ldots, g_i\,[j-1] \leftrightsquigarrow a$, то, используя существование вывода $g_i \overset{|}{\rightleftharpoons} \ldots \overset{|}{\rightleftharpoons} a\mathsf{pref}(g_i, j-i)\,\overline{\mathsf{pref}}(g_i, j)$ и лемму 5.2.6, перейдём к следующему шагу: $h_{i+1} = \mathsf{tl}(h_i)$, $g_{i+1} = \mathsf{pref}(g_i, j-1)\,\overline{\mathsf{pref}}(g_i, j)$. В противном случае по лемме 5.2.2 оператор a не может появиться в левой позиции, если к цепочке g_i применять только коммутацию, а значит, соотношение $[\![h_i]\!] \preceq [\![g_i]\!]$ не выполнено. □

Рассмотрим вершину v графа Γ. Состояния $v.s_1$, $v.s_2$ кодируются минимальными представителями этих состояний. По лемме 5.4.1 код состояния $v.s_2$ — это либо оператор, либо пустое слово. Рассмотрим дугу $v \xrightarrow{\ell} v'$. Для цепочек h, h', кодирующих состояния s_1, s'_1 соответственно, верно: $|h'| \leq |h| + 1$. В процессе обхода алгоритмом исследуется не более $\mathsf{O}\big(N^5\big)$ вершин графа Γ (лемма 5.7.1). Это значит, что длина цепочки, кодирующей состояние данных $v.s_1$, не превышает $\mathsf{O}\big(N^5\big)$. Тогда полиномиальность определения вершины v' следует

из лемм 5.7.2, 5.7.3.

При исследовании очередной вершины v прежде всего происходит сравнение её со всеми уже исследованными вершинами v'. Наибольшая сложность здесь вносится проверкой равенства $v.s_1 = v.s_2$ — полиномиальность такой проверки обосновывается леммой 5.7.3. Проверки того, является ли вершина опровергающей, появились ли в исследованном фрагменте графа Г циклы и применимы ли леммы 5.6.1, 5.6.2 к группе, в которую добавлена вершина, также полиномиальны.

Таким образом, предложенный алгоритм завершает свою работу за время, полиномиальное относительно размеров программ. Это позволяет считать обоснованной полиномиальную разрешимость проблемы эквивалентности пропозициональных последовательных программ на упорядоченных шкалах с коммутативностью и подавлением.

Теорема 5.7.1. *Пусть $\mathcal{F}$ — упорядоченная шкала с коммутативностью и подавлением. Тогда проблема $\mathcal{F}$-эквивалентности пропозициональных последовательных программ разрешима за время, полиномиальное относительно размеров этих программ.*

Глава 6

Эквивалентность линейных унарных рекурсивных программ на упорядоченных полугрупповых шкалах

Данная глава посвящена:

- описанию достаточных условий полиномиальной разрешимости проблемы эквивалентности линейных унарных рекурсивных программ на упорядоченных полугрупповых шкалах;
- описанию быстрых (полиномиальных по времени работы относительно размеров программ) алгоритмов, решающих проблему эквивалентности линейных унарных рекурсивных программ на упорядоченных полугрупповых шкалах, удовлетворяющих описанным достаточным условиям, с обоснованием корректности и сложности этих алгоритмов.

Результаты данной главы опубликованы в работе [38] в журнале из перечня ВАК.

В данной главе используются следующие понятия и утверждения:

- общие понятия, описанные в главе 2;
- понятия и утверждения, относящиеся к линейным унарным рекурсивным программам (раздел 3.2);
- понятия, введённые при описании подхода к исследованию проблемы эквивалентности (раздел 4.2).

Исследование проблемы эквивалентности программ на шкале $\mathcal{F}$ разбивается на следующие этапы:

- на структуру шкалы $\mathcal{F}$ накладываются ограничения, определяющие общий вид различий пар термов (раздел 6.1) — эти ограничения включаются в достаточные условия полиномиальной разрешимости проблемы эквивалентности;

- описывается граф совместных вычислений, в нём выделяются опровергающие маршруты, и проверка $\mathcal{F}$-эквивалентности программ сводится к проверке наличия таких маршрутов в графе (раздел 6.2);

- определяются ограничения на структуру шкалы $\mathcal{F}$, при которых проблема $\mathcal{F}$-эквивалентности линейных унарных рекурсивных программ оказывается разрешимой (раздел 6.3) и полиномиально разрешимой (раздел 6.4, теорема 6.4.1).

Для краткости в данном главе будем называть линейные унарные рекурсивные программы просто программами. Далее считаем заданными:

- конечные алфавиты операторов $\mathfrak{O}$ и логических условий $\mathfrak{L}$;

- программы π_1, π_2;

- числа $n.q = \max\{|\pi_1|.q, |\pi_2|.q\}$, $n.D = \max\{|\pi_1|.D, |\pi_2|.D\}$;

- упорядоченную полугрупповую шкалу $\mathcal{F}$.

Используя утверждения 3.2.1, 3.2.2, считаем программы π_1, π_2 нормализованными. Исключениями являются описания решающих алгоритмов (в них явно обозначен шаг нормализации программ) и формулировки теорем 6.3.1, 6.4.1 (в них явно выписаны все используемые предположения).

6.1. Критериальные системы

Различие линейных термов основывается на понятии критериальной системы. *Критериальная система* — это четверка $\mathcal{K} = (W, U, w^+, w^*)$, где:

- W — моноид (называемый далее *критериальным*) с операцией $\circ$ и нейтральным элементом e;
- U — подмоноид моноида W;
- w^+, w^* — выделенные элементы моноида W.

К критериальной системы *для шкалы* $\mathcal{F}$ предъявляются следующие требования. Пусть $\mathcal{M}$ — определяющий моноид шкалы $\mathcal{F}$. Тогда:

1. существует гомоморфизм (далее называемый *определяющим гомоморфизмом*) $\varphi : \mathcal{M} \times \mathcal{M} \to U$, такой что

$$w^+ \circ \varphi(s_1, s_2) \circ w^* = e \quad \Leftrightarrow \quad s_1 = s_2;$$

2. уравнение $w^+ \circ w \circ X = e$, где $w \in U$, имеет не более одного решения относительно X в классе смежности $U \circ w^*$;
3. уравнение $X \circ w \circ w^* = e$, где $w \in U$, имеет не более одного решения относительно X в классе смежности $w^+ \circ U$.

На основе критериальной системы естественным образом определяется различие линейных термов. Пусть в процессе совместного выполнения программами были получены линейные термы t_1, t_2. Рассмотрим цепочки $t_i{}^{\downarrow}$ и $t_i{}^{\downarrow}$ этих термов. Используя определяющий гомоморфизм φ, заменим эти цепочки на элементы $\varphi\big(t_1{}^{\downarrow}, t_2{}^{\downarrow}\big)$, $\varphi\big(t_1{}^{\downarrow}, t_2{}^{\downarrow}\big)$ моноида U. Добавим к ним элементы w^+, w^*, обозначающие соответственно начало и конец совместной работы, — получим элементы $\psi = w^+ \circ \varphi\big(t_1{}^{\downarrow}, t_2{}^{\downarrow}\big)$, $\rho = \varphi\big(t_1{}^{\downarrow}, t_2{}^{\downarrow}\big) \circ w^*$. Элемент ψ назовем левой разностью, а ρ — правой разностью. Различие линейных термов t_1, t_2 состоит из их обрабатываемых символов $t_i{}^{\downarrow}$, левой разности ψ и правой разности ρ.

Если в результате совместной работы программами получены вычисления, оканчивающиеся базовыми термами h_1, h_2, то различие результатов этих вычислений — $w^+ \circ \varphi(h_1, h_2) \circ w^*$. Таким образом, в процессе совместной работы

достаточно сохранять левую разность ψ и правую разность ρ, если обеими программами достигнуты их выходы, то проверить равенство $\psi \circ \rho = e$.

Ограничения на существование решений уравнений в критериальном моноиде облегчают анализ различий линейных термов и содержательно могут быть переформулированы так. Предположим, что в некоторый момент совместной работы программами выполнены цепочки h_1, h_2, а в конце работы получены вычисления с результатами $h_1 g_1$, $h_2 g_2$. Тогда по левой разности $w^+ \circ \varphi([\![h_1]\!], [\![h_2]\!])$ однозначно определяется правая разность $\varphi([\![g_1]\!], [\![g_2]\!]) \circ w^*$ — и обратно, по правой разности определяется левая разность.

Приведём примеры шкал, для которых можно построить критериальную систему. В статье [9] введено понятие критериальной системы для уравновешенной полугрупповой шкалы (то есть удовлетворяющей требованию: если $[\![h]\!] = [\![g]\!]$, то $|h| = |g|$). Это понятие схоже с приведённым в данном разделе, однако имеет ряд существенных отличий, связанных с использованием критериальных систем для исследования проблемы эквивалентности только на уравновешенных шкалах. В статье приведено несколько примеров критериальных систем, например, для свободной шкалы и для частично коммутативной шкалы (полугрупповой шкалы, определяемой произвольным наборот соотношений коммутативности). Анализ приведённых примеров позволяет заключить, что они удовлетворяют требованиям к критериальной системе, описанным в данном разделе.

Для полноты описания приведём пример упорядоченной полугрупповой шкалы $\hat{\mathcal{F}}$, такой что она не является уравновешенной и для неё существует критериальная система. Например, такой является шкала $\hat{\mathcal{F}}$, определяющий моноид которой образован множеством операторов $\{a, b\}$ и задаётся единственным определяющим соотношением $aa = b$. Упорядоченность и неуравновешенность такой шкалы очевидны. Критериальная система для шкалы $\hat{\mathcal{F}}$ может иметь следующий вид. Критериальный моноид есть моноид целых чисел с операцией сложения. Элементы моноида, отмечающие начало и конец вычисления,

фиктивны (равны нолю). Пусть n_g^a — число вхождений оператора a в слово g. Состояния данных $[\![h_1]\!]$, $[\![h_2]\!]$ вкладываются в критериальный моноид следующим образом: $\varphi([\![h_1]\!], [\![h_2]\!]) = n_{h_1}^a + 2 \cdot n_{h_1}^b - n_{h_2}^a - 2 \cdot n_{h_2}^b$. Можно легко убедиться том, что выполняются все требования, предъявляемые к критериальной системе для шкалы $\hat{\mathcal{F}}$.

Далее считаем заданными:

- критериальную систему $\mathcal{K} = (W, U, w^+, w^*)$ для шкалы $\mathcal{F}$;
- операцию $\circ$ и нейтральный элемент e критериального моноида W;
- определяющий гомоморфизм φ.

Основываясь на частичном порядке состояний данных, можно разбить элементы моноида U на классы по признаку сравнимости пар состояний данных, описываемых этими элементами. Для описания такого разбиения потребуется следующая лемма.

Лемма 6.1.1. *Пусть s_1, s_2, s_3, s_4 — состояния данных шкалы $\mathcal{F}$, такие что $w^+ \circ \varphi(s_1, s_2) = w^+ \circ \varphi(s_3, s_4)$. Тогда:*

$$\begin{aligned} s_1 \preceq s_2 \quad &\Leftrightarrow \quad s_3 \preceq s_4; \\ s_1 \succeq s_2 \quad &\Leftrightarrow \quad s_3 \succeq s_4. \end{aligned}$$

Справедливость леммы 6.1.1 очевидным образом следует из описания определяющего гомоморфизма. Лемма 6.1.1 позволяет разбить класс смежности $w^+ \circ U$ на четыре подкласса:

$$\begin{aligned} U_= &= \left\{ w^+ \circ \varphi(s_1, s_2) \mid s_1 = s_2 \right\}; \\ U_\prec &= \left\{ w^+ \circ \varphi(s_1, s_2) \mid s_1 \prec s_2 \right\}; \\ U_\succ &= \left\{ w^+ \circ \varphi(s_1, s_2) \mid s_1 \succ s_2 \right\}; \\ U_\parallel &= \left\{ w^+ \circ \varphi(s_1, s_2) \mid s_1 \| s_2 \right\}. \end{aligned}$$

Введённые подклассы будут лежать в основе описания графа совместных вычислений.

6.2. Граф совместных вычислений

Чтобы сделать определение графа совместных вычислений понятнее, опишем стратегию совместного выполнения программ. Пусть в процессе совместного выполнения программами получены линейные термы t_1, t_2.

- Если ${t_1}^{\downarrow} \neq \lambda$, ${t_2}^{\downarrow} \neq \lambda$ и либо $[\![{t_1}^{\downarrow}]\!] = [\![{t_2}^{\downarrow}]\!]$, либо $[\![{t_1}^{\downarrow}]\!] \parallel [\![{t_2}^{\downarrow}]\!]$, то активны обе программы.
- Если ${t_1}^{\downarrow} \neq \lambda$ и либо ${t_2}^{\downarrow} = \lambda$, либо $[\![{t_1}^{\downarrow}]\!] \prec [\![{t_2}^{\downarrow}]\!]$, то активна первая программа.
- Если ${t_2}^{\downarrow} \neq \lambda$ и либо ${t_1}^{\downarrow} = \lambda$, либо $[\![{t_1}^{\downarrow}]\!] \succ [\![{t_2}^{\downarrow}]\!]$, то активна вторая программа.
- Иначе обе программы неактивны.

Опишем граф совместных вычислений $\Gamma = \Gamma(\pi_1, \pi_2, \mathcal{F})$. Вершина v графа Γ включает в себя компоненты $v.\phi_1$, $v.\phi_2$, $v.\psi$, $v.\rho$:

- $\phi_i \in \pi_i.H \cup \{\lambda\}$ — обрабатываемый символ текущего терма программы π_i;
- $\psi \in w^+ \circ U$ — левая разность текущих термов;
- $\rho \in U \circ w^*$ — правая разность текущих термов.

Корнем графа объявляется вершина v_{root}, для которой верно: $v_{root}.\phi_1 = \pi_1.q$, $v_{root}.\phi_2 = \pi_2.q$, $v_{root}.\psi = w^+$, $v_{root}.\rho = w^*$. Разобьем все вершины v графа совместных вычислений на пять классов, основываясь на разбиении класса смежности $w^+ \circ U$:

- $v \in V_{=}$, если $v.\phi_1 \neq \lambda$, $v.\phi_2 \neq \lambda$ и $v.\psi \in U_{=}$;
- $v \in V_{\parallel}$, если $v.\phi_1 \neq \lambda$, $v.\phi_2 \neq \lambda$ и $v.\psi \in U_{\parallel}$;
- $v \in V_{\prec}$, если $v.\phi_1 \neq \lambda$ и либо $v.\psi \in U_{\prec}$, либо $v.\phi_2 = \lambda$;
- $v \in V_{\succ}$, если $v.\phi_2 \neq \lambda$ и либо $v.\psi \in U_{\succ}$, либо $v.\phi_1 = \lambda$;
- в противном случае $v \in V_{\emptyset}$.

На основании этого разбиения опишем разметку дуг графа Γ. Метки дуг графа Γ выбираются из множества $(\mathfrak{L} \cup \{\varepsilon\})^2$. Содержательно, составляющая $\ell\,[i]$ метки ℓ есть логическое условие, по которому программа π_i совершает переход согласно стратегии совместного выполнения; значение ε означает неактивность программы (и отсутствие перехода).

