Федеральное государственное бюджетное учреждение науки
Институт математики им. С. Л. Соболева
Сибирского отделения Российской академии наук

На правах рукописи

Нечесов Андрей Витальевич

Полиномиальная вычислимость в семантическом программировании

Специальность 1.1.5 —
«Математическая логика, алгебра, теория чисел
и дискретная математика.»

Диссертация на соискание учёной степени
кандидата физико-математических наук

Научный руководитель:
академик РАН, д.ф.-м.н., профессор
Гончаров Сергей Савостьянович

Новосибирск — 2022

Оглавление

Стр.

Стр.

Введение

0.1 Постановка задачи и актуальность темы диссертации.

В 21 веке информационные технологии сделали серьезный скачок в своем развитии. Бурными темпами начали развиваться такие направления как криптовалюты, блокчейны, умные контракты, децентрализованные системы, нейронные сети, искусственный интеллект, робототехника и т.д. Во всех этих направлениях исследований требуется, чтобы все процессы происходили быстро и четко, без задержек и зацикливаний. Одной из важнейших характеристик для этого является понятие полиномиальной временной сложности. Т.е. процесс вычисления и выдача результата должны быть осуществлены не позднее чем через время равное некоторому полиному от длины входных данных. Далеко не все алгоритмы на сегодняшний день удовлетворяют этой характеристике.

Проблема 1. *Проблема быстродействия, целостности, адаптивности и надежности интеллектуальных систем. [5]*

Но порой одной полиномиальной временной сложности недостаточно, особенно, если это касается направления искусственного интеллекта [33]. Где нужно не только быстро выдать конечный результат, но и объяснить человеку на понятном ему языке, почему был выдан этот результат. К сожалению, большинство реализованных на сегодняшний день алгоритмов не могут этого сделать или делают это крайне плохо. Почти все они реализованы с помощью нейронных сетей [28] или других алгоритмов машинного обучения [21], которые натренированы на определенных данных и могут выдавать только конечный результат и то с достаточно большой долей погрешности. Эти сети не могут логически объяснить человеку, почему этот результат был выбран. Поэтому такого типа интеллектуальные системы чаще всего не могут быть использованы на передовых и ответственных направлениях таких как: проведение серьезных хирургических операций, управление роботами, управление автомобилями и т.д.

Возникает так называемая проблема черного ящика в искусственном интеллекте. Необходимо не просто выдать результат быстро и правильно, но еще

логически обосновать его. Поэтому второй характеристикой для всех этих процессов служит объяснительная составляющая.

Проблема 2. *Проблема черного ящика в ИИ. [18]*

Для решения этой проблемы в ИИ было представлено направление объяснительного искусственного интеллекта (далее XAI). Интеллектуальные системы на основе XAI не просто выдают результат, но еще и объясняют человекопонятным языком как этот результат получен. Для этих целей как нельзя лучше подходят высокоуровневые языки программирования в которых видна логика исполнения программы. Особое место среди этих языков занимают логические языки программирования базирующиеся на основных конструкциях математической логики, таких как формулы и термы.

Проблема 3. *Проблема p-полноты для языков программирования. [36]*

Большинство высокоуровневых языков программирования (Паскаль, Си++) [15], в том числе и логических языков программирования (PROLOG [34], Libretto [13] и т.д.) являются Тьюринг полными [30]. Другая часть языков, наоборот, являются очень сильно урезанными и программы на них выполняются либо за линейное время, либо за полиномиальное время, где степень полинома крайне низкая. Основная проблема p-полноты - это создать язык программирования такой, что любая программа, реализованная в этом языке, будет исполняться за полиномиальное время от длины входных данных. И в то же время любой полиномиальный алгоритм может быть реализован при помощи некоторой программы этого языка.

Концепция семантического программирования

В 70-х - 80-х годах прошлого столетия академиками Ю.Л.Ершовым и С.С.Гончаровым, а также профессором Д.И.Свириденко была разработана концепция семантического программирования. Основная идея заключалась в разработке логического языка программирования с использованием основных конструкций математической логики, таких как формулы и термы. В качестве исполняющего устройства выступала наследственно-конечная списочная надстройка $HW(\mathfrak{M})$ сигнатуры σ [22; 23; 25], в которой все функции и предикаты

вычислимы. А в качестве Σ-программ выступали синтаксические конструкции задаваемые с помощью Σ-определений вида [8]:

$$\underline{def}\ Q(\overline{x}) :\ \Phi(\overline{x},Q,F)$$

а также вида

$$\underline{def}\ F(x_1,\ldots,x_{n-1}) = x_n :\ \Phi(\overline{x},Q,F)$$

где Q и F - наборы позитивно входящих в Φ предикатных и функциональных переменных. При этом в последнем случае формула Φ определяет функциональное равенство. Но построенный логический язык программирования в рамках концепции семантического программирования являлся Тьюринг полным и не решал проблему p-полноты.

В рамках семантического программирования уже реализованы несколько языков программирования таких как: d0sl [20], sDSL [27], Libretto [13] и язык документных моделей [12;14]. Но основная проблема, что ни для одного из них не доказана p-полнота. Более того, Libretto вообще является Тьюринг полным языком. Поэтому стояла задача, как в рамках концепции семантического программирования найти такие ограничения, чтобы получить язык со свойством p-полноты.

Проблема 4. *Как в рамках семантического программирования увеличить выразительную силу языка и найти подходящие синтаксические конструкции, при которых вычислительная сложность логических программ не выходила бы за рамки полиномиальной и при этом любой алгоритм полиномиальной сложности мог бы быть реализован подходящей программой в этом языке?*

Для ответа на первую часть вопроса такие синтаксические конструкции были найдены Гончаровым. Он предложил рассматривать только формулы с ограниченными кванторами определенного вида (Δ_0-формулы), а также ввел понятие условного терма [6].

Далее последовал ряд работ в которых были введены другие типы термальных расширений: рекурсивные термы, p-рекурсивные термы, ограниченные Δ_0-термы и т.д. [7;24;26]. Кроме рекурсивных термов, термальные расширения с помощью остальных перечисленных термов не выводят за рамки полиномиальности [7]. Более того, понятие логической программы также изменилось, если раньше ключевым элементом логической программы была Σ-формула

(Δ_0-формула), то теперь под программой стал пониматься некоторый специальный терм. Понятие специального терма индуктивно задается с помощью термального расширения стандартных термов условными термами, p-рекурсивными термами и другими видами термов.

В работе [31] было показано, что сложность проверки истинности любой Δ_0-формулы и сложность вычисления любого терма является полиномиальной в p-вычислимой наследственно-конечной списочной надстройке $HW(\mathfrak{M})$ сигнатуры σ. Так как понятие терма пришлось индуктивно расширить, то автоматически расширилось и понятие Δ_0-формулы. Уже при этих термальных расширениях возник вопрос, а все ли полиномиальные алгоритмы можно реализовать с помощью этих программ? Этот вопрос оставался открытым несколько лет.

Только в 2021 году в нашей совместной работе [36] с Гончаровым удалось построить необходимую синтаксическую конструкцию p-итерационного терма и термально расширить существующий язык L_1. В полученном языке L вычислительная сложность [29; 32] каждой логической программы является полиномиальной и, более того, для любого полиномиального алгоритма существует подходящая L-программа реализующая его. Это удалось достичь, благодаря моделированию работы машины Тьюринга на первых $h(|x|)$ тактах с помощью p-итерационного терма, где $h(|x|)$ – некоторый многочлен от длины входных данных.

Еще одним важным шагом было понимание того, что с помощью списочных элементов $HW(M)$ можно кодировать индуктивно задаваемые объекты любой природы. Выделять же эти элементы будут некоторые новые одноместные предикаты. При этом новые элементы могут индуктивно формульно определяться через базовые элементы основного множества первоначальной модели. Если перенестись в плоскость высокоуровневых языков программирования, то индуктивно определяемыми элементами через базовые элементы 0 и 1, могут выступать натуральные числа, наследственно-конечные списки, массивы, матрицы, слова, предложения и т.д. И для того, чтобы правильно определить такие конструкции и не выйти за рамки полиномиальной сложности нам нужны обоснованные математические оценки.

Одним из важнейших результатов в теории Σ-определимости [10] является классическая теорема Ганди [10; 17], в которой по любой Σ-формуле, индуктивно расширяющей предикат, строится специальный оператор, наименьшая неподвижная точка которого Σ-определима. Если мы хотим применить этот

результат в семантическом программировании, то здесь возникает вопрос, как для новых типов элементов, индуктивно задаваемых через базовые элементы основного множества модели, применить классическую теорему Ганди?

На все вышепоставленные вопросы в этой работе даются ответы. Более того, были исследованы вопросы связанные с существованием полиномиально вычислимых представлений для множества выводов в порождающих грамматиках [16], которые играют ключевую роль в построении синтаксиса различных лингвистических языков, а также для построения языков программирования таких как Pascal, C++, PHP, Solidity, Java и т.д. Очень важно, чтобы существовали быстрые алгоритмы проверки, является ли некоторый программный код программой в соответствующем языке и какова вычислительная сложность этой проверки.

Далее, встал вопрос о сложности распознавания некоторых синтаксических объектов из логики предикатов первого порядка. К этим объектам относятся термы, формулы, линейные доказательства и доказательства в виде дерева из логики предикатов первого порядка. Более того, встал вопрос о существовании полиномиально вычислимых представлений для L-программ и L-формул построенного нами языка L.

В диссертации получены результаты, которые дают ответы на вышепоставленные вопросы. Более того, исследован достаточно большой класс множеств математических объектов, для которых существуют полиномиально вычислимые списочные представления. Это позволяет использовать данные синтаксические конструкции в p-полных языках программирования без увеличения вычислительной сложности программ.

Особенно хотелось бы отметить результат о существовании полиномиально вычислимого списочного представления для множества доказательств. Данный результат позволяет внедрять проверку действий того или иного интеллектуального устройства в рамках заданных начальных понятий и логических правил. Корректность и человекопонятность рассуждений является важной составляющей для объяснительного искусственного интеллекта, а также для робототехники.

0.2 Основные результаты диссертации.

1. Решена проблема равенства классов P и L.
2. Построен полиномиальный аналог классической теоремы Ганди о наименьшей неподвижной точке с начальными p-вычислимыми условиями.
3. Получена серия результатов о существовании полиномиально вычислимых представлений:
   - для множества термов и множества формул ИП.
   - для множества L-программ и множества L-формул.
   - для множества доказательств ИП в виде дерева.
   - для множества линейных доказательств ИП.
   - для множества выводов в порождающих грамматиках.

Результаты пунктов 1 – 2 доказаны в соавторстве со своим научным руководителем С.С. Гончаровым [35; 36]. Результаты пункта 3 получены автором диссертации лично [37; 38].

Вклад соавтора в основной результат 1: в работе [24] Гончаровым и Свириденко были введены p-рекурсивные термы, а также, немного ранее, Гончаровым было введено понятие условного терма в [6]. Термальные расширения первоначального логического языка этими конструкциями не выводили за рамки полиномиальности. Далее, Гончаровым была поставлена проблема p-полноты полученного языка. Это решение было найдено Гончаровым и Нечесовым (автором диссертации) в работе [36]. Гончаровым было предложено построить терм прямо по машине Тьюринга, вычисляющей некоторую полиномиально вычислимую функцию. Нечесовым была найдена конструкция p-итерационного терма, являющаяся специальным случаем ограниченной рекурсии, термальное расширение с помощью которой оказалось достаточным для реализации всех полиномиально вычислимых функций. Нечесовым также было доказано, что данное термальное расширение L не выводит за рамки полиномиальности. А в силу того, что любая полиномиально вычислимая функция реализуется с помощью подходящей L-программы, то отсюда автоматически вытекает равенство классов P и L.

Вклад соавтора в основной результат 2: Нечесовым было предложено построить полиномиальный аналог классической теоремы Ганди, который бы мог единообразно применяться к индуктивно задаваемым конструкциям. Данные

синтаксические конструкции появлялись в работах Гончарова и его соавторов [6;7;24;26]. Нечесовым были построены полиномиально вычислимые порождающие семейства формул, но доказательство в общем случае не проходило. В результате совместных обсуждений было выработано более точное построение локального шага оператора замыкания, которое и позволило Нечесову получить окончательное доказательство полиномиально аналога теоремы Ганди о наименьшей неподвижной точке.

0.3 Новизна и научная значимость.

В работе все полученные результаты новые. Работа носит теоретико-практический характер. Результаты работы могут быть использованы как для дальнейших исследований, так и для реализации методологии построения программ в высокоуровневых языках, базирующейся на теории семантического программирования.

Построенный в работе логический язык программирования L является первым p-полным высокоуровневым языком. До этого момента либо высокоуровневые языки программирования были Тьюринг полными, либо имели сильные ограничения, что не позволяло реализовать ряд алгоритмов полиномиальной вычислительной сложности. Новый же язык логического программирования L помимо того, что является p-полным, также имеет сильные выразительные свойства, что позволяет использовать его в алгоритмах объяснительного искусственного интеллекта. Более того, в качестве базовых библиотек могут быть использованы модули полиномиальной вычислительной сложности реализующие работу полносвязных нейронных сетей прямого распространения с полиномиально вычислимыми функциями активации, модули реализующие построение элементов с помощью порождающих грамматик и их производных.

Доказанный в работе полиномиальный аналог теоремы Ганди дал новый способ доказательства полиномиальной вычислимости для индуктивно задаваемых синтаксических конструкций с помощью построения подходящих p-вычислимых GNF-систем. Как оказалось, в процессе изучения этого направления, свойство индуктивного задания через вновь определяемые элементы присуще очень многим множествам объектов. Наиболее важными из них являются спис-

ки, массивы, компьютерные программы, логические формулы и термы, доказательства в исчислении предикатов первого порядка и т.д. Поэтому применение данной теоремы может сильно облегчить понимание того какие индуктивные конструкции следует использовать, а какие нет.

Еще одним важным результатом является разработка нового метода, основанного на использовании PAG-теоремы, p-итерационных термов и свойств L-программ и L-формул. Данный метод позволяет показывать полиномиальную вычислительную сложность множеств и алгебраических структур не прибегая к построению машин Тьюринга и подсчету числа тактов работы. Для этого достаточно построить подходящую L-программу или подходящую L-формулу.

0.4 Методы исследования.

В работе используются классические методы теории моделей, теории алгоритмов, базовые методы семантического программирования, методы формульной определимости и теории сложности.

0.5 Апробация работы.

Результаты диссертации были представлены на конференциях:

- международная конференция «Мальцевские чтения» Новосибирск, 2019, 2022
- международная конференция: «IEEE International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)», 2022
- «Global Summit on Robotics and Artificial Intelligence», Prague, 2022
- международная школа-семинар «Синтаксис и семантика логических систем». Владивосток, 2022
- международная конференция «Актуальные задачи математики, механики и информатики», Караганда, 2022

- международная конференция «Current State and Development Perspectives of Digital Technologies and Artificial Intelligence», Самарканд, 2022
- международная конференция «Mathematical Logic and Computer Science», Астана, 2022

Также результаты работы докладывались на семинаре «Теория вычислимости» в 2021 году в Институте математики им. Соболева СО РАН и в 2022 году на математическом семинаре в Иркутском Государственном Университете.

0.6 Публикации.

Основные результаты автора по теме диссертации опубликованы в 4-х работах [35–38] из журналов списка ВАК. Из них 2 работы [35; 36] опубликованы в соавторстве со своим научным руководителем С.С. Гончаровым и 2 работы [37; 38] опубликованы автором лично.

Работа автора [39] из журнала списка ВАК является англоязычной версией работы автора [37].

Более того, по тематике диссертации у автора есть 2 работы [40; 41] в сборниках конференций не из списка ВАК.

0.7 Объем и структура работы.

Диссертация состоит из введения, 4 глав и заключения. Полный объём диссертации составляет 80 страниц. Список литературы содержит 41 наименование.

Глава 1. Логический язык программирования

1.1 Предварительные сведения

Пусть Σ – некоторый конечный алфавит и A,$B \subseteq \Sigma^*$, где Σ^* – множество конечных слов над алфавитом Σ. Функция $f : A \to B$ – полиномиально вычислима [1–4], если существует детерминированная одноленточная или многоленточная машина Тьюринга T над алфавитом Σ реализующая эту функцию и числа C,$p \in N$ такие, что для любого $w \in A$ значение функции $f(w)$ вычисляется на T за не более чем $C \cdot |w|^p$ шагов. Будем говорить, что функция p-вычислима, если она полиномиально вычислима.

Понятие p-вычислимости распространяется также на множества, алгоритмы, предикаты [19; 35]. В том числе можно определить и p-вычислимые неподвижные точки, заданные как n-ки множеств вида $\Gamma_w = (Q_1, \ldots, Q_n)$. Подразумевается, что каждая проекция на j-ю координату $I_j(\Gamma_w)$ является p-вычислимым множеством. Модель $\mathfrak{M}$ конечной сигнатуры σ является p-вычислимой, если p-вычислимыми являются основное множество модели, а также все операции и отношения.

Под полиномиально вычислимым представлением множества будем понимать такую инъективную кодировку в некотором алфавите Σ, что в этой кодировке сложность распознавания элементов этого множества является полиномиальной.

Под наследственно-конечной списочной надстройкой $HW(\mathfrak{M})$ сигнатуры σ будем понимать модель, получаемую из модели $\mathfrak{M}$ некоторой конечной сигнатуры σ_0, добавлением к основному множеству M множества наследственно-конечных списков $HW(M)$ индуктивно построенных из элементов множества M, а также операций и отношений для работы с новыми элементами. Также считаем, что в базовой модели $\mathfrak{M}$ по умолчанию уже есть стандартная модель арифметики $\mathfrak{N}$, а также элементы q и d, которыми мы будем кодировать различные множества переменных.

Введем ряд обозначений для новых сигнатурных символов сигнатуры σ:

1) сигнатурные символы для работы с натуральными числами:

$$\sigma_N = \{Number^{(1)}, \leqslant^{(2)}, +, -, \times, 0, 1, s^{(1)}\}$$

где $Number$ – предикат, выделяющий натуральные числа, $\leqslant$ – стандартное отношение меньше или равно, константы 0 и 1, s – стандартная функция следования. Введем также обозначение $\underline{n}$ для $s^n(0)$, где функция s последовательно применяется n раз.

2) сигнатурные символы для работы со списками [35]:

$$\sigma_W = \{tail^{(1)}, cons^{(2)}, head^{(1)}, conc^{(2)}, nil, \in^{(2)}, \subseteq^{(2)}, first^{(1)}, second^{(1)}\} \cup \\ \cup \{NumElements^{(1)}, getElement^{(2)}, changeListElements^{(3)}\}$$

где $tail^{(1)}$ – операция, которая выдает список без последнего элемента, $cons^{(2)}$ – операция, которая в первый список добавляет в качестве последнего элемента второй, $head^{(1)}$ – выдает последний элемент списка, $conc^{(2)}$ – конкатенация двух списков, nil – константа выделяющая пустой список, $\in^{(2)}$ – бинарное отношение быть элементом списка, $\subseteq^{(2)}$ – бинарное отношение быть начальным сегментом списка, $first^{(1)}$ и $second^{(1)}$ выдают первый и второй элемент списка соответственно, $NumElements^{(1)}$ – количество элементов в списке, $getElement^{(2)}$ – выдает i-й элемент списка, $changeListElements^{(3)}$ - меняет местами два элемента списка стоящих на i-и и j-м месте.

3) вспомогательные операции и константные символы:

$$\sigma_S = \{ \underline{false}, |\ |^{(1)}, listStr^{(1)}, \cdot\}$$

где $\underline{false}$ – константа, выделяющая $false$ элемент (добавим этот элемент в множество M), $|\ |$ – операция длины строки, $listStr$ - переводит список w вида $< a_1, \ldots, a_k >$, где все $a_i \in M$ в слово $a_1 \ldots a_k$, $\cdot$ – операция конкатенации строк.

4) в некоторых главах данной работы будет требоваться закодировать формулы и термы исчисления предикатов первого порядка над некоторой конечной сигнатурой σ^{ext} с помощью списков из основного множества модели $HW(\mathfrak{M})$ сигнатуры σ. Поэтому для таких случаев добавим множество константных символов σ_{ext}:

$$\sigma_{ext} = \{\underline{f_1}, \ldots, \underline{f_n}, \underline{P_1}, \ldots, \underline{P_k}, \underline{c_1}, \ldots, \underline{c_h}\}$$

где:

$$\sigma^{ext} = \{f_1, \ldots, f_n, P_1, \ldots, P_k, c_1, \ldots, c_h\}$$

5) также для списочного представления различных формул и термов сигнатуры σ^{ext} добавим также множество константных символов для логических

связок, кванторов и символа d для выделения переменных в формулах сигнатуры σ_{ext}:

$$\sigma_l = \{\underline{\&}, \underline{\vee}, \underline{\rightarrow}, \underline{\neg}, \underline{\forall}, \underline{\exists}, \underline{(}, \underline{)}, d\} \cup \{,\}$$

6) множество константных символов для ограниченных кванторов, условных и p-итерационных термов и символ q для выделения переменных в L-программах и L-формулах:

$$\sigma_L = \{\underline{\exists \in}, \underline{\forall \in}, \underline{\exists \subseteq}, \underline{\forall \subseteq}, \underline{\exists \leqslant}, \underline{\forall \leqslant}, \underline{\leqslant}, \underline{Cond}, \underline{Iteration}, q\}$$

Без ограничения общности считаем, что наборы с 1) по 3) сигнатурных символов включены в σ по умолчанию. Наборы с 4) по 6) добавляются в случае, если мы работаем с синтаксическими объектами некоторой фиксированной внешней сигнатуры σ^{ext}.

