Московский Государственный Университет
имени М.В. Ломоносова
Российская Академия Наук
Международная Академия Технологических Наук
Российская Академия Естественных Наук

Интеллектуальные Системы.

Теория и приложения

ТОМ **29** ВЫПУСК 4 * **2025**

МОСКВА

УДК 519.95; 007:159.955 ISSN 2411–4448
ББК 32.81 Издаётся с 1996 г.

В журнале «Интеллектуальные системы. Теория и приложения» публикуются научные достижения в области теории и приложений интеллектуальных систем, новых информационных технологий и компьютерных наук.
Издание журнала осуществляется под эгидой МГУ имени М.В. Ломоносова, Научного Совета по комплексной проблеме «Кибернетика» РАН, Отделения «Математическое моделирование технологических процессов» МАТН.
Учредитель журнала: ООО «Интеллектуальные системы».
Журнал входит в список изданий, включенных ВАК РФ в реестр публикаций материалов по кандидатским и докторским диссертациям по математике и механике. Входит в дополнительный перечень научных изданий, в которых могут быть опубликованы результаты диссертаций, защищаемых в МГУ.

Индекс подписки на журнал: 64559 в каталоге НТИ «Роспечать».

Адрес редакции: 119991, Москва, ГСП-1, Ленинские Горы, д. 1, механико-математический факультет, комн. 12-01.
Адрес издателя: 115230, Россия, Москва, Хлебозаводский проезд, д. 7, стр. 9, офис 9. Тел. +7 (495) 939-46-37, e-mail: mail@intsysmagazine.ru

*) Прежнее название журнала: «Интеллектуальные системы».

ОГЛАВЛЕНИЕ

Александр Сергеевич Подколзин (к 75-летию со дня рождения)

7 декабря 2025 года исполняется 75 лет профессору кафедры математической теории интеллектуальных систем механико-математического факультета МГУ, доктору физико-математических наук Александру Сергеевичу Подколзину.

Редакционная коллегия журнала «Интеллектуальные системы. Теория и приложения» от всей души поздравляет выдающегося российского ученого-математика с этим замечательным юбилеем!

Вся профессиональная жизнь Александра Сергеевича неразрывно связана с Московским университетом. Окончив механико-математический факультет в 1972 году, он прошел славный путь от ассистента до профессора, посвятив служению альма-матер более полувека. За эти годы А. С. Подколзин стал признанным во всем мире ученым и создателем научной школы в области теории автоматов и интеллектуальных систем.

Научная биография А. С. Подколзина — это история непрерывного восхождения к вершинам знания. Уже в 1975 году, в рекордные сроки, он защитил кандидатскую диссертацию, посвященную сложнейшим вопросам теории клеточных автоматов. Фундаментальные результаты этих исследований легли в основу монографии «Теория однородных структур», ставшей классикой для специалистов. Написанная им в соавторстве книга «Введение в теорию автоматов» и по сей день остается непревзойденным учебником, на котором воспитано не одно поколение математиков и кибернетиков.

Новую эпоху в области искусственного интеллекта открыла его докторская диссертация «Компьютерное моделирование процессов решения математических задач», защищенная в 1995 году. Александр Сергеевич совершил настоящий научный подвиг, создав уникальный компьютерный решатель, моделирующий логические процессы. В отличие от стандартных систем, этот решатель не имеет мировых аналогов и обеспечивает лидирующее положение российской науки. Действующие версии решателей охватывают такие области, как элементарная алгебра, геометрия, математический анализ и дифференциальные уравнения.

Это издание, насчитывающее к настоящему времени десять томов, представляет собой уникальную энциклопедию знаний, не имеющую равных в мировой науке.

Уникальность научного таланта А. С. Подколзина заключается в гармоничном сочетании фундаментальной теории с важнейшими прикладными разработками. Его алгоритмы нашли применение в области оптимального синтеза микросхем, позволив кардинально повысить производительность проектирования промышленных чипов. Новизна и прак-

тическая значимость этих идей подтверждены более чем 30 патентами США. Признанием выдающегося вклада А. С. Подколзина в науку стало его избрание членом Академии технологических наук России.

Александр Сергеевич — не только выдающийся исследователь, но и блестящий педагог, Учитель с большой буквы. Его лекционные курсы «Теория автоматов», «Компьютерное моделирование логических процессов» и «Теория дискретных функций» пользуются неизменным успехом у студентов. Легендарный семинар «Элементы кибернетики», которым он руководит, стал настоящей кузницей научных кадров. Под чутким руководством А. С. Подколзина выросла целая плеяда талантливых ученых — им подготовлено 2 доктора и 11 кандидатов наук.