- Если $v \in V_{=}$, то из вершины v исходит ровно по одной дуге с меткой (σ, σ) для каждого логического условия σ.
- Если $v \in V_{\parallel}$, то из вершины v исходит ровно по одной дуге с меткой (σ_1, σ_2) для каждой пары логических условий σ_1, σ_2.
- Если $v \in V_{\prec}$, то из вершины v исходит ровно по одной дуге с меткой (σ, ε) для каждого логического условия σ.
- Если $v \in V_{\succ}$, то из вершины v исходит ровно по одной дуге с меткой (ε, σ) для каждого логического условия σ.
- Иначе ($v \in V_{\emptyset}$) из вершины v не исходит ни одной дуги.

Для завершения описания графа Γ опишем вершину v', в которую ведет дуга с меткой ℓ из вершины v:

- если $\ell\,[i] = \varepsilon$, то $v'.\phi_i = \phi_i$ и $s_i = \hat{s}_i = s$;
- если $\ell\,[i] = \sigma \in \mathfrak{L}$ и $v.\phi_i \xrightarrow{\sigma}_{\pi_i} t$, то $v'.\phi_i = t^{\downarrow}$, $s_i = [\![t^{\downarrow}]\!]$ и $\hat{s}_i = [\![t^{\downarrow}]\!]$;

- $v'.\psi = v.\psi \circ \varphi(s_1, s_2)$, $v'.\rho = \varphi(\hat{s}_1, \hat{s}_2) \circ v.\rho$.

Далее словом "маршрут" будем обозначать (как конечные, так и бесконечные) маршруты графа Г, начинающиеся в его корне. Запись $|\Omega|$ будет использоваться для обозначения длины маршрута, то есть числа вершин этого маршрута. Чтобы поставить соответствие между маршрутами и парами трасс программ π_1, π_2, введем понятие проекции маршрута. *Проекция* $\mathsf{pr}(\Omega)$ маршрута Ω может быть определена индуктивно следующим образом:

- $\mathsf{pr}(\Omega) = (\mathsf{pr}_1(\Omega), \mathsf{pr}_2(\Omega))$;
- $\mathsf{pr}_i(v_{root}) = q_i$;
- если $\ell[i] = \varepsilon$, то $\mathsf{pr}_i\left(\Omega \xrightarrow{\ell} v\right) = \mathsf{pr}_i(\Omega)$;
- если $\ell[i] = \sigma \in \mathfrak{L}$, то $\mathsf{pr}_i\left(\Omega \xrightarrow{\ell} v\right) = \mathsf{pr}_i(\Omega) \xrightarrow{\sigma}_{\pi_i} t$;
- если Ω — бесконечный маршрут, то $\mathsf{pr}_i(\Omega)$ есть предел последовательности $\mathsf{pr}_i(\mathsf{pref}(\Omega, 1)), \mathsf{pr}_i(\mathsf{pref}(\Omega, 2)), \ldots$.

Отметим, что терм t в предпоследнем пункте определения проекции однозначно задается логическим условием σ. Составляющую $\mathsf{pr}_1(\Omega)$ проекции $\mathsf{pr}(\Omega)$ будем называть 1-проекцией, а составляющую $\mathsf{pr}_2(\Omega)$ — 2-проекцией. В следующих далее леммах 6.2.1–6.2.4 сформулированы основные свойства маршрутов, показывающие их связь с трассами программ π_1, π_2.

Лемма 6.2.1. *Для любого маршрута его* 1*-проекция и* 2*-проекция являются* $\mathcal{F}$*-совместными трассами программ* π_1, π_2 *соответственно.*

Доказательство. То, что компонентами проекции маршрута являются трассы программ π_1, π_2, очевидным образом следует из определения проекции и описания графа Г. Покажем совместность этих трасс. Предположим противное: существует маршрут Ω, проекцией которого являются несовместные трассы. Тогда по утверждению 3.2.4 в проекции этого маршрута содержатся термы $t_1 = \mathsf{pr}_1(\Omega)[i_1]$, $t_2 = \mathsf{pr}_2(\Omega)[i_2]$, такие что:

- $i_1 < |\mathsf{pr}_1(\Omega)|$, $i_2 < |\mathsf{pr}_2(\Omega)|$;
- $[\![t_1{}^{\downarrow}]\!] = [\![t_2{}^{\downarrow}]\!]$;
- ни для какого логического условия σ соотношения $t_1 \xrightarrow{\sigma}_{\pi_1} \mathsf{pr}_1(\Omega)\,[i_1+1]$, $t_2 \xrightarrow{\sigma}_{\pi_2} \mathsf{pr}_2(\Omega)\,[i_2+1]$ не выполняются одновременно.

Рассмотрим кратчайший префикс Ω' маршрута Ω, такой что его 1-проекция оканчивается термом t_1 или 2-проекция — термом t_2. Без ограничения общности считаем, что 1-проекция маршрута Ω' оканчивается термом t_1. Пусть t_2' — последний терм 2-проекции маршрута Ω'. Тогда $t_2'{}^{\downarrow} \preceq t_2{}^{\downarrow}$, а значит, существует цепочка h, для которой верно $t_2{}^{\downarrow} = t_2'{}^{\downarrow}h$. Рассмотрим кратчайший префикс Ω'' маршрута Ω, такой что его 2-проекция оканчивается термом t_2. По упорядоченности шкалы $\mathcal{F}$ все вершины маршрута Ω'', начиная с последней вершины его префикса Ω' и оканчивая предпоследней вершиной маршрута Ω'', принадлежат множеству $V_{\succ}$. Следовательно, при достраивании маршрута Ω' до маршрута Ω'' терм t_1 остается последним термом 1-проекции. Так как $[\![t_1{}^{\downarrow}]\!] = [\![t_2{}^{\downarrow}]\!]$, последняя вершина маршрута Ω'' принадлежит множеству $V_{=}$, а значит, следующий после Ω'' переход в маршруте Ω делается по метке (σ, σ), $\sigma \in \mathfrak{L}$. □

Тот факт, что проекцией маршрута является пара трасс программ, будет в дальнейшем использоваться без ссылок на лемму 6.2.1.

Лемма 6.2.2. *Пусть Ω — маршрут, оканчивающийся в вершине v. Тогда последние термы трасс $\mathsf{pr}_1(\Omega)$, $\mathsf{pr}_2(\Omega)$ имеют вид $h_1\ v.\phi_1\ g_1$, $h_2\ v.\phi_2\ g_2$ соответственно, где $w^+ \circ \varphi(h_1, h_2) = v.\psi$ и $\varphi(g_1, g_2) \circ w^* = v.\rho$.*

Лемма 6.2.2 легко обосновывается индукцией по длине маршрута Ω.

Лемма 6.2.3. *Пусть cp_1, cp_2 — конечные $\mathcal{F}$-совместные вычисления программ π_1, π_2 соответственно. Тогда существует конечный маршрут Ω, такой что $\mathsf{pr}_1(\Omega) = cp_1$ и $\mathsf{pr}_2(\Omega) = cp_2$.*

Доказательство. Выберем $\mathcal{F}$-интерпретацию $\mathcal{I}$, в которой реализуются вычисления cp_1, cp_2. Построим маршрут Ω, начиная с корня и выбирая дуги согласно описанной в начале раздела стратегии совместного выполнения программ в интерпретации $\mathcal{I}$. Для завершения обоснования отметим, что первый пункт стратегии отвечает вершинам $V_=$ (равенство текущих состояний данных) и $V_\parallel$ (несравнимость текущих состояний данных), второй пункт — вершинам $V_\prec$, третий пункт — вершинам $V_\succ$, а последний (четвертый) пункт — вершинам $V_\emptyset$. □

Лемма 6.2.4. *Пусть:*

- cp_1, cp_2 — *$\mathcal{F}$-совместные вычисления программ* π_1, π_2 *соответственно;*
- $i \in \{1, 2\}$*;*
- *вычисление* cp_i *бесконечно;*
- *вычисление* cp_{3-i} *конечно.*

Тогда существует бесконечный маршрут Ω*, такой что* $\mathsf{pr}_i(\Omega) = cp_i$*, а* $\mathsf{pr}_{3-i}(\Omega)$ *есть префикс вычисления* cp_{3-i}*.*

Обоснование леммы 6.2.4 совпадает с обоснование леммы 6.2.3 с учётом определения проекции бесконечного маршрута.

Согласно подходу к проверке $\mathcal{F}$-эквивалентности программ, описанному в разделе 4.2, среди всех маршрутов графа Г следует выделить опровергающие. Маршрут будем называть опровергающим в следующих случаях:

- он оканчивается в вершине v, для которой верно $v.\phi_1 = v.\phi_2 = \lambda$ и $v.\psi \circ v.\rho \neq e$;
- он бесконечен, и для всех его вершин v, кроме, быть может, конечного числа, верно $v.\psi \in U_\succ$ и $v.\phi_1 \neq f_\infty$ (бесконечный опровергающий маршрут *типа 1*);

- он бесконечен, и для всех его вершин v, кроме, быть может, конечного числа, верно $v.\psi \in U_{\prec}$ и $v.\phi_2 \neq f_\infty$ (бесконечный опровергающий маршрут *типа 2*);

Содержательно:

- конечный опровергающий маршрут отвечает совместному выполнению программ, в результате которого получаются конечные совместные вычисления, имеющие различные результаты;

- бесконечный опровергающий маршрут типа i отвечает совместному выполнению программ, в результате которого программой π_{3-i} строится бесконечное вычисление, а программой π_i — конечная трасса, которую можно достроить до конечного вычисления и такая что текущее состояние данных ее последнего терма превосходит текущие состояния данных всех термов вычисления программы π_{3-i}.

Теорема 6.2.1. *Программы π_1, π_2 $\mathcal{F}$-эквивалентны тогда и только тогда, когда граф $\Gamma(\pi_1, \pi_2, \mathcal{F})$ не содержит опровергающих маршрутов.*

Доказательство. ($\Rightarrow$): Пусть граф Γ содержит опровергающий маршрут Ω. Покажем, что программы π_1, π_2 не являются $\mathcal{F}$-эквивалентными.

Случай 1: маршрут Ω конечен. По лемме 6.2.1 1-проекция и 2-проекция этого маршрута — совместные трассы программ π_1, π_2 соответственно. По лемме 6.2.2 эти трассы являются вычислениями, имеющими различные результаты.

Случай 2: Ω — бесконечный опровергающий маршрут типа i, $i \in \{1, 2\}$. Для определённости положим $i = 1$. По лемме 6.2.1 1-проекция и 2-проекция этого маршрута — совместные бесконечное вычисление программы π_1 и конечная трасса программы π_2 соответственно. Если трасса $\mathsf{pr}_2(\Omega)$ является вычислением, то обоснование завершено. Пусть теперь трасса $\mathsf{pr}_2(\Omega)$ не является вычислением, и пусть она оканчивается термом t_2. По определению опровергающего

маршрута верно неравенство $t_2{}^{\downarrow} \neq f_\infty$, и для любого терма t_1 вычисления $\mathsf{pr}_1(\Omega)$ верно неравенство $[\![t_1{}^{\downarrow}]\!] \prec [\![t_2{}^{\downarrow}]\!]$. Следовательно, трассу $\mathsf{pr}_2(\Omega)$ можно продолжить до конечного вычисления cp, и по упорядоченности шкалы и утверждению 3.2.4 вычисление cp обязательно будет совместно с вычислением $\mathsf{pr}_1(\Omega)$.

($\Leftarrow$): Пусть программы π_1, π_2 не являются $\mathcal{F}$-эквивалентными. Покажем, что граф Γ содержит хотя бы один опровергающий маршрут. Неэквивалентность программ означает существование их совместных вычислений cp_1, cp_2, имеющих различные результаты.

Случай 1: вычисления cp_1, cp_2 оканчиваются базовыми термами h_1, h_2 соответственно, и $[\![h_1]\!] \neq [\![h_2]\!]$. По лемме 6.2.3 существует конечный маршрут Ω, проекцией которого является пара (cp_1, cp_2). По лемме 6.2.2 для последней вершины v этого маршрута верно: $v.\phi_1 = v.\phi_2 = \lambda$, $v.\psi \circ v.\rho = w^+ \circ \varphi(h_1, h_2) \circ w^* \neq e$.

Случай 2: вычисления cp_1, cp_2 бесконечны. Тогда их результаты равны $\bot$, а значит, совпадают, чего быть не может.

Случай 3: вычисление cp_i бесконечно, а вычисление cp_{3-i} конечно, $i \in \{1, 2\}$. По лемме 6.2.4 граф Γ содержит бесконечный опровергающий маршрут типа i. □

6.3. Разрешимость проблемы эквивалентности

В данном разделе приведены утверждения, позволяющие ограничить число вершин графа совместных вычислений, которые достаточно исследовать для ответа на вопрос об $\mathcal{F}$-эквивалентности программ π_1, π_2. Этими утверждениями предоставляется экспоненциальная относительно размеров программ оценка числа вершин, и кроме того, сложность исследования графа Γ зависит от структуры шкалы $\mathcal{F}$ и критериальной системы $\mathcal{K}$. Ценность решающего алгоритма, основанного на исследовании графа совместных вычислений, состоит в том, что полиномиальный решающий алгоритм получается из него незначительной мо-

дификацией при наложении более строгих ограничений на структуру шкалы и критериального моноида. Принцип ограничения числа исследуемых вершин графа совместных вычислений приведен в леммах 6.3.1–6.3.4.

Лемма 6.3.1. *Пусть в из корня графа* Γ *достижима вершина* v*, такая что* $\phi_i = f_\infty$ *и* $\phi_{3-i} \neq f_\infty$*, где* $i \in \{1, 2\}$*. Тогда программы* π_1*,* π_2 *не являются* $\mathcal{F}$*-эквивалентными.*

Доказательство. Для определённости положим $i = 1$. Рассмотрим маршрут Ω, оканчивающийся в вершине v. Далее в доказательстве маршрут Ω будет достроен до бесконечного опровергающего маршрута — по теореме 6.2.1 получим неэквивалентность программ π_1, π_2. Обоснованием возможности построения такого маршрута (то есть выбора исходящей дуги на каждом шаге построения) в каждом из рассматриваемых далее случаев является лемма 6.2.2, определяющая вид трасс проекции конечного маршрута.

Случай 1: $v.\phi_2 = \lambda$. Продолжим маршрут Ω произвольным образом до бесконечного, используя незавершаемость символа $v.\phi_1$. Полученный маршрут является бесконечным опровергающим маршрутом типа 1.

Случай 2: $v.\phi_2 \neq \lambda$. По нормализованности программы π_2 функциональный символ $v.\phi_2$ является её завершаемым заголовком. Значит, существует конечный $v.\phi_2$-путь программы π_2, оканчивающийся базовым термом. Для определённости положим, что это путь вида

$$pt = v.\phi_2 \xrightarrow{\sigma_1}_{\pi_2} t_1 \xrightarrow{\sigma_2}_{\pi_2} \ldots \xrightarrow{\sigma_m}_{\pi_2} t_m.$$

Будем достраивать маршрут до бесконечного поэтапно, начиная с первого этапа. На j-м этапе:

- пока это возможно, выбирается исходящая дуга с меткой (σ, ε) для произвольно выбранного логического условия σ;
- затем, если маршрут не достроен до бесконечного, выбирается исходящая дуга с меткой (σ_j, σ_j) или (ε, σ_j), после чего начинается следующий этап.