Под логическим языком семантического программирования будем понимать формальный язык над фиксированным алфавитом, предназначенный для записи логических программ и формул. В общем, будем подразумевать, что данный язык определяет набор лексических, синтаксических и логических правил, определяющих непосредственно саму программу и порядок ее исполнения. В качестве исполняющего устройства будет выступать p-вычислимая наследственно-конечная списочная надстройка $HW(\mathfrak{M})$ сигнатуры σ. Под логической программой, если это не оговорено особо, мы будем подразумевать некоторый терм. Так как, в дальнейшем, понятие терма и формулы будет задаваться в некоторых термальных расширениях, то будет изменяться и понятие программы. Процессом же исполнения программы будет являться процесс вычисления терма. Если у нас уже есть некоторый логический язык L и в нем формализовано понятие терма и формулы, то будем называть любой терм из этого языка L-программой, а любую формулу L-формулой.

Так как основной интерес в этой работе представляют алгоритмы полиномиальной сложности, то основной задачей этой главы является построение L-программ и L-формул обладающих этим свойством. Первоначально, под языком L_0 будем подразумевать логический язык над p-вычислимой наследственно-конечной списочной надстройкой $HW(\mathfrak{M})$ сигнатуры σ, в котором L_0-программами являются стандартные термы исчисления предикатов первого порядка. А L_0-формулами – Δ_0-формулы с ограниченными кванторами вида $\exists x \in y$, $\forall x \in y$, $\exists x \subseteq y$, $\forall x \subseteq y$, $\exists x \leqslant y$, $\forall x \leqslant y$ [8]. Так определенные L_0-программы и L_0-формулы обладают свойством p-вычислимости [31]. Но у

них есть один недостаток, данным конструкциям не хватает выразительной силы. И не понятно, насколько богатым является этот язык, все ли алгоритмы полиномиальной сложности можно реализовать на нем?

Для этих целей, в следующих главах будут представлены различные виды термальных расширений (с помощью условных и р-итерационных термов) языка L_0. При таком расширении новый язык L становится достаточно богатым, обладает мощными выразительными свойствами и любая L-программа является р-вычислимой. Но самое важное, как будет показано далее, в этом языке можно реализовать любой алгоритм полиномиальной сложности.

1.2 Условные термы

В этом разделе будет впервые расширено понятие терма с помощью индуктивно определимой конструкции условного терма [6]. Это естественная конструкция является аналогом «if then else» из высокоуровневых языков программирования.

В качестве основной модели выступает р-вычислимая наследственно-конечная списочная надстройка $HW(\mathfrak{M})$ сигнатуры σ. Пусть t_0,t_1,...,t_{n+1} - L_0-программы и $\varphi_0,\ldots,\varphi_n$ - L_0-формулы [7]. Тогда взаимной индукцией можно определить концепцию L_1-программы, условного терма и L_1-формулы следующим образом:

Базис индукции: Любая L_0-программа является L_1-программой, любая L_0-формула является L_1-формулой.

Шаг индукции: Если t_0,...,t_{n+1} - L_1-программы и $\varphi_0,\ldots,\varphi_n$ - L_1-формулы, то следующая синтаксическая конструкция будет определять понятие условного терма $t(v)$ (или $Cond(t_0,\ldots,t_{n+1},\varphi_0,\ldots,\varphi_n)$) следующим образом:

$$t(\overline{v}) = \begin{cases} t_0(\overline{v}), \text{ если } HW(\mathfrak{M}) \models \varphi_0(\overline{v}) \\ t_1(\overline{v}), \text{ если } HW(\mathfrak{M}) \models \varphi_1(\overline{v})\&\neg\varphi_0(\overline{v}) \\ \ldots \\ t_n(\overline{v}), \text{ если } HW(\mathfrak{M}) \models \varphi_n(\overline{v})\&\neg\varphi_0(\overline{v})\&\ldots\&\neg\varphi_{n-1}(\overline{v}) \\ t_{n+1}(\overline{v}), \text{ иначе} \end{cases} \tag{1.1}$$

которая в свою очередь является и L_1-программой.

Если $t_1, \ldots, t_n$ – L_1-программы, а F – n-местный функциональный символ сигнатуры σ, то выражение $F(t_1, \ldots, t_n)$ является L_1-программой и других способов построения L_1-программ и условных термов нет.

L_1-формулы на индуктивном шаге:
1) Если t и q – L_1-программы, то $t = q$ – L_1-формула.
2) Если $t_1, \ldots, t_n$ – L_1-программы, а P – n-местный предикатный символ сигнатуры σ, то $P(t_1, \ldots, t_n)$ – L_1-формула.
3) Если Φ и Ψ – L_1-формулы, тогда $\Phi\&\Psi$, $\Phi \vee \Psi$ и $\Phi \to \Psi$ – L_1-формулы.
4) Если Φ – L_1-формула, то $\neg\Phi$ – L_1-формула.
5) Если Φ – L_1-формула и $t(\overline{x})$ - L_1-программа, то $\exists y \in t(\overline{x})\Phi$, $\forall y \in t(\overline{x})\Phi$, $\exists y \subseteq t(\overline{x})\Phi$, $\forall y \subseteq t(\overline{x})\Phi$, $\exists y \leqslant t(\overline{x})\Phi$, $\forall y \leqslant t(\overline{x})\Phi$ – L_1-формулы.
Других L_1-формул нет.

С точностью до переобозначений, введенных в данной работе, можно сформулировать следующее утверждение.

Утверждение 1.1. *(Оспичев, Пономарев) [31] Вычислительная сложность любой L_1 - программы и сложность проверки истинности любой L_1-формулы в p-вычислимой наследственно-конечной списочной надстройке $HW(\mathfrak{M})$ конечной сигнатуры σ является полиномиальной.*

Теорема 1.2. *(Гончаров) [6] Существует алгоритм, строящий по любой L_1-формуле φ такую L_0-формулу ψ, что:*

$$HW(\mathfrak{M}) \models \forall \overline{v}\ \varphi(\overline{v}) \Leftrightarrow \psi(\overline{v})$$

По сути, теорема 1.2 говорит о том что данное термально-формульное расширение является консервативным. Но при этом логический язык становится более выразительным. Более того, сохраняются и полиномиальные свойства проверки истинности любой L_1-формулы в модели $HW(\mathfrak{M})$.

1.3 P-итерационные термы

Пусть $HW(\mathfrak{M})$ – p-вычислимая модель конечной сигнатуры σ. Пусть L_1–логический язык, определенный в 1.2. В этом языке все L_1-программы полино-

миально вычислимы и сложность проверки истинности L_1-формул полиномиальна [31].

Индуктивно расширим язык L_1 до языка L следующим образом:

Определим индуктивно концепцию L-программы и p-итерационного терма $Iteration_{g,\varphi}$, а также концепцию L-формулы следующим образом:

Базис индукции: Любая L_1-программа является L-программой, любая L_1-формула является L-формулой.

Шаг индукции: Пусть $g(x)$ – L-программа, $\varphi(x)$ – L-формула. Причем существует константа C такая, что для любого x из области определения g выполнено:

$$|g(x)| \leqslant |x| + C \tag{1.2}$$

Введем обозначение: $g^i(x)$ означает i раз примененная функция g, т.е. $g^i(x) = g(g^{i-1}(x))$, при этом $g^0(x) = x$.

Тогда p-итерационным термом будем называть следующую синтаксическую конструкцию:

$$Iteration_{g,\varphi}(w,n) = \begin{cases} g^i(w), \text{ если существует} \\ \quad i \leqslant n \ HW(\mathfrak{M}) \models \varphi(g^i(w)) \\ \quad \text{и } \forall j < i \ HW(\mathfrak{M}) \not\models \varphi(g^j(w)) \\ false, \text{ иначе} \end{cases} \tag{1.3}$$

Если t_1,t_2 – L-программы, тогда:

$$Iteration_{g,\varphi}(t_1,t_2)$$

является L-программой.

Если $t_1,\ldots,t_{n+1}$ - L-программы, а $\varphi_1,\ldots,\varphi_n$ – L-формулы, тогда:

$$Cond(t_1,\varphi_1,\ldots,t_n,\varphi_n,t_{n+1})$$

также является L-программой.

Если $t_1,\ldots,t_n$ - L-программы, а F – n-местный функциональный символ сигнатуры σ, то выражение $F(t_1,\ldots,t_n)$ является L-программой и других способов построения L-программ и p-итерационных термов нет.

L-формулы на индуктивном шаге:

1) Если t и q – L-программы, то $t = q$ – L-формула.

2) Если $t_1,\ldots,t_n$ – L-программы, а P – n-местный предикатный символ сигнатуры σ, то $P(t_1,\ldots,t_n)$ – L-формула.

3) Если Φ и Ψ – L-формулы, тогда $\Phi\&\Psi$, $\Phi \vee \Psi$ и $\Phi \to \Psi$ – L-формулы.
4) Если Φ – L-формула, то $\neg\Phi$ – L-формула.
5) Если Φ – L-формула и $t(\overline{x})$ - L-программа, то $\exists y \in t(\overline{x})\Phi$, $\forall y \in t(\overline{x})\Phi$, $\exists y \subseteq t(\overline{x})\Phi$, $\forall y \subseteq t(\overline{x})\Phi$, $\exists y \leqslant t(\overline{x})\Phi$, $\forall y \leqslant t(\overline{x})\Phi$ – L-формулы.
Других L-формул нет.

Взаимной индукцией определим понятие сложности для L-программ и L-формул.

Сложность для L-программ:
1) $r(x) = 0$ и $r(c) = 0$, где x – переменная, c – константа сигнатуры σ.
2) $r(f(t_1,\ldots,t_k)) = \sum_{i=1}^{k} r(t_i) + 1$, где $t_1,\ldots,t_k$ – L-программы и $f \in \sigma$.
3) $r(Cond(t_1,\varphi_1,\ldots,t_{n+1})) = \sum_{i=1}^{n+1} r(t_i) + \sum_{i=1}^{n} r(\varphi_i) + 1$, где $t_1,\ldots,t_{n+1}$ – L-программы и $\varphi_1,\ldots,\varphi_n$ – L-формулы.
4) $r(Iteration_{g,\varphi}(t_1,t_2)) = r(g)+r(\varphi)+r(t_1)+r(t_k)+1$, где t_1 и t_2 – L-программы, g – L-программа удовлетворяющая условию (1.2), φ– L-формула.

Сложность для L-формул:
1) $r(P(t_1,\ldots,t_n)) = \sum_{i=1}^{n} r(t_i) + 1$
2) $r(t_1 = t_2) = r(t_1) + r(t_2) + 1$
3) $r(\Phi\&\Psi)) = r(\Phi) + r(\Psi) + 1$
4) $r(\Phi \vee \Psi)) = r(\Phi) + r(\Psi) + 1$
5) $r(\Phi \to \Psi)) = r(\Phi) + r(\Psi) + 1$
6) $r(\neg\Phi)) = r(\Phi) + 1$
7) $r(\exists x \delta t\Phi)) = r(t) + r(\Phi) + 1$, где $\delta \in \{\in, \subseteq, \leqslant\}$
8) $r(\forall x \delta t\Phi)) = r(t) + r(\Phi) + 1$, где $\delta \in \{\in, \subseteq, \leqslant\}$

Теорема 1.3. *В p-вычислимой наследственно-конечной списочной надстройке $HW(\mathfrak{M})$ сигнатуры σ:*
1) Вычислительная сложность любой L-программы является полиномиальной.
2) Вычислительная сложность проверки истинности любой L-формулы является полиномиальной.

Доказательство. Одновременной индукцией по сложности L-программ и L-формул докажем это утверждение:

Базис индукции: Если сложность L-программы t равна 0, значит t является либо переменной, либо константой. А следовательно имеет полиномиальную вычислительную сложность.

Если сложность L-формулы Φ равна 0, то множество таких формул пусто. Следовательно, предположение верно.

Шаг индукции: Пусть индукционное предположение верно для всех L-программ и L-формул у которых сложность не превосходит n, покажем это и для $n+1$.

Пусть t – L-программа и $r(t) = n+1$, рассмотрим следующие варианты:
1) t имеет вид $f(t_1,\ldots,t_n)$, где $f \in \sigma$. Тогда по индукционному предположению все t_i – p-вычислимы, а следовательно p-вычислима и L-программа $f(t_1,\ldots,t_n)$.
2) t имеет вид $Cond(t_1,\varphi_1,\ldots,t_{n+1})$, тогда из индукционного предположения для $t_1,\ldots,t_{n+1}$ и $\varphi_1,\ldots,\varphi_n$ вытекает p-вычислимость L-программы $Cond(t_1,\varphi_1,\ldots,t_{n+1})$.
3) t имеет вид $Iteration_{g,\varphi}(t_1,t_2)$. То в силу индукционного предположения для g, t_1, t_2, φ и условия на g (1.2) получаем, что для любого k верно следующее:

$$|g^{k+1}(w)| \leqslant |g^k(w)| + C \leqslant |w| + C\cdot(k+1) \tag{1.4}$$

учитывая то, что для любого $k \leqslant n$ выполнено неравенство:

$$|g^k(w)| \leqslant |w| + C\cdot n \leqslant C\cdot(|w|+n) \tag{1.5}$$

оценим сложность вычисления произвольного терма вида $Iteration_{g,\varphi}(w,n)$:

$$\begin{aligned} &t(Iteration_{g,\varphi}(w,n)) \leqslant \textstyle\sum_{i=1}^{n}(t(g(\delta_{i-1})) + t(\varphi(g(\delta_i)))) \leqslant \\ &\leqslant \textstyle\sum_{i=1}^{n}(C\cdot(C\cdot(|w|+n))^p + C\cdot(C\cdot(|w|+n))^p) \leqslant \\ &\leqslant 2\cdot n\cdot C^{p+1}\cdot(|w|+n)^p \leqslant C_1\cdot(|w|+n)^{p+1} \end{aligned} \tag{1.6}$$

где $\delta_i = g^i(w)$ и $g^0(w) = w$.

В силу того, что t_1 и t_2 – p-вычислимы, вытекает p-вычислимость и для L-программы вида $Iteration_{g,\varphi}(t_1,t_2)$.

Для L-формул вида $t_1 = t_2$, $\Phi\&\Psi$, $\Phi\vee\Psi$, $\Phi\to\Psi$, $\neg\Phi$, как и для L-программ все доказывается тривиально с помощью индуктивного предположения. Нетривиальный случай - это формулы вида $\exists x\delta t\Phi$ и $\forall x\delta t\Phi$, где $\delta \in \{\in, \subseteq, \leqslant\}$.

Не нарушая общности докажем, что сложность проверки истинности L-формулы вида $\exists y \in t(x)\Phi(x,y)$ полиномиальна, для остальных случаев все доказывается аналогично.

Из индукционного предположения имеем, что t и Φ – p-вычислимы. Получаем, что для любого $w \in HW(M)$ $|t(w)| \leqslant C\cdot|w|^p$. Поэтому, количество

элементов в списке $t(w)$ не превышает $C \cdot |w|^p$. Для каждого элемента списка $t(w)$ необходимо проверить истинность L-формулы Φ.

$$t(\exists y \in t(w)\Phi(w,y)) \leqslant \sum_{v \in t(w)} t(\Phi(w,v)) \leqslant C \cdot |w|^p \cdot C \cdot (|w| + C \cdot |w|^p)^p \leqslant C_1 \cdot |w|^{p^2+p}$$

□

1.4 Машины Тьюринга и машинные слова

Обозначим за T_f детерминируемую машину Тьюринга над конечным алфавитом Σ представляющую полиномиальную функцию f. Пусть $h(x)$ – многочлен мажорирующий число тактов этой машины Тьюринга до ее остановки, S множество символов $\{1,B\}$, Q – множество состояний машины Тьюринга, где q_1 – начальное состояние и q_0 конечное состояние. Обозначим за P_{T_f} программу машины тьюринга T_f реализующую функцию f. Поскольку любая программа машины Тьюринга записывается через конечное множество пятерок вида:

$$\sigma : Q \times S \to Q \times S \times \{R,L\} \tag{1.7}$$

то, не ограничивая общности, будем считать, что все такие элементы программы P_{T_f} представлены в виде списков вида $< q_{i_1},s_{j_1},q_{i_2},s_{j_2},\beta >$.

Набор символов находящихся на ленте, состояние машины Тьюринга T_f и ее положение головки однозначно определяются с помощью конфигурации. Так следующая последовательность символов:

$$c_i : \ s_{-m_i} \dots s_{-2}\ s_{-1}\ q_{k_i}\ s_0\ s_1 \dots s_{n_i} \tag{1.8}$$

будет называться конфигурацией машины Тьюринга на i-м шаге вычислений с помощью Тьюринговой программы.

Если конфигурация машины Тьюринга T_f меняется с c_i на c_{i+1} тогда:

$$|c_i| - 1 \leqslant |c_{i+1}| \leqslant |c_i| + 1 \tag{1.9}$$

Из (1.9) следует, что финальная конфигурация c_{final} машины Тьюринга T при работе над словом w имеет следующее ограничение на длину:

$$|c_{final}| \leqslant |c_0| + h(|w|) = d(|w|) \tag{1.10}$$

Используя вид конфигурацию c_i из (1.8) определим машинное слово w_{c_i} следующим образом:

$$w_{c_i} : << s_{-m_i}, \ldots, s_{-1} >, \underline{q_{k_i}}, < s_0, \ldots, s_{n_i} >> \tag{1.11}$$

где $\underline{q_{k_i}}$ представляет собой строку $q \ldots q$ длины $k_i + 1$, $s_j \in \{B,1\}$.

Определим $w_{c_i,k}$ как k-й элемент списка, представляющего машинное слово w_{c_i}, где $k \in \{1,2,3\}$

Если конфигурация c_i имеет форму $q_{k_i} s_0 \ldots s_{n_i}$, тогда машинное слово w_{c_i} имеет вид:

$$w_{c_i} : <<>, \underline{q_{k_i}}, < s_0, \ldots, s_{n_i} >> \tag{1.12}$$

Замечание 1.4.1. *1) Из неравенств* (1.9) *следует, что существует* $C \in N$ *такое, что:*

$$|w_{c_i}| - C \leqslant |w_{c_{i+1}}| \leqslant |w_{c_i}| + C \tag{1.13}$$

2) Из неравенств (1.10) *и* (1.13) *следует*

$$|w_{c_{final}}| \leqslant |w_{c_0}| + C \cdot h(|x|) \leqslant r'(|x|) \text{ для некоторого полинома } r'(|x|).$$

Определим новую бинарную операцию $\otimes$ на машинных словах вида w_{c_i} и списочных элементах $< \underline{q_1},s_1,\underline{q_2},s_2,\beta >$ из P_{T_f}:

Случай 1: элемент имеет вид $< \underline{q_1},s_1,\underline{q_2},s_2,R >$

$$w_{c_i} \otimes < \underline{q_1},s_1,\underline{q_2},s_2,R >= \begin{cases} w_{c_j}, & \text{если } head(tail(w_{c_i})) = \underline{q_1} \text{ и } first(head(w_{c_i})) = s_1 \\ false, & \text{иначе} \end{cases} \tag{1.14}$$

и $w_{c_j} =< w_{c_j,1},w_{c_j,2},w_{c_j,3} >$, где

$$w_{c_j,1} = cons(first(w_{c_i}),s_2);\ w_{c_j,2} = \underline{q_2};$$
$$w_{c_j,3} = tail\_l(head(w_{c_i}));$$

Случай 2: элемент имеет вид $< \underline{q_1},s_1,\underline{q_2},s_2,L >$

$$w_{c_i} \otimes < \underline{q_1},s_1,\underline{q_2},s_2,L >= \begin{cases} w_{c_j}, & \text{если } head(tail(w_{c_i})) = \underline{q_1} \text{ и } first(head(w_{c_i})) = s_1 \\ false, & \text{иначе} \end{cases} \tag{1.15}$$

и $w_{c_j} =< w_{c_j,1},w_{c_j,2},w_{c_j,3} >$, где

$$w_{c_j,1} = tail(first(w_{c_i}));\ w_{c_j,2} = \underline{q_2};$$
$$w_{c_j,3} = cons\_l(cons\_l(tail\_l(head(w_{c_i})),s_2),head(first(w_{c_i})));$$

Замечание 1.4.2. *Операция* $\otimes$ *- p-вычислима.*

1.5 Решение проблемы P=L

Введем обозначение $\underline{q_i}$ – для строки символов q длины i. Тогда для каждой пятерки вида $< q_{i_1},s_{j_1},q_{i_2},s_{j_2},\beta >$ из программы детерминированной машины Тьюринга P_{T_f} существует подходящая L-программа $v(\underline{q_{i_1}},s_{j_1},\underline{q_{i_2}},s_{j_2},\beta)$ следующего вида:

$$v(\underline{q_{i_1}},s_{j_1},\underline{q_{i_2}},s_{j_2},\beta) = cons(cons(cons(cons(cons(nil,\underline{q_{i_1}}),s_{j_1}),\underline{q_{i_2}}),s_{j_2}),\beta) \quad (1.16)$$

Теорема 1.4. *(Решение проблемы P=L)*
Пусть $HW(\mathfrak{M})$ *– p-вычислимая модель конечной сигнатуры* σ *тогда:*
1) Любая L-программа имеет полиномиальную вычислительную сложность.
2) Для любой полиномиально вычислимой функции существует подходящая L-программа реализующая ее.