Нельзя не отметить и вклад Александра Сергеевича в сохранение научных традиций Московского университета. Основанная им династия продолжается в его дочери: унаследовав математический дар отца и окончив кафедру МаТИС, она достойно несет эстафету служения науке, работая на механико-математическом факультете МГУ.

Глубокая научная принципиальность, такт, искренность и мудрость снискали Александру Сергеевичу безграничное уважение коллег и любовь учеников.

В день Вашего 75-летия мы желаем Вам, дорогой Александр Сергеевич, крепкого здоровья, неиссякаемой творческой энергии для новых свершений и радости от общения с молодыми талантами! Пусть Ваш жизненный путь еще долгие годы освещает дорогу российской науке!

Редакционная коллегия журнала
«Интеллектуальные системы. Теория и приложения»

Часть 1
Общие проблемы теории интеллектуальных систем

Введение в логические процессы. Алгоритмический язык ЛОС

А. С. Подколзин[1]

В статье описывается алгоритмический язык ЛОС (логическое описание ситуаций). Описываются основные операторы языка, редактор программ ЛОС, и показано как можно отлаживать программы на ЛОС.

Ключевые слова: решатель математических задач, логические процессы, логический язык, логическая формализация задач.

1. Введение

Данная статья является третьей в цикле статей, посвященных практикуму по решателю математических задач. Она является логическим продолжением статей [1, 2]. В данной статье мы опишем алгоритмический язык ЛОС (логическое описание ситуаций), его основные операторы, редактор программ ЛОС и покажем, как можно отлаживать программы на ЛОС. Решатель и заложенные в него принципы подробно описаны в монографиях [3, 4, 5, 6, 7, 8, 9, 10, 11].

2. Алгоритмический язык ЛОС

2.1. Структуры данных и общая схема организации программы на языке ЛОС

Программы приемов решателя записываются на языке ЛОС (Логический Описатель Ситуаций) и реализуются интерпретатором этого языка. Язык ЛОС по своему уровню сопоставим с ПРОЛОГом, однако специально адаптирован к изложенной выше схеме сканирования задач и имеет много принципиальных отличий от ПРОЛОГа. Для логической системы он является языком низкого уровня, и запись на нем приемов “вручную” предпринимается крайне редко. Фактически приемы записываются на языке более высокого уровня, ГЕНОЛОГе, и далее компилятор ГЕНОЛОГа создает на основе этой записи исполняемую ЛОС-программу приема.

[1] *Подколзин Александр Сергеевич* — д.ф.м.н., профессор каф. математической теории интеллектуальных систем мех.-мат. ф-та МГУ, e-mail: alexander.p@yandex.ru.

Podkolzin Alexander Sergeevich — Dr. of Sc., Professor, Lomonosov Moscow State University, Faculty of Mechanics and Mathematics, Chair of Mathematical Theory of Intellectual Systems.

Вместе с тем, основу языка ЛОС составляют около 200 предикатов и операций, которые могут рассматриваться не только как алгоритмический, но и как вполне развитый логический язык для описаний ситуаций в логико-сетевых структурах данных. Поэтому в тех случаях, когда речь идет не о работе с объектами предметной области, а о работе с объектами структурного уровня (иными словами, когда предметной областью являются сами структуры данных), логические уровни ЛОСа и ГЕНОЛОГа, по существу, совпадают, и программирование приемов непосредственно на ЛОСе становится вполне оправданным.

В языке ЛОС выделяются следующие основные типы объектов, обрабатываемых программой:

а) Символы переменных и логические символы. Максимально возможное число переменных и логических символов определяется конкретной версией интерпретатора; используемая в настоящее время версия допускает применение $2^{19}-1$ логических символов и стольких же переменных. Можно вводить, удалять либо изменять название ранее введенных логических символов. Каждое название логического символа представляет собой произвольную последовательность символов расширенного алфавита (допускаются русские и латинские буквы, цифры и пр.), длина которой не более 24. Собственно в программных файлах ЛОСа логические символы представлены при помощи своих номеров, а названия их хранятся в отдельном файле, что позволяет изменять эти названия, не изменяя программ.