Построение обязательно завершается не более чем за $(m+1)$ этап. Полученный маршрут является бесконечным опровергающим маршрутом типа 1. □

Лемма 6.3.2. *Пусть из корня графа* Γ *достижима вершина* v*, такая что* $v.\phi_i$ *— существенно рекурсивный заголовок программы* π_i *и* $v.\phi_{3-i}$ *— граничный символ программы* π_{3-i}*. Тогда программы* π_1*,* π_2 *не являются* $\mathcal{F}$*-эквивалентными.*

Доказательство. Для определённости положим $i = 1$. Рассмотрим маршрут Ω, оканчивающийся в вершине v. Опишем способ достраивания маршрута Ω до бесконечного опровергающего маршрута — по теореме 6.2.1 получим неэквивалентность программ π_1, π_2. Обоснованием возможности построения такого маршрута (то есть выбора исходящей дуги на каждом шаге построения) является лемма 6.2.2, определяющая вид трасс проекции конечного маршрута.

По существенной рекурсивности заголовка $v.\phi_1$ существует бесконечный $v.\phi_1$-путь программы π_1. Пусть, для определенности, этот путь имеет вид

$$pt = v.\phi_1 \xrightarrow{\sigma_1}_{\pi_1} t_1 \xrightarrow{\sigma_2}_{\pi_1} \dots \ .$$

Будем достраивать маршрут до бесконечного поэтапно, начиная с первого этапа. На j-м этапе:

1. пока это возможно, выбирается исходящая дуга с меткой (ε, σ) для произвольно выбранного логического условия σ;

2. затем выбирается исходящая дуга с меткой (σ_j, σ_j) или (σ_j, ε), после чего начинается следующий этап.

По выбору символа $v.\phi_2$ каждый этап построения конечен, при этом число этапов бесконечно. Полученный маршрут является бесконечным опровергающим маршрутом типа 1. □

Лемма 6.3.3. *Пусть из корня графа* Γ *достижимы вершины* v_1, v_2, *такие что* $v_1.\phi_1 = v_2.\phi_1$, $v_1.\phi_2 = v_2.\phi_2$, $v_1.\psi = v_2.\psi$, $v_1.\rho \neq v_2.\rho$ *и символы* $v_1.\phi_1$, $v_1.\phi_2$ *завершаемы. Тогда программы* π_1, π_2 *не являются* $\mathcal{F}$*-эквивалентными.*

Доказательство. Рассмотрим маршрут Ω_1, оканчивающийся в вершине v_1. По завершаемости заголовка $v_1.\phi_1$ существует конечный $v_1.\phi_1$-путь pt программы π_1. Пусть этот путь имеет вид

$$pt = v_1.\phi_1 \xrightarrow{\sigma_1}_{\pi_1} t_1 \xrightarrow{\sigma_2}_{\pi_1} \ldots \xrightarrow{\sigma_m}_{\pi_1} t_m.$$

Достроим маршрут Ω_1 до бесконечного либо оканчивающегося в вершине из множества $V_\emptyset$ поэтапно, начиная с первого этапа. На i-м этапе:

- выбирается, пока это возможно, исходящая дуга с меткой (ε, σ) для произвольно выбранного логического условия σ;
- затем, если требуемый маршрут не получен, выбирается дуга с меткой (σ_i, σ_i) или (σ_i, ε), после чего начинается следующий этап.

Если в результате построен бесконечный маршрут, то он является опровергающим маршрутом типа $(3 - i)$. Предположим теперь, что в результате построен конечный маршрут Ω_1'. Тогда рассмотрим маршрут Ω_2, оканчивающийся в вершине v_2, и продолжим его до маршрута Ω_2', выбирая исходящие дуги с теми же метками, что и при продолжении маршрута Ω_1. Это можно сделать по определению графа совместных вычислений, а именно потому что при переходе от вершины $\dot{v}$ к вершине $\ddot{v}$ по дуге с меткой ℓ значения $\ddot{v}.\phi_1$, $\ddot{v}.\phi_2$, $\ddot{v}.\psi$ не зависят от значения $\dot{v}.\rho$. Для последних вершин v_1', v_2' маршрутов Ω_1', Ω_2' верно: $v_1'.\phi_1 = v_1'.\phi_2 = v_2'.\phi_1 = v_2'.\phi_2 = \lambda$, $v_1'.\psi = v_2'.\psi$ и (по ограничениям критериальной системы) существует элемент w моноида U, такой что $v_1'.\psi \circ v_1'.\rho = v_1.\psi \circ w \circ v_1.\rho \neq v_2.\psi \circ w \circ v_2.\rho = v_2'.\psi \circ v_2'.\rho$. Следовательно, хотя бы один из маршрутов Ω_1', Ω_2' является опровергающим. $\square$

Лемма 6.3.4. *Пусть из корня графа* Γ *достижимы вершины* v_i, $i \in \{1, \ldots, n.D^2 + 1\}$, *такие что:*

- *компоненты* $v_i.\phi_1$ *всех вершин* v_i, $i \in \{1, \ldots, n.D^2 + 1\}$, *совпадают;*
- *компоненты* $v_i.\phi_2$ *всех вершин* v_i, $i \in \{1, \ldots, n.D^2 + 1\}$, *совпадают;*
- $v_1.\phi_1$, $v_1.\phi_2$ — *существенно рекурсивные завершаемые заголовки программ* π_1, π_2 *соответственно;*
- *элементы* $v_i.\psi$, $i \in \{1, \ldots, n.D^2 + 1\}$, *попарно различны.*

Тогда программы π_1, π_2 *не являются* $\mathcal{F}$*-эквивалентными.*

Доказательство. Схема доказательства такова. Пусть $\phi_1 = v_1.\phi_1$, $\phi_2 = v_1.\phi_2$. Прежде всего строятся ϕ_1-путь pt_1 программы π_1 и ϕ_2-путь pt_2 программы π_2, оканчивающиеся в крайних заголовках этих программ и такие что $|pt_1| \leq n.D$ и $|pt_2| \leq n.D$. Затем показывается, что хотя бы один из маршрутов, оканчивающихся в вершинах v_i, можно продолжить по тем же логическим условиям, по которым строились пути pt_1, pt_2, при этом избегая вершин множества $V_=$. Затем маршрут продолжается еще на одну вершину v так, чтобы один из символов $v.\phi_1$, $v.\phi_2$ оказался существенно рекурсивным заголовком, тогда как другой — граничным символом. Неэквивалентность программ в этом случае следует из леммы 6.3.2.

Описанные выше пути pt_1, pt_2 существуют по определению существенно рекурсивных завершаемых заголовков. Пусть $t^j_m = pt_j\,[m]$ для $j \in \{1, 2\}$ и $1 \leq m \leq |pt_j|$. Покажем, что хотя бы для одного индекса I, $1 \leq I \leq n.D^2 + 1$, верно следующее: для любых индексов m_1, m_2, $1 \leq m_1 \leq |pt_1|$, $1 \leq m_2 \leq |pt_2|$, верно неравенство $v_I.\psi \circ \varphi\Big(\Big[\!\Big[t^1_{m_1}{}^{\downarrow}\Big]\!\Big], \Big[\!\Big[t^2_{m_2}{}^{\downarrow}\Big]\!\Big]\Big) \circ w^* \neq e$. Чтобы показать это, достаточно заметить, что по свойствам критериальной системы для каждой пары индексов m_1, m_2 существует не более одного индекса i_{m_1,m_2}, для которого верно равенство $v_{i_{m_1,m_2}}.\psi \circ \varphi\Big(\Big[\!\Big[t^1_{m_1}{}^{\downarrow}\Big]\!\Big], \Big[\!\Big[t^2_{m_2}{}^{\downarrow}\Big]\!\Big]\Big) \circ w^* = e$. Существование искомого индекса I следует

из того, что число различных пар индексов m_1, m_2 не превосходит $n.D^2$, тогда как индекс I может принимать $(n.D^2+1)$ значение.

Пусть $t^j_m \xrightarrow{\sigma^j_m}_{\pi_j} t^j_{m+1}$ для $j \in \{1,2\}$ и $1 \leq m < |pt_j|$. Выберем произвольным образом маршрут, оканчивающийся в вершине v_I. Продолжим этот маршрут следующим образом. При добавлении очередной вершины выбирается дуга с меткой ℓ, где либо $\ell\,[p] = \varepsilon$, либо:

- подсчитывается число l дуг, выбранных на предыдущих шагах, для меток ℓ' которых верно $\ell'\,[2] \neq \varepsilon$;
- выбирается логическое условие $\ell\,[p] = \sigma^p_{l+1}$.

Добавление вершин прекращается, как только появляется необходимость выбрать (несуществующее) логическое условие $\sigma^p_{|pt_p|}$. Пусть в результате получен маршрут Ω, оканчивающийся в вершине v'. По выбору индекса I верно неравенство $v'.\psi \notin U_=$. Тогда возможен один из следующих вариантов.

Случай 1: $v'.\psi \in U_\prec$. Тогда $v'.\phi_1$ — крайний заголовок программы π_1, а $v'.\phi_2$ — существенно рекурсивный заголовок программы π_2. Продолжим маршрут на одну вершину по метке (σ, ε) так, чтобы символ $\pi_1.D(v'.\phi_1, \sigma)^\downarrow$ оказался граничным символом программы π_1.

Случай 2: $v'.\psi \in U_\succ$. Рассуждения данного случая аналогичны случаю (**1**) с заменой индексов 1 на 2 и обратно.

Случай 3: $v'.\psi \in U_\parallel$. Один из символов $v'.\phi_1$, $v'.\phi_2$ является крайним, а другой — существенно рекурсивным. Пусть, для определенности, крайним символом является $v'.\phi'_1$. Тогда продолжим маршрут на одну вершину по метке (σ_1, σ_2) так, чтобы было верно: $\pi_2.D(v'.\phi_2, \sigma_2)^\downarrow$ — существенно рекурсивный заголовок программы π_2; $\pi_1.D(v'.\phi_1, \sigma_1)^\downarrow$ — граничный символ программы π_1. □

Опишем алгоритм проверки $\mathcal{F}$-эквивалентности, основываясь на свойствах графа Γ и леммах, описанных в данном разделе.

На первом этапе алгоритма происходит нормализация программ π_1, π_2. По утверждению 3.2.1 этот этап можно провести за время $\mathsf{O}\big(n.q + n.D^4\big)$, и при этом размер запросов результирующих программ равен единице, а размер описаний тел функций не превосходит $N.D = \mathsf{O}\big(n.q + n.D^2\big)$.

Второй этап алгоритма состоит в произвольном обходе графа Γ, начиная с корня этого графа. Пусть в процессе обхода посещена вершина v.

- Если $v.\phi_1 = v.\phi_2 = f_\infty$, то дуги, исходящие из вершины v, игнорируются в дальнейшем обходе.

- Если ровно один из символов $v.\phi_1$, $v.\phi_2$ совпадает с f_∞, то обход завершается и по лемме 6.3.1 констатируется неэквивалентность программ.

- Если ровно один из символов ϕ_1, ϕ_2 существенно рекурсивен, то обход завершается и по лемме 6.3.2 констатируется неэквивалентность программ.

- Если ни одно из предыдущих условий не выполнено и ранее была посещена вершина v', такая что $v'.\phi_1 = v.\phi_1$, $v'.\phi_2 = v.\phi_2$, $v'.\psi = v.\psi$ и $\rho' \neq \rho$, то обход завершается и по лемме 6.3.3 констатируется неэквивалентность программ.

- Если ни одно из предыдущих условий не выполнено и ранее были посещены вершины v_i, $i \in \big\{1, \ldots, N.D^2\big\}$, такие что $v_i.\phi_1 = v.\phi_1$, $v_i.\phi_2 = v.\phi_2$ и элементы $v_i.\psi$ попарно различны и отличны от ψ, то обход завершается и по лемме 6.3.4 констатируется неэквивалентность программ.

- Иначе обход продолжается, пока не будут исчерпаны посещаемые вершины.

В процессе обхода также отслеживается наличие опровергающих маршрутов.

- Если посещена вершина v, такая что $v.\phi_1 = v.\phi_2 = \lambda$ и $v.\psi \circ v.\rho \neq e$, то обход завершается и констатируется неэквивалентность программ.

- Если в процессе обхода построен цикл, такой что либо все его вершины принадлежат множеству $V_{\prec}$ и символ $v.\phi_2$ всех его вершин v завершаем, либо все его вершины принадлежат множеству $V_{\succ}$ и символ $v.\phi_1$ всех его вершин v завершаем, то обход завершается и констатируется неэквивалентность программ.

- Если обход завершён и программы не признаны неэквивалентными, то констатируется эквивалентность программ.

Оценим число вершин, накапливаемых в процессе обхода. Назовём вершину v граничной, если оба символа $v.\phi_1$, $v.\phi_2$ граничны, и внутренней иначе. В процессе обхода посещается не более $N.D^2 \cdot \left(N.D^2+1\right)$ внутренних вершин: первый множитель отвечает всевозможным парам заголовков программ, хотя бы один из которых не является граничным, а второй — ограничению числа вершин с заданными заголовками (одна, две или $\left(N.D^2+1\right)$ в зависимости от типов заголовков). По определению граничности символов из каждой граничной вершины достижимы только другие граничные вершины, причем не более $|\mathfrak{L}|^{2N.D}$. Из каждой внутренней вершины по одной дуге достигается не более $|\mathfrak{L}|^2$ граничных вершин. Следовательно, в процессе обхода посещается не более $N.D^2 \cdot \left(N.D^2+1\right) \cdot |\mathfrak{L}|^{2N.D+2}$ граничных вершин: если корень графа Γ является граничной вершиной, то граф Γ содержит только граничные вершины, и их не более $|\mathfrak{L}|^{2N.D}$; иначе перед посещением граничной вершины обязательно посещается внутренняя.

В процессе обхода элементы критериального моноида $\psi = w^+ \circ \varphi(\llbracket h_1 \rrbracket, \llbracket h_2 \rrbracket)$ и $\rho = \varphi(\llbracket h_1 \rrbracket, \llbracket h_2 \rrbracket) \circ w^*$ могут быть представлены парой цепочек h_1, h_2. Для определения того, посещается ли вершина графа впервые, и подсчета числа вершин в леммах 6.3.3, 6.3.4 используется проверка равенства элементов критериального моноида.

Проблема равенства элементов критериального моноида *Для це-*

почек h_1, h_2, h_3, h_4 *проверить, выполняются ли равенства:*

$$\begin{aligned} w^+ \circ \varphi([\![h_1]\!], [\![h_2]\!]) &= w^+ \circ \varphi([\![h_3]\!], [\![h_4]\!]), \\ \varphi([\![h_1]\!], [\![h_2]\!]) \circ w^* &= \varphi([\![h_3]\!], [\![h_4]\!]) \circ w^*, \\ w^+ \circ \varphi([\![h_1]\!], [\![h_2]\!]) \circ w^* &= w^+ \circ \varphi([\![h_3]\!], [\![h_4]\!]) \circ w^*. \end{aligned}$$

Элементы, приписанные корню графа Г, представлены пустыми цепочками. При переходе из вершины v в вершину v' к цепочкам, представляющим элемент ψ, дописывается не более чем по одному оператору, а к цепочкам, представляющим элемент ρ — не более чем по $N.D$ операторов. Всего дописывание происходит не более $\left(N.D^2 \cdot \left(N.D^2 + 1\right) + N.D\right)$ раз: первое слагаемое — максимальное число внутренних вершин, второе — максимальное число посещаемых друг за другом граничных вершин.