Доказательство. 1) результат автоматически следует из теоремы 1.3
2) Пусть $f(x)$ – произвольная p-вычислимая функция, вычислительная сложность которой ограничена некоторым полиномом $h(|x|)$. Рассмотрим машину Тьюринга T_f над алфавитом Σ с программой P_{T_f} реализующую ее. И пусть l_f – список всех пятерок вида $v(\underline{q_{i_1}},s_{j_1},\underline{q_{i_2}},s_{j_2},\beta)$ (1.16), где $(q_{i_1},s_{j_1},q_{i_2},s_{j_2},\beta) \in P_{T_f}$ и $\beta \in \{R,L\}$.

Определим L-формулу φ следующим образом:

$$\varphi(\mathbf{x}) :\ Final(\mathbf{x}) \quad (1.17)$$

где предикат $Final(w)$ истинен, если w является машинным словом вида:

$$<< s_{-m},\ldots,s_{-1} >,\underline{q_0},< s_1,\ldots,s_k >>$$

Заметим, что предикат $Final$ – p-вычислим.

Введем обозначение $tail^k(\mathbf{x})$ для операции $tail$ примененной k раз к $\mathbf{x}$. Тогда L-программу $g(\mathbf{x})$ для p-итерационного терма можно задать с помощью условного терма [7] следующим способом:

$$g(\mathbf{x}) = \begin{cases} \mathbf{x} \otimes l_1, \text{ где } (l_1 = first(l_f))\&(\mathbf{x} \otimes l_1 \neq false) \\ \dots \\ \mathbf{x} \otimes l_i, \text{ где } (l_i = head(tail^{k-i}(l_f)))\&(\mathbf{x} \otimes l_i \neq false) \\ \dots \\ \mathbf{x} \otimes l_k, \text{ где } (l_k = head(l_f))\&(\mathbf{x} \otimes l_k \neq false) \\ \mathbf{x}, \text{ иначе} \end{cases} \tag{1.18}$$

Определим L-программу $mw(x)$, которая по любому слову $w \in \Sigma^*$ строит машинное представление вида $<<> ,\underline{q_1},w^* >$ следующим образом:

$$mw(x) : \ cons(cons(cons(nil,nil),\underline{q_1}),strList(x)) \tag{1.19}$$

Заметим, что функция $mw(w)$ переводит слово w в машинное слово, соответствующее начальной конфигурации c_0 машины Тьюринга T_f.

Определим p-вычислимую функцию $value(x)$, которая по машинному слову строит соответствующее слово на ленте машины Тьюринга T_f следующим образом:

$$value(w) : \ listStr(conc(first(w),head(w))) \tag{1.20}$$

С помощью p-итерационного терма $Iteration_{g,n}$ с заданными g (1.18) и φ (1.17), который удовлетворяет условиям теоремы 1.3, можно построить следующую L-программу реализующую функцию $f(x)$:

$$value(t(mw(\mathbf{x}),h(|\mathbf{x}|))) \tag{1.21}$$

□

1.6 Модификация p-итерационных термов

В параграфе 1.5 было показано, что термальное расширение с помощью p-итерационных термов не выводит за рамки полиномиальности и более того, любой полиномиальный алгоритм реализуется подходящей L-программой.

Пусть у нас задан p-итерационный терм:

$$Iteration_{g,\varphi}(w,n) = \begin{cases} g^i(w), \text{ если существует} \\ \quad i \leqslant n\ HW(\mathfrak{M}) \models \varphi(g^i(w)) \\ \quad \text{и } \forall j < i\ HW(\mathfrak{M}) \not\models \varphi(g^j(w)) \\ false, \text{ иначе} \end{cases}$$

Переопределим для него условия на g и φ.

Пусть для фиксированного $n \in N$ L-программа $g(x)$ задается следующим образом:

$$g(w) = \begin{cases} w^*, \text{ если } w =< w_1, \ldots, w_n > \\ false, \text{ иначе} \end{cases} \tag{1.22}$$

где $w^* =< w_1^*, \ldots, w_n^* >$ и при этом выполнены следующие условия (с точностью до перестановки):
1) $|w_1^*| \leqslant |w_1| + C \cdot \sum_{i=2}^{n} |w_i|^p$
2) $|w_i^*| \leqslant |w_i|$, для любого $i \in [2, \ldots, n]$

Утверждение 1.5. *Терм $Iteration_{g,\varphi}$ из (1.3) с L-программой g (1.22), удовлетворяющей условиям 1) и 2), является p-вычислимым.*

Доказательство. Посчитаем вычислительную сложность для $Iteration_{g,\varphi}(w,m)$. Из условий 1)-2) получаем, что для любого k верно следующее:

$$\begin{aligned} &|first(g^{k+1}(w))| \leqslant |first(g^k(w))| + C \cdot \textstyle\sum_{i=2}^{n} |getElement(g^k(w),i)|^p \leqslant \\ &\leqslant |first(g^k(w))| + C \cdot \textstyle\sum_{i=2}^{n} |w_i|^p \leqslant \\ &\leqslant |w_1| + C \cdot (k+1) \cdot \textstyle\sum_{i=2}^{n} |w_i|^p \leqslant C \cdot (k+1) \cdot |w|^p \end{aligned} \tag{1.23}$$

и при $k+1 \leqslant m$ получаем, что:

$$\begin{aligned} &|g^{k+1}(w)| \leqslant |first(g^{k+1}(w))| + \textstyle\sum_{i=2}^{n} |getElement(g^{k+1}(w),i)| + n + 2 \leqslant \\ &\leqslant C \cdot (k+1) \cdot |w|^p + \textstyle\sum_{i=2}^{n} |w_i| + n + 2 \leqslant \\ &\leqslant C \cdot m \cdot |w|^p + |w| \leqslant C \cdot (m+1) \cdot |w|^p \end{aligned} \tag{1.24}$$

где $n+2$ - это количество запятых и треугольных скобок в списке w

Учитывая оценку (1.24) посчитаем сложность нашего p-итерационного терма вида $Iteration_{g,\varphi}(w,m)$:

$$\begin{aligned} & t(Iteration_{g,\varphi}(w,m)) \leqslant \textstyle\sum_{i=1}^{m}(t(g(\delta_{i-1})) + t(\varphi(\delta_i))) \leqslant \\ & \leqslant \textstyle\sum_{i=1}^{m}(C \cdot (C \cdot (m+1) \cdot |w|^p)^p + C \cdot (C \cdot (m+1) \cdot |w|^p)^p) \leqslant \\ & \leqslant 2m \cdot C \cdot (C \cdot (m+1) \cdot |w|^p)^p \leqslant C_1 \cdot (m+1)^{p+1} \cdot |w|^{p^2} \leqslant \\ & \leqslant C_1 \cdot (m + |w|)^{p^2+p+1} \end{aligned} \tag{1.25}$$

где $\delta_i = g^i(w)$ и $g^0(w) = w$.

□

Глава 2. GNF-системы и PAG-теорема

2.1 Классическая теорема Ганди

Вся вторая глава посвящена решению вопросов сложности для индуктивно задаваемых множеств и объектов. В этой главе ключевым результатом является построение полиномиального аналога для классической теоремы Ганди. Сама же классическая теорема Ганди [10] представлена следующим образом.

Пусть $\Omega =< w,0,s,+,\cdot,\leqslant >$ – алгебраическая система сигнатуры $\sigma_0 =< 0,s^1,+^2,\cdot^2,\leqslant^2>$. Обозначим через σ^* расширение сигнатуры σ_0, полученное добавлением символов для всех Σ-функций на Ω и константных символов для всех элементов w, а через Ω^* – соответствующее обогащение Ω.

Определим оператор $\Gamma^{\Omega^*}_{\Phi[\overline{x}]}$ следующим образом:

$$\Gamma^{\Omega^*}_{\Phi[\overline{x}]}(Q) = \{< a_0,\ldots,a_{k-1} > \mid\ < \Omega^*,Q >\models \Phi(a_0,\ldots,a_{k-1})\}$$

где $Q \subseteq w^k$.

С оператором $\Gamma^{\Omega^*}_{\Phi[\overline{x}]}$ свяжем последовательность:

$$\Gamma_0,\ \Gamma_1,\ldots,\Gamma_\alpha,\ldots,\ \text{где } \alpha \text{ – ординал}$$

k-местных предикатов на $|\Omega^*|$ следующим образом:

$$\Gamma_0 = \varnothing,\ \Gamma_{\alpha+1} = \Gamma^{\Omega^*}_{\Phi[\overline{x}]},\ \Gamma_\alpha = \cup_{\beta<\alpha}\Gamma_\beta$$

Пусть α – наименьший ординал такой, что $\Gamma_{\alpha+1} = \Gamma_\alpha$, тогда $\Gamma_* = \Gamma_\alpha$ – наименьшая неподвижная точка.

Теорема 2.1. *(классическая теорема Ганди) [10]*
Пусть $\Phi(P^+)$ – Σ-формула сигнатуры $\sigma^* \cup < P^k >$, в которую предикатный символ P входит позитивно, и пусть $\overline{x} = x_0,\ldots,x_{k-1}$ – список попарно различных переменных такой, что $FV(\Phi) \subseteq \{\overline{x}\}$. Тогда наименьшая неподвижная точка $\Gamma_* \subseteq w^k$ оператора $\Gamma^{\Omega^*}_{\Phi[x]}$ является Σ-предикатом на Ω^*.

2.2 Предварительные сведения

Пусть Σ_0 некоторый конечный алфавит и $\Sigma = \Sigma_0 \cup \{<\ ,\ >\} \cup \{,\}$ - новый алфавит, полученный добавлением треугольных скобок и запятой.

Расщеплением слова будем называть следующую частичную функцию:

$$R : \Sigma^* \to (\Sigma \cup \{\#\})^*$$

такую, что:

$$R(w) = \begin{cases} w_1\#...\#w_n, \text{ где } w =< w_1, \ldots, w_n > \text{ и } w_i \text{ удовлетворяет (1) или (2)} \\ false, \text{ иначе} \end{cases}$$

(1) $w_i \in \Sigma_0^*$

(2) w_i начинается с левой скобки и число левых скобок равно числу правых, причем для любого собственного начального подслова слова w_i число левых скобок больше числа правых.

Утверждение 2.2. *Расщепление любого слова - единственно.*

Доказательство. Докажем методом от противного. Пусть существуют два различных расщепления $R(w) = w_1\# \ldots \#w_n$ и $R(w) = l_1\# \ldots \#l_k$ такие что $w =< w_1, \ldots, w_n >$ и $w =< l_1, \ldots, l_k >$. Тогда, по определению, либо w_1 и $l_1 \in \Sigma_0^*$, либо w_1 и l_1 начинаются с левой скобки и число левых и правых скобок для каждого слова одинаково. В первом случае, w_1 и l_1 совпадают. Во втором случае, либо w_1 является подсловом l_1, либо l_1 является подсловом w_1. Из того что никакое начальное собственное подслово не может иметь равное количество левых и правых скобок, следует равенство этих слов. Аналогично показывается равенство и для других слов w_i и l_i. □

Утверждение 2.3. *Сложность расщепления $R(x)$ равна $O(|x|)$.*

Доказательство. Данное утверждение можно доказать с помощью построения машины Тьюринга T с 2-мя полулентами над алфавитом $\Sigma \cup \{1,B,\#\}$, где B – пустой символ. И пусть существует 2 конечных состояния: q_0 и q_1. Вычисленное значение будет находиться на 1-й ленте, если машина закончила свою работу в состоянии q_1 и $false$, если в состоянии q_0.

(1) На 1-й ленте будет выписано слово w.

(2) На 2-й ленте будет храниться разница между количеством левых и правых скобок в слове w.

Опишем алгоритм работы 2-х ленточной машины T:

(1) Если первый символ на 1-й ленте не является левой скобкой, тогда машина T останавливает свою работу в финальном состоянии q_1. Иначе, T заменяет этот символ на B и переходит к следующим шагам.

(2) Если второй символ в слове w из Σ_0, тогда T читает слово w, до тех пор пока не встретит символ не из Σ_0. Если этот символ не запятая или правая скобка, тогда машина T останавливает свою работу в состоянии q_1.

(3) Если второй символ не из Σ_0 и не является левой скобкой, тогда машина T останавливает свою работу в состоянии q_1.

(4) Если машина T считывает левую скобку на 1-й ленте в слове w, тогда T добавляет символ 1 на 2-ю ленту. Если же встречает правую скобку в слове w, тогда машина T стирает последний символ 1 со второй ленты.

(5) Если на 2-й ленте нет 1 и машина T считывает с 1-й ленты правую скобку, тогда T заменяет эту скобку на символ B и если после этой скобки не идет больше никаких символов в слове w, то останавливается в состоянии q_0, иначе в состоянии q_1.

(6) Если машина T встречает запятую на 1-й ленте и на 2-й ленте нет символов вида 1, тогда T заменяет эту запятую на # символ.

Вычислительная сложность $R(w)$:

(1) машина T посимвольно считывает слово w на 1-й ленте переодически заменяя запятые символов #, или заменяя скобки на символ B. Число таких шагов не превосходит $|w|$.

(2) На 2-й ленте машина T добавляет или удаляет символ 1. Число таких шагов не превосходит $|w|$.

(3) Все шаги в пунктах (1) and (2) делаются одновременно.

Отсюда вытекает, что вычислительная сложность $R(w)$ не превосходит $O(|w|)$. □

Индуктивно определим понятие ранг элемента $r(w)$ для $w \in \Sigma^*$:

$$r(w) = \begin{cases} 0, \text{ если } R(w) = false \\ sup\{r(w_1), \ldots, r(w_k)\} + 1, \text{ если } R(w) = w_1 \# \ldots \# w_k \end{cases}$$

2.3 GNF-системы

Так же как и в главе 1 будем рассматривать p-вычислимую модель $HW(\mathfrak{M})$ сигнатуры σ. Считаем, что все переменные x_i кодируются в виде $q \dots q$ длины i. Чтобы индуктивно с помощью подходящих формул задать новые типы элементов в модели, будем использовать следующий подход. Добавим конечное число одноместных предикатов $P_1, \dots, P_n$ в модель. Первоначально эти предикаты выделяют некоторые p-вычислимые множества, возможно пустые. Далее, будем индуктивно расширять их. В качестве инструмента для расширения используются соответствующие порождающие семейства $F_{P_1}, \dots, F_{P_n}$, каждое из которых состоит из счетного или конечного числа L-формул специального вида. Для всей этой конструкции, можно определить понятие GNF-системы, описанной в работе [38]. GNF-системы специального вида играют важную роль в доказательстве полиномиальной вычислимости тех или иных множеств.

Под GNF-системой будем понимать следующую восьмерку:

$$G = (\Sigma, \sigma, \Omega, L, HW(\mathfrak{M}), P, F, G), \text{ где} \tag{2.1}$$

1) Σ - некоторый конечный алфавит и $M \subseteq \Sigma^*$

2) $\Omega = \Sigma \cup \sigma \cup \sigma_L$ – расширенный алфавит.

3) L - логический язык описанный выше, в котором все L-формулы и L-программы являются элементами Ω^*.

4) P - конечное множество расширяемых предикатов $P_i \in \sigma$, где $i \in [1, \dots, n]$.

5) F - конечное множество порождающих семейств F_{P_i} $(i \in [1, \dots, n])$

6) G - конечное множество функций:

$$\gamma_i : \Sigma^* \to \Omega^* \cup \{false\}, \text{ где } i \in [1, \dots, n]$$

каждая γ_i строит по любому элементу из Σ^* некоторую L-формулу $\Phi_k(x_1, \dots, x_{n_k})$ из порождающего семейства $F_i \in F$, либо выдает $false$.

Будем говорить, что *GNF-система является p-вычислимой*, если выполнены следующие условия:

1) *Свойство предикатной отделимости и позитивности*

Для любого порождающего семейства и любой L-формулы $\Phi_k(x_1, \dots, x_{n_k})$ из него, если в эту L-формулу входит предикат P_i, то только позитивно [10] и только в виде $P_i(x_j)$, для некоторой переменной x_j. Более того, предикаты

P_i, где $i \in [1,\ldots,n]$ не участвуют ни в каких L-программах присутствующих в L-формулах из порождающих семейств. Свойство предикатной отделимости: в любую L-формулу для любой переменной x_i не могут входить одновременно две подформулы вида $P_j(x_i)$ и $P_m(x_i)$ для любых $j,m \in [1,\ldots,n]$.

2) *Свойство равномерной ограниченности*

Для любой L-формулы $\Phi_k(x_1,\ldots,x_{n_k})$ порождающего семейства F_{P_i} существуют такие константы $C,p \in N$, что для любых $\varepsilon_i \in \{0,1\}$ и $w_1,\ldots,w_{n_k} \in HW(M)$ сложность проверки истинности формулы $\Phi_k^{\varepsilon_1\ldots\varepsilon_n}(w_1,\ldots,w_{n_k})$ не превосходит:

$$t(HW(\mathfrak{M}) \models \Phi_k^{\varepsilon_1\ldots\varepsilon_n}(w_1,\ldots,w_{n_k})) \leqslant C \cdot | < w_1,\ldots,w_{n_k} > |^p$$

где $\Phi_k^{\varepsilon_1\ldots\varepsilon_n}(x_1,\ldots,x_{n_k})$ – L-формула полученная из $\Phi_k(x_1,\ldots,x_{n_k})$ заменой всех вхождений вида $P_j(x_i)$ на $\varepsilon_i \in \{0,1\}$. Заметим, что для каждого i число различных вхождений вида $P_j(x_i)$ в формуле $\Phi_k(x_1,\ldots,x_{n_k})$ не превосходит 1.

3) *Свойство единственности порождения*

Для любых двух L-формул $\Phi_1(x_1,\ldots,x_k)$, $\Phi_2(x_1,\ldots,x_k) \in F_{P_i}$, любых $w_1,\ldots,w_k \in \Sigma^*$, и любых $\overline{\varepsilon_1} = \varepsilon_{11},\ldots\varepsilon_{1k}$, $\overline{\varepsilon_2} = \varepsilon_{21},\ldots\varepsilon_{2k}$, где все $\varepsilon_{1i},\varepsilon_{2j} \in \{0,1\}$, выполняется следующее свойство единственности порождения:

$$HW(\mathfrak{M}) \not\models \Phi_1^{\overline{\varepsilon_1}}(w_1,\ldots,w_k)\&\Phi_2^{\overline{\varepsilon_2}}(w_1,\ldots,w_k)$$

4) *Свойство p-вычислимости функций из G*

Каждая функция $\gamma_i : HW(M) \to \Omega^* \cup \{false\}$, $i \in [1,\ldots,n]$ является p-вычислимой и по любому $w \in \Sigma^*$ либо строит потенциальную порождающую L-формулу $\Phi_k(x_1,\ldots,x_{n_k}) \in F_{P_i}$, либо выдает $false$.

Замечание 2.3.1. *Под потенциальной порождающей L-формулой для элемента вида* $w =< w_1,\ldots,w_{n_k} >$ *понимается L-формула* $\Phi_k(x_1,\ldots,x_{n_k}) \in F_{P_i}$ *такая, что существуют* $\varepsilon_1,\ldots,\varepsilon_{n_k} \in \{0,1\}$ *при которых:*

$$HW(\mathfrak{M}) \models \Phi_k^{\varepsilon_1\ldots\varepsilon_{n_k}}(w_1,\ldots,w_{n_k})$$

Замечание 2.3.2. *Если количество L-формул в некотором порождающем семействе конечно, то свойство равномерной ограниченности выполнено автоматически.*

Замечание 2.3.3. *Свойство предикатной отделимости и позитивности для любого порождающего семейства часто следует из самого вида L-формул содержащихся в нем.*

Замечание 2.3.4. *p-вычислимость любой функции из G для конечного порождающего семейства L-формул очень часто следует автоматически, если выполнено еще свойство единственности порождения.*

Замечание 2.3.5. *Если при проверке истинности L-формулы, значение какой-либо L-программы, входящей в эту L-формулу, равно $false$, то считаем, что эта формула ложна.*

Замечание 2.3.6. *Пусть в p-вычислимой GNF-системе расширяемый предикат P_i не имеет начальных условий или начальные условия задаются с помощью некоторой L-формулы. И пусть множество L-формул пороождающего семейства F_{P_i} конечно и никакие расширяемые предикаты $P_1, \ldots, P_n$ не входят ни в какие L-формулы семейства F_{P_i}. Тогда существует L-формула, которая конструируется из L-формул семейства F_{P_i} и L-формулы определяющей начальные условия истинности предиката P_i, задающая этот предикат.*

2.4 Лемма о подстановке

Лемма 2.4. *(о подстановке)*
Пусть задана p-вычислимая GNF-система G и пусть $\Phi_m(x_1, \ldots, x_k)$ – потенциально порождающая L-формула для w из некоторого порождающего семейства F_{P_i}. Предположим, что сложность всех функций $\gamma_i(x)$ из G ограничена полиномом вида $C \cdot |x|^p$, для некоторых подходящих C и p. Тогда $\Phi_m(w_1, \ldots, w_k)$ строится по w за время не превосходящее $4 \cdot C \cdot |w|^{p+1}$.