б) Термы, построенные из переменных и логических символов (заголовок операции — произвольный логический символ, число операндов — любое). Терм длины 1 отождествляется с набором длины 1 (см. пункт в) и отличается от входящего в него логического символа.

в) Наборы $(a_1, \ldots, a_n)$ ранее определенных допустимых объектов.

г) Вхождения символов в термы и разрядов в наборы.

Заметим, что для скобок не предусмотрены специальные логические символы, а терм $\varphi(a_1 \ldots a_n)$ не является набором входящих в него логических символов и скобок — в структурах данных интерпретатора с ЛОСа скобки представлены неявным образом (указаны длины подтермов и ссылки на внешние операции). Тем не менее, во многих случаях терм можно рассматривать как последовательность составляющих его логических символов и символов переменных с опущенными скобками, применяя при этом операторы, предназначенные для работы с наборами. Для обозначения пустого набора выделен логический символ “пустоеслово”, воспринимаемый рядом операторов языка как набор длины 0.

Все логические символы пронумерованы натуральными числами. При использовании их в качестве заголовков операторов языка ЛОС выделяются непосредственно реализуемые логические символы (номера от 1 до 240) и программно реализуемые логические символы. С непосредственно реализуемыми символами связаны соответствующие процедуры интерпретатора; выполнение операторов с программно реализуемыми заголовками происходит при помощи программ языка ЛОС.

Представление чисел в решателе осуществляется четырьмя различными способами. С одной стороны, для быстрых выкладок с целыми числами диапазона от -259 до 65535 используется кодирование их логическими символами (логический символ "0" имеет номер 260). С другой стороны, отличное от цифры десятичное число $a_1 \dots a_n, a_{n+1} \dots a_{n+m}$ кодируется набором $(a_1 a_2 \dots a_n, a_{n+1} \dots a_{n+m})$, где запятая является логическим символом (изменение знака числа происходит путем добавления либо удаления логического символа "минус" в начале набора), а цифра отождествляется с обозначающим ее логическим символом. При работе с термами применяется представление указанного числа в виде "величина $(a_1 \dots a_n, a_{n+1} \dots a_{n+m})$" (отрицательного числа — в виде "минус (величина $(a_1 \dots a_n, a_{n+1} \dots a_{n+m})$)"; цифры — в виде однобуквенного терма). Наконец, для приближенных вычислений с математическим сопроцессором и для вычислений с применением обычной 32-битной машинной арифметики используются форматы "вещественное с плавающей запятой", "комплексное с плавающей запятой" (64-битное представление как для вещественной, так и для мнимой части), "целое со знаком" (32 бита) и "длинное целое со знаком" (массив 32-битных значений, имеющий переменную длину). Числа в указанных форматах кодируются специальным образом: первое — набором длины 2, второе — набором длины 4, третье — набором длины 1, четвертое - набором переменной длины. Эти наборы несколько отличаются от обычных наборов ЛОСа, и работать с ними нужно только при помощи операторов, специально предназначенных для машинной арифметики.

Программе доступна совокупность данных, образованная, с одной стороны, значениями ее переменных (входные данные и вспомогательные объекты, формируемые программой), а с другой стороны, некоторая совокупность внешних данных, описывающих цепь задач, сложившуюся в процессе решения предложенной системе задачи. Задача Z представляет собой набор $(p_0, p_1, \dots, p_n)$, где p_0 - логический символ, определяющий тип задачи; p_1 — набор посылок задачи; p_2 — набор весов посылок (логических символов от "0" до "20"); p_3 — набор наборов объектов, представляющих собой комментарии к посылкам, и т.д. Цепь задач представляет собой упорядоченную последовательность задач $(Z_0, Z_1, \dots, Z_N)$, где Z_0 — ис-

ходная задача; Z_N — текущая решаемая задача; Z_{i+1} — подзадача задачи Z_i $(i = 1, \ldots, N-1)$. Эта последовательность сама не образует допустимого объекта (набора задач) и обращение к ней осуществляется через специальные операторы языка. Такие операторы, в частности, позволяют определить по каждому элементу Z_i цепи задач его текущий m_i и максимальный M_i уровни (логические символы от "0" до "20"). Исходная задача Z_0 имеет фиктивный характер и создается автоматически при включении системы. Она представляет собой задачу на исследование с единственной фиктивной однобуквенной посылкой "вход"; при сканировании этой посылки запускается написанная на языке ЛОС программа интерфейса, позволяющая сформировать фактически решаемую задачу Z_1. Список общих комментариев к посылкам задачи Z_0 используется в качестве "доски объявлений", доступной любой процедуре системы. Главным образом, эта доска объявлений используется процедурами интерфейса.