Для определения типа вершины (чтобы выбрать исходящую дугу и для проверить наличие бесконечных опровергающих маршрутов) необходимо сравнить состояния данных, задаваемые цепочками, представляющими элемент ψ. Чтобы определить, в каком из отношений $=$, $\prec$, $\succ$, $\|$ находятся состояния данных s_1, s_2, достаточно проверить, находятся ли они в отношениях $s_1 \preceq s_2$ и $s_2 \preceq s_1$.

Проблема достижимости состояний данных *Для цепочек* h_1, h_2 *проверить, справедливо ли соотношение* $[\![h_1]\!] \preceq [\![h_2]\!]$.

Для определения типа вершины достаточно сделать две проверки достижимости состояний данных, подав на вход цепочки, представляющие элемент ψ. Граничность, завершаемость и существенная рекурсивность заголовков схем может быть определена перед началом обхода за время, полиномиальное относительно размеров программ (как было отмечено в разделе 3.2.5).

Таким образом:

- длина цепочек, кодирующих элементы $v.\rho$ вершин v в процессе обхода, не превосходит $\mathsf{O}\left(N.D^5\right)$;

- длина цепочек, кодирующих элементы $v.\psi$ вершин v в процессе обхода, не превосходит $\mathsf{O}\left(N.D^4\right)$;
- алгоритмом посещается не более $\mathsf{O}\left(N.D^4\right)$ внутренних вершин и не более $2^{\mathsf{O}(N.D)}$ граничных;
- если равенство элементов критериального моноида и достижимость состояний данных проверяется за конечное время, то все операции, производимые над вершинами, завершаются за конечное время.

Описание и анализ алгоритма позволяют считать обоснованной следующую теорему.

Теорема 6.3.1. *Пусть:*

- $\mathcal{F}$ *— полугрупповая упорядоченная шкала;*
- $\mathcal{K} = (W, U, w^+, w^*)$ *— критериальная система для шкалы* $\mathcal{F}$*;*
- *проблема равенства элементов критериального моноида разрешима;*
- *проблема достижимости состояний данных разрешима.*

Тогда проблема $\mathcal{F}$*-эквивалентности линейных унарных рекурсивных программ* π_1, π_2 *также разрешима.*

Замечание. *Описанный алгоритм в общем случае не является полиномиальным по двум причинам:*

1. *число исследуемых вершин графа совместных вычислений в общем случае экспоненциально относительно размеров программ;*
2. *проверки равенства и достижимости, используемые алгоритмом, могут иметь высокую сложность.*

6.4. Полиномиальная разрешимость проблемы эквивалентности

Для получения полиномиального алгоритма, решающего проблему $\mathcal{F}$-эквивалентности линейных унарных рекурсивных программ, на шкалу $\mathcal{F}$ и критериальную систему $\mathcal{K} = (W, U, w^+, w^*)$ налагаются дополнительные ограничения:

- проблема равенства элементов критериального моноида W разрешима за полиномиальное время (относительно размеров входных цепочек);
- проблема достижимости состояний данных разрешима за полиномиальное время (относительно размеров входных цепочек);
- критериальный моноид является группой.

Первые два ограничения позволяют не заботиться о сложности решения проблем равенства и достижимости, лежащих в основе решающего алгоритма. Последнее ограничение позволяет снизить число граничных вершин, посещаемых при обходе графа совместных вычислений, с экспоненциального до полиномиального. Такое снижение числа вершин основываетя на сформулированных далее леммах 6.4.1, 6.4.2. Напомним, что в формулировках лемм подразумеваются нормализованные линейные программы с размерами описаний тел функций, не превосходящими $n.D$.

Лемма 6.4.1. *Пусть из корня графа* Γ *достижимы вершины* v_i, $i \in \{1, \ldots, n.D^2 + 2\}$, *такие что:*

- *символы* $v_i.\phi_1$, $i \in \{1, \ldots, n.D^2 + 2\}$, *совпадают;*
- *символы* $v_i.\phi_2$, $i \in \{1, \ldots, n.D^2 + 2\}$, *совпадают;*
- $v_1.\phi_1$, $v_1.\phi_2$ — *граничные символы своих программ;*

- *элементы* $v_i.\psi^{-1} \circ v_i.\rho^{-1}$, $i \in \{1, \ldots, n^2+2\}$, *попарно различны.*

Тогда программы π_1, π_2 *не являются* $\mathcal{F}$*-эквивалентными.*

Доказательство. Пусть Ω_i — маршрут, оканчивающийся в вершине v_i, $i \in \{1, \ldots, n^2+2\}$. Продолжим трассы $\mathsf{pr}_1(\Omega_1)$, $\mathsf{pr}_2(\Omega_1)$ до (конечных) вычислений кратчайшим образом, то есть добавляя как можно меньше термов. Пусть в результате получены трассы $\mathsf{pr}_j(\Omega_1) \xrightarrow{\sigma_1^j}_{\pi_j} t_1^j \xrightarrow{\sigma_2^j}_{\pi_j} \ldots \xrightarrow{\sigma_{m_j}^j}_{\pi_j} t_{m_j}^j$, $j \in \{1,2\}$. Продолжим трассы проекций остальных маршрутов Ω_i, используя те же последовательности логических условий σ_m^j. Теми же рассуждениями, что и в лемме 6.3.4, можно показать, что хотя бы в двух парах (из $\left(n^2+2\right)$ построенных пар) вычисления $\mathcal{F}$-совместны. Пусть это пары вычислений, построенные для маршрутов Ω_{I_1}, Ω_{I_2}.

Пусть $v_1.\phi_j$-путь схемы π_j, построенный по логическим условиям σ_m^j, $1 \leq m \leq m_j$, оканчивается базовым термом h_j. По леммам 6.2.2, 6.2.3 из вершины v_{I_p} достижима вершина $\dot{v}_p$, такая что $\dot{v}_p.\phi_1 = \dot{v}_p.\phi_2 = \lambda$ и $\dot{v}_p.\psi \circ \dot{v}.\rho = v_{I_p}.\psi \circ \varphi(\llbracket h_1 \rrbracket, \llbracket h_2 \rrbracket) \circ v_{I_p}.\rho$, $p \in \{1,2\}$. Если $v_{I_1}.\psi \circ \varphi(\llbracket h_1 \rrbracket, \llbracket h_2 \rrbracket) \circ v_{I_1}.\rho = v_{I_2}.\psi \circ \varphi(\llbracket h_1 \rrbracket, \llbracket h_2 \rrbracket) \circ v_{I_2}.\rho = e$, то $\varphi(\llbracket h_1 \rrbracket, \llbracket h_2 \rrbracket) = v_{I_1}.\psi^{-1} \circ v_{I_1}.\rho^{-1} = v_{I_2}.\psi^{-1} \circ v_{I_2}.\rho^{-1}$, чего не может быть по условию леммы. □

Лемма 6.4.2. *Пусть из корня графа* Γ *достижимы вершины* v_i, $i \in \{1, \ldots, n.D^2+2\}$, *такие что:*

- *символы* $v_i.\phi_1$, $i \in \{1, \ldots, n.D\}$, *совпадают;*
- *символы* $v_i.\phi_2$, $i \in \{1, \ldots, n.D\}$, *совпадают;*
- $v_1.\phi_1$, $v_2.\phi_2$ — *граничные символы своих программ;*
- *элементы* $v_i.\psi^{-1} \circ v_i.\rho^{-1}$, $i \in \{1, \ldots, n.D^2+2\}$, *совпадают;*
- *через вершины* $v_1, \ldots, v_{n.D^2+1}$ *не проходит ни одного опровергающего маршрута.*

Тогда через вершину $v_{n.D^2+2}$ также не проходит ни одного опровергающего маршрута.

Доказательство. Предположим, что через вершину $v_{n.D^2+2}$ проходит опровергающий маршрут Ω. Пусть $\Omega' = \Omega_{n.D^2+2}$ — префикс этого маршрута, оканчивающийся в вершине $v_{n.D^2+2}$. По граничности символов $v_{n.D^2+2}.\phi_1$, $v_{n.D^2+2}.\phi_2$ маршрут Ω конечен. Тогда по леммам 6.2.1, 6.2.2 трассы $\mathsf{pr}_1(\Omega)$, $\mathsf{pr}_2(\Omega)$ являются $\mathcal{F}$-совместными вычислениями программ π_1, π_2 соответственно, имеющими различные результаты. По определению проекции маршрута трассы $\mathsf{pr}_1(\Omega')$, $\mathsf{pr}_2(\Omega')$ являются префиксами вычислений $\mathsf{pr}_1(\Omega)$, $\mathsf{pr}_2(\Omega)$ соответственно. Пусть $\mathsf{pr}_j(\Omega)[m] \xrightarrow{\sigma^j_m}_{\pi_j} \mathsf{pr}_j(\Omega)[m+1]$, где $|\mathsf{pr}_j(\Omega')| \leq m < |\mathsf{pr}_j(\Omega)|$, $j \in \{1,2\}$. Продолжим проекции маршрутов Ω_i, оканчивающихся в вершинах v_i, $i \in \{1,\ldots,n^2+1\}$, по тем же логическим условиям σ^j_m до вычислений cp^i_1, cp^i_2. Теми же рассуждениями, что и в лемме 6.3.4, можно получить индекс I, для которого вычисления cp^I_1, cp^I_2 $\mathcal{F}$-совместны. По лемме 6.2.3 существует маршрут, проекцией которого являются эти вычисления. Пусть этот маршрут оканчивается вершиной v^I, где $v^I.\phi_1 = v^I.\phi_2 = \lambda$, и пусть маршрут Ω оканчивается вершиной v. Теми же рассуждениями, что и в лемме 6.4.1, получим цепочки h_1, h_2, для которых верно $v.\psi \circ v.\rho = v_{n.D^2+2}.\psi \circ \varphi([\![h_1]\!], [\![h_2]\!]) \circ v_{n.D^2+2}.\rho$ и $v^I.\psi \circ v^I.\rho = v_I.\psi \circ \varphi([\![h_1]\!], [\![h_2]\!]) \circ v_I.\rho$. Если $v_I.\psi \circ \varphi([\![h_1]\!], [\![h_2]\!]) \circ v_I.\rho = e$, то по условию леммы верны равенства $\varphi([\![h_1]\!], [\![h_2]\!]) = v_I.\psi^{-1} \circ v_I.\rho^{-1} = v_{n.D^2+2}.\psi^{-1} \circ v_{n.D^2+2}.\rho^{-1}$, а значит, $v.\psi \circ v.\rho = v_{n.D^2+2}.\psi \circ \varphi([\![h_1]\!], [\![h_2]\!]) \circ v_{n^2+2}.\rho = e$, чего быть не может. Следовательно, предположение о том, что через вершину $v_{n.D^2+2}$ проходит опровергающий маршрут, неверно. □

Модифицируем решающий алгоритм, предложенный в разделе 6.3, следующим образом. Добавим в него два условия. Пусть в процессе обхода посещена вершина v, где $v.\phi_1$, $v.\phi_2$ — граничные символы.

1. Если ранее были посещены вершины v_i, $i \in \{1,\ldots,N.D^2+1\}$, такие что $v_i.\phi_1 = v.\phi_1$, $v_i.\phi_2 = v.\phi_2$ и элементы $v_i.\psi^{-1} \circ v_i.\rho^{-1}$ попарно различны и

отличны от $v.\psi^{-1} \circ v.\rho^{-1}$, то обход завершается и по лемме 6.4.1 констатируется неэквивалентность программ.

2. Если ранее были посещены вершины v_i, $i \in \{1, \ldots, N.D^2 + 1\}$, такие что $v_i.\phi_1 = v.\phi_1$, $v_i.\phi_2 = v.\phi_2$ и $v_i.\psi^{-1} \circ v_i.\rho^{-1} = v.\psi^{-1} \circ v.\rho^{-1}$, то дуги, исходящие из вершины v, игнорируются в дальнейшем обходе согласно лемме 6.4.2.

Чтобы учесть дополнительные условия в оценке сложности решающего алгоритма, добавим к проблеме равенства элементов критериального моноида следующую подпроблему: для цепочек $h_1, \ldots, h_8$ определить, справедливо ли равенство $(w^+ \circ \varphi(h_1, h_2))^{-1} \circ (\varphi(h_3, h_4) \circ w^*)^{-1} = (w^+ \circ \varphi(h_5, h_6))^{-1} \circ (\varphi(h_7, h_8) \circ w^*)^{-1}$.

Отличие оценок сложности модифицированного алгоритма во введенных ограничениях от исходного алгоритма состоит в следующем.

- В процессе обхода в модифицированном алгоритме посещается $\mathsf{O}\left(N.D^6\right)$ граничных вершин (в отличие от $2^{\mathsf{O}(N.D)}$ граничных вершин в исходном алгоритме). Таким образом, всего посещается $\mathsf{O}\left(N.D^6\right)$ вершин графа Γ.

- Все операции, производимые над каждой вершиной, завершаются за полиномиальное время (относительно размеров исходных программ).

Описание и анализ модифицированного алгоритма позволяют считать обоснованной следующую теорему.

Теорема 6.4.1. *Пусть:*

- $\mathcal{F}$ *— полугрупповая упорядоченная шкала;*

- $\mathcal{K} = (W, U, w^+, w^*)$ *— критериальная система для шкалы* $\mathcal{F}$*;*

- *моноид* W *является группой;*

- *проблема равенства элементов критериального моноида разрешима за полиномиальное время;*
- *проблема достижимости состояний данных разрешима за полиномиальное время.*

Тогда проблема $\mathcal{F}$-эквивалентности линейных унарных рекурсивных программ разрешима за время, полиномиальное относительно размеров программ.

Глава 7

Сильная эквивалентность металинейных унарных рекурсивных программ

Данная глава посвящена:

- обоснованию полиномиальной разрешимости проблемы сильной эквивалентности металинейных унарных рекурсивных программ;
- описанию быстрого (полиномиального по времени работы относительно размеров программ) алгоритма, решающего проблему эквивалентности, с обоснованием корректности и сложности этого алгоритма.

Результаты данной главы опубликованы в работе [39] в журнале из перечня ВАК.

В данной главе используются следующие понятия и утверждения:

- общие понятия, описанные в главе 2;
- понятия и утверждения, относящиеся к металинейным унарным рекурсивным программам (раздел 3.2);
- понятия, введённые при описании подхода к исследованию проблемы эквивалентности (раздел 4.2).

Для краткости в данной главе будем называть металинейные унарные рекурсивные программы просто программами.

В разделе 7.1 приведены понятия и утверждения, облегчающие анализ сильной эквивалентности программ. В числе прочего они позволяют описать и исследовать граф совместных вычислений (раздел 7.2), позволяющий свести проверку сильной эквивалентности программ к проверке достижимости вершины в графе. Граф совместных вычислений в общем случае бесконечен, однако в

разделе 7.3 приводятся утверждения, позволяющие ограничить число вершин графа, которые достаточно исследовать для ответа на вопрос о сильной эквивалентности программ. На основе этих утверждений в разделе 7.4 описывается и анализируется алгоритм, проверяющий сильную эквивалентность программ за время, полиномиальное относительно размеров этих программ, и формулируется и обосновывается теорема 7.4.1, констатирующая полиномиальную разрешимость проблемы сильной эквивалентности металинейных унарных рекурсивных программ.