Доказательство. Рассмотри машину Тьюринга T состоящую из 3-х полулент. На 1-ю и 2-ю ленту подадим слово w вида $< w_1, \ldots, w_k >$. Далее, на первой ленте T запускает алгоритм реализующий p-вычислимую функцию расщепления $R(w)$ и выдает слово $w_1 \# \ldots \# w_k$. После этого, машина T на 2-й ленте запускает алгоритм реализующий функцию γ_i и получает потенциально порождающую L-формулу $\Phi_m(x_1, \ldots, x_k)$. Заметим, что переменная x_i представлена строкой символов q длины i. Далее, T возвращает головки 1-й и 2-й ленты в крайние левые состояния и посимвольно копирует слово со 2-й ленты на 3-ю, пока не встретится символ q. В этом случае при каждом считывании символа q

на 2-й ленте, кроме самого первого символа q, машина T сдвигает головку 1-й ленты вправо, пока не считает следующий символ #. Когда строка q символов будет считана, необходимо скопировать слово w_i, идущее после обозреваемого символа #, с 1-й ленты на 3-ю.

Вычислительная сложность:

1) Необходимо на 1-й ленте по слову w построить слово $w^{\#}$ вида $w_1\#\ldots\#w_k$. Время на построение по утверждению 2.3 не превосходит $C\cdot|w|$ шагов (считаем что для этого можно использовать вспомогательные ленты). Длина слова $w^{\#}$ не превосходит длины слова w.

2) На 2-й ленте по слову w с помощью функции γ_i строится потенциальная порождающая формула $\Phi_m(x_1,\ldots,x_k)$. Так как наша GNF-система является p-вычислимой, то длина данной L-формулы не превосходит $C\cdot|w|^p$.

3) Посчитаем вычислительную сложность построения слова вида $\Phi_m(w_1,\ldots,w_k)$. Возьмем C как максимум из констант в пунктах 1) и 2).

а) Головка 2-й ленты всегда движется только вправо или стоит на месте. Количество тактов не превосходит длины $\Phi_m(x_1,\ldots,x_k)$, которая в свою очередь не превосходит $C\cdot|w|^p$.

б) Головка 1-й ленты может при этом ходить влево или вправо, либо стоять на месте. Суммарное количество тактов не превосходит количество символов q встречающихся в слове 2-й ленты умноженное на две длины слова $w^{\#}$. Число тактов работы машины T на 3-й ленте не превосходит:

$$2\cdot|w|\cdot|\Phi_m(x_1,\ldots,x_k)|\leqslant 2\cdot C\cdot|w|^{p+1}$$

Суммируя общее число тактов по каждой из лент получим утверждение леммы.

□

2.5 Наименьшие неподвижные точки операторов

Для одноместных предикатов $P_1,\ldots,P_n$ индуктивно определим процесс расширения их множеств истинности следующим образом:

1) *шаг 0:* Вначале $P_1=\varnothing,\ldots,P_n=\varnothing$.

2) *шаг* $m\cdot n+i$*:* пусть на шаге $m\cdot n+i-1$, где $i\in[1,\ldots,n]$, одноместные

предикаты $P_1,\ldots,P_n$ выделяют некоторые подмножества элементов $Q_1,\ldots,Q_n$ из $HW(M)$ соответственно.

Тогда будем говорить, что на шаге $m \cdot n + i$ истинность предиката P_i расширяется до $Q_i^* \subseteq \Sigma^*$, если:

$$Q_i^* = Q_i \cup \cup_{\Phi_k \in F_{P_i}} \{< w_1,\ldots,w_{n_k} > \ | HW(\mathfrak{M}) \models \Phi_k(w_1,\ldots,w_{n_k})\} \tag{2.2}$$

Весь этот процесс индуктивного расширения истинности предикатов $P_1,\ldots,P_n$ можно представить в виде монотонного локально-конечного оператора:

$$\Gamma^{HW(\mathfrak{M})}_{F_{P_1},\ldots,F_{P_n}} : P(HW(M))^n \to P(HW(M))^n$$

следующим образом:

$$\Gamma^{HW(\mathfrak{M})}_{F_{P_1},\ldots,F_{P_n}}(Q_1,\ldots,Q_n) = (Q_1^*,\ldots,Q_n^*) \tag{2.3}$$

где Q_i^* получается с помощью Q_i из равенства (2.2).

У этого оператора существует наименьшая неподвижная точка, которая достигается за w шагов. [35]

Определим наименьшую неподвижную точку Γ_w следующим образом:

$$\Gamma_0 = (\varnothing,\ldots,\varnothing),\ldots,\Gamma_{k+1} = \Gamma^{HW(\mathfrak{M})}_{F_{P_1},\ldots,F_{P_n}}(\Gamma_k),\ldots,\Gamma_w = \cup_{k<w}\Gamma_k \tag{2.4}$$

Также можно определить наименьшую неподвижную точку с р-вычислимыми начальными условиями:

$$\Gamma_0(Q) = (Q_1,\ldots,Q_n),\ldots,\Gamma_{k+1}(Q) = \Gamma^{HW(\mathfrak{M})}_{F_{P_1},\ldots,F_{P_n}}(\Gamma_k),\ldots,\Gamma_w(Q) = \cup_{k<w}\Gamma_k(Q) \tag{2.5}$$

где $Q = (Q_1,\ldots,Q_n)$ и все множества $I_j(Q)$ – р-вычислимы, $j \in [1,\ldots,n]$

2.6 Обобщенная PAG-теорема с начальными условиями

Теорема 2.5. *(обобщенная PAG-теорема с р-вычислимыми начальными условиями)*
Пусть задана р-вычислимая GNF-система G вида (2.1), тогда наименьшая неподвижная точка с р-вычислимыми начальными условиями $\Gamma_w(Q)$ (2.5) оператора $\Gamma^{HW(\mathfrak{M})}_{F_{P_1},\ldots,F_{P_n}}$ из (2.3) является р-вычислимой.

Доказательство. Чтобы показать р-вычислимость неподвижной точки, достаточно показать, что все проекции вида $I_j(\Gamma_w(Q))$ р-вычислимы. А для этого необходимо показать, что сложность проверка истинности формул вида $P_j(w)$ полиномиальна.

Индукцией по сложности $r(w)$ покажем что:

$$t(P_j(w)) \leqslant 36 \cdot C \cdot (r(w) + 1) \cdot |w|^{p+1}$$

где C и p максимальные константы, которые участвуют в построении полинома вида $C \cdot |x|^p$. Данный полином ограничивает сложность всех р-вычислимых конструкций в р-вычислимой GNF-системе G.

Базис индукции: $r(w) = 0$, тогда утверждение верно в силу того, что элемент не является списком и поэтому достаточно проверить только его принадлежность к множеству Q_j.

Шаг индукции: пусть утверждение верно при всех $r(w) < n$, докажем это и для $r(w) = n$.

Пусть $(\Phi_{n_k}(w_1, \ldots, w_k))^{(s)}$ – слово, получаемое из слова $\Phi_{n_k}(w_1, \ldots, w_k)$ заменой первого вхождения подслова вида $P_j(w_i)$ на специальную строку символов s той же длины, что и $P_j(w_i)$.

Рассмотрим многоленточную машину Тьюринга T, состоящую из нескольких полулент. 1-я полулента будет основной, все остальные вспомогательные. И, далее, машина T, используя разделительные символы # на 1-й ленте будет строить слова вида:

$$P_j(w)$$

$$P_j(w)\#\Phi_{n_k}(w_1, \ldots, w_k)$$

$$P_j(w)\#(\Phi_{n_k}(w_1, \ldots, w_k))^{(s)}\#P_m(w_i)$$

$$P_j(w)\#(\Phi_{n_k}(w_1, \ldots, w_k))^{(s)}\#\varepsilon_i$$

$$(P_j(w)\#(\Phi_{n_k}(w_1, \ldots, w_k))^{(s)})^{\varepsilon_i}$$

где $\gamma_j(w) = \Phi_{n_k}(x_1, \ldots, x_k)$ и $P_m(w_i)$ – первое вхождение слова вида $P_j(w_i)$ в слово $\Phi_{n_k}(w_1, \ldots, w_k)$.

Работу машины T на 1-й ленте можно рассматривать как работу со стеком, в котором работа всегда происходит с последним элементом: удаление, изменение последнего элемента или добавление в конец нового элемента.

Далее, используя индукционное предположение, машина T будет вычислять значение истинности ε_i формулы $P_m(w_i)$. После того как это значение

истинности будет вычислено, машина T заменяет вхождение строки символов s на строку где первым стоит символ ε_i и дальше идут специальные символы $\square$. После этого машина T продолжает искать дальше вхождения вида $P_j(w_i)$ в слове $(\Phi_{n_k}(w_1,\ldots,w_k))^{(s)})^{\varepsilon_i}$.

В силу p-вычислимости GNF-системы, количество вхождений вида $P_j(w_i)$ в слово $\Phi_{n_k}(w_1,\ldots,w_k)$ не превосходит k. Причем для каждого w_i существует только одно вхождение вида $P_j(w_i)$.

Поэтому, в силу индукционного предположения, суммарная сложность вычисления этих конструкций следующая:

$$\sum_{i=1}^{k} t(P_{j_i}(w_i)) \leqslant \sum_{i=1}^{k} 36 \cdot C \cdot (r(w_i)+1) \cdot |w_i|^{p+1} \leqslant 36 \cdot C \cdot ((r(w)+1)-1) \cdot |w|^{p+1}$$

Машина T осуществляет также работу по копированию элементов вида $P_j(w_i)$ в конец 1-й ленты. Для этого нужно скопировать первое слово вида $P_j(w_i)$ на вспомогательную ленту, пройти всю формулу на 1-й ленте, добавить специальный символ # и скопировать это слово со вспомогательной ленты. Суммарно машина T должна сделать k таких проходов, где $k \leqslant |w|$.

По лемме 2.4 имеем, что сложность построения $\Phi_{n_k}(w_1,\ldots,w_k)$ не превосходит $4 \cdot C \cdot |w|^{p+1}$, при этом:

$$|\Phi_{n_k}(w_1,\ldots,w_k)| \leqslant C \cdot |w|^p + p \leqslant 2 \cdot C \cdot |w|^p$$

поэтому общее количество тактов для копирования слов вида $P_j(w_i)$ не превосходит:

$$2 \cdot |w| \cdot 2 \cdot C \cdot |w|^p = 4 \cdot C \cdot |w|^{p+1}$$

После того как все значения истинности ε_i для всех $P_j(w_i)$ найдены и подставлены в слово $\Phi_{n_k}(w_1,\ldots,w_k)$, необходимо скопировать эту формулу на вспомогательную ленту при этом игнорируя символы $\square$ и вычислить ее значение истинности, затем вычисленное значение подставить вместо этого слова на 1-й ленте. В силу p-вычислимости GNF-системы сложность проверки истинности этой формулы не превосходит $C \cdot |w|^p$.

Суммарное количество тактов на копирование, вычисление и подстановку вычисленного значения не превосходит: $10 \cdot C \cdot |w|^{p+1}$.

Суммируя все вышесказанное получим, что:

$$t(P_j(w)) \leqslant 36 \cdot C \cdot (r(w)+1) \cdot |w|^{p+1}$$

Индукционное предположение доказано и так как $r(w)+1$ всегда меньше длины w получаем, что сложность проверки истинности $P_j(w)$ на $HW(\mathfrak{M})$ не превосходит $36 \cdot C \cdot |w|^{p+2}$. Тем самым получаем, что предикаты P_j, где $j \in [1,\ldots,n]$ являются p-вычислимыми, а следовательно p-вычислимой является и наименьшая неподвижная точка Γ_w оператора $\Gamma^{HW(\mathfrak{M})}_{F_{P_1},\ldots,F_{P_n}}$.

□

Следствие 2.5.1. *Пусть в p-вычислимой GNF-системе с полиномиально вычислимыми начальными условиями вычислительная сложность всех базовых функций, предикатов, а также функций γ_i для построения порождающих формул не превосходит $O(|x|^p)$. Тогда вычислительная сложность наименьшей неподвижной точки не превосходит $O(|x|^{p+2})$.*

Глава 3. Полиномиально вычислимые представления

В данной главе получена серия результатов о существовании полиномиально вычислимых представлений:

– для множества термов и множества формул ИП.

– для множества L-программ и множества L-формул.

– для множества доказательств ИП в виде дерева.

– для множества линейных доказательств ИП.

– для множества выводов в порождающих грамматиках.

Везде далее речь пойдет о полиномиально вычислимых представлениях над элементами основного множества p-вычислимой модели $HW(\mathfrak{M})$ конечной сигнатуры σ. Где сигнатура σ содержит в себе, как и в главе 1, сигнатуры σ_N, σ_W, σ_S. Под представлением некоторого множества над $HW(M)$ будем понимать некоторое произвольное кодирование его элементов с помощью элементов множества $HW(M)$. Чаще всего такая кодировка будет инъективной.

Основная цель этого раздела – это собрать воедино все полученные автором результаты о тех множествах синтаксических конструкций (термы, формулы и т.д.), которые задаются индуктивно и имеют полиномиально вычислимые списочные представления. Большинство этих конструкций являются базовыми понятиями математической логики, теории алгоритмов и теории программирования. Особенно бы хотелось отметить существование полиномиально вычислимых представлений для множества доказательств исчисления предикатов первого порядка как в виде дерева, так и линейных.

В данной главе было не только важно доказать существование полиномиально вычислимых представлений выше перечисленных конструкций, но и показать технику построения и применения p-вычислимых GNF-систем, обобщенной PAG-теоремы и построения соответствующих L-программ и L-формул.

Существование полиномиально вычислимых списочных представлений для всех этих синтаксических объектов, позволяет нам использовать их в логическом языке программирования L без нарушения полиномиальной вычислительной сложности программ. Тем самым увеличивая выразительную силу языка L.

3.1 Представления для термов и формул ИП

Пусть задана некоторая конечная произвольная сигнатура:

$$\sigma^{ext} = \{f_1^{m_1}, \ldots, f_k^{m_k}, P_1^{m_{k+1}}, \ldots, P_n^{m_{k+n}}, c_1, \ldots, c_m\} \tag{3.1}$$

Далее, в этой сигнатуре будут рассматриваться термы и формулы ИП [11]. Построим соответствующую сигнатуры σ_{ext}, состоящую из константных символов вида:

$$\sigma_{ext} = \{\underline{f_1}, \ldots, \underline{f_k}, \underline{P_1}, \ldots, \underline{P_n}, \underline{c_1}, \ldots, \underline{c_m}\}$$

расширим с помощью этих константных символов основную сигнатуру модели $HW(\mathfrak{M})$ и добавим аналогичные символы в множество M модели $HW(\mathfrak{M})$. Основная цель – это закодировать все формулы и термы с помощью списков $HW(M)$, а также порождать эти списки с помощью формул порождающего семейства.

Пусть предикат Var выделяет элементы $\underline{x_i^d}$, где $\underline{x_i^d}$ – код переменной x_i в виде строки символов d длины i. Пусть также предикат $Const$ выделяет элементы $c_h \in M$, где $\underline{c_h} \in \sigma_{ext}$. Заметим, что эти предикаты являются p-вычислимыми.

Покажем, что множество термов сигнатуры σ^{ext} в ИП имеет полиномиально вычислимое представление. Для этого определим индуктивно понятие терма с помощью предиката $Term$. Из его p-вычислимости и будет автоматически следовать существование полиномиально вычислимого представления. Вначале зададим p-вычислимые начальные условия для предиката $Term$:

$$Term = Var \cup Const$$

Далее, построим порождающее семейство F_{Term}:

$$\{(x_1 = \underline{f_i})\&Term(x_2)\& \ldots \&Term(x_{m_i+1}) \mid i \in [1, \ldots, k]\} \tag{3.2}$$

Лемма 3.1. *$Term$ – p-вычислимый предикат.*

Доказательство. Для этого достаточно показать, что следующая GNF-система (2.1)

$$G = (\Sigma, \sigma, \Omega, L, HW(\mathfrak{M}), \{Term\}, \{F_{Term}\}, \{\gamma\})$$

является p-вычислимой.

Свойство предикатной отделимости и позитивности выполнено по построению. Свойство равномерной ограниченности вытекает из конечного числа формул в порождающем семействе. Свойство единственности порождения также выполнено, так как разные формулы порождают разные списочные элементы. Каждая порождающая формула порождает списки в которых на первом месте, в качестве элемента списка, стоит уникальный $\underline{f_i}$.

Свойство p-вычислимости функции γ следует из того, что каждая L-формула порождающего семейства имеет свое отличие. Она порождает только те элементы, в которых на первом месте стоит уникальный символ $\underline{f_i}$, поэтому легко построить полиномиально вычислимую функцию, которая по виду списочного элемента построит его потенциальную порождающую формулу или выдаст $false$.

Следовательно, по теореме 2.5, предикат $Term$ является p-вычислимым. □

Далее, покажем, что множество формул ИП сигнатуры σ^{ext} имеет полиномиально вычислимое представление.

Зададим порождающее семейство для предиката $Formula$:

$$
\begin{aligned}
&(x_1 = \underline{=})\&Term(x_2)\&Term(x_3)\\
&(x_1 = \underline{\&})\&Formula(x_2)\&Formula(x_3)\\
&(x_1 = \underline{\vee})\&Formula(x_2)\&Formula(x_3)\\
&(x_1 = \underline{\rightarrow})\&Formula(x_2)\&Formula(x_3)\\
&(x_1 = \underline{\neg})\&Formula(x_2)\\
&(x_1 = \underline{\exists})\&Var(x_2)\&Term(x_3)\&Formula(x_4)\\
&(x_1 = \underline{\forall})\&Var(x_2)\&Term(x_3)\&Formula(x_4)\\
&\{(x_1 = \underline{P_j})\&_{i=2}^{n_j+1} Term(x_i) \mid j \in [1,\ldots,n]\}
\end{aligned}
$$

Лемма 3.2. *$Formula$ – p-вычислимый предикат.*

Доказательство. В силу того, что порождающее семейство $F_{Formula}$ содержит конечное число L-формул, то доказательство этой леммы почти полностью совпадает с доказательством леммы 3.1. Поэтому все свойства p-вычислимой GNF-системы, также как и в случае с предикатом $Term$, выполнены. Следовательно, предикат $Formula$ является p-вычислимым. □

Замечание 3.1.1. *Так как мы определили кодировки для всех синтаксических конструкций ИП (переменных, констант, термов, формул), то везде далее, если не будет возникать разночтений, будем вместо код терма, формулы, переменной и константы писать просто терм, формула, переменная, константа.*

Определим предикат $TermVar$ следующим образом:

$$TermVar(\underline{t^d},\underline{x^d}) = \begin{cases} true, \text{ если } \underline{x^d} \in FV(\underline{t^d}) \\ false, \text{ иначе} \end{cases} \tag{3.3}$$

где $FV(\underline{t^d})$ – множество свободных переменных терма $\underline{t^d}$.

Лемма 3.3. *Предикат $TermVar$ – p-вычислим*

Доказательство. Вначале, необходимо проверить корректность входящих параметров с помощью p-вычислимых предикатов $Term$ и Var. После этого только останется проверить точное вхождение строки символов d, кодирующей переменную x, в терм $\underline{t^d}$. Под точным вхождением подразумевается, что непосредственно ни слева, ни справа от такого вхождения больше нет вхождения символа d. Это, в свою очередь, легко проверяется с помощью многоленточной машины Тьюринга T. Причем за полиномиальное время от длины списка вида $< \underline{t^d},\underline{x^d} >$. □

Определим предикат $FormulaFreeVar$ следующим образом:

$$FormulaFreeVar(\underline{\Phi^d},\underline{x^d}) = \begin{cases} true, \text{ если } \underline{x^d} \in FV(\underline{\Phi^d}) \\ false, \text{ иначе} \end{cases} \tag{3.4}$$

Лемма 3.4. *$FormulaFreeVar$ – p-вычислимый предикат.*

Доказательство. Все рассуждения почти полностью такие же как и в лемме 3.3. Единственное отличие будет в том, что нам потребуется дополнительно завести на вспомогательной ленте счетчик треугольных скобок. Если при работе над словом $\underline{\Phi^d}$ встретился квантор $\underline{\forall}$ или квантор $\underline{\exists}$ после которого идет переменная $\underline{x^d}$, то необходимо открыть счетчик треугольных скобок на вспомогательной ленте(если подсчет уже не ведется). Для этого на вспомогательную ленту счетчика треугольных скобок машина T добавляет в конец символ 1, если T считала левую треугольную скобку и стирает 1 с конца, если правую. Пока

есть символы 1 на этой ленте, то на основной ленте машина T пропускаем все переменные. T начинает отслеживать свободную переменную, только если лента счетчика треугольных скобок пустая. Также легко видеть, что вышеописанный алгоритм имеет полиномиальную вычислительную сложность. □

3.2 Представления для L-формул и L-программ

Покажем теперь, что множества L-формул и L-программ конечной сигнатуры σ имеет полиномиально вычислимые представления. Рассмотрим p-вычислимую модель $HW(\mathfrak{M})$ сигнатуры $\sigma \cup \sigma_l \cup \sigma_L$, где сигнатуры σ_l и σ_L определены в главе 1 и имеют вид:

$$\sigma_l = \{\underline{\&}, \underline{\vee}, \underline{\rightarrow}, \underline{\neg}, \underline{\forall}, \underline{\exists}, \underline{(}, \underline{)}, d\} \cup \{,\}$$

$$\sigma_L = \{\underline{\exists \in}, \underline{\forall \in}, \underline{\exists \subseteq}, \underline{\forall \subseteq}, \underline{\exists \leqslant}, \underline{\forall \leqslant}, \underline{\leqslant}, \underline{Cond}, \underline{Iteration}, q\}$$

Также добавим для константных символов сигнатур σ_l и σ_L представителей в множество M с теми же именами. Обозначим новое множество за M' Тогда, можно закодировать L-формулы и L-программы в виде элементов $HW(M')$. Более того, можно задать порождающие формулы, которые будут порождать индуктивно L-формулы и L-программы.