Помимо указанных здесь основных типов данных, предусмотрены различные типы данных внешнего характера (файлы с древовидными информационными конструкциями, файлы программ ЛОСа и др.). Более подробно эти типы данных описываются ниже, при рассмотрении операторов языка ЛОС. Для работы с внешними данными используются ссылки, представленные посредством данных основных типов (как правило, наборы логических символов). Перечень внешних типов данных системы является, по существу, открытым, так как организация интерпретатора языка позволяет сравнительно легко подключать дополнительные непосредственно реализуемые операторы, работающие с данными нового типа.

Программа решателя организована по энциклопедическому принципу и распадается на программы отдельных логических символов. Программа логического символа φ имеет древовидную структуру и состоит из совокупности пронумерованных натуральными числами текстов, называемых фрагментами этой программы. Каждый фрагмент программы логического символа φ представляет собой последовательность $P_1, \ldots, P_n$ термов специального вида, называемых операторами языка. В большинстве случаев эти термы представляют собой просто утверждения логического языка, значениями переменных которых служат данные указанных выше типов.

Два специальных типа операторов, а именно "ветвь(N)" и "иначе(N)", где N — натуральное число, называются операторами перехода (для краткости скобки в операторах перехода опускаются, т.е. применяется запись "ветвь N", "иначе N"). Если в i - м фрагменте встречается оператор "ветвь j" либо "иначе j", то говорим, что этот фрагмент имеет ссылку на j - й фрагмент. Граф ссылок между фрагментами программы

является деревом с корнем в 1-м фрагменте; 1-й фрагмент называется также началом программы символа φ. Оператор "иначе N" располагается в фрагменте только после некоторого другого оператора, не являющегося оператором перехода. Во избежание путаницы сразу заметим, что в интерфейсе программного редактора ЛОСа применяется не глобальная, как сказано выше, а локальная нумерация фрагментов программы: ссылки из каждого фрагмента на подфрагменты пронумерованы там независимым образом последовательными натуральными числами (по мере появления их в тексте фрагмента), начинающимися с 1. Указанная выше глобальная нумерация фрагментов введена здесь лишь для удобства описания языка; фактически в файле программы ЛОСа ссылка из фрагмента на подфрагмент задается смещением этого подфрагмента в файле.

В операторах используется специальный класс подтермов, называемых операторными выражениями. В большинстве случаев эти подтермы суть некоторые выражения логического языка. Входящие в операторные выражения переменные обозначаются в программе ЛОСа посредством буквы "x" с номером: x1, x2, ... По определению, каждое однобуквенное выражение является операторным. Синтаксис операторных выражений $\varphi(t_1 \dots t_n)$ определяется независимо для каждого символа φ. Непосредственно реализуемые символы φ рассматриваются в следующем разделе данной главы; в случае программно реализуемого символа все операнды $t_1, \dots, t_n$ сами должны быть операторными выражениями (число n может варьироваться, при условии, что программа способна самостоятельно определить его по заведомо доступным ей данным). Аналогичным образом, синтаксис операторов $\varphi(t_1 \dots t_n)$, $n \geq 0$, определяется независимо для каждого символа φ, причем в случае программно реализуемого символа φ все $t_1, \dots, t_n$ суть операторные выражения. При определении оператора вида $\varphi(t_1 \dots t_n)$ указывается разбиение множества его свободных переменных на входные и выходные переменные; выходная переменная имеет (кроме случаев $\varphi \in \{$ "и", "или", "альтернатива" $\}$) ровно одно вхождение, являющееся вхождением некоторого операнда $t_i (i = 1, \dots, n)$. В некоторых случаях допускаются различные варианты такого разбиения, причем интерпретатор ЛОСа при реализации оператора выбирает то из них, для которого значения всех входных переменных уже определены программой, а всех выходных — не определены.

При обращении к программе логического символа φ, а также на каждом шаге выполнения этой программы выделяется некоторый набор переменных, значения которых считаются определенными, причем программа способна самостоятельно отличать их от прочих переменных.