Далее считаем заданными:

- конечные алфавиты операторов $\mathfrak{O}$ и логических условий $\mathfrak{L}$;
- программы π_1, π_2;
- числа $n.q = \max\{|\pi_1|.q, |\pi_2|.q\}$, $n.D = \max\{|\pi_1|.D, |\pi_2|.D\}$;
- свободную шкалу $\mathcal{F}$.

Замечание. *Сильная эквивалентность программ (по определению, данному в разделе 3.2.3) означает их эквивалентность на свободной шкале $\mathcal{F}$. Символ для обозначения свободной шкалы вводится явно, так как будет использоваться в записи понятий "$\mathcal{F}$-интерпретация" и "$\mathcal{F}$-совместность".*

7.1. Вспомогательные понятия и утверждения

Согласно утверждению 3.2.1, приведённому в разделе 3.2.6, без ограничения общности считаем программы π_1, π_2 нормализованными. Определение нормализованной программы приведено в том же разделе. Одно из преимуществ нормализованных программ, как было отмечено в разделе 3.2.6, состоит в том, что в них отсутствуют конечные безрезультатные вычисления. Таким образом, все вычисления программ делятся на бесконечные и результативные.

Следующие леммы позволяют не рассматривать тривиальные (с точки зрения эквивалентности программ) случаи.

Лемма 7.1.1. *Если оба запроса $\pi_1.q$, $\pi_2.q$ содержат тупиковый символ, то программы π_1, π_2 сильно эквивалентны.*

Лемма 7.1.2. *Если ровно один из запросов $\pi_1.q$, $\pi_2.q$ содержит тупиковый символ, то программы π_1, π_2 не являются сильно эквивалентными.*

Обоснование лемм 7.1.1, 7.1.2 довольно просто. Из завершаемости заголовков нормализованной программы, отличных от тупикового, следует, что в запросе нормализованной рекурсивной программы содержится тупиковый символ тогда и только тогда, когда все вычисления этой программы бесконечны. Значит, в лемме 7.1.1 все вычисления программ π_1, π_2 бесконечны, а в лемме 7.1.2 ровно у одной программы есть конечное вычисление. Следовательно, в лемме 7.1.1 все вычисления безрезультатны, а в лемме 7.1.2 существует $\mathcal{F}$-интерпретация, в которой вычисление ровно одно из программ конечно (утверждение 3.2.3), а следовательно, результаты вычислений программ π_1, π_2 в этой модели различны. Основываясь на леммах 7.1.1, 7.1.2, далее полагаем, что запросы $\pi_1.q$, $\pi_2.q$ не содержат тупикового символа.

Базовый терм h назовем *раскрытием* терма t в программе π, если существует t-путь этой программы, оканчивающийся термом h. Раскрытие будем называть *минимальным*, если оно имеет наименьшую (среди всех раскрытий) длину. Для каждого терма t определим его *вес* $\|t\|_\pi$ в программе π следующим образом:

- если терм t имеет хотя бы одно раскрытие в программе π, то $\|t\|_\pi$ — длина минимального раскрытия терма t в программе π;
- вес остальных термов t' бесконечен ($\|t'\|_\pi = \infty$).

Лемма 7.1.3. *Пусть π — программа, t — терм и $t \to_\pi t'$. Тогда:*

1. *все раскрытия терма t' в программе π являются раскрытиями терма t в этой программе;*

2. $\|t\|_\pi \leq \|t'\|_\pi$;

3. $\|t\|_\pi = \|t'\|_\pi$ *тогда и только тогда, когда хотя бы одно минимальное раскрытие терма t является минимальным раскрытием терма t'.*

Доказательство. Для обонования первого пункта достаточно заметить, что если tr — t'-путь, то $t \to_\pi tr$ — t-путь. Остальные пункты этой леммы следуют из первого с учетом определений веса и минимального раскрытия. □

Лемма 7.1.4. *Пусть tr — t-путь программы π, оканчивающийся термом t'. Тогда $\|t\|_\pi \leq \|t'\|_\pi$. Если при этом t' — базовый терм, то равенство достигается в том и только в том случае, если t' — минимальное раскрытие терма t.*

Лемма 7.1.4 напрямую следует из леммы 7.1.3.

Наложим на программы π_1, π_2 следующее ограничение: f_∞ — единственный функциональный символ, являющийся заголовком обеих программ. Это ограничение может быть удовлетворено переименованием заголовков программ и потому не ограничивает общности рассуждений. Используя бесконечность веса символа f_∞ и уникальность остальных заголовков программ π_1, π_2, обозначим записью $\|t\|$ вес терма $t \in (\mathfrak{D} \cup \pi_i.H)^*$ в программе π_i.

Лемма 7.1.5. *Если $\|\pi_1.q\| \neq \|\pi_2.q\|$, то программы π_1, π_2 не являются сильно эквивалентными.*

Доказательство. Без ограничения общности положим $\|\pi_1.q\| < \|\pi_2.q\|$. Пусть h — минимальное раскрытие терма $\pi_1.q$. По утверждению 3.2.3 существует $\mathcal{F}$-интерпретация, в которой реализуется вычисление программы π_1 с результатом h. Результат вычисления программы π_2 в этой же модели либо не определен, либо представляет собой базовый терм длины строго больше $|h|$. □

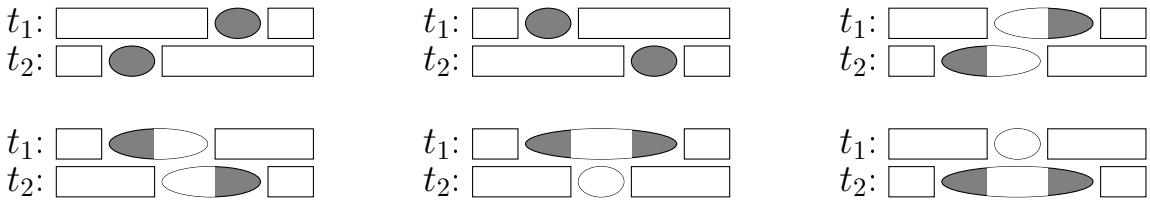


Рис. 7.1. Режим синхронной работы

Основываясь на лемме 7.1.5, далее без ограничения общности считаем, что $\|\pi_1.q\| = \|\pi_2.q\|$.

7.2. Граф совместных вычислений

Чтобы сделать описание графа совместных вычислений понятнее, предварим его описаниями различия пар термов и стратегии совместного выполнения. Чтобы описать различие термов t_1, t_2 и стратегию совместного выполнения, рассмотрим несколько случаев.

Случай 1 — *режим синхронной работы* — схематично изображен на рисунке 7.1. В этом случае предполагается, что выполненные цепочки термов совпадают и что ни в один из термов не входит тупиковый символ. Терм t изображается в виде идущих слева направо прямоугольника ($t^{\downarrow}$), овала ($t^{\downarrow\downarrow}$) и ещё одного прямоугольника ($t^{\downarrow\Downarrow}$). Длины фигур пропорциональны весам соответствующих частей терма. На рисунке 7.1 перечислены всевозможные соотношения весов частей термов t_1, t_2. В различии термов сохраняются слова, отвечающие прямоугольникам и тёмным частям овала. При этом предполагается, что цепочки, отвечающие светлым частям овалов, равны. Стратегия совместного выполнения в этом случае гласит, что обе программы активны.

Случай 2 — *режим опережения* программы π_i программой π_{3-i} — схематично изображен на рисунке 7.2. В этом случае предполагается, что выполненная цепочка терма t_i является собственным префиксом выполненной цепочки терма t_{3-i} и что ни в один из термов не входит тупиковый символ. Отличие от

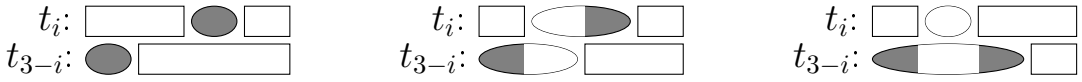


Рис. 7.2. Режим опережения

случая (**1**) состоит в следующем. Терм t_{3-i} изображается овалом (цепочка h, для которой верно $t_{3-i}{}^{\downarrow} = t_i{}^{\downarrow}h$) и следующим за ним прямоугольником ($t_{3-i}{}^{\Downarrow}$). Стратегия совместного выполнения гласит, что активна только программа π_i. В остальном описание данного случая совпадает со случаем (**1**).

Случай 3: $t_1 = t_2 \in \mathsf{BTerms}$. Тогда обе программы неактивны, и термам сопоставляется особое различие, означающее, что построены вычисления с совпадающими результатами.

Случай 4: в каждый из термов t_1, t_2 входит тупиковый символ. Тогда обе программы неактивны, и термам сопоставляется особое различие, означающее, что при продолжении совместной работы не могут быть получены вычисления с различными результатами.

Случай 5: условия случаев (**1**)–(**4**) не выполнены. Тогда обе программы неактивны, и термам сопоставляется особое различие, означающее, что обязательно существует способ продолжения совместной работы, приводящий к вычислениям с различными результатами.

Опишем граф $\Gamma = \Gamma(\pi_1, \pi_2, \mathcal{F})$ совместных вычислений программ π_1, π_2. Вершина v графа Γ включает в себя компоненты $v.\phi_1$, $v.\phi_2$, $v.u_1$, $v.u_2$, $v.w_1$, $v.w_2$, $v.R_1$, $v.R_2$:

- $v.\phi_i \in \pi_i.H \cup \{\lambda\}$: в синхронном режиме и режиме опережения программы π_i это обрабатываемый функциональный символ, а в режиме опережения программы π_{3-i} — символ λ, $i \in \{1, 2\}$;

- $v.u_1, v.u_2, v.w_1, v.w_2 \in \mathsf{BTerms}$: как в синхронном режиме, так и в режиме опережения отвечают тёмным частям овалов на рисунках 7.1, 7.2;

- $v.R_i \in \pi_i.H^*$: в синхронном режиме и режиме опережения программы π_i

это остаток отложенной части терма, а в режиме опережения программы π_{3-i} — остаток терма, $i \in \{1, 2\}$.

Корень v_{root} графа Γ определяется так: $v_{root}.\phi_1 = \pi_1.q^{\downarrow}$, $v_{root}.\phi_2 = \pi_2.q^{\downarrow}$, $v_{root}.u_1 = v_{root}.u_2 = v_{root}.w_1 = v_{root}.w_2 = \lambda$, $v_{root}.R_1 = \pi_1.q^{\downarrow}$, $v_{root}.R_2 = \pi_2.q^{\downarrow}$. Для каждой вершины v определен тип $\mathsf{type}(v)$, равный:

- $\tau_{=}$, если $v.\phi_1 \neq \lambda$ и $v.\phi_2 \neq \lambda$;
- $\tau_{\prec}$, если $v.\phi_1 \neq \lambda$ и $v.\phi_2 = \lambda$;
- $\tau_{\succ}$, если $v.\phi_1 = \lambda$ и $v.\phi_2 \neq \lambda$;
- $\tau_{\|}$, если $v.\phi_1 = \lambda$ и $v.\phi_2 = \lambda$.

Помимо описанных вершин в графе Γ содержатся также вершины v_∞ и $v_\nsim$, причем $\mathsf{type}(v_\infty) = \mathsf{type}(v_\nsim) = \tau_{\|}$. Содержательно, вершины v_∞, $v_\nsim$ отвечают случаям (**4**) и (**5**) в описании различия термов. Здесь же отметим, что случаю (**3**) отвечает вершина v, такая что $v.\phi_1 = v.\phi_2 = v.u_1 = v.u_2 = v.w_1 = v.w_2 = v.R_1 = v.R_2 = \lambda$.

Метки дуг графа Γ выбираются из множества $(\mathfrak{L} \cup \{\varepsilon\})^2$. Содержательно, составляющая $\ell\,[i]$ метки ℓ есть логическое условие, по которому программа π_i совершает переход согласно стратегии совместного выполнения; значение ε означает неактивность программы (и отсутствие перехода).

- Если $\mathsf{type}(v) = \tau_{=}$, то из вершины v исходит ровно по одной дуге с меткой (σ, σ) для каждого логического условия σ.
- Если $\mathsf{type}(v) = \tau_{\prec}$, то из вершины v исходит ровно по одной дуге с меткой (σ, ε) для каждого логического условия σ.
- Если $\mathsf{type}(v) = \tau_{\succ}$, то из вершины v исходит ровно по одной дуге с меткой (ε, σ) для каждого логического условия σ.

- Иначе ($\mathsf{type}(v) = \tau_{\|}$) из вершины v не исходит ни одной дуги.

Для завершения описания графа Γ определим вершину v', в которую ведет дуга с меткой ℓ из вершины Γ.

Этап 1: переход. По завершении этапа определяется либо вершина v', либо промежуточная вершина v''. В последнем случае вершина v' определяется на следующем этапе.

Для краткости будем использовать следующую запись: $\Delta_i = \|v.R_i\| - \|v.R_{3-i}\|$, где $i \in \{1, 2\}$.

Случай 1 (режим синхронной работы): $\ell\,[1] = \ell\,[2] = \sigma \in \mathfrak{L}$. Пусть $v.\phi_1 \xrightarrow{\sigma}_{\pi_1} t_1$ и $v.\phi_2 \xrightarrow{\sigma}_{\pi_2} t_2$. Если ${t_1}^{\downarrow} = {t_2}^{\downarrow} = f_\infty$, то этап завершается: $v' = v_\infty$. Если выполняется хотя бы одно из условий: $\mathsf{hd}(t_1) \neq \mathsf{hd}(t_2)$; $\|v.\phi_1\| - \|v.\phi_2\| \neq \|t_1\| - \|t_2\|$; ровно один из символов ${t_1}^{\downarrow}$, ${t_2}^{\downarrow}$ является тупиковым — то этап завершается: $v' = v_\infty$. Иначе этап продолжается: $v''.\phi_1 = {t_1}^{\downarrow}$, $v''.\phi_2 = {t_2}^{\downarrow}$, $v''.R_1 = v.R_1$, $v''.R_2 = v.R_2$.

Оставшиеся компоненты вершины v'' можно определить так. Рассмотрим цепочки h_1, h_2: если $t_i \notin \mathsf{BTerms}$, то $h_i = {t_i}^{\downarrow}$, иначе $t_i = \mathsf{tl}(t_i)$. Выберем индекс $j \in \{1, 2\}$ так, чтобы выполнялось неравенство $\Delta_j \leq 0$.

Подслучай 1.а: $|v.w_j| = \Delta_{3-j}$. Тогда $v''.w_1 = v.w_1$ и $v''.w_2 = v.w_2$.

Пусть $u^1 = h_1\ v.u_1$, $u^2 = h_2\ v.u_2$ и индекс k выбран так, чтобы выполнялось неравенство $|u^k| \geq |u^{3-k}|$. Если $u^{3-k} \neq \mathsf{suf}\big(u^k, |u^{3-k}|\big)$, то этап завершается: $v' = v_\infty$. Иначе $v''.u_k = \overline{\mathsf{suf}}\big(u^k, |u^{3-k}|\big)$ и $v''.u_{3-k} = \lambda$.

Подслучай 1.б: $|v.w_j| < \Delta_{3-j}$. Тогда $v''.w_j = \mathsf{suf}(h_j\ v.w_j, \Delta_{3-j})$ и $v''.w_{3-j} = \lambda$. Далее повторяются рассуждения подслучая (**1.а**) для слов $u^j = \overline{\mathsf{suf}}(h_j\ v.w_j, \Delta_{3-j})$ и $u^{3-j} = h_{3-j}\ v.u_{3-j}$.