Пусть $Const_L$ – предикат выделяющий элементы в M', которые являются интерпретациями для констант из $\sigma \cup \sigma_L$.
Пусть Var_L – предикат выделяющий элементы вида $q \ldots q$ произвольной конечной длины из M'.

Предикат $Term_L$ будет задаваться с помощью порождающего семейства из (3.2) (только теперь все символы $\underline{f_i} \in \sigma$) с начальными условиями:

$$Term_L = Var_L \cup Const_L$$

Тогда рассмотрим два порождающих семейства для предикатов $Formula_L$ и $Program_L$ соответственно.

Начальные условия для предикатов:

$$Formula_L = \varnothing,\ Program_L = Term_L$$

Семейство $F_{Formula_L}$ имеет вид:

$$(x_1 = \underline{=})\&Program_L(x_2)\&Program_L(x_3)$$
$$(x_1 = \underline{\&})\&Formula_L(x_2)\&Formula_L(x_3)$$
$$(x_1 = \underline{\vee})\&Formula_L(x_2)\&Formula_L(x_3)$$
$$(x_1 = \underline{\rightarrow})\&Formula_L(x_2)\&Formula_L(x_3)$$
$$(x_1 = \underline{\neg})\&Formula_L(x_2)$$
$$(x_1 = \underline{\forall \in})\&Var_L(x_2)\&Program_L(x_3)\&Formula_L(x_4)$$
$$(x_1 = \underline{\exists \in})\&Var_L(x_2)\&Program_L(x_3)\&Formula_L(x_4)$$
$$(x_1 = \underline{\forall \subseteq})\&Var_L(x_2)\&Program_L(x_3)\&Formula_L(x_4)$$
$$(x_1 = \underline{\exists \subseteq})\&Var_L(x_2)\&Program_L(x_3)\&Formula_L(x_4)$$
$$(x_1 = \underline{\forall \leqslant})\&Var_L(x_2)\&Program_L(x_3)\&Formula_L(x_4)$$
$$(x_1 = \underline{\exists \leqslant})\&Var_L(x_2)\&Program_L(x_3)\&Formula_L(x_4)$$
$$\{(x_1 = \underline{P_j})\&_{i=2}^{n_j+1} Program_L(x_i) \mid j \in [1,\ldots,n]\}$$

Семейство $F_{Program_L}$ имеет вид:

$$(x_1 = \underline{Iteration})\&Program_L(x_2)\&Program_L(x_3)\&Program_L(x_4)\&Formula_L(x_5)$$
$$\{(x_1 = \underline{f_j})\&(\&_{j=2}^{n_j+1} Program_L(x_j)) \mid j \in [1,\ldots,k]\}$$
$$\{(x_1 = \underline{Cond})\&_{i=1}^{k}(Program_L(x_{2i})\&Formula_L(x_{2i+1}))\&Program_L(x_{2k+2}) \mid k \in N\}$$

Лемма 3.5. *$Formula_L$ и $Program_L$ – р-вычислимые предикаты.*

Доказательство. Покажем, что следующая GNF-система

$$G = (\Sigma,\sigma,\Omega,L,HW(\mathfrak{M}),\{Formula_L, Program_L\},\{F_{Formula_L},F_{Program_L}\},\{\gamma_1,\gamma_2\})$$

является р-вычислимой.

Свойства предикатной отделимости и позитивности, а также свойство единственности порождения выполнены для каждого из семейств по построению.

Докажем, что выполнено также свойство равномерной ограниченности. Для первого семейства это выполнено в силу конечного числа L-формул входящих в него. Для второго семейства достаточно показать это для счетного числа L-формул вида:

$$\underline{\Phi_k^{\varepsilon}}: \ \{(x_1 = \underline{Cond})\&_{i=1}^{k}(\varepsilon_{2i}\&\varepsilon_{2i+1})\&\varepsilon_{2k+2} \mid k \in N\} \tag{3.5}$$

где $\varepsilon_i \in [0,1]$.

Существует $C \in N$ такая, что для любого означивания $w_1,\ldots,w_{2k+2}$ выполнено неравенство:

$$|\underline{\Phi}_k^{\bar{\varepsilon}}(w_1,\ldots,w_{2k+2})| \leqslant C \cdot | < w_1,\ldots,w_{2k+2} > |$$

и тогда, в силу того, что в формуле $\underline{\Phi}_k^{\bar{\varepsilon}}$ все конъюктивные члены, кроме первого, либо 0, либо 1, оценка на сложность проверки истинности формулы будет линейная:

$$t(\underline{\Phi}_k^{\bar{\varepsilon}}(w_1,\ldots,w_{2k+2})) \leqslant C_1 \cdot C \cdot | < w_1,\ldots,w_{2k+2} > | \leqslant C_2 \cdot | < w_1,\ldots,w_{2k+2} > |$$

Для γ_1 свойство p-вычислимости выполнено в силу конечного числа L-формул и свойства единственности порождения для семейства $F_{Formula_L}$.

Покажем, что функция γ_2 является p-вычислимой. Не нарушая общности, будем считать, что она строит только формулы вида:

$$\underline{\Phi_k} : \ \{(x_1 = \underline{Cond})\&_{i=1}^{k}(Program_L(x_{2i})\& \\ \&Formula_L(x_{2i+1}))\&Program_L(x_{2k+2}) \mid k \in N\}$$

где x_i есть слово вида $q\ldots q$ длины i.

Количество символов q в формуле $\underline{\Phi_k}$ не более чем: $(1 + (2k + 2))/2 \cdot (2k + 2)$, поэтому функция γ_2 по $w =< w_1,\ldots,w_{2k+2} >$ строит потенциальную порождающую формулу со следующим ограничением на длину:

$$|\underline{\Phi_k}(x_1,\ldots,x_{2k+2})| \leqslant C \cdot | < w_1,\ldots,w_{2k+2} > | + (1 + (2k + 2))/2 \cdot (2k + 2) \leqslant \\ \leqslant C \cdot |w| + |w|^2/2$$

Чтобы доказать p-вычислимость функции γ_2 необходимо сделать следующее:

1) расщепить w на $w_1\#\ldots\#w_{2k+2}$. Вычислительная сложность не превосходит $C_1 \cdot |w|$ шагов. [35]

2) если $w_1 \neq \underline{Cond}$, то закончить работу, иначе заменить все слова w_i на символы 1 и получить слово $1\#\ldots\#1$. Вычислительная сложность не превосходит $C_2 \cdot |w|$ шагов.

3) Построить многоленточную машину Тьюринга которая будет считывать слово $1\#\ldots\#1$ и по нему строить формулу: $\underline{\Phi_k}(x_1,\ldots,x_{2k+2})$. Для этого на 2-й ленте будем выписывать строку символов q которая будет на единицу больше длины считанных с 1-й ленты символов #. На 3-ю ленту, при считывании нового символа # с 1-й ленты будет выписываться слово:

$$(Program_L(x_{2i})\&Formula_L(x_{2i+1}))$$

Суммарная вычислительная сложность пунктов 1)-3) не превосходит $C_4 \cdot |w|^2$ для подходящего $C_4 \in N$. Отсюда получаем, что γ_2 – p-вычислимая функция.

Все свойства p-вычислимой GNF-системы выполнены. □

3.3 Вспомогательные утверждения ИП

Как и ранее, под исчислением предикатов сигнатуры Σ (ИП$^\Sigma$) будем понимать исчисление предикатов определенное в [11]. Формулами ИП$^\Sigma$ будут формулы сигнатуры σ. Секвенциями ИП$^\Sigma$ называются последовательности следующих 4 типов:

$$\Phi_0, \ldots, \Phi_n \vdash \Psi;\ \Phi_0, \ldots, \Phi_n \vdash;\ \vdash \Psi;\ \vdash \tag{3.6}$$

где $\Phi_0, \ldots, \Phi_n, \Psi$ - формулы ИП$^\Sigma$.

Пусть $\underline{x_1^d}, \ldots, \underline{x_n^d}$ - переменные, $\underline{t_1^d}, \ldots, \underline{t_n^d}$ - термы. Тогда под записью $(\underline{\Phi^d})^{\underline{x_1^d},\ldots,\underline{x_n^d}}_{\underline{t_1^d},\ldots,\underline{t_n^d}}$ будем понимать результат подстановки термов $\underline{t_1^d}, \ldots, \underline{t_n^d}$ вместо всех свободных вхождений переменных $\underline{x_1^d}, \ldots, \underline{x_n^d}$ соответственно. Причем предполагается, что ни одно свободное вхождение в $\underline{\Phi^d}$ переменной $\underline{x_i^d}$ не входит в подформулу $\underline{\Phi^d}$ вида $\underline{\forall y^d \Phi_1^d}$ или $\underline{\exists y^d \Phi_2^d}$ для $\underline{y^d} \in FV(\underline{t_i^d})$.

Замечание 3.3.1. *Хотя формулы, термы и переменные заданы через свои списочные кодировки, но понятия свободного вхождения переменной, ее замены на терм и так далее легко переносятся и сюда.*

Определим предикат *checkAddmissible* следующим образом:

$$checkAddmissible(\underline{\Phi^d}, \underline{x^d}, \underline{t^d}) = \begin{cases} true, \text{ если ни одно свободное вхождение } \underline{x^d} \\ \quad \text{не входит в подформулу } \underline{\Phi^d} \text{ вида} \\ \quad \underline{\forall y\ \Phi_1^d} \text{ или } \underline{\exists y\ \Phi_2^d} \text{ для } \underline{y^d} \in FV(\underline{t^d}) \\ false, \text{ иначе} \end{cases} \tag{3.7}$$

Лемма 3.6. *checkAddmissible – p-вычислимый предикат.*

Доказательство. Для начала, по терму $\underline{t^d}$ создадим список всех переменных, входящих в него, в том порядке, в котором они встречаются в терме. Переменные могут повторяться. Список же будет иметь вид: $< \underline{y_1^d}, \ldots, \underline{y_k^d} >$ и он строится

за полиномиальное время от длины терма $\underline{t^d}$ с помощью 2-х ленточной машины Тьюринга. Где на 2-ю ленту с 1-й копируются только переменные, запятые между ними и граничные треугольные скобки.

Далее, построим p-вычислимую характеристическую функцию для предиката *checkAddmissible* с помощью p-итерационного терма.

g будет работать со словом вида:

$$< 1,\underline{\Phi^d},\underline{x^d}, < \underline{y_1^d}, \ldots ,\underline{y_i^d} >> \tag{3.8}$$

следующим образом:
а) Берет последнюю переменную из последнего элемента списка (3.8) (в данном случае это $\underline{y_i^d}$) и проверяет входит ли $\underline{x^d}$ свободно в подформулы вида $\underline{\exists y_i^d \Theta}$, $\underline{\forall y_i^d \Theta}$ формулы $\underline{\Phi^d}$. Если входит, то g выдает:

$$< 0,\underline{\Phi^d},\underline{x^d}, < \underline{y_1^d}, \ldots ,\underline{y_i^d} >>$$

если же не входит, то g выдает:

$$< 1,\underline{\Phi^d},\underline{x^d}, < \underline{y_1^d}, \ldots ,\underline{y_{i-1}^d} >>$$

Несложно видеть, что так определенная функция g – p-вычислимая так как доказательство почти полностью совпадает с доказательством леммы 3.4. Только здесь ищем квантор по $\underline{y_i^d}$ вместо $\underline{x^d}$.

L-формула φ будет иметь следующий вид:

$$(first(x) = 0) \vee (head(x) = nil)$$

Отсюда следует, что p-итерационный терм, задаваемый с помощью g и φ, описанных выше, удовлетворяет свойствам 1) и 2) утверждения 1.5. □

Определим функцию $changeFreeVar$ следующим образом:

$$changeFreeVar(\underline{w},\underline{x^d},\underline{t^d}) = \begin{cases} \underline{\Psi^d}, \text{ если } \underline{w} \text{ формула вида } \underline{\Phi^d} \\ \quad \text{и формула } \underline{\Psi^d} \text{ получается из формулы } \underline{\Phi^d} \\ \quad \text{заменой всех вхождений свободной} \\ \quad \text{переменной } \underline{x^d} \text{ в } \underline{\Phi^d} \text{ на терм } \underline{t^d} \\ \underline{t_1^*}, \text{ если } \underline{w} \text{ есть терм вида } \underline{t_1^d} \\ \quad \text{и терм } \underline{t_1^*} \text{ получается из терма } \underline{t_1^d} \\ \quad \text{заменой всех вхождений} \\ \quad \text{переменной } \underline{x^d} \text{ в } \underline{t_1^d} \text{ на терм } \underline{t^d} \\ false, \text{ иначе} \end{cases} \tag{3.9}$$

Лемма 3.7. *changeFreeVar – p-вычислимая функция.*

Доказательство. Реализуем эту функцию с помощью модифицированного p-итерационного терма.

Если L-программа $g(x)$ получает на вход элемент вида $< \underline{\Phi^d},\underline{x^d},\underline{t^d} >$, то она ищет первое вхождение свободной переменной $\underline{x^d}$ в $\underline{\Phi^d}$ и заменяет его на $\underline{t^d}$. Этот алгоритм реализуется почти также, как и в лемме 3.4, только здесь помимо того, чтобы найти свободную переменную, нужно еще ее заменить.

Значением будет $< \underline{\Phi^*},\underline{x^d},\underline{t^d} >$, где формула $\underline{\Phi^*}$ получается из $\underline{\Phi}$ заменой одного вхождения переменной $\underline{x^d}$ на терм $\underline{t^d}$.

Для терма еще проще, если g получает на вход элемент вида $< \underline{t_1^d},\underline{x^d},\underline{t^d} >$, то просто ищем первое вхождение переменной $\underline{x^d}$ и заменяем его на $\underline{t^d}$.

L-формула $\varphi(x)$ имеет вид:

$$\neg FormulaFreeVar(first(x_1),second(x_1))$$

Для p-итерационного терма, задаваемого с помощью g и φ, условия 1)и 2) утверждения 1.5 выполняются и поэтому он является p-вычислимым. □

Пусть $\underline{\Phi^d}$ – списочная кодировка формулы сигнатуры σ. Определим функцию *change* следующим образом:

$$change(\underline{\Phi^d},\underline{x^d},\underline{t^d}) = \begin{cases} \underline{\Psi^d}, \text{ если } (\underline{\Phi^d})^{\frac{x^d}{t^d}} = \underline{\Psi^d} \\ false, \text{ иначе} \end{cases} \tag{3.10}$$

Лемма 3.8. *change – p-вычислимая функция.*

Доказательство. Построим L-программу реализующую функцию *change*:

$$Cond(changeFreeVar(\underline{\Phi^d}, \underline{x^d}, \underline{t^d}),\ checkAddmissible(\underline{\Phi^d}, \underline{x^d}, \underline{t^d}), false)$$

□

Пусть $\underline{\Psi^d}$ и $\underline{\Theta^d}$ списочные кодировки формул сигнатуры σ. Определим функцию *gtp* следующим образом:

$$gtp(\underline{\Psi^d},\underline{\Theta^d}, \underline{x^d}) = \begin{cases} \underline{t^d}, \text{ если существуют терм } \underline{t^d} \text{ и такие слова} \\ \quad \alpha = \alpha_1 \dots \alpha_n,\ \beta = \beta_1 \dots \beta_k,\ \theta = \theta_1 \dots \theta_m, \text{ что} \\ \quad \alpha \cdot \underline{x^d} \cdot \beta = \underline{\Psi^d} \text{ и } \alpha \cdot \underline{t^d} \cdot \theta = \underline{\Theta^d}, \text{ при этом} \\ \quad \alpha_n \neq d,\ \beta_1 \neq d \text{ и } \theta_1 \neq d \\ false, \text{ иначе} \end{cases} \tag{3.11}$$

где все α_i, β_j и θ_k из алфавита Σ.

Лемма 3.9. *gtp – p-вычислимая функция.*

Доказательство. Первое, что нужно сделать, это проверить входящие параметры с помощью p-вычислимых предикатов $Formula$ и Var. Далее, рассмотрим 5-ти ленточную машину Тьюринга T. На первые три ленты выпишем соответственно $\underline{\Psi^d}$, $\underline{\Theta^d}$ и $\underline{x^d}$. Далее, машина T посимвольно сравнивает слова на 1-й и 2-й ленте. Как только появились разные символы, то передвигает головки 1-й и 2-й ленты обратно влево, пока встречаются символы d. Затем, на 1-й ленте, нужно считать всю строку символов из d, при этом сравнивая посимвольно со словом $\underline{x^d}$ на 3-й ленте. Если на 1-й ленте стоит переменная $\underline{x^d}$, то машина пытается считать терм $\underline{t^d}$ со 2-й ленты с того символа, который сейчас обозревает головка. Это либо переменная, либо константа, либо некоторый терм, начинающийся с функционального символа. Т.е. может встречаться либо строка символов из d, либо константный символ, либо список, который кодирует терм. Для первых двух случаев, просто считываем и копируем ответ на 5-ю ленту. Для третьего случая, нужно завести счетчик треугольных скобок. При считывании со 2-й ленты левой треугольной скобки машина T добавляет символ 1 на 4-ю ленту и удаляет 1 при считывании правой треугольной скобки. Параллельно копируя посимвольно слово со 2-й ленты на 5-ю, пока головка 4-й ленты не будет обозревать пустой символ (количество правых и левых скобок сравнялось). Тем самым машина выпишет на 5-ю ленту терм $\underline{t^d}$. Несложным

суммированием всех тактов, которые делают головки каждой ленты получаем, что суммарное количество тактов работы машины Тьюринга не превосходит полинома от длины входных данных. □

Пусть $\underline{\Phi^d}$, $\underline{\Psi^d}$ две списочные кодировки формул сигнатуры σ^{ext}. Определим функцию gt следующим образом:

$$gt(\underline{\Phi^d},\underline{\Psi^d},\underline{x^d}) = \begin{cases} \underline{t^d}, \text{ если существует терм } \underline{t^d} \text{ , т.ч. } (\underline{\Phi^d})^{\underline{x^d}}_{\underline{t^d}} = \underline{\Psi^d} \\ false, \text{ иначе} \end{cases} \tag{3.12}$$

Лемма 3.10. *gt – p-вычислимая функция.*

Доказательство. Применим обозначение := (по определению есть):

$$tp := gtp(\underline{\Phi^d},\underline{\Psi^d},\underline{x^d});$$

Следующая L-программа:

$$Cond(tp,\ change(\underline{\Phi^d},\underline{x^d},tp) = \underline{\Psi^d},\ false) \tag{3.13}$$

реализует функцию gt.

В силу того, что функции gtp и $change$ – p-вычислимы и gt реализуется L-программой 3.13, получаем наше утверждение. □

Определим функцию $ChangeWithCond$ следующим образом:

$$ChangeWithCond(\underline{\Theta^d},\underline{\Psi^d},\underline{x^d},\underline{y^d}) = \begin{cases} \underline{\Psi^*}, \text{ если существует такая формула } \underline{\Phi^d} \text{ и} \\ \quad \text{переменная } \underline{z^d}, \text{ что} \\ \quad \underline{\Psi^d} = (\underline{\Phi^d})^{\underline{z^d}}_{\underline{y^d}},\ \underline{\Theta^d} = (\underline{\Phi^d})^{\underline{z^d}}_{\underline{x^d}} \\ \quad \text{и формула } \underline{\Psi^*} \text{ получается из } \underline{\Psi^d} \\ \quad \text{заменой всех свободных вхождений} \\ \quad \text{переменной } \underline{y^d} \text{ на } \underline{x^d}, \text{ при условии,} \\ \quad \text{что в формуле } \underline{\Theta^d} \text{ на этих местах} \\ \quad \text{стоит переменная } \underline{x^d} \\ false, \text{ иначе} \end{cases} \tag{3.14}$$

Замечание 3.3.2. *Так как L-формулы закодированы списками, то слова "в формуле на этих местах" стоит читать как "на соответствующих позициях элементов списка" (элементов элементов списка и т.д.).*

Лемма 3.11. *ChangeWithCond – p-вычислимая функция.*

Доказательство. Для доказательства этого утверждения построим p-итерационный терм следующего вида. Функция g будет действовать следующим образом:

$$g(<\underline{\Theta^d},\underline{\Psi^d},\underline{x^d},\underline{y^d}>) =<\underline{\Theta^d},\underline{\Psi^{**}},\underline{x^d},\underline{y^d}>$$

она ищет первое свободное вхождение $\underline{y^d}$ в формуле $\underline{\Psi^d}$, при котором на этом же месте в формуле $\underline{\Theta^d}$ стоит $\underline{x^d}$, и заменяет его на $\underline{x^d}$. То что эта функция p-вычислима легко доказывается с помощью многоленточной машины Тьюринга. L-формула же φ будет иметь вид:

$$\neg FormulaFreeVar(second(x_1),\underline{y^d})$$

Поэтому для p-итерационного терма $Iteration_{g,\varphi}(<\underline{\Theta^d},\underline{\Psi^d},\underline{x^d},\underline{y^d}>,|\underline{\Psi^d}|)$ все условия 1) и 2) утверждения 1.5 выполнены. Единственное, что стоит заметить, что эти условия выполнены для элементов, где первый и второй элементы списка поменяны местами. □

3.4 Представления для секвенций и аксиом ИП

Определим p-вычислимый предикат Sec, который будет выделять множество списочных кодов всех секвенций из [11] вида:

$$\Phi_0,\ldots,\Phi_n \vdash \Psi;\ \Phi_0,\ldots,\Phi_n \vdash;\ \vdash \Psi;\ \vdash$$

где Φ_i - формулы ИП сигнатуры σ^{ext}.