Обращения к программе символа φ бывают следующих трех типов:

1) Обращение в процессе сканирования текущей решаемой задачи Z_N. Если φ — тип задачи и произошло обращение к процедуре, объединяющей приемы решения задач типа φ без привязки к конкретному вхождению логического символа в задачу (см. пункт 1 описания процедуры сканирования), то определены значения переменных $x1$ (задача Z_N) и $x5$ (текущий уровень m_N этой задачи). Если в задаче выделено конкретное вхождение символа φ (пункт 3 описания процедуры сканирования), то определены $x1 = Z_N$, $x2$ — данное вхождение символа φ в текущую посылку либо условие задачи, $x3$ — вхождение текущей посылки либо условия задачи в список посылок либо условий (при отсутствии такового — в саму задачу), $x4$ равно 0, если просматривается посылка задачи, и равно 1 в противном случае, $x5 = m_N$. Таким образом, тройка $(x2, x3, x4)$ определяет координаты вхождения в задачу выделенного символа φ. Если произошло обращение к процедуре символа, являющегося названием типа задачи (пункт 1 процедуры сканирования), то значения $x2, x3, x4$ равны логическому символу 0.

2) Обращение при выполнении программно реализуемого оператора либо нахождении значения операторного выражения $\varphi(t_1 \dots t_n)$. В этом случае определены все переменные $xi, 1 \le i \le n$, для которых t_i не является выходной переменной оператора. Остальные переменные $xi, i \in \{1, \dots, n\}$, считаются не определенными и должны оставаться таковыми вплоть до момента присвоения им специальными заключительными операторами результатов выполнения программы перед выходом из нее.

3) Приемы решения задач часто используют вспомогательные процедуры, распадающиеся на систему независимых подпроцедур, связанных с различными логическими символами. Такие процедуры обеспечивают получение информации справочного характера, связанной с соответствующими логическими символами (например, определяют арность символа; находят область допустимых значений операции с заданным заголовком, и т.п.). Эти процедуры называем далее справочниками. Каждая из них обозначается специальным логическим символом, отличным от символа “0” — типом данного справочника. Тип справочника ψ рассматривается как характеристика обращения к программе символа φ при попытке получить информацию, связанную с этим символом. Чтобы определить, что имеет место именно такой тип обращения, используется вспомогательный оператор “обращение(ψ)”, истинный лишь в этом случае. Значения переменных $x1, \dots, xn$, определенных при обращениях к справочнику типа ψ, задаются осуществляющим запрос операторным выражением “справка($\psi\varphi t_1 \dots t_n$)” ($x_i$ получает значение операторного выражения t_i).

Операторы языка ЛОС делятся на проверочные, перечисляющие и смешанного проверочно-перечисляющего типа. Проверочный оператор, при

указании значений его входных переменных, либо принимает значение "ложь", либо однозначным образом определяет значения выходных переменных и принимает значение "истина". Перечисляющий оператор, при определении значений его входных переменных, генерирует некоторую конечную последовательность (возможно,пустую) наборов значений выходных переменных, принимая после каждой выдачи очередного набора значение "истина", после чего принимает значение "ложь". Операторы смешанного типа функционируют либо в проверочном, либо в перечисляющем режиме, в зависимости от конкретных значений их входных переменных. Оператор "ветвь N" по определению считается перечисляющим. Для возвращения к перечисляющему оператору и получения от него очередного набора значений выходных переменных при реализации программы создается стэк перечисляющих операторов $(Q_1 \dots Q_m)$, которые относятся, вообще говоря, к различным фрагментам программы символа φ, находящимся на пути от начала программы к текущему фрагменту.

Реализация очередного оператора P_i текущего фрагмента $P_1, \dots, P_n$ программы символа φ происходит следующим образом:

1) Если оператор имеет вид $\psi(t_1 \dots t_n)$, где ψ — программно реализуемый символ, то последовательно определяются значения входных операторных выражений t_i и осуществляется обращение к программе символа ψ; действия в случае непосредственно реализуемого ψ описаны в следующем разделе. Если по окончании выполнения оператора он принимает значение "ложь", то переход к пункту 2. В противном случае оказываются определены значения всех выходных переменных этого оператора. Если при реализации оператора P_i имел место режим перечисления, то оператор заносится в конец стэка перечисляющих операторов. Если $i < n$, то переход к выполнению следующего оператора P_{i+1}, иначе — переход к пункту 3.