Случай 2 (режим опережения программы π_i программой π_{3-i}): $\ell\,[i] = \sigma \in \mathfrak{L}$, $\ell\,[3-i] = \varepsilon$. Пусть $v.\phi_i \xrightarrow{\sigma}_{\pi_i} t$. Если выполнено хотя бы одно из условий: $\mathsf{hd}(t) \neq \mathsf{hd}(v.u_{3-i})$; $\|t\| \neq \|v.\phi_i\|$; $t^{\downarrow} = f_\infty$ — то этап завершается: $v' = v_\infty$. Иначе этап продолжается: $v''.f_i = t^{\downarrow}$, $v''.f_{3-i} = \lambda$, $v''.R_1 = v.R_1$, $v''.R_2 = v.R_2$.

Рассмотрим цепочку h: если $t \notin \mathsf{BTerms}$, то $h = t^{\downarrow}$, иначе $h = \mathsf{tl}(t)$.

Подслучай 2.а: либо $\Delta_i \geq 0$, либо $\Delta_i < 0$ и $|v.w_i| = \Delta_{3-i}$. Тогда $v''.w_1 = v.w_1$ и $v''.w_2 = v.w_2$. Если $\mathsf{suf}(v.u_{3-i}, |h|) \neq h$, то этап завершается: $v' = v_\infty$. Иначе $v''.u_i = \lambda$ и $v''.u_{3-i} = \overline{\mathsf{suf}}(v.u_{3-i}, |h|)$.

Подслучай 2.б: $\Delta_i < 0$, $|v.w_i| < \Delta_{3-i}$. Тогда $v''.w_i = \mathsf{suf}(h\ v.w_i, \Delta_{3-i})$ и $v''.w_{3-i} = \lambda$. Пусть $u = \overline{\mathsf{suf}}(h\ v.w_i, \Delta_{3-i})$. Если $\mathsf{suf}(v.u_{3-i}, |u|) \neq u$, то этап завершается: $v' = v_\infty$. Иначе $v''.u_i = \lambda$ и $v''.u_{3-i} = \overline{\mathsf{suf}}(v.u_{3-i}, |u|)$.

Этап 2: приведение.

Случай 1: $v''.\phi_1 = v''.\phi_2 = v''.w_1 = v''.w_2 = \lambda$, $v''.R_1 \neq \lambda$, $v''.R_2 \neq \lambda$. Тогда $v'.\phi_1 = \mathsf{hd}(v''.R_1)$, $v'.\phi_2 = \mathsf{hd}(v''.R_2)$, $v'.R_1 = \mathsf{tl}(v''.R_1)$, $v'.R_2 = \mathsf{tl}(v''.R_2)$ и $v'.u_1 = v'.u_2 = v'.w_1 = v'.w_2 = \lambda$.

Случай 2: $v''.\phi_1 = v''.\phi_2 = v''.w_i = \lambda$, $v''.w_{3-i} \neq \lambda$, где $i \in \{1, 2\}$. Тогда $v'.\phi_i = \lambda$, $v'.\phi_{3-i} = \mathsf{hd}(v''.R_{3-i})$, $v'.R_i = v''.R_i$, $v'.R_{3-i} = \mathsf{tl}(v''.R_{3-i})$, $v'.u_i = \mathsf{pref}(v''.w_i, \|v'.\phi_i\|)$, $v'.w_i = \overline{\mathsf{pref}}(v''.w_i, \|v'.\phi_i\|)$ и $v'.u_{3-i} = v'.w_{3-i} = \lambda$.

Случай 3: $v''.\phi_i = v''.u_i = \lambda$, $v''.\phi_{3-i} \neq \lambda$, где $i \in \{1, 2\}$. Тогда $v'.\phi_i = \mathsf{hd}(v''.R_i)$, $v'.\phi_{3-i} = v''.\phi_{3-i}$, $v'.R_i = \mathsf{tl}(v''.R_i)$, $v'.R_{3-i} = v''.R_{3-i}$, $v'.u_{3-i} = \mathsf{pref}(v''.w_{3-i}, \|v'.\phi_i\| - \|v'.\phi_{3-i}\|)$, $v'.w_{3-i} = \overline{\mathsf{pref}}(v''.w_{3-i}, \|v'.\phi_i\| - \|v'.\phi_{3-i}\|)$ и $v'.u_i = v'.w_i = \lambda$.

Во всех остальных случаях $v' = v''$.

Внимательное изучение этапов пересчёта вершины показывает, что он в точности соответствует описанию совместной работы программ в контексте различия термов и стратегии совместного выполнения, описанных в начале данного раздела. Этим обосновывается следующая лемма.

Лемма 7.2.1. *Пусть из корня графа* Γ *достижима вершина* v. *Тогда:*

- $v.\phi_1 \neq f_\infty$, $v.\phi_2 \neq f_\infty$;
- *хотя бы одна из цепочек* $v.u_1$, $v.u_2$ *пуста;*
- *хотя бы одна из цепочек* $v.w_1$, $v.w_2$ *пуста;*

- $|v.u_1| \le n.D$, $|v.u_2| \le n.D$;
- $|v.w_1| \le n.D$, $|v.w_2| \le n.D$;
- $\|v.\phi_1\ v.u_1\ v.w_1\ v.R_1\| = \|v.\phi_2\ v.u_2\ v.w_2\ v.R_2\|$;
- *если* $v.u_{3-i} = \lambda$, *то* $\|v.\phi_i\ v.u_i\| \le \|v.\phi_{3-i}\|$;
- *если* $v.w_{3-i} = \lambda$, *то* $\|v.w_i\ v.R_i\| \le \|v.R_{3-i}\|$.

Далее словом "маршрут" будем обозначать (как конечные, так и бесконечные) маршруты графа Γ, начинающиеся в его корне. Запись $|\Omega|$ будет использоваться для обозначения длины маршрута Ω, то есть числа вершин этого маршрута. Чтобы поставить соответствие между маршрутами и парами трасс программ π_1, π_2, введём понятие проекции маршрута. *Проекция* $\mathsf{pr}(\Omega)$ маршрута Ω может быть определена индуктивно следующим образом:

- $\mathsf{pr}(\Omega) = (\mathsf{pr}_1(\Omega)\,, \mathsf{pr}_2(\Omega))$;
- $\mathsf{pr}_i(v_{root}) = \pi_i.q$;
- если $\ell\,[i] = \varepsilon$, то $\mathsf{pr}_i\Big(\Omega \xrightarrow{\ell} v\Big) = \mathsf{pr}_i(\Omega)$;
- если $\ell\,[i] = \sigma \in \mathfrak{L}$, то $\mathsf{pr}_i\Big(\Omega \xrightarrow{\ell} v\Big) = \mathsf{pr}_i(\Omega) \xrightarrow{\sigma}_{\pi_i} t$;
- если Ω — бесконечный маршрут, то $\mathsf{pr}_i(\Omega)$ есть предел последовательности $\mathsf{pr}_i(\mathsf{pref}(\Omega, 1))\,, \mathsf{pr}_i(\mathsf{pref}(\Omega, 2))\,, \ldots$.

Отметим, что терм t в предпоследнем пункте определения проекции однозначно задается логическим условием σ. Составляющую $\mathsf{pr}_1(\Omega)$ проекции $\mathsf{pr}(\Omega)$ будем называть 1-проекцией, а составляющую $\mathsf{pr}_2(\Omega)$ — 2-проекцией. В леммах 7.2.2–7.2.8 сформулированы основные свойства маршрутов, показывающие их связь с трассами программ π_1, π_2.

Лемма 7.2.2. *Для любого маршрута его* 1*-проекция и* 2*-проекция являются совместными трассами программ* π_1, π_2 *соответственно.*

Лемма 7.2.2 легко обосновывается индукцией по длине маршрута Ω с использованием утверждения 3.2.4 и описания стратегии совместного выполнения. Тот факт, что проекцией маршрута является пара трасс программ, будет в дальнейшем использоваться без ссылок на лемму 7.2.2.

Лемма 7.2.3. *Пусть Ω' — префикс маршрута Ω, такой что $|\Omega| - |\Omega'| > n.D$. Тогда $|\mathsf{pr}_i(\Omega)| > |\mathsf{pr}_i(\Omega')|$, $i \in \{1, 2\}$.*

Доказательство. Для определенности положим $i = 1$. Пусть маршрут Ω' оканчивается вершиной v. Если $\mathsf{type}(v) \in \{\tau_=, \tau_\prec\}$, то достаточно добавить к нему одну вершину, чтобы увеличить длину 1-проекции. Пусть теперь $\mathsf{type}(v) = \tau_\succ$. При добавлении одной вершины к маршруту Ω' обязательно увеличивается длина 2-проекции. По построению графа Г верно следующее: если добавить одну вершину к маршруту Ω', то длина цепочки u_1 последней вершины маршрута уменьшится хотя бы на единицу. По лемме 7.2.1 длина этой цепочки всегда ограничена сверху числом $n.D$, а снизу — единицей. Следовательно, при добавлении $n.D$ вершин к маршруту Ω' хотя бы один раз изменяется тип его последней вершины. Достаточно добавить одну вершину после изменения типа, чтобы увеличить длину 1-проекции. □

Лемма 7.2.4. *Пусть Ω — маршрут, оканчивающийся вершиной v, $v \notin \{v_\infty, v_\nsim\}$. Тогда существуют цепочки h, g, такие что трассы $\mathsf{pr}_1(\Omega)$, $\mathsf{pr}_2(\Omega)$ оканчиваются термами $h\ v.\phi_1\ v.u_1\ g\ v.w_1\ v.R_1$, $h\ v.\phi_2\ v.u_2\ g\ v.w_2\ v.R_2$ соответственно.*

Лемма 7.2.4 легко обосновывается индукцией по длине маршрута Ω.

Лемма 7.2.5. *Пусть Ω — маршрут, оканчивающийся вершиной v_∞. Тогда последние термы трасс $\mathsf{pr}_1(\Omega)$, $\mathsf{pr}_2(\Omega)$ содержат хотя бы по одному вхождению тупикового символа.*

Лемма 7.2.5 следует из леммы 7.2.4 и построения графа совместных вычислений.

Лемма 7.2.6. *Пусть cp_1, cp_2 — $\mathcal{F}$-совместные вычисления программ π_1, π_2 соответственно. Тогда существует маршрут Ω, такой что:*

- *если маршрут Ω оканчивается в одной из вершин $v_\sim$, v_∞, то $\mathsf{pr}_i(\Omega)$ есть префикс вычисления cp_i, $i \in \{1, 2\}$;*
- *иначе $\mathsf{pr}_i(\Omega) = cp_i$, $i \in \{1, 2\}$.*

Лемма 7.2.6 легко обосновывается индукцией по длине маршрута Ω с привлечением леммы 7.2.3 для случая бесконечного маршрута Ω.

Лемма 7.2.7. *Пусть:*

- *Ω — маршрут, оканчивающийся вершиной v, $v \notin \{v_\infty, v_\sim\}$;*
- *$v.\phi_1, v.\phi_2 \in \mathfrak{F}$;*
- *$\sigma \in \mathfrak{L}$;*
- *$\|\pi_1.D(v.\phi_1, \sigma)\| - \|\pi_2.D(v.\phi_2, \sigma)\| \neq \|v.\phi_1\| - \|v.\phi_2\|$.*

Тогда 1-проекция и 2-проекция маршрута Ω могут быть продолжены до $\mathcal{F}$-совместных вычислений программ π_1, π_2 соответственно, имеющих различные результаты.

Доказательство. По леммам 7.2.2, 7.2.4 трассы $\mathsf{pr}_1(\Omega)$, $\mathsf{pr}_2(\Omega)$ $\mathcal{F}$-совместны и оканчиваются термами $t_1 = h\ v.\phi_1\ v.u_1\ g\ v.w_1\ v.R_1$, $t_2 = h\ v.\phi_2\ v.u_2\ g\ v.w_2\ v.R_2$ соответственно, где $h, g \in \mathsf{BTerms}$. Пусть $t_1 \xrightarrow{\sigma}_{\pi_1} t_1'$ и $t_2 \xrightarrow{\sigma}_{\pi_2} t_2'$. Используя утверждение 3.2.4, несложно убедиться, что трассы $tr_1 = \mathsf{pr}_1(\Omega) \xrightarrow{\sigma}_{\pi_1} t_1'$, $tr_2 = \mathsf{pr}_2(\Omega) \xrightarrow{\sigma}_{\pi_2}$ $\mathcal{F}$-совместны. Положим без ограничения общности, что справедливо неравенство $\|\pi_1.D(v.\phi_1, \sigma)\| - \|\pi_2.D(v.\phi_2, \sigma)\| < \|v.\phi_1\| - \|v.\phi_2\|$. Продолжим трассу tr_1 до вычисления программы π_1, оканчивающегося минимальным раскрытием $\dot{h}_1$ терма t_1'. Используя утверждение 3.2.4, продолжим трассу tr_2 произвольным $\mathcal{F}$-совместным образом до вычисления cp. Если вычисление

cp бесконечно, то обоснование закончено. Иначе вычисление *cp* оканчивается некоторым базовым термом $\dot{h}_2$. По лемме 7.1.4 и проведенным рассуждениям справедливы соотношения $0 = \|t_1\| - \|t_2\| < \|t'_1\| - \|t'_2\| \leq \|\dot{h}_1\| - \|\dot{h}_2\|$, а значит, $\dot{h}_1 \neq \dot{h}_2$. □

Лемма 7.2.8. *Пусть:*

- Ω — *маршрут, оканчивающийся вершиной* v*;*
- $i \in \{1, 2\}$*;*
- $v.\phi_i \in \mathfrak{F}$, $v.\phi_{3-i} = \lambda$*;*
- $\sigma \in \mathfrak{L}$*;*
- $\|\pi_i.D(v.\phi_i, \sigma)\| \neq \|v.\phi_i\|$.

Тогда 1*-проекция и* 2*-проекция маршрута* Ω *могут быть продолжены до* $\mathcal{F}$*-совместных вычислений программ* π_1, π_2 *соответственно, имеющих различные результаты.*

Обоснование леммы 7.2.8 повторяет обоснование леммы 7.2.7 с единственным различием: $(3 - i)$-проекция маршрута Ω является вычислением, и поэтому не изменяется в процессе рассуждений.

Согласно методу, изложенному в разделе 4.2, для проверки эквивалентности программ π_1, π_2 следует выделить среди маршрутов опровергающие. Опровергающими являются маршруты, оканчивающиеся в вершине v_∞. Таким образом, проверка сильной эквивалентности программ π_1, π_2 сводится к проверке достижимости вершины v_∞ из корня графа Γ:

Теорема 7.2.1. *Программы* π_1, π_2 *сильно эквивалентны тогда и только тогда, когда вершина* v_∞ *не достижима из корня графа* $\Gamma(\pi_1, \pi_2, \mathcal{F})$.

Доказательство. ($\Rightarrow$): Предположим, что из корня графа Γ достижима вершина v_∞. Из лемм 7.2.7, 7.2.8 и описания графа Γ следует, что проекция маршрута Ω, оканчивающегося в вершине v_∞, может быть продолжена до совместных вычислений, имеющих различные результаты.

($\Leftarrow$): Предположим, что программы π_1, π_2 не являются сильно эквивалентными. Тогда существуют $\mathcal{F}$-совместные вычисления cp_1, cp_2 программ π_1, π_2 соответственно, такие что:

- либо оба этих вычисления конечны и оканчиваются различными базовыми термами;
- либо одно из этих вычислений конечно, тогда как другое бесконечно.