Для этого построим следующее порождающее семейство F_{Sec}:

$(x_1 = \underline{\vdash})\&Formula(x_2)\&(NumElements(x_3) \geqslant 1)\&(\forall w \in x_3\ Formula(w))$
$(x_1 = \underline{\vdash})\&(x_2 = nil)\&(NumElements(x_3) \geqslant 1)\&(\forall w \in x_3\ Formula(w))$
$(x_1 = \underline{\vdash})\&Formula(x_2)\&(x_3 = nil)$
$(x_1 = \underline{\vdash})\&(x_2 = nil)\&(x_3 = nil)$

Это порождающее семейство порождает списочные элементы вида:

$$\begin{array}{c} <\underline{\vdash},\underline{\Psi},<\underline{\Phi_0},\ldots,\underline{\Phi_n}>>\ ,\ \ <\underline{\vdash},<>>\ ,<\underline{\Phi_0},\ldots,\underline{\Phi_n}>>\ , \\ <\underline{\vdash},\underline{\Psi},<>>,<\underline{\vdash},<>\ ,<>> \end{array} \tag{3.15}$$

Более того, Sec – р-вычислимый предикат по замечанию 2.3.6.

Определим предикат $Axiom$, который будет выделять множество всех списочных кодов для аксиом [11] следующим образом:
1) аксиоме $\Phi \vdash \Phi$, Φ – формула ИП, сопоставим список $< \underline{\vdash}, \underline{\Phi^d}, < \underline{\Phi^d} >>$
2) аксиоме $\vdash x \approx x$, x – переменная, сопоставим список $< \underline{\vdash}, < \underline{=}, \underline{x^d}, \underline{x^d} > , <>>$
3) $x \approx y$, $(\Phi)^z_x \vdash (\Phi)^z_y$, x,y,z – переменные, Φ – формула ИП, удовлетворяющая условию на запись $(\Phi)^z_x$ сопоставим список:

$$< \underline{\vdash}, (\underline{\Phi^d})^{\underline{z^d}}_{\underline{y^d}}, << \underline{=}, \underline{x^d}, \underline{y^d} > , (\underline{\Phi^d})^{\underline{z^d}}_{\underline{x^d}} >>$$

Порождающее семейство F_{Axioms} имеет вид:

$$1)\ (x_1 = \underline{\vdash}) \& Formula(x_2) \& (NumElements(x_3) = 1) \& (first(x_3) = x_2)$$

$$2)\ (x_1 = \underline{\vdash}) \& Formula(x_2) \& first(x_2) = (\underline{=}) \&$$
$$\& Var(second(x_2)) \& (head(x_2) = second(x_2)) \& (x_3 = nil)$$

$$3)\ (x_1 = \underline{\vdash}) \& Formula(x_2) \& Formula(second(x_3)) \& (first(first(x_3)) = \underline{=}) \&$$
$$\& Var(second(first(x_3))) \& Var(head(first(x_3))) \&$$
$$\& (ChangeWithCond(second(x_3), x_2, second(first(x_3))),$$
$$head(first(x_3))) = second(x_3))$$

поэтому по замечанию 2.3.6 $Axiom$ – р-вычислимый предикат.

3.5 Доказательства ИП в виде дерева

Индуктивно определим предикат $Proof$, который будет выделять доказательства секвенций в виде дерева, аналогично тому как это описано в работе [11], но только здесь вместо секвенций и доказательств представлены их списочные кодировки.

Списочное представление доказательства для секвенции $\underline{C}$, которая является аксиомой, будет иметь вид:

$$\underline{C^{pr}} : < \underline{C}, <>>$$

Списочное представление доказательства для секвенции $\underline{C}$, которая получается по одному из правил вывода из секвенций $\underline{C_1}, \ldots, \underline{C_n}$ будет иметь вид:

$$\underline{C^{pr}} :< \underline{C}, \underline{C_1^{pr}}, \ldots, \underline{C_n^{pr}} >$$

где $\underline{C_1^{pr}}, \ldots, \underline{C_n^{pr}}$ – списочные представления доказательств секвенций $\underline{C_1}, \ldots, \underline{C_n}$ соответственно.

Для удобства и наглядности, вначале будем выписывать правило вывода из [11], а уже ему сопоставлять соответствующую L-формулу из F_{Proof}.

1) $\dfrac{\Gamma \vdash \Phi;\ \Gamma \vdash \Psi}{\Gamma \vdash \Phi \& \Psi}$:

$$\begin{aligned} &Sec(x_1)\&Proof(x_2)\&Proof(x_3)\& \\ &\quad \&(head(x_1) = head(first(x_2)))\&(head(x_1) = head(first(x_3)))\& \\ &\quad \&(first(second(x_1)) = \underline{\&})\&(second(second(x_1)) = second(first(x_2)))\& \\ &\quad \&(head(second(x_1)) = second(first(x_3))) \end{aligned}$$

2) $\dfrac{\Gamma \vdash \Phi \& \Psi}{\Gamma \vdash \Phi}$:

$$\begin{aligned} &Sec(x_1)\&Proof(x_2)\&(head(x_1) = head(first(x_2)))\& \\ &\quad \&(first(second(first(x_2))) = \underline{\&})\&(second(second(first(x_2))) = second(x_1)) \end{aligned}$$

3) $\dfrac{\Gamma \vdash \Phi \& \Psi}{\Gamma \vdash \Psi}$:

$$\begin{aligned} &Sec(x_1)\&Proof(x_2)\&(head(x_1) = head(first(x_2)))\& \\ &\quad \&(first(second(first(x_2))) = \underline{\&})\&(head(second(first(x_2))) = second(x_1)) \end{aligned}$$

4) $\dfrac{\Gamma \vdash \Phi}{\Gamma \vdash \Phi \vee \Psi}$:

$$\begin{aligned} &Sec(x_1)\&Proof(x_2)\&(head(x_1) = head(first(x_2)))\& \\ &\quad \&(first(second(x_1)) = \underline{\vee})\&(second(second(x_1)) = second(first(x_2))) \end{aligned}$$

5) $\dfrac{\Gamma \vdash \Psi}{\Gamma \vdash \Phi \vee \Psi}$:

$$\begin{aligned} &Sec(x_1)\&Proof(x_2)\&(head(x_1) = head(first(x_2)))\& \\ &\quad \&(first(second(x_1)) = \underline{\vee})\&(head(second(x_1)) = second(first(x_2))) \end{aligned}$$

6) $\dfrac{\Gamma,\Phi \vdash X;\ \Gamma,\Psi \vdash X;\ \Gamma \vdash \Phi \vee \Psi}{\Gamma \vdash X}$:

$$Sec(x_1)\&Proof(x_2)\&Proof(x_3)\&Proof(x_4)\&$$
$$\&(head(x_1) = tail(head(first(x_2))))\&(head(x_1) = tail(head(first(x_3))))\&$$
$$\&(head(x_1) = head(first(x_4)))\&$$
$$\&(cons(cons(cons(nil,\underline{\vee}),head(head(first(x_2)))),head(head(first(x_3)))) =$$
$$= second(first(x_4)))\&$$
$$\&(second(x_1) = second(first(x_2)))\&(second(x_1) = second(first(x_3)))$$

7) $\dfrac{\Gamma,\Phi \vdash \Psi}{\Gamma \vdash \Phi \to \Psi}$:

$$Sec(x_1)\&Proof(x_2)\&(head(x_1) = tail(head(first(x_2))))\&$$
$$\&(first(second(x_1)) = \underline{\to})\&(second(second(x_1)) = head(head(first(x_2))))\&$$
$$\&(head(second(x_1)) = second(first(x_2)))$$

8) $\dfrac{\Gamma \vdash \Phi;\ \Gamma \vdash \Phi \to \Psi}{\Gamma \vdash \Psi}$:

$$Sec(x_1)\&Proof(x_2)\&Proof(x_3)\&(head(x_1) = head(first(x_2)))\&$$
$$\&(head(x_1) = head(first(x_3)))\&(first(second(first(x_3))) = \underline{\to})\&$$
$$\&(second(second(first(x_3))) = second(first(x_2)))\&$$
$$\&(head(second(first(x_3))) = second(x_1))$$

9) $\dfrac{\Gamma,\neg\Phi \vdash}{\Gamma \vdash \Phi}$:

$$Sec(x_1)\&Proof(x_2)\&(head(x_1) = tail(head(first(x_2))))\&$$
$$\&(first(head(head(first(x_2)))) = \underline{\neg})\&$$
$$\&(second(head(head(first(x_2)))) = second(x_1))\&$$
$$\&(second(first(x_2)) = nil)$$

10) $\dfrac{\Gamma \vdash \Phi;\ \Gamma \vdash \neg\Phi}{\Gamma \vdash}$:

$$Sec(x_1)\&Proof(x_2)\&Proof(x_3)\&(head(x_1) = head(first(x_2)))\&$$
$$\&(head(x_1) = head(first(x_3)))\&(second(x_1) = nil)\&$$
$$\&(first(second(first(x_3))) = \underline{\neg})\&$$
$$\&(second(second(first(x_3))) = second(first(x_2)))$$

11) $\dfrac{\Gamma,\Phi,\Psi,\Gamma_1 \vdash X}{\Gamma,\Psi,\Phi,\Gamma_1 \vdash X}$:

$$Sec(x_1)\&Proof(x_2)\&(second(x_1) = second(first(x_2)))\&$$
$$\&(\exists k < NumElements(head(x_1))$$
$$(getElement(head(x_1),k) \neq getElement(head(x_1),s(k)))\&$$
$$\&changeListElements(head(x_1),k,s(k)) = head(first(x_2)))$$

12) $\dfrac{\Gamma \vdash \Phi}{\Gamma,\Psi \vdash \Phi}$:

$$Sec(x_1)\&Proof(x_2)\&(second(x_1) = second(first(x_2)))\&$$
$$\&(tail(head(x_1)) = head(first(x_1)))$$

13) $\dfrac{\Gamma \vdash \Phi}{\Gamma \vdash \forall x\ \Phi}$, где x не входит в члены Γ свободно:

$$Sec(x_1)\&Proof(x_2)\&(first(second(x_1)) = \underline{\forall})\&$$
$$\&(head(second(x_1)) = second(first(x_2)))\&$$
$$\&(\forall w \in head(x_1)\ \neg FormulaFreeVar(w,second(second(x_1))))$$

14) $\dfrac{\Gamma,(\Phi)_t^x \vdash \Psi}{\Gamma,\forall x\Phi \vdash \Psi}$:

$$Sec(x_1)\&Proof(x_2)\&(second(x_1) = second(first(x_2)))\&$$
$$\&(tail(head(x_1)) = tail(head(first(x_2))))\&$$
$$\&(first(head(head(x_1))) = \underline{\forall})\&$$
$$\&(change(head(head(head(x_1))),second(head(head(x_1))),$$
$$gt(head(head(head(x_1))),head(head(first(x_2))),second(head(head(x_1))))$$
$$) = head(head(first(x_2))))$$

15) $\dfrac{\Gamma \vdash (\Phi)_t^x}{\Gamma \vdash \exists x\Phi}$:

$$Sec(x_1)\&Proof(x_2)\&(head(x_1) = head(first(x_2)))\&$$
$$\&(first(second(x_1)) = \underline{\exists})\&$$
$$\&(change(head(second(x_1)),second(second(x_1)),$$
$$gt(head(second(x_1)),second(first(x_2)),second(second(x_1)))$$
$$) = second(first(x_2)))$$

16) $\dfrac{\Gamma,\Phi \vdash \Psi}{\Gamma,\exists x\Phi \vdash \Psi}$, где x не входит в Ψ и члены Γ свободно:

$$\begin{aligned}
&Sec(x_1)\&Proof(x_2)\&(second(x_1) = second(first(x_2)))\& \\
&\quad\&(tail(head(x_1)) = tail(head(first(x_2))))\& \\
&\quad\&(head(head(head(x_1))) = head(head(first(x_2))))\& \\
&\quad\&(first(head(head(x_1))) = \exists)\& \\
&\quad\&\neg FormulaFreeVar(second(first(x_2)), second(head(head(x_1))))\& \\
&\quad\&(\forall y \in tail(head(x_1))\ \neg FormulaFreeVar(y, second(head(head(x_1)))))
\end{aligned}$$

Теорема 3.12. *$Proof$ – p-вычислимый предикат.*

Доказательство. Покажем, что следующая GNF-система

$$G = (\Sigma,\sigma,\Omega,L,HW(\mathfrak{M}),\{Proof\},\{F_{Proof}\},\{\gamma\})$$

является p-вычислимой.

Начальные условия для предиката $Proof$ – p-вычислимы, так как предикат $Axiom$ – p-вычислим.

1) Свойство предикатной отделимости и позитивности выполнено. Это видно по виду порождающих L-формул.

2) Свойство равномерной ограниченности следует из конечного числа порождающих L-формул.

3) Свойство единственности порождения также легко проверяется, так как каждая L-формула порождает уникальные элементы. Часть формул порождает 2-х элементные списки, часть 3-х и часть 4-х. При этом в каждом случае есть еще некоторое уникальное отличие. Например, в 1) это символ $\underline{\&}$, который находится как $first(second(x_1))$, у других L-формул другие уникальные отличия. По этим отличиям функция γ и строит подходящую порождающую формулу для элемента w. Так как формул конечное число, то функция γ легко задается подходящей L-формулой.

4) Свойство p-вычислимости вытекает из конечного числа порождающих L-формул и единственности порождения.

Осталось применить обобщенную PAG-теорему с p-вычислимыми начальными условиями и получим, что $Proof$ – p-вычислимый предикат. □

3.6 Линейные доказательства ИП

Покажем, что множество линейных доказательств в ИП [11] также имеет полиномиально вычислимое представление. Для этого заведем предикат $lProof$, который будет задаваться с помощью своего порождающего семейства следующим образом.

Пусть $\underline{C_i}$ – списочный код секвенции C_i. Тогда все L-формулы из порождающего семейства F_{Proof} теоремы 3.12 можно преобразовать в L-формулы порождающего семейства F_{lProof} по следующему алгоритму:

1) Везде вместо предиката $Proof$ будет стоять предикат Sec.

2) Вместо $first(x_i)$, где x_i удовлетворяет $Proof(x_i)$, теперь будет стоять x_i.

Например, 1-я L-формула семейства F_{Proof} преобразуется в L-формулу семейства F_{lProof} следующим образом:

1) $\dfrac{\Gamma \vdash \Phi;\ \Gamma \vdash \Psi}{\Gamma \vdash \Phi \& \Psi}$:

$$\begin{aligned}\Phi_1 :\ & Sec(x_1)\&Sec(x_2)\&Sec(x_3)\& \\ & \&(head(x_1) = head(x_2))\&(head(x_1) = head(x_3))\& \\ & \&(first(second(x_1)) = \underline{\&})\&(second(second(x_1)) = second(x_2))\& \\ & \&(head(second(x_1)) = second(x_3))\end{aligned}$$

вместо:

$$\begin{aligned}& Sec(x_1)\&Proof(x_2)\&Proof(x_3)\& \\ & \&(head(x_1) = head(first(x_2)))\&(head(x_1) = head(first(x_3)))\& \\ & \&(first(second(x_1)) = \underline{\&})\&(second(second(x_1)) = second(first(x_2)))\& \\ & \&(head(second(x_1)) = second(first(x_3)))\end{aligned}$$

Семейство F_{lProof} состоит из L-формул Φ_j, $j \in [1,\ldots,16]$.

Следствие 3.12.1. *$lProof$ – p-вычислимый предикат.*

Доказательство. Пусть $l =< \underline{C_0},\ldots,\underline{C_n} >$ – некоторое линейное доказательство. В силу того, что $lProof$ единственный расширяемый предикат, ни в какую L-формулу порождающего семейства F_{lProof} предикат $lProof$ не входит и L-формул порождающего семейства конечное число, то все эти L-формулы

можно записать с помощью одной L-формулы следующего вида:

$$\forall i \leqslant NumElements(l)$$
$$\exists i_1,\ldots,i_{k_1} \leqslant i\ \Phi_1(getElement(l,i),getElement(l,i_1),\ldots,getElement(l,i_{k_1}))\vee$$
$$\ldots$$
$$\vee\exists i_1,\ldots,i_{k_{16}} \leqslant i\ \Phi_{16}(getElement(l,i),getElement(l,i_1),\ldots,getElement(l,i_{k_{16}}))$$

Эта L-формула и задает предикат $lProof$. □

3.7 Порождающие грамматики

Под порождающей грамматикой [16; 39] (далее просто грамматика) будем понимать четверку следующего вида $G = (V,N,S,P)$, где V - конечный алфавит, называемый терминальным алфавитом, N - конечный алфавит, называемый нетерминальным алфавитом, причем пересечение V и N пусто; S - выделенный символ нетерминального алфавита, называемый аксиомой; P - конечное множество правил вывода или продукций. Обозначим за D объединение терминальных и нетерминальных алфавитов V и N. Порождающая грамматика позволяет выводить (порождать) цепочки языка из некоторой начальной цепочки с помощью определенных правил замены. Каждое правило вывода в порождающей грамматике является упорядоченной парой (α,β) цепочек в алфавите $V \cup N$, причем цепочка α должна содержать вхождение хотя бы одного символа нетерминального алфавита; цепочку α называют левой, цепочку β – правой частью правила вывода. Правило вывода принято записывать в таком виде: $\alpha \to \beta$.

Цепочка δ непосредственно выводима в грамматике G из цепочки γ (обозначение $\gamma \vdash_G \delta$ или просто $\gamma \vdash \delta$), если существуют такие цепочки σ и ρ и такое правило вывода $\alpha \to \beta$ из P, что $\gamma = \sigma\alpha\rho$, а $\delta = \sigma\beta\rho$.

Под отношением выводимости (обозначение $\gamma \vdash^*_G \delta$ или просто $\gamma \vdash^* \delta$) понимается выводимость цепочки δ из цепочки γ в грамматике G так, что существуют цепочки $\alpha_0 = \gamma$, $\alpha_1,\ldots,$ $\alpha_n = \delta$, где $n \geqslant 0$ для которых выполняется следующее:

$$\alpha_0 \vdash \alpha_1 \vdash \cdots \vdash \alpha_{n-1} \vdash \alpha_n \tag{3.16}$$

Язык, порождаемый грамматикой G, – это множество $L(G)$ всех выводимых из аксиомы грамматики терминальных цепочек вида:

$$L(G) = \{x : S \vdash_G^* x, x \in V^*\} \tag{3.17}$$

Обозначим за $L(G)_{\vdash}^{V^*}$ множество выводов из аксиомы S цепочек $x \in V^*$ закодированных в виде списков следующего вида:

$$L(G)_{\vdash}^{V^*} = \{<<< \cdots << S >, \alpha_1 >, \cdots >, \alpha_{n_x-1} >, x >: \\ S \vdash \alpha_1 \vdash \cdots \vdash \alpha_{n_x-1} \vdash x\} \tag{3.18}$$

Обозначим за $L(G)_{\vdash}^{D^*}$ множество выводов из аксиомы S цепочек вида $x \in D^*$:

$$L(G)_{\vdash}^{D^*} = \{<<< \cdots << S >, \alpha_1 >, \cdots >, \alpha_{n_x-1} >, x >: \\ S \vdash \alpha_1 \vdash \cdots \vdash \alpha_{n_x-1} \vdash x\} \tag{3.19}$$

Пусть $HW(M)$ есть p-вычислимая модель сигнатуры σ, где $M = D^*$. Добавим элементы из D в сигнатуру σ в качестве константных символов с естественной интерпретацией в основное множество. Более того, с помощью функции конкатенации строк, которая определена только для элементов из множества D, для любой цепочки $\alpha \in D^*$ можно построить термальное подходящее выражение α интерпретируемое в элемент α.

Определим предикат Q:

$$Q(\alpha,\beta) = \begin{cases} true, \text{ если } \alpha \vdash_G \beta \\ false, \text{ иначе} \end{cases}$$

Утверждение 3.13. *Q - p-вычислимый предикат.*

Доказательство. Грамматика G содержит конечное число правил вывода, поэтому может быть применен следующий алгоритм. Введем следующее обозначение $\alpha \vdash_G^{i,\sigma\to\delta} \beta$ которое означает, что цепочка β получается из α при применении правила вывода $\sigma \to \delta$ к цепочке α с i-го символа. Обозначим за $s(\alpha,i,\sigma,\delta)$ следующую функцию:

$$s(\alpha,i,\sigma,\delta) = \begin{cases} \beta, \text{ если } \alpha \vdash_G^{i,\sigma\to\delta} \beta \\ false, \text{ иначе} \end{cases}$$

Не сложно видеть, что найдутся такие константы C и p, что для всех правил вывода вида $\sigma_j \to \delta_j$ грамматики G вычислительная сложность функции s не превосходит $C \cdot |\alpha|^p$.