Пусть Ω — маршрут, определяемый леммой 7.2.6. Хотя бы одно из вычислений cp_1, cp_2 конечно, а значит, и машрут Ω конечен (лемма 7.2.3). По леммам 7.2.1, 7.2.4 маршрут Ω может оканчиваться только в вершинах v_∞, v_∞. По лемме 7.2.5 маршрут Ω не может оканчиваться в вершине v_∞. □

7.3. Ограничение числа вершин графа совместных вычислений

Полиномиальный обход графа совместных вычислений, требуемый методом, описанным в разделе 4.2, основывается на следующем факте: число вершин графа совместных вычислений сильно эквивалентных программ полиномиально относительно размеров этих программ. Этот факт основывается на приведенных далее леммах 7.3.1, 7.3.2.

Лемма 7.3.1. *Пусть из корня графа* Γ *достижимы вершины* v_1, v_2, *такие что:*

- $v_1.\phi_1 = v_2.\phi_1$, $v_1.\phi_2 = v_2.\phi_2$, $v_1.R_1 = v_2.R_1$, $v_1.R_2 = v_2.R_2$;

- $|v_1.u_1| = |v_2.u_1|$, $|v_1.u_2| = |v_2.u_2|$;
- $(v_1.u_1, v_1.u_2) \neq (v_2.u_1, v_2.u_2)$.

Тогда программы π_1, π_2 не являются сильно эквивалентными.

Доказательство. По лемме 7.2.1 без ограничения общности положим $v_1.u_1 = v_2.u_1 = \lambda$. Рассмотрим маршруты Ω_1, Ω_2, оканчивающиеся в вершинах v_1, v_2 соответственно. По лемме 7.2.4 проекцией маршрута Ω_i, $i \in \{1, 2\}$, являются совместные трассы, оканчивающиеся термами вида $t_1^i = h^i\ v_i.\phi_1\ g^i\ v_i.w_1\ v_i.R_1$, $t_2^i = h^i\ v_i.\phi_2\ v_i.u_2\ g^i\ v_i.w_2\ v_i.R_2$ для первой и второй программ соответственно, где $h^1, h^2, g^1, g^2 \in \mathsf{BTerms}$.

Продолжим трассу $\mathsf{pr}_1(\Omega_1)$ до вычисления cp_1^1, результат которого есть кратчайшее раскрытие $h^1 h_1 g_1^1$ терма t_1^1, где $h_1, g_1^1 \in \mathsf{BTerms}$ и $|h| = \|v_1.\phi_1\|$. Пусть $cp_1^1\,[m+i-1] \xrightarrow{\sigma_i}_{\pi_1} cp_1^1\,[m+i]$, где $m = |\mathsf{pr}_1(\Omega_1)|$ и $1 \leq i \leq |cp_1^1| - |\mathsf{pr}_1(\Omega_1)|$. Продолжим трассу $\mathsf{pr}_1(\Omega_2)$ до вычисления cp_1^2 по тем же логическим условиям: $cp_1^2\,[m+i-1] \xrightarrow{\sigma_i}_{\pi_1} cp_1^2\,[m+i]$, где $m = |\mathsf{pr}_1(\Omega_2)|$ и $1 \leq i \leq |cp_1^2| - |\mathsf{pr}_1(\Omega_2)|$. Результатом вычисления cp_1^2 является базовый терм $h^2 h_1 g_1^2$, где $g_1^2 \in \mathsf{BTerms}$.

Используя утверждение 3.2.4, продолжим трассу $\mathsf{pr}_2(\Omega_i)$, где $i \in \{1, 2\}$, произвольным образом до вычисления cp_2^i, $\mathcal{F}$-совместного с cp_1^i. Если какое-либо из вычислений cp_2^1, cp_2^2 бесконечно, то обоснование леммы завершено. Пусть теперь оба вычисления cp_2^1, cp_2^2 конечны. Если вес результата какого-либо из вычислений cp_2^i отличен от веса результата вычисления cp_1^i, то обоснование леммы завершено. Пусть теперь веса в каждой паре результатов совпадают. Тогда результаты вычислений cp_2^i, где $i \in \{1, 2\}$, имеют вид $h^i\ h_2^i\ v_i.u_2\ g_2^i$, где $|h_2^i\ v_i.u_2| = |h_1|$. По условию верно $|v_1.u_2| = |v_2.u_2|$ и $v_1.u_2 \neq v_2.u_2$, а значит, справедливо хотя бы одно из неравенств $h_2^i\ v_i.u_2 \neq h_1$, где $i \in \{1, 2\}$. □

Лемма 7.3.2. *Пусть из корня графа Γ достижимы вершины v_1, v_2, где:*

- $v_1.\phi_1 = v_2.\phi_1$, $v_1.\phi_2 = v_2.\phi_2$, $v_1.R_1 = v_2.R_1$, $v_1.R_2 = v_2.R_2$, $v_1.u_1 = v_2.u_1$, $v_1.u_2 = v_2.u_2$;

- $|v_1.w_1| = |v_2.w_1|$, $|v_1.w_2| = |v_2.w_2|$,
- $(v_1.w_1, v_1.w_2) \neq (v_2.w_1, v_2.w_2)$.

Тогда программы π_1, π_2 не являются сильно эквивалентными.

Доказательство. По лемме 7.2.1 без ограничения общности считаем, что $v_1.w_1 = v_2.w_1 = \lambda$. Пусть Ω_i — маршрут, оканчивающийся в вершине v_i, $i \in \{1, 2\}$. По лемме 7.2.4 проекцией маршрута Ω_i являются совместные трассы, оканчивающиеся термами вида $t_1^i = h^i\ v_i.\phi_1\ v_i.u_1\ g^i\ v_i.R_1$ и $t_2^i = h^i\ v_i.\phi_2\ v_i.u_2\ g^i\ v_i.w_2\ v_i.R_2$ для первой и второй программ соответственно, где $h^1, h^2, g^1, g^2 \in \mathsf{BTerms}$. Определим вычисления cp_i^j в точности так, как это сделано в доказательстве леммы 7.3.1. Представим результат вычислений cp_1^i в виде $h^i h_1$, где $h_1 \in \mathsf{BTerms}$. Повторяя рассуждения леммы 7.3.1, считаем, что что результаты вычислений cp_2^i определены, совпадают по весу с результатами соответствующих вычислений cp_1^i и имеют вид $h^i h_2^i v_i.w_2 g_2^i$, где $|v_1.w_2 g_2^1| = |v_2.w_2 g_2^2|$. По условию верно $|v_1.w_2| = |v_2.w_2|$ и $v_1.w_2 \neq v_2.w_2$, а значит, справедливо хотя бы одно из неравенств $h_1 \neq h_2^i\ v_i.w_2\ g_2^i$, $i \in \{1, 2\}$. □

7.4. Полиномиальная разрешимость проблемы эквивалентности

Из свойств графа Γ и лемм раздела 7.3 естественным образом следует алгоритм проверки эквивалентности программ π_1, π_2, состоящий в обходе графа совместных вычислений $\Gamma(\pi_1, \pi_2, \mathcal{F})$. Пусть π_1, π_2 — металинейные рекурсивные программы (не обязательно нормализованные).

Первый этап алгоритма состоит в нормализации программ и отсечении вырожденных случаев (леммы 7.1.1, 7.1.2, 7.1.5). По утверждению 3.2.1 при наличии весов всех заголовков программ π_1, π_2 этот этап может быть проведен за время $\mathsf{O}\big(n.q + n.D^4\big)$, при этом размер запросов результирующих программ не превосходит $n.q$, а размер описаний тел функций — $N.D = \mathsf{O}\big(n.q + n.D^2\big)$.

Второй этап алгоритма состоит в произвольном обходе графа Г, начиная с корня. Если в процессе обхода посещена вершина $v_\sim$, то обход прекращается и по теореме 7.2.1 констатируется неэквивалентность программ. Если при обходе посещено более $n.q^2 \cdot N.D^3$ вершин графа Г, то обход прекращается и по леммам 7.3.1, 7.3.2 констатируется неэквивалентность программ. Если непосещенные вершины исчерпаны и неэквивалентность не констатирована, то по теореме 7.2.1 констатируется эквивалентность программ.

Для определения того, посещалась ли ранее заданная вершина v, достаточно сравнить слова, содержащиеся в вершине v, с соответствующими словами, содержащимися во всех посещенных вершинах. Перевычисление вершины при переходе по дуге графа Г делается по определению в предположении о том, что известны веса всех заголовков программ.

Веса заголовков программ могут быть вычислены до начала обхода графа Г с помощью, например, алгоритма Дейкстры [60]. Более подробно:

- вершины графа суть заголовки программы и выделенная вершина λ;
- каждая дуга $f \xrightarrow{s} f'$ определяется соотношениями $f' \to_\pi afh$ и $s = |op| + |h|$;
- каждая дуга $\lambda \xrightarrow{s} f$ определяется соотношениями $f \to_\pi h$, $h \in \mathsf{BTerms}$, и $s = |h|$;
- алгоритм находит минимальное взвешенное расстояние от вершины λ до всех остальных вершин;
- полученные расстояния назначаются весами соответствующих заголовков.

Так как число вершин, посещаемых при обходе, полиномиально и все действия над вершинами, как и действия перед началом обхода, могут быть сделаны за полиномиальное время, можно считать обоснованной полиномиальную

разрешимость проблемы сильной эквивалентности металинейных унарных рекурсивных программ.

Теорема 7.4.1. *Проблема сильной эквивалентности металинейных унарных рекурсивных программ разрешима за время, полиномиальное относительно размеров программ.*

Глава 8

Заключение

По итогам исследования, проведённого в рамках данной диссертационной работы, были получены следующие результаты.

1. Разработаны полиномиальные алгоритмы проверки эквивалентности пропозициональных последовательных программ на произвольных упорядоченных шкалах с коммутативностью и подавлением. Строго обоснованы корректность и полиномиальность разработанных алгоритмов. С использованием этих алгоритмов доказана полиномиальная разрешимость проблемы эквивалентности пропозициональных последовательных программ на произвольных упорядоченных шкалах с коммутативностью и подавлением.

2. Установлены достаточные условия полиномиальной разрешимости проблемы эквивалентности линейных унарных рекурсивных программ на упорядоченных полугрупповых шкалах. Разработаны полиномиальные алгоритмы, проверяющие указанную эквивалентность при выполнении установленных достаточных условий. Корректность и полиномиальность разработанных алгоритмов строго обоснованы.

3. Разработаны полиномиальные алгоритмы проверки сильной эквивалентности металинейных унарных рекурсивных программ. Корректность и полиномиальность разработанных алгоритмов строго обоснована. С использованием этих алгоритмов доказана полиномиальная разрешимость проблемы сильной эквивалентности металинейных унарных рекурсивных программ.

Полученные результаты вносят вклад в исследование быстрой разрешимости проблемы эквивалентности в моделях вычислений, важной для общего

понимания устройства моделей вычислений и для исследования многих актуальных практических задач программирования, так или иначе использующих сравнение функциональности программ. Разнообразие вычислительных моделей бесконечно велико, и поэтому не существует полного разграничения моделей с разрешимой и неразрешимой проблемами эквивалентности, как и моделей с разрешимой и быстро разрешимой проблемами эквивалентности. При этом полученные результаты показывают действенность и перспективность разработанного подхода для получения быстрых алгоритмов проверки эквивалентности для широких классов семантик и для различных моделей вычислений.

Список литературы

1. Буда А. О., Иткин В. Э. Сводимость эквивалентности программ к термальной эквивалентности // Системное и теоретическое программирование. — 1974. — Т. 1. — С. 293–324.
2. Глушков В. М. Теория автоматов и формальные преобразования микропрограмм // Кибернетика. — 1965. — № 5. — С. 1–9.
3. Глушков В. М., Летичевский А. А. Теория дискретных преобразователей // Избранные вопросы алгебры и логики. — 1973. — С. 5–39.
4. Ершов А. П. Об операторных схемах Янова // Проблемы кибернетики. — 1967. — Вып. 20. — С. 181—200.
5. Ершов А. П. Современное состояние теории схем программ // Проблемы кибернетики. — 1973. — Вып. 27. — С. 87–110.
6. Захаров В. А. Аппроксимация абстрактных семантик формальными моделями программ // Дискретная математика. — 1998. — Т. 10, вып. 4. — С. 119–141.
7. Захаров В. А. Быстрые алгоритмы разрешения эквивалентности операторных программ на уравновешенных шкалах // Математические вопросы кибернетики. — 1998. — Вып. 7. — С. 303–324.
8. Захаров В. А. Быстрые алгоритмы разрешения эквивалентности пропозициональных операторных программ на упорядоченных полугрупповых шкалах // Вестник Московского университета. Серия 15. Вычислительная математика и кибернетика. — 1999. — № 3. — С. 29–35.
9. Захаров В. А. Об эффективной разрешимости проблемы эквивалентности линейных унарных рекурсивных программ // Математические вопросы кибернетики. — 1999. — Вып. 8. — С. 255–273.
10. Захаров В. А. Проверка эквивалентности программ при помощи двухленточных автоматов // Кибернетика и системный анализ. — 2010. — N 4. — С. 39–48.

11. Захаров В. А., Новикова Т. А. Полиномиальный по времени алгоритм проверки логико-термальной эквивалентности программ // Труды Института системного программирования РАН. — 2012. — Т. 22. — С. 435–455.
12. Захаров В. А., Подымов В. В. Об одной полугрупповой модели программ, определяемой при помощи двухленточных автоматов // Научные ведомости Белгородского государственного университета. Серия История, экономика, политология, информатика. — 2010. — Т. 14, № 7. — С. 94–101.
13. Захаров В. А., Подымов В. В. Об эквивалентности металинейных унарных рекурсивных программ // Материалы XI Международного семинара "Дискретная математика и ее приложения", посвященного 80-летию со дня рождения академика О. Б. Лупанова. — 2012. — С. 157–159.
14. Иткин В. Э. Логико-термальная эквивалентность схем программ // Кибернетика. — 1972. — № 1. — С. 5–27.
15. Калужнин Л. А. Об алгоритмимзации математических задач // Проблемы кибернетики. — 1959. — Вып. 2. — С. 51–67.
16. Карпов Ю. Г. Model checking. Верификация моделей программ. — СПб.: БХВ-Петербург, 2010.
17. Котов В. Е., Сабельфельд В. К. Теория схем программ. — М.: Наука, 1991.
18. Летичевский А. А. Функциональная эквивалентность дискретных преобразователей I // Кибернетика. — 1969. — № 2. — С. 5–15.
19. Летичевский А. А. Функциональная эквивалентность дискретных преобразователей II // Кибернетика. — 1970. — № 2. — С. 14–28.
20. Летичевский А. А. Функциональная эквивалентность дискретных преобразователей III // Кибернетика. — 1972. — № 1. — С. 1–4.
21. Лисовик Л. П. Проблема эквивалентности для преобразователей над размеченными деревьями // Доклады АН УССР. — 1980. — № 6. — С. 77–79.
22. Лисовик Л. П. Металинейные схемы с засылками констант // Программирование. — 1985. — № 2. — С. 29–38.
23. Ляпунов А. А. О логических схемах программ // Проблемы кибернети-

ки. — 1968. — Вып. 1. — С. 46–74.