Для каждой цепочки символов δ в грамматике G можно сопоставить терм δ сигнатуры σ. Далее, по всем правилам вывода вида $\sigma_i \to \delta_i$ грамматики G можно сконструировать терм l сигнатуры σ, который содержит всю информацию о правилах вывода грамматики G:

$$l = cons(\ldots cons(cons(nil,\underline{< \sigma_1,\delta_1 >}),\ldots,\underline{< \sigma_n,\delta_n >}) \tag{3.20}$$

где $\underline{< \sigma_i,\delta_i >}$ кодировка правила $\sigma_i \to \delta_i$ в виде списка $cons(cons(nil,\sigma_i),\delta_i)$.

Следующая L-программа, задаваемая с помощью условного терма $Cond$ [6], является характеристической функцией предиката Q. Она сравнивает две цепочки α, β и выдает результат 1, если цепочка β непосредственно следует из α и 0 иначе.

$$Cond(1, \Phi(\alpha,\beta),0)$$

где $\Phi(\alpha,\beta)$ имеет вид

$$\exists w \in l\ \exists i \leqslant |\alpha|\ s(\alpha, i, head(tail(w)), head(w)) = \beta$$

Из того что любая L-программа имеет полиномиальную вычислительную сложность, следует утверждение нашей теоремы. □

Добавим этот предикат Q в нашу модель.

Теорема 3.14. *$L(G)^{D^*}_{\vdash}$ из (3.19) является p-вычислимым множеством.*

Доказательство. Пусть $G = (V,N,S,P)$ грамматика и $HW(\mathfrak{M})$ есть p-вычислимая модель сигнатуры σ, где $M = \{V \cup N\}^*$. Так как грамматика G содержит только одну аксиому S и набор правил P в ней конечен, то добавим в модель p-вычислимый предикат $Axiom$ для выделения аксиомы S.

Для нового предиката $L(G)^{D^*}_{\vdash}$ выделяющего первоначально пустое множество элементов в $HW(M)$ построим порождающее семейство $F_{L(G)^{D^*}_{\vdash}}$, которое будет состоять из 2-х L-формул следующего вида:

$$F_{L(G)^{D^*}_{\vdash}} = \{Axiom(x_1), L(G)^{D^*}_{\vdash}(x_1)\&Q(head(x_1),x_2)\} \tag{3.21}$$

Так как семейство состоит из конечного числа порождающих L-формул, то свойство равномерной ограниченности выполнено автоматически. Свойство предикатной отделимости также выполнено, так как здесь всего один расширяемый предикат $L(G)^{D^*}_{\vdash}$. Более того, его вхождения в L-формулы порождающего семейства если есть, то позитивны и только в виде $L(G)^{D^*}_{\vdash}(x_i)$. Условных и p-итерационых термов нет в L-формулах порождающего семейства, поэтому в их

построениях расширяемый предикат также не участвует. Порождающее семейство $F_{L(G)^{D^*}_{\vdash}}$ из (3.21) порождает все слова вида (3.19).

Функция γ имеет вид:

$$\gamma(w) = \begin{cases} Axiom(w_1), \text{ если } w =< w_1 > \\ L(G)^{D^*}_{\vdash}(w_1)\&Q(head(w_1),w_2), \text{ если } w =< w_1,w_2 > \\ false, \text{ иначе} \end{cases} \tag{3.22}$$

Как видно из построения (3.22) функция $\gamma(x)$ является p-вычислимой.

Тем самым была задана p-вычислимая GNF-система и, после применения обобщенной PAG-теоремы 2.5 получаем, что $L(G)^{D^*}_{\vdash}$ является p-вычислимым множеством. □

Теорема 3.15. *$L(G)^{V^*}_{\vdash}$ из (3.18) является p-вычислимым множеством.*

Доказательство. В теореме 3.14 было показано, что предикат $L(G)^{D^*}_{\vdash}$ из (3.19) является p-вычислимым.

Определим p-вычислимый предикат $Terminal$ как:

$$Terminal(w) = \begin{cases} true, \text{ если } w \in V^* \\ false, \text{ иначе} \end{cases}$$

Построим L-программу с помощью условного терма $Cond$ из [6] следующим образом:

$$Cond(1, Terminal(head(x))\&L(G)^{D^*}_{\vdash}(x), 0) \tag{3.23}$$

В силу того, что любая L-программа (3.23) имеет полиномиальную вычислительную сложность и в дополнение к этому является характеристической функцией множества $L(G)^{V^*}_{\vdash}$ получаем, что предикат $L(G)^{V^*}_{\vdash}$ является p-вычислимым. □

Глава 4. Объектно-ориентированный язык L*

Результаты этой главы являются важным практическим применением теоретических результатов полученных в предыдущих главах. В данной главе будет представлен новый высокоуровневый объектно-ориентированный язык программирования L*, который является консервативным расширением p-полного языка L. Таким образом L*-программы могут применяться для реализации любых алгоритмов полиномиальной вычислительной сложности в таких направлениях как искусственный интеллект, робототехника и умные контракты. Более того, синтаксис данного языка максимально приближен к синтаксису таких популярных языков программирования как C++, PHP, JavaScript, что обеспечивает программистам быстрый вход в разработку.

4.1 L*-программы и виртуальные машины.

Пусть $HW(\mathfrak{M})$ – p-вычислимая наследственно-конечная списочная надстройка сигнатуры σ. Под виртуальной машиной будем подразумевать следующую тройку:

$$V = (HW(\mathfrak{M}),\sigma, < Fl,Cl,Hl,Sl,Ll >)$$

где Fl будем называть списком L*-определимых функций, Cl – списком L*-определимых классов, Hl – вспомогательный список, Sl – список, выполняющий роль экрана для виртуальной машины V, Ll – список, выполняющий роль ленты для виртуальной машины V.

Взаимной индукцией определим понятие L(V)-программы, L(V)-формулы, а также L*-программы.

Индуктивно определим понятие L(V)-программы:

(1) $t(x)$ – L(V)-программа, если $t(x)$ – L-программа

(2) $f(t_1,\ldots,t_n)$ – L(V)-программа, если $t_1,\ldots,t_n$ – L(V)-программы и $f \in \sigma$

(3) $f(t_1,\ldots,t_n)$ – L(V)-программа, если $t_1,\ldots,t_n$ – L(V)-программы и функция f задается следующим образом:

$$[mod]\ function\ f(x_1,\ldots,x_n)\{p\}$$

где p – L*-программа, являющаяся телом функции, $mod \in \{public, protected, private\}$ – необязательный модификатор доступа. Более детальное описание смотреть в параграфе 4.2

(4) $Cond(t_1, \varphi_1, \dots)$ – L(V)-программа, если $t_1, \dots, t_{n+1}$ – L(V)-программы и $\varphi_1, \dots, \varphi_n$ – L(V)-формулы

(5) $Iteration_{g,\varphi}(t_1, t_2)$ – L(V)-программа, если g – L-программа, t_1 и t_2 – L(V)-программы, φ – L-формула

(6) $obj.m(t_1, \dots, t_k)$ – L(V)-программа, если obj – это объект некоторого класса $Class$ у которого есть метод $m(x_1, \dots, x_k)$, $t_1, \dots, t_{n+1}$ – L(V)-программы. Более детальное описание смотреть в параграфе 4.3

Индуктивно определим понятие L(V)-формулы:

(1) $\varphi(x)$ – L(V)-формула, если $\varphi(x)$ – L-формула

(2) $\varphi : P(t_1, \dots, t_n)$ – L(V)-формула, если $t_1, \dots, t_n$ – L(V)-программы и $P \in \sigma$

(3) $\varphi : t_1 = t_2$ – L(V)-формула, если t_1, t_2 – L(V)-программы

(4) $\varphi : \varphi_1 \delta \varphi_2$ – L(V)-формула, если φ_1, φ_2 – L(V)-формулы и $\delta \in \{\&, \vee, \rightarrow\}$

(5) $\varphi : \neg\varphi_1$ – L(V)-формула, если φ_1 – L(V)-формула

(6) $\varphi : \exists x \delta t \varphi_1$ and $\forall x \delta t \varphi_1$ – L(V)-формулы, если $\delta \in \{\in, \subseteq, \leqslant\}$, t – L(V)-программа, φ_1 – L(V)-формула

Индуктивно определим понятие L*-программы:

(1) $y := t(x);$ – L*-программа, если $t(x)$ – L(V)-программа

(2) $this.y := t(x);$ – L*-программа, если $t(x)$ – L(V)-программа

(3) $return\ t(x);$ – L*-программа, если $t(x)$ – L(V)-программа

(4) $print(t(x));$ – L*-программа, если $t(x)$ – L(V)-программа

(5) $p_1 \cdot p_2$ – L*-программа, если p_1, p_2 – L*-программы и $\cdot$ – операция конкатенации

(6) если p^* – L*-программа задающая функцию $Func$ (подробнее 4.2) тогда:

$$[mod]\ function\ Func(x_1, \dots, x_k)\ \{p^*\} \tag{4.1}$$

– L*-программа, где $mod \in \{public, protected, private\}$ – модификатор доступа

(7) если p^* – L*-программа задающая класс (подробнее 4.3) тогда:

$$class\ ClassName\ [extends\ parentClass]\{p^*\} \tag{4.2}$$

– L*-программа.

(8) инициализация объекта:
если $t_1, \ldots, t_k$ – L(V)-программы, тогда:

$$object \ := \ new \ ClassName(t_1, \ldots, t_k); \tag{4.3}$$

– L*-программа. Более детальное описание смотреть в параграфе 4.3.

Будем говорить, что функция f – L*-определима, если она определяется синтаксической конструкцией вида (4.1).

Будем говорить, что класс $Class$ – L*-определим, если он определяется синтаксической конструкцией вида (4.2).

Также будем называть L(V)-формулы и L(V)-программы нестандартными (или нестандартными L(V)-конструкциями), если в их построении участвуют L*-определимые функции или объекты, использующие методы или свойства L*-определимых классов. Синтаксическую конструкцию инициализиции объекта класса (4.3) также будем относить к нестандартным синтаксическим конструкциям.

4.2 L*-определимые функции

Функция в языке L* будет задаваться с помощью специального слова $function$, после которого идет имя функции и набор входящих параметров. Далее, в фигурных скобках стоит некоторая специальная L*-программа p^*, которая называется телом функции.

Синтаксическая конструкция задания функции:

$$[mod] \ function \ Func(x_1, \ldots, x_k) \ \{p^*\}$$

Внутри L*-программы p^* не могут определяться другие функции и классы.

Запись $[mod]$ обозначает необязательный параметр модификатора $mod \in \{public|protected|private\}$, который добавляется только в случае, когда функция выступает в роли метода класса (в случае отсутствия модификатора считаем, что стоит $public$).

4.3 Классы, свойства, методы, объекты

В этой главе, как и в большинстве высокоуровневых языков под классом будем понимать синтаксическую конструкцию которая описана следующим образом. Вначале идет специальное слово *class*, затем идет имя класса, далее необязательное наследование свойств и методов родительского класса с помощью [*extends parentClass*] и, далее, в фигурных скобках L*-программа p^* задающая свойства и методы класса.

Синтаксическая конструкция задания класса:

$$class\ ClassName\ [extends\ parentClass]\{p^*\}$$

где L*-программа p^* тело класса $ClassName$.

Создание объекта класса определяется с помощью специального слова *new* после которого идет имя класса и набор параметров для инициализации объекта класса с помощью конструктора класса.

Синтаксическая конструкция инициализации объекта класса:

$$object := new\ Class(a_1, \ldots, a_k);$$

Для вызова метода $m_1(x)$ класса $Class$ в контексте объекта *object* используется запись вида $object.m_1(a)$ при некотором означивании a переменной x. Специальная переменная *this*, используется только внутри методов класса. Через эту переменную можно обращаться к свойствам и методам объекта, вызывающего текущий метод класса. Более того, каждое свойство или метод класса используют специальные модификаторы доступа:

$$mod \in \{public, private, protected\}$$

Если модификатор отсутствует, то по умолчанию считаем, что стоит *public*. Свойства и методы с модификатором *public* доступны из любого класса наследника. Также эти свойства и методы доступны, когда объект текущего класса или любого класса наследника вызывает их. Модификатор доступа *protected* используется, чтобы свойства или методы с этим модификатором были доступны только у классов наследников. Свойства и методы с модификатором *private* доступны только внутри самого класса и больше нигде.

В каждом классе может быть задан конструктор класса, который вызывается при инициализации объекта класса. Этот конструктор также может наследоваться классами наследниками, если стоит соответствующий модификатор доступа *public* или *protected*. Для задания конструктора класса используется специальное слово *constructor*.

4.4 Компиляция L*-программ

В этом параграфе будет рассмотрена компиляция L*-программ. Для этого, не нарушя общности будем считать везде далее, что все модификаторы доступа для функций и методов классов принимают значение *public*. Аналог рассуждения также можно будет применять, если рассматривать все модификаторы доступа вида:

$$mod \in \{public,private,protected\}$$

Пусть V – виртуальная машина, p – L*-программа. Определим функцию компиляции:

$$C :<< Fl,Cl > ,p > \rightarrow << Fl^*,Cl^* > ,p^* >$$

которая переводит L*-программу p в L*-программу p^* следующим образом:

1) В p последовательно удаляются куски кода, которые задают функцию или класс. Эти куски кода конвертируются в списки следующего вида:

$$< funcName,[mod], < x_1,\ldots,x_k > ,p^* > \tag{4.4}$$

где $mod \in \{public\}$ и

$$\begin{array}{l} < className, \\ \quad << y_1, < mod_{y_1} >> , \ldots , < y_m, < mod_{y_m} >>> , \\ \quad\quad << m_1,[mod_{m_1}], < x_1,\ldots,x_k > ,p_1^* > , \ldots , \\ \quad\quad\quad < m_k,[mod_{m_k}], < x_1,\ldots,x_k > ,p_k^* >>> \end{array} \tag{4.5}$$

где y_i - свойства класса, mod_{y_i} – соответствующие модификаторы доступа для них, m_j - методы класса, mod_j – соответствующие модификаторы доступа для них.

2) затем списки вида 4.4 и 4.5 добавляются соответственно как элементы в списки Fl и Cl. Эти списки после компиляции программы остаются неизменными при дальнейшем исполнении откомпилированной программы.

В конечном итоге L*-программа p^* не содержит никаких конструкций задающих функции или классы. Причем и сами L*-определимые функции и классы из Fl и Cl соответственно не содержат в себе задания других L*-определимых конструкций.

Замечание 4.4.1. *C – p-вычислимая функция.*

4.5 Исполнение L*-программ

Пусть $p^*(x_1,\ldots,x_n)$ – L*-программа, полученная в результате компиляции L*-программы $p(x_1,\ldots,x_n)$.

Индуктивно определим понятие исполнения L*-программы p^* виртуальной машиной V:

1) V добавляет в Ll элемент следующего вида:

$$<< Vl_{p^*},Ol_{p^*},Hl_{p^*},Rl_{p^*} > ,p^* > \tag{4.6}$$

где Vl_{p^*} - список инициализированных переменных, Ol_{p^*} - список инициализированных объектов, Hl_{p^*} – вспомогательный список, Rl_{p*} – список который хранит вычисленное значение.

Вначале Vl_{p^*} имеет вид:

$$Vl_{p^*} : << x_1,w_1 > , \ldots , < x_n,w_n >>$$

где w_i значения означенных входящих переменных x_i в L*-программе p^*. Заметим, что входящие переменные у L*-программ p и p* совпадают.

2) пусть на некотором шаге в списке Ll в качестве последнего элемента стоит элемент вида 4.6 для L*-программы q^*. Тогда V построчно считывает q^* и сохраняет вычисленные значения переменных в соответствующий список Vl_{q^*} пока не встретит нестандартную L(V)-программу которая не содержит в качестве параметров другие нестандартные L(V)-программы. Тогда V заменяет эту нестандартную L(V)-программу на новую, нигде не используемую до этого переменную z и добавляет эту переменную в Hl_{q^*} в виде $< var,z >$.

3) Если найденная нестандартная L(V)-программа имеет вид $f(t_1,\ldots,t_n)$, тогда новый элемент вида:

$$<< Vl_{q^*},Ol_{q^*},Hl_{q^*},Rl_{q^*} > ,q^* >$$

добавляется в конец списка Ll, где f – L*-определимая функция и q^* – тело функции f. В список же Vl_{q^*} добавляются вычисленные значения L-программ $t_1,\ldots,t_n$.

Если же нестандартная L(V)-программа имеет вид $object.m(t_1,\ldots,t_n)$, тогда новый элемент вида:

$$<< Vl_{q^*},Ol_{q^*},Hl_{q^*},Rl_{q^*} > ,q^* >$$

добавляется в конец списка Ll, где q^* – это тело метода m из L*-определимого класса которому принадлежит объект $object$. В список же Vl_{q^*} добавляются вычисленные значения L-программ $t_1,\ldots,t_n$. В список Ol_{q^*} копируется список, который содержит инициализируемые свойства объекта $object$.

Пусть $t_1,\ldots,t_n$ – L-программы. Если машина V считывает конструкцию вида:

$$object_1 = new\ ClassName(t_1,\ldots,t_n);$$

тогда она добавляет новый элемент вида:

$$<< Vl_{q^*},Ol_{q^*},Hl_{q^*},Rl_{q^*} > ,q^* >$$

в конец списка Ll, где q^* – тело конструктора класса $ClassName$ и в Vl_{q^*} добавляются вычисленные значение L-программ $t_1,\ldots,t_n$. В список Hl_{p^*} добавляется список вида $< object,z >$, где z – переменная, которая до этого не встречалась в программе q^*. Машина V также заменяет слово $ClassName(t_1,\ldots,t_n)$ на переменную z так, чтобы получилась запись вида:

$$object_1 = new\ z;$$

В дальнейшем, инициализированное значение z объекта $object_1$ будет записано в конец списка Ol_{q^*}, если до этого этот объект не был инициализирован и добавлен в Ol_{q^*}. В противном случае, значение инициализированного объекта будет перезаписано.

4) Рассмотрим другие варианты исполнения L*-программы q^* виртуальной машиной V:

(1) $y := t(x)$; – V добавляет $< y,w_1 >$ в Vl_{q^*}, где w_1 вычисленное значение L-программы $t(x)$.

(2) $this.y := t(x)$; – V берет первый элемент (объект) из списка Ol_{p^*} и свойству y приписывает значение w, где w – это вычисленное значение L-программы $t(x)$.

(3) $y := this.x$; – V берет первый элемент из списка Ol_{p^*} и переменной y приписывает значение свойства x объекта, списочный код которого содержится в первом элементе списка Ol_{p^*}.

(4) $return\ t(x)$; – V добавляет вычисленное значение w L-программы $t(x)$ в список Rl_{q^*} и вместо L*-программы p^* подставляет пустой список. Далее, если в списке Ll есть и другие элементы, тогда если в предыдущем элементе списка Ll в списке Hl_{r*} хранится запись вида $< var,y >$, то V добавляет элемент $< y,w >$ в список Vl_{r^*}, если же в Hl_{r*} хранится запись вида $< object,y >$, то V добавляет элемент $< y,w >$ в последний элемент списка Ol_{r^*}.

(5) $print(t(x))$; – V добавляет вычисленное значение L-программы $t(x)$ в список Sl.

Машина V останавливает свою работу только в следующих случаях:
1) в процессе исполнения откомпилированной программы появляется значение $false$, как значение при вычислении функции или метода некоторого класса. 2) Когда в списке Ll останется только один элемент и при этом, вместо L*-программы p^* будет стоять пустой список. Вычисленное же значение при этом будет храниться в списке Rl_{p^*}.

Замечание 4.5.1. *Машина V при работе над откомпилированной L*-программой p всегда останавливается.*

4.6 Сложность L*-программ

Взаимной индукцией определим понятие сложности для L(V)-программ, L(V)-формул и L*-программ.

Индуктивно определим сложность $r(t)$ для L(V)-программы t:

(1) $r(c) = 1$, где $c \in \sigma$

(2) $r(x) = 1$, где x – переменная.

(3) $r(f(t_1,\ldots,t_n)) = \sum_{i=1}^{n} r(t_i) + 1$, где $f \in \sigma$.

(4) $r(f(t_1,\ldots,t_n)) = \sum_{i=1}^{n} r(t_i) + r(p^*) + 1$, где $f \in Fl$ и p^*– тело L*-определимой функции f.

(5) $r(Cond(t_1,\varphi_1,\ldots)) = \sum_{i=1}^{n+1} r(t_i) + \sum_{i=1}^{n} r(\varphi_i) + 1$

(6) $r(Iteration_{g,\varphi}(t_1,t_2)) = r(g) + r(\varphi) + r(t_1) + r(t_2) + 1$.

(7) $r(object.m(t_1,\ldots,t_n)) = \sum_{i=1}^{n} r(t_i) + r(p^*) + 1$,

m – метод класса, которому принадлежит объект $object$, p^* – тело метода m.

Индуктивно определим сложность для L(V)-формул:

(1) $r(\varphi) = \sum_{i=1}^{n} r(t_i) + 1$, если $\varphi : P(t_1,\ldots,t_n)$ $(P \in \sigma)$

(2) $r(\varphi) = r(t_1) + r(t_2) + 1$, если $\varphi : t_1 = t_2$

(3) $r(\varphi) = r(\varphi_1) + r(\varphi_2) + 1$, если $\varphi : \varphi_1 \delta \varphi_2$, где $\delta \in \{\&, \vee, \rightarrow\}$

(4) $r(\varphi) = r(\varphi_1) + 1$, если $\varphi : \neg\varphi_1$

(5) $r(\varphi) = r(t) + r(\varphi_1) + 1$, если $\varphi : \exists x \delta t \varphi_1$ или $\varphi : \forall x \delta t \varphi_1$,
где $\delta \in \{\in, \subseteq, \leqslant\}$.

Индуктивно определим сложность для L*-программ:

(1) $r(p) = r(t) + 1$, если $p :\ y := t(x)$;

(2) $r(p) = r(t) + 1$, если $p :\ this.y := t(x)$;

(3) $r(p) = r(t) + 1$, если $p :\ return\ t(x)$;

(4) $r(p) = r(t) + 1$ если $p :\ print(t(x))$;

(5) $r(p) = r(p_1) + r(p_2) + 1$, если $p :\ p_1 \cdot p_2$

(6) $r(p) = \sum_{i=1}^{n} r(t_i) + r(p^*) + 1$, if

$$p :\ object\ =\ new\ ClassName(t_1,\ldots,t_k);$$

где p^* – тело конструктора класса $ClassName$.

4.7 Консервативность расширения

Будем говорить, что машина V делает шаг, если в процессе исполнения программы она выполняет одно из следующих условий:

1) При считывании нестандартной L(V)-программы, в которую в качестве параметров не входят другие нестандартные L(V)-программы, машина V заменяет

ее на новую переменную и, далее, выписывает тело данной L(V)-программы с соответствующими параметрами на ленту Ll.
2) При считывании конструкции $return\ t(x)$; – V подсчитывает значение $t(x)$ и записывает его в соответствующий список Rl_{q^*}. Если же в списке Ll есть еще элементы, то V возвращает это значение в предпоследний элемент в соответствии с правилами исполнения, описанными в параграфе 4.5. В противном случае машина V подставляет вместо L*-программы q^* пустой список и останавливает свою работу.
3) Если V считывает первую строку L*-программы, в которую не входят нестандартные L(V)-конструкции, то в зависимости от вида этой строки она делает соответствующие преобразования, которые описаны в параграфе 4.5.

Пусть p некоторая L*-программа. Пусть V – виртуальная машина с начальными состояниями, полученными в результате компиляции L*-программы p.

Лемма 4.1. *(о шагах) Пусть V – виртуальная машина с заданными списками Fl и Cl, полученными в результате компиляции L*-программы p, а p^*– соответствующая откомпилированная программа. Тогда существует константа C такая, что при любых означиваниях входящих переменных x_i L*-программы p^* количество шагов машины V при работе над p^* не превосходит C.*

Доказательство. Это утверждение докажем индукцией по сложности L*-программ q^* появляющихся на ленте Ll в виде 4.6 для машины V с заданными списками Fl и Cl.

Для $r(q^*) = 1$ – утверждение верно, в силу отсутствия таких программ.

Предположим, что индукционное предположение верно для любых L*-программ q^* сложности не выше чем n. Покажем это для L*-программы p^* такой, что $r(p^*) = n + 1$.

Если в L*-программе p^* в первой строке не содержится никаких нестандартных L(V)-конструкций, то, без ограничения общности, считаем, что первая строка в p^* имеет вид $y := t(t_1, \ldots, t_n))$, где t и $t_1, \ldots, t_n$ – L-программы. Тогда машина V считывает первую строку и присваивает вычисленное значение переменной y. Далее, записывает это значение в Vl_{p^*} и удаляет первую строку из программы p^* переходя к программе p^{**}. Сложность p^{**} ниже чем сложность p^* поэтому можно применить индукционное предположение.

Если же в L*-программе p^* в первой строке встречаются нестандартные L(V)-конструкции, то машина V находит первую такую конструкцию, в которую в качестве параметров не входят другие нестандратные L(V)-конструкции и заменяет ее на специальную переменную z. При этом L*-программа p^* преобразуется в L*-программу p^{**}. Далее, V добавляет новый элемент в конец списка Ll, в который входит L*-программа q^* являющаяся телом этой нестандартной L(V)-конструкции. Сложность q^* будет ниже, чем сложность p^*. В силу индукционного предположения, для q^* такая константа C_1 существует. Значит машина V сделает C_1 шагов и исполнит q^*. После этого V подставит это значение и, далее, необходимо уже исполнить за C_2 шагов преобразованную программу p^{**} у которой сложность также ниже чем у p^*. Общее число шагов при исполнении L*-программы p^* не превосходит $C_1 + C_2$. □

Лемма 4.2. *(об ограничениях) Пусть V – виртуальная машина с заданными списками Fl и Cl, полученными в результате компиляции L*-программы $p(x_1,\ldots,x_n)$. Тогда существуют константы C и k такие, что для любых инициализаций w_i входящих переменных x_i, любой конфигурации машины V при работе над p^* выполнялось следующее неравенство:*

$$|l_i| \leqslant C \cdot | < w_1,\ldots,w_n > |^k$$

где l_i – значение некоторой переменной z, которое вычисляется в процессе исполнения машиной V L*-программы p^*.

Доказательство. Доказательство проведем индукцией по сложности L*-программы q^*, которая появляется в списке вида 4.6. Этот список, в свою очередь, является последним элементом в списке Ll.
Шаг индукции: $r(q^*) = 1$: утверждение верно в силу отсутствия таких программ.
Шаг индукции: пусть утверждение верно для любой L*-программы q^* сложности не выше чем n, покажем это для L*-программы p^{**} сложности $n+1$.

Пусть $r(p^*) = n+1$. И пусть последний списочный элемент, содержащий p^*, в списке Ll имеет вид:

$$<< Vl_{p^*},Ol_{p^*},Hl_{p^*},Rl_{p^*} > ,p^* > \tag{4.7}$$

где Vl_{p^*} содержит списки вида $< x_i,w_i >$.

Машина V считывает построчно L*-программу p^*.

1) Если первой строкой встретилась нестандартная L(V)-программа не содержащая других нестандартных L(V)-программ, то V заменяет ее на новую переменную z и выписывает новый элемент вида 4.7 только для q^* в конец списка Ll. В этом новом элементе будет содержаться L*-программа q^* являющаяся телом этой нестандратной L(V)-конструкции. Заметим, сложность q^* ниже p^*. В силу индукционного предположения получим, что существуют C_1 и k_1 такие, что новые значения для переменных, получаемые в процессе исполнения L*-программы q^* ограничены значениями l_i, которыми были означены входящие переменные программы q^*:

$$|m_i| \leqslant C_1 \cdot | < l_1, \ldots, l_n > |^{k_1}$$

но в силу того, что существуют C_2 и k_2 такие, что:

$$|l_i| \leqslant C_2 \cdot | < w_1, \ldots, w_k > |^{k_2}$$

получаем, что существуют C_{q^*} и k_{q^*} такие, что верно

$$|m_i| \leqslant C_{q^*} \cdot | < w_1, \ldots, w_k > |^{k_{q^*}}$$

После исполнения L*-программы q^* в p^* будет подставлено значение b вместо переменной z, которое удовлетворяет следующему неравенству:

$$|b| \leqslant C_{q^*} \cdot | < w_1, \ldots, w_k > |^{k_{q^*}}$$

При этом L*-программа p^* преобразуется в программу меньшей сложности p^{**}, для которой можно уже применить индукционное предположение, но для инициализированных переменных:

$$< x_1, w_1 > , \ldots , < x_k, w_k > , < z, b >$$

поэтому получаем, что существуют константы $C_{p^{**}}$ и $k_{p^{**}}$, что вычисляемое значение v любой новой переменной y в программе p^{**} удовлетворяет неравенству:

$$|v| \leqslant C_{p^{**}} \cdot | < w_1, \ldots, w_k, b > |^{k_{p^{**}}} \leqslant C_{p^{**}} \cdot (C_{q^*} \cdot | < w_1, \ldots, w_k > |^{k_{q^*}+1})^{k_{p^{**}}}$$

В силу того, что выбор L*-программ q^* и p^{**} не зависит от инициализированных входящих значений $w_1, \ldots, w_k$ переменных $x_1, \ldots, x_k$ в программе p^* получаем, что наше индукционное предположение верно.

2) Если же первая строка является инициализацией объекта и не содержит нестандратных L(V)-программ, тогда выписывается тело конструктора класса в элемент вида 4.7 и все рассуждения пункта 1) повторяются.

3) Если же в строке идет вычисление значения для переменной и в эту строку не входят никакие нестандартные L(V)-конструкции, то машина V просто высчитывает значение для этой переменной и заносит в Vl. Далее используется индукционное предположение.

□

Теорема 4.3. *(о p-вычислимости L^*-программ)*
Любая L^*-программа является p-вычислимой.

Доказательство. Рассмотрим p-вычислимую наследственно-конечную списочную надстройку $HW(M)$ сигнатуры σ такую, что любая L^*-программа p является элементом $HW(M)$. Пусть p^* – L^*-программа, которая получается в результате компиляции L^*-программы p.

Для доказательства этой теоремы достаточно построить модифицированный p-итерационный терм следующим образом:
1) g пошагово моделирует работу виртуальной машины V.
2) φ – проверяет список Ll. Как только останется один элемент и вместо начальной программы p^* будет стоять пустой список, то машина V останавливает свою работу.

Входящие значения для модифицированного p-итерационного терма $Iteration_{g,\varphi}$ являются w и C:
1) w имеет вид:

$$\begin{aligned} &<< Vl_{p^*}, Ol_{p^*}, Hl_{p^*}, Rl_{p^*} >, \\ &\quad < w_1, \ldots, w_n >, p^*, Fl, Cl, Sl, Ll > \end{aligned} \tag{4.8}$$

где Vl_{p^*} состоит из элементов $< x_1, w_1 >$, Ol_{p^*} и Hl_{p^*} – первоначально пустые списки.
2) По лемме 4.1 существует константа C ограничивающая общее число шагов машины V. Этой константой ограничим число итераций в $Iteration_{g,\varphi}$.

В силу того, что временная сложность затраченная на один шаг машины V является полиномиальной относительно длины списка вида (4.8). Отсюда следует, что q также p-вычислима.

Из р-вычислимости функции g и условий лемм 4.1 и 4.2 следует, что условия 1) и 2) для модифицированного р-итерационного терма выполнены, а следовательно L*-программа p является р-вычислимой. □

Теорема 4.4. *(о консервативности расширения)*
Язык L^* является консервативным расширением языка L.

Доказательство. В силу того, что язык L является р-полным и того что любая L*-программа является р-вычислимой (теорема 4.3) автоматически следует консервативность расширения. □

Заключение

В диссертации решены следующие проблемы:

1. Совместно с Гончаровым автором был разработан p-полный логический язык программирования L и тем самым была решена проблема равенства классов P и L.
2. Совместно с Гончаровым автором был доказан полиномиальный аналог классической теоремы Ганди о наименьшей неподвижной точке.
3. Автором было показано существование полиномиально вычислимых представлений для базовых конструкций ИП: множества термов ИП, множества формул ИП, а также для множества доказательств в ИП (как линейных, так и в виде дерева).
4. Автором было показано существование полиномиально вычислимых представления для L-формул и L-программ.
5. Автором также было показано существование полиномиально вычислимых представлений для множества выводов в порождающих грамматиках.
6. Совместно с Гончаровым, для выполнения поставленных задач на основе GNF-систем, PAG-теоремы и L-программ, автором был создан метод и разработаны техники, которые позволяют исследовать полиномиальную сложность с точки зрения формульной определимости и построения подходящих L-программ.
7. Совместно с Гончаровым на базе логического языка L был разработан p-полный объектно-ориентированный язык L^* синтаксис которого очень схож с синтаксисами других высокоуровневых объектно-ориентированных языков программирования.

Построение p-полного языка L дало толчок к использованию данных теоретических наработок в практических целях. Это касается таких направлений как: объяснительный искусственный интеллект, умные города [40], смарт контракты, робототехника и т.д. В любых этих направлениях требуется описать работу того или иного процесса. И для этих целей как нельзя лучше подходит построенный нами логический язык программирования L. Помимо практического применения автора интересуют и теоретические продолжения исследований. Это касается более полной формализации понятия блокчейна, алгоритмов кон-

сенсуса и теоретическое описание различных устойчивых децентрализованных систем. Все это планируется реализовать в рамках концепции семантического программирования.

В заключение хочу выразить свою благодарность и большую признательность своему научному руководителю Гончарову С. С. за поддержку, помощь, обсуждение результатов и научное руководство. А также за тот неоценимый научный опыт, который он передал мне в процессе нашей совместной работы. Также хочу поблагодарить Алаева П. Е. за консультации, корректировки и анализ ряда моих результатов. Выражаю благодарность Одинцову С. П. за активную поддержку моих исследований и становления меня как ученого. Выражаю особую признательность Бутурлакину А. А. за помощь и консультации связанные с вопросами алгебры. Хочу поблагодарить весь коллектив кафедры алгебры и логики, кто сделал данную работу возможной. Также я благодарен своей жене Нечесовой Н. С. за неизменную поддержку и своей маме Нечесовой Е. И. за веру в меня.

Список литературы

[1] *Алаев П.Е.* Структуры, вычислимые за полиномиальное время. I. *Алгебра и логика.* **2016**. *55(6)*, стр.647–669. https://doi.org/10.17377/alglog.2016.55.601

[2] *Алаев П.Е.* Структуры, вычислимые за полиномиальное время. II. *Алгебра и логика.* **2017**. *56(6)*, стр.651–670. https://doi.org/10.17377/alglog.2017.56.601

[3] *Алаев П.Е.* Существование и единственность структур, вычислимых за полиномиальное время. *Алгебра и логика.* **2016**. *55(1)*, стр.106–112. https://doi.org/10.17377/alglog.2016.55.107

[4] *Алаев П.Е., Селиванов В.Л.* Поля алгебраических чисел, вычислимые за полиномиальное время. I *Алгебра и логика.* **2019**. *58(6)*, стр.673–705. https://doi.org/10.33048/alglog.2019.58.601

[5] *Анцыферов С.С.* Проблемы искусственного интеллекта. *Проблемы искусственного интеллекта.* **2015**. *0(1)*, стр.5-12

[6] *Гончаров С.С.* Условные термы в семантическом программировании. *Сибирский математический журнал.* **2017**. *58(5)*, стр.794–800. https://doi.org/10.17377/smzh.2017.58.506

[7] *Гончаров С.С., Свириденко Д.И.* Логический язык описания полиномиальной вычислимости. *Доклады Академии Наук.* **2019**. *485(1)*, 11–14. https://doi.org/10.31857/S0869-5652485111-14

[8] *Гончаров С.С., Свириденко Д.И.* Σ-программирование. *Вычислимые системы.* **1985**. *485(1)*, 3–29.

[9] *Гончаров С.С., Свириденко Д.И.* Семантическое моделирование и искусственный интеллект. *Сибирский философский журнал.* **2018**. *16(4)*, стр.5-25. https://doi.org/10.25205/2541-7517-2018-16-4-5-25

[10] *Ершов Ю.Л.* Определимость и вычислимость. *Новосибирск.* **2000**

[11] *Ершов Ю.Л, Палютин Е.А.* Математическая логика. *Наука.* **1987**

[12] *Малых А.А., Манцивода А.В.* Документное моделирование. *Изв. ИГУ. Серия: Математика.* **2017**. *21*, стр.89-107. https://doi.org/10.26516/1997-7670.2017.21.89

[13] *Малых А.А., Манцивода А.В.* Libretto: язык программирования как средство логического объектного моделирования. *Сборник докладов международной конференции «Мальцевские чтения»*, Новосибирск, **2011**. стр.130-131

[14] *Манцивода А.В., Пономарев Д.К.* Формализация документных моделей средствами семантического моделирования *Изв. ИГУ. Серия: Математика.* **2019**. *27*, стр.36–54. https://doi.org/10.26516/1997-7670.2019.27.36.

[15] *Пратт Т., Зелковиц М.* Языки программирования: разработка и реализация. *4-е изд. СПб.: Питер.* **2003**

[16] *Тестелец Я. Г.* Введение в общий синтаксис. Глава XI. Порождающая грамматика: от правил к ограничениям. *РГГУ.* **2001**. ISBN 5-7281-0343-X.

[17] *Barwise J.* Admissible Sets and Structures. *Springer.* **1975**

[18] *Bathaee Y.* The artificial intelligence black box and the failure of intent and causation. *Harvard Journal of Law & Technology.* **2018**. *31(2)*

[19] *Cenzer D., Remmel J.* Polynomial-time versus recursive models. *Ann. Pure Appl. Log.*. **1991**. *54*, 17–58.

[20] d0sl – programming language. URL: https://d0sl.org/

[21] *Deisenroth, M.P.; Faisal, A.A.; Ong. C.S.* Mathematics for Machine Learning *Published by Cambridge University Press*, **2020**

[22] *Ershov Y.L., Goncharov S.S., Sviridenko D.I.* Semantic programming. *In Proceedings of the Information Processing 86* **1986**. *10*, p.1113–1120

[23] *Ershov Y.L., Goncharov S.S., Sviridenko D.I.* Semantic foundations of programming. *Lecture Notes in Computer Science.* **1987**. *278*, p.116–122.

[24] *Goncharov S.S., Sviridenko D.I.* Recursive terms in semantic programming. *Sib. Math. J.* **2018**. *59*, p.1014–1023. https://doi.org/10.1134/S0037446618060058

[25] *Goncharov S.S., Sviridenko D.I.* Theoretical aspects of Σ-programming. *Lecture Notes in Computer Science.* **1986**. *215*, p.169–179.

[26] *Goncharov S.S., Ospichev S.S., Ponomaryov D.K., Sviridenko D.I.* The expressiveness of looping terms in the semantic programming. *Sib. Electron. Math. Rep.* **2020**. *17*, p.380–394. https://doi.org/10.33048/semi.2020.17.024

[27] *Gumirov V., Matyukov P., Palchunov D.* Semantic Domain-specific Languages. *SIBIRCON.* **2019**. https://doi.org/10.1109/SIBIRCON48586.2019.8958237

[28] *Hagan, M.T; Demuth, H.B.; Beale, M.H; Jesus, O.D.* Neural Network Design *Martin Hagan*, **2014**

[29] *Lewis H., Papadimitriou C.* Elements of the Theory of Computation. *Prentice-Hall: Upper Saddle River, NJ, USA.* **1998**

[30] *Michaelson, G.* Programming Paradigms, Turing Completeness and Computational Thinking. *The Art, Science, and Engineering of Programming.* **2020**. *4(3)*. https://doi.org/10.22152/programming-journal.org/2020/4/4

[31] *Ospichev S.S.; Ponomaryov D.K.* On the complexity of formulas in semantic programming. *Sib. Electron. Math. Rep.* **2018**. *15*, p.987–995. https://doi.org/10.17377/semi.2018.15.083

[32] *Papadimitriou C.* Computational complexity. *Addison-Wesley.* **1994**

[33] *Russel, S.J; Norvig, P.* Artificial Intelligence - A Modern Approach (3rd Edition) *Prentice Hall*, **2010**

[34] *Wielemaker, J; Hildebrand, M; Ossenbruggen, J.* Using Prolog as the Fundament for Applications on the Semantic Web. *Proceedings of the ICLP2007 Workshop on Applications of Logic Programming to the Web, Semantic Web and Semantic Web Services (ALPSWS2007)*, **2007**. p.1–16, RWTH Aachen.

Работы автора по теме диссертации из журналов списка ВАК

[35] *Goncharov, S.S.; Nechesov, A.V.* Polynomial analogue of Gandy's fixed point theorem. *Mathematics* **2021**. *9(17):2102*. https://doi.org/10.3390/math9172102

[36] *Goncharov S.S., Nechesov A.V.* Solution of the problem P = L. *Mathematics.* **2022**. *10(1):113*. https://doi.org/10.3390/math10010113

[37] *Нечесов, А.В.* Некоторые вопросы полиномиально вычислимых представлений для порождающих грамматик и форм Бэкуса-Наура. *Математические труды.* **2022**. *25(1)*, стр.134-151. https://doi.org/10.33048/mattrudy.2022.25.106

[38] *Нечесов, А.В.* Семантическое программирование и полиномиально вычислимые представления. *Математические труды.* **2022**. *25(2)*, стр.174-202. https://doi.org/10.33048/mattrudy.2022.25.208

[39] *Nechesov, A.V.* Some Questions on Polynomially Computable Representations for Generating Grammars and Backus–Naur Forms. *Sib. Adv. Math.* **2022**. *32*, p.299–309. https://doi.org/10.1134/S1055134422040058

Работы автора по теме диссертации из журналов не из списка ВАК

[40] *Nechesov A.V.; Safarov R.A.* Web 3.0 and smart cities. *Сборник докладов республиканской научно-технической конференции «Современное состояние и перспективы развития цифровых технологий и искусственного интеллекта». Самарканд, Узбекистан*, **2022**. ч.1, стр.248-253

[41] *Гончаров С.С., Нечесов А.В.* Объектно-ориентированный логический язык программирования для искусственного интеллекта и робототехники. *Proceedings of the International Conference «Mathematical Logic and Computer Science», ENU, Астана, Казахстан*, **2022**. стр.191-195