24. Пентус А. Е., Пентус М. Р. Теория формальных языков. — М.: Изд-во ЦПИ при механико-математическом ф-те МГУ, 2004.
25. Подловченко Р. И. Иерархия моделей программ // Программирование. — 1981. — № 2. — С. 3–14.
26. Подловченко Р. И. Полугрупповые модели программ // Программирование. — 1981. — № 4. — С. 3–13.
27. Подловченко Р. И. Рекурсивные программы и иерархия их моделей // Программирование. — 1991. — № 6. — С. 44–51.
28. Подловченко Р. И. Абстрактные программы с процедурами и конечные автоматы с магазином // Интеллектуальные системы. — 1997. — Т. 2, вып. 1–4. — С. 275–295.
29. Подловченко Р. И. От схем Янова к теории моделей программ // Математические вопросы кибернетики. — 1998. — Вып. 7. — С. 281–302.
30. Подловченко Р. И. О схемах программ с перестановочными и монотонными операторами // Программирование. — 2003. — № 5. — С. 46–54.
31. Подловченко Р. И. Алгебраические модели программ и автоматы // Математические вопросы кибернетики. — 2006. — Вып. 15. — С. 47–56.
32. Подловченко Р. И. К вопросу о полиномиальной сложности проблемы эквивалентности в алгебраических моделях программ // Кибернетика и системный анализ. — 2012. — № 5. — С. 17–24.
33. Подловченко Р. И. Об одном классе алгебраических моделей программ, представляющем практический интерес // Программирование. — 2013. — № 3. — С. 15–28.
34. Подловченко Р. И., Долгих Б. А. Двухступенчатое моделирование программ с процедурами // Математические вопросы кибернетики. — 2004. — Вып. 12. — С. 47–56.
35. Подловченко Р. И., Захаров В. А. Полиномиальный по сложности алгоритм, распознающий коммутативную эквивалентность схем программ //

Доклады РАН, серия Информатика. — 1998. — Т. 362, № 6. — С. 27–31.

36. Подловченко Р. И., Молчанов А. Э. Разрешимость эквивалентности в перегородчатых моделях программ // Моделирование и анализ информационных систем. — 2014. — Т. 21, № 2. — С. 56–70.
37. Подымов В. В. О проверке эквивалентности последовательных и рекурсивных программ на упорядоченных полугрупповых шкалах // Материалы X Международной конференции “Интеллектуальные системы и компьютерные науки”. — 2011. — С. 295–298.
38. Подымов В. В. Алгоритм проверки эквивалентности линейных унарных рекурсивных программ на упорядоченных полугрупповых шкалах // Вестник Московского университета. Серия 15. Вычислительная математика и кибернетика. — 2012. — № 4. — С. 37–43.
39. Подымов В. В. О проверке сильной эквивалентности металинейных унарных рекурсивных программ // Вестник Московского университета. Серия 15. Вычислительная математика и кибернетика. — 2013. — № 1. — С. 21–27.
40. Подымов В. В. Быстрый алгоритм проверки эквивалентности программ с коммутативными и подавляемыми операторами // Материалы XVII международной конференции “Проблемы теоретической кибернетики” — 2014. — С. 234–237.
41. Подымов В. В., Захаров В. А. Полиномиальный алгоритм проверки эквивалентности в модели программ с перестановочными и подавляемыми операторами // Труды Института системного программирования РАН. — 2014. — Т. 26, вып. 3. — С. 145–166.
42. Роганов Е. А. Основы информатики и программирования. — М.: МГИУ, 2001.
43. Сабельфельд В. К. Полиномиальная оценка сложности распознавания логико-термальной эквивалентности // ДАН СССР. — 1979. — Т. 249, № 4. — С. 793–796.
44. Щербина В. Л., Захаров В. А. Эффективные алгоритмы проверки эквива-

лентности программ в моделях, связанных с обработкой прерываний // Вестник Московского университета. Серия 15. Вычислительная математика и кибернетика. — 2008. — № 2. — С. 33–41.

45. Янов Ю. И. О логических схемах алгоритмов // Проблемы кибернетики. — 1958. — Вып. 1. — С. 75–127.
46. Aho A. V., Lam M. S., Sethi R., Ullman J. D. Compilers: principles, techniques, and tools. Second edition. — Addison-Wesley, 2007.
47. Ashcroft E., Manna Z., Pnueli A. Decidable properties of monadic functional schemes // Journal of the ACM. — 1973. — Vol. 20, no 3. — P. 489–499.
48. Baier C., Catoen J.-P. Principles of model checking. — MIT Press, 2008.
49. Bannwart F., Müller P. Changing Programs Correctly: Refactoring with Specifications // Proceedings of the 14th International Symposium on Formal Methods. — 2006. — P. 492–507.
50. Bird R. The equivalence problem for deterministic two-tape automata // Journal of Computer and System Science. — 1973. — Vol. 7, no 4. — P. 218–236.
51. Böhm S., Göller S. Language Equivalence of Deterministic Real-Time One-Counter Automata Is NL-Complete // Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science. — 2011 — P. 194–205.
52. Böhm S., Göller S., Jancar P. Equivalence of Deterministic One-Counter Automata is NL-complete // Proceedings of the 45th annual ACM symposium on Theory of computing. — 2013. — P. 131–140.
53. Böhm S., Göller S., Jancar P. Bisimulation equivalence and regularity for real-time one-counter automata // Journal of Computer and System Sciences. — 2014. — Vol. 80, no 4. — P. 720–743.
54. Christodorescu M., Jha S. Static Analysis of Executables to Detect Malicious Patterns // Proceedings of the 12th USENIX Security Symposium. — 2003. — P. 169–186.
55. Christodorescu M., Jha S., Seshia S., Song D., Bryant R. E. Semantics-Aware

Malware Detection // Proceedings of the 2005 IEEE Symposium on Security and Privacy. — 2005. — P. 32–46.

56. Clarke E. M., Grumberg O., Peled D. Model Checking. — MIT Press, 1999.
57. Christensen S., Hüttel H., Stirling C. Bisimulation equivalence is decidable for all context-free processes // Information and Computation. — 1995. — Vol. 121, no 2. — P. 143–148.
58. Collberg C., Thomborson C., Low D. A Taxonomy of Obfuscating Transformations. Technical Report 148, Department of Computer Science, University of Auckland, New Zealand, 1997.
59. Collberg C., Thomborson C., Low D. Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs // Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. — 1998. — P. 184–196.
60. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to algorithms, thrird edition. — MIT press, 2009.
61. Courcelle B. On jump deterministic pushdown automata // Mathematical Systems theory. — 1977. — No 11. — P. 87–109.
62. Courcelle B. A representation of trees by languages I // Theoretical Computer Science. — 1978. — Vol. 6. — P. 255–279.
63. Courcelle B. A representation of trees by languages II // Theoretical Computer Science. — 1978. — Vol. 7. — P. 25–55.
64. Engelfriet J., Maneth S., Seidl H. Deciding equivalence of top-down XML transformations in polynomial time // Journal of Computer and System Sciences. — 2009. — Vol. 75, no 5. — P. 271–286.
65. Fowler M., Beck K., Brant J., Opdyke W. Refactoring: Improving the Design of Existing Code. — Addison-Wesley, 1999.
66. Friedman E. P. Equivalence problems for deterministic languages and monadic recursion schemes // Journal of Computer and System Sciences. — 1977. — Vol. 14, no 3. — P. 342–359.

67. Gallier J. H. DPDA's in "Atomic normal form" and applications to equivalence problems // Theoretical Computer Science. — 1981. — Vol. 14, no 2. — P. 155–186.
68. Garey M. R., Johnson D. S. Computers and Intractability; A Guide to the Theory of NP-Completeness — W. H. Freeman & Co., 1990.
69. Garland S. J., Luckham D. C. Program schemes, recursion schemes and formal languages // Journal of Computer and System Sciences. — 1973. — Vol. 7, no 2. P. 119–160.
70. Goldwasser S., Rothblum G. N. On best-possible obfuscation // Journal of Cryptology. — 2014. — Vol. 27, no 3. — P. 480–505.
71. Göller S. The Fixed-Parameter Tractability of Model Checking Concurrent Systems // Proceedings of the 22nd EACSL Annual Conference on Computer Science Logic. — 2013. — P. 332–347.
72. Harel D., Kozen D., Tiuryn J. Dynamic logic. — MIT Press, 2000.
73. Harju T., Karhumäki J. The equivalence problem of multitape finite automata // Theoretical Computer Science. — 1991. — Vol. 78, no 2. — P. 347–355.
74. Harrison J. Handbook of Practical Logic and Automated Reasoning. — Cambridge University Press, 2009.
75. Hopcroft J. E., Karp R. M. A linear algorithm for testing equivalence of finite automata. Technical Report, Cornell University, Computer Science Department, TR 71–114. — 1971. — P. 5.
76. Hopcroft J. E., Motwani R., Ullman J. D. Introduction to automata theory, languages, and computation — international edition (2. ed). — Addison-Wesley, 2003.
77. Hunt H. B., Constable R. L., Sahni S. On the computational complexity of program scheme equivalence // SIAM Journal of Computing. — 1980. — Vol. 9, no 2. — P. 396–416.
78. Jančar P., Kučera A., Mayr R. Deciding bisimulation-like equivalences with

finite-state processes // Theoretical Computer Science. — 2011. — Vol. 258. — P. 409–443.

79. 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.
80. Lacey D., Jones N. D., Wyk E. V., Frederiksen C. C. Proving correctness of compiler optimizations by temporal logic // Proceedings of the 29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages. — 2002. — P. 283–294.
81. Lerner S., Millstein T. D., Chambers C. Automatically proving the correctness of compiler optimizations // Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation. — 2003. — P. 220–231.
82. Li C. M., Anbulagan. Heuristics Based on Unit Propagation for Satisfiability Problems // Proceedings of the 15th International Joint Conference on Artificial Intelligence. — 1997. — P. 366–371.
83. Luckham D. C., Park D. M., Paterson M. S. On formalized computer programs // Journal of Computer and System Science. — 1970. — Vol. 4, no 3. — P. 220–249.
84. Marques-Silva J. P., Sakallah K. A. GRASP: A New Search Algorithm for Satisfiability // Proceedings of International Conference on Computer-Aided Design. — 1996. — P. 220–227.
85. McCarthy J. Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I // Communications of the ACM. — 1960. — Vol. 3, no 4. — P. 184–195.
86. Milner R. Communication and concurrency. — Prentice Hall, 1989.
87. Mitsch S., Quesel J.-D., Platzer A. Refactoring, Refinement, and Reasoning - A Logical Characterization for Hybrid Systems // Proceedings of the 19th

International Symposium on Formal Methods. — 2014. — P. 481–496.

88. Moskewicz M. W., Madigan C. F., Zhao Y., Zhang L., Malik S. Chaff: Engineering an Efficient SAT Solver // Proceedings of the 38th Design Automation Conference. — 2001. — P. 530–535.
89. Paterson M. S. Program schemata // Machine Intelligence. — 1968. — Vol. 3. — P. 19–31.
90. Paterson M. S., Hewitt C. T. Comparative Schematology // Proceedings of the ACM Conference on Concurrent Systems and Parallel Computation. — 1970. — P. 119–127.
91. Preda M. D., Christodorescu M., Jha S., Debray S. K. A semantics-based approach to malware detection // Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. — 2007. — P. 377–388.
92. Rabin M. O., Scott D. Finite automata and their decision problems // IBM Journal of Research and Development. — 1959. — Vol. 3, no 2. — P. 114–125.
93. Rahli V., Bickford M., Anand A. Formal Program Optimization in Nuprl Using Computational Equivalence and Partial Types // Proceedings of the 4th Conference on Interactive Theorem Proving. — 2013. — P. 261–278.
94. Rice H. G. Classes of Recursively Enumerable Sets and Their Decision Problems // Transactions of the American Mathematical Society. — 1953. — Vol. 74. — P. 358–366.
95. Rosen B. K. Program equivalence and context-free grammars // Journal of Computer and System Science. — 1975. — Vol. 11, no 3. — P. 358–374.
96. Rutledge J. D. On Ianov's program schemata // Journal of the ACM. — 1964. — Vol. 11, no 1. — P. 1–9.
97. Sabelfeld V. K. The logic-termal equivalence is polynomial-time decidable // Information Processing Letters. — 1980. — Vol. 10, no 2. — P. 57–62.
98. Schäfer M., Ekman T., de Moor O. Challenge proposal: verification of refactorings // Proceedings of the 3rd Workshop on Programming Languages

Meets Program Verification. — 2009. — P. 67–72.

99. Schnoebelen Ph. Bisimulation and Other Undecidable Equivalences for Lossy Channel Systems // Proceedings of the 4th International Symposium on Theoretical Aspects of Computer Software. — 2001. — P. 385–399.
100. Sénizergues G. The equivalence problem for deterministic pushdown automata is decidable // Lecture Notes in Computer Science. — 1997. — Vol. 1256. — P. 271–281.
101. Sénizergues G. The equivalence problem for t-turn DPDA is co-NP // Lecture Notes in Computer Science. — 2003. — Vol. 2719. — P. 478–489.
102. de Souza R. On the Decidability of the Equivalence for k-Valued Transducers // Proceedings of the 12th Internationsl Conference on Developments in Language Theory. — 2008. — P. 252–263.
103. Srba J. Visibly Pushdown Automata: From Language Equivalence to Simulation and Bisimulation // Proceedings of the 20th International Workshop on Computer Science Logic. — 2006. — P. 89–103.
104. Strong H. R. Translating recursive equations into flow-charts // Journal of Computer and System Science. — 1971. — Vol. 5, no 3. — P. 254–285.
105. Szor P. The Art of Computer Virus Research and Defense. — Addison-Wesley Professional, 2005.
106. Valiant L. G. The Equivalence Problem for Deterministic Finite-Turn Pushdown Automata // Information and Control. — 1974. — Vol. 25, no 2. — P. 123–133.
107. Wakatsuki M., Tomita E., Nishino T. A Polynomial-Time Algorithm for Checking the Equivalence for Real-Time Deterministic Restricted One-Counter Transducers Which Accept by Final State // Proceedings of the 14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. — 2013. — P. 459–465.
108. Webster M., Malcolm G. Detection of metamorphic and virtualization-based malware using algebraic specification // Journal in Computer Virology. —

2009. — Vol. 5, no 3. P. 221–245.

109. Wong W., Stamp M. Hunting for metamorphic engines // Journal in Computer Virology. — 2006. — Vol. 2, no 3. P. 211–229.
110. Worrell J. Revisiting the Equivalence Problem for Finite Multitape Automata // Proceedings of the 40th International Colloquium on Automata, Languages and Programming. — 2013. — P. 422–433.
111. Young P. Optimization Among Provably Equivalent Programs // Journal of the ACM. — 1977. — Vol. 2, no 4. P. 93–700.
112. Zakharov V. A. An efficient and unified approach to the decidability of equivalence of propositional program schemes // Lecture Notes in Computer Science. — 1998. — Vol. 1443. — P. 247–258.
113. Zakharov V. A. On the decidability of the equivalence problem for monadic recursive programs // Theoretical Informatics and Applications. — 2000. — Vol. 34, no 2, P. 157-–171.
114. Zakharov V. A. Two-tape machinery for the equivalence checking of sequential programs // Proceedings of the International Workshop on Program Understanding. — 2009. — P. 28–40.
115. Zakharov V. A., Zakharyaschev I. M. On the equivalence checking problem for a model of programs related with muti-tape automata // Lecture Notes in Computer Science. — 2005. — Vol. 3317. — P. 293—305.