2) Если P_{i+1} имеет вид "иначе N", то осуществляется переход к первому оператору N - го фрагмента программы. Если стэк перечисляющих операторов пуст, то выполнение программы символа φ завершается. При этом, если происходило сканирование задачи, то делается очередной шаг сканирования; если вычислялось операторное выражение $\varphi(\theta_1 \dots \theta_k)$, то выдается значение "0" (логический символ); если же обрабатывался оператор $\varphi(\theta_1 \dots \theta_k)$, то он получает значение "ложь". Если стэк перечисляющих операторов $(Q_1 \dots Q_m)$ непуст, то осуществляется возвращение к последнему оператору Q_m, который удаляется из стэка. При этом значения всех переменных, номера которых больше номера последней определенной до обращения к Q_m переменной, становятся не определены. Оператор Q_m определяет очередной набор значений своих выходных переменных либо, при отсутствии такого набора, принимает значение

“ложь”, и далее повторяется выполнение программы начиная с оператора, следующего за Q_m. Особо выделяется случай оператора Q_m вида “ветвь N”. В этом случае просто выполняется переход к первому оператору N - го фрагмента программы.

3) В языке ЛОС имеется несколько специальных операторов, размещаемых обычно в конце фрагментов программы и используемых для управления ходом выполнения программы. Если P_i представляет собой один из таких заключительных операторов, то выполняются следующие действия:

а) P_i = “выход”. В этом случае программа символа φ осуществляет реализацию проверочного оператора, который получает значение “истина”.

б) P_i = “стоп”. В этом случае программа осуществляет реализацию оператора (не обязательно проверочного), который получает значение “ложь”.

в) P_i = “результат($xk_1\ t_1\ xk_2\ t_2 \dots xk_s\ t_s$)”. Программа осуществляет реализацию перечисляющего оператора $\varphi(\tau_1 \dots \tau_p)$; выходным переменным $\tau_{k_1}, \dots, \tau_{k_s}$ этого оператора присваиваются значения операторных выражений $t_1, \dots, t_s$. Заметим, что при реализации проверочного оператора $\varphi(\tau_1 \dots \tau_p)$ присвоение значений выходным переменным происходит таким же образом, но оператор “результат($\dots$)” располагается непосредственно перед завершающим оператором “выход”.

г) P_i = “продолжение”. Этот оператор является тождественно ложным, т.е. происходит возвращение к последнему оператору Q_m стэка перечисляющих операторов либо (при пустом стэке) выход из программы.

д) P_i = “обрыв” либо P_i = “сброс(k)”; k - натуральное число. Происходит возвращение к оператору Q_{m-k} (в первом случае $k = 1$) из стэка $(Q_1 \dots Q_m)$ перечисляющих операторов; если такого оператора нет, то — выход из программы, как при пустом стэке.

е) P_i = “ответ(t)”. В этом случае программа либо выполняла сканирование задачи, и тогда определяется ответ t на эту задачу, либо вычисляла операторное выражение (в частности, вида “справка($\psi\, \varphi \dots$)”), получающее значение операторного выражения t.

ж) P_i = “пересмотр”. В этом случае программа выполняет сканирование задачи, которое возобновляется повторно при нулевом текущем уровне задачи.

з) P_i = “контроль”. Программа выполняет сканирование задачи. Если эта задача имеет хотя бы один терм (посылку либо условие) с нулевым весом (что бывает при изменении терма либо занесении нового терма), то начинается повторное сканирование задачи при нулевом текущем уровне.

В противном случае — переход к последнему перечисляющему оператору Q_m (если его нет, то выход из программы).

Если P_n не является оператором одного из указанных видов, то либо имеет место сканирование задачи, и тогда осуществляется очередной шаг сканирования, либо происходит реализация проверочного оператора без выходных переменных, который получает значение "истина".

2.2. Основные операторы языка ЛОС

2.2.1. Общие операторы

Язык ЛОС имеет свыше 200 реализуемых интерпретатором операторов, ориентированных главным образом на обработку данных сетевого и логического типов. Часть этих операторов ("выход", "стоп", "результат($x1\ t_1 \ldots xn\ t_n$)", "ответ(t)", "обрыв", "сброс(n)", "продолжение", "контроль", "пересмотр") была рассмотрена в предыдущем разделе. Приведем еще ряд простых операторов управляющего характера.

Операторы "решить", "программа", "обращение(ψ)" принимают значение "истина", соответственно, если программа символа φ применяется при сканировании задачи, при реализации оператора и при вычислении операторного выражения t; в последнем случае $\psi \neq$ "0" означает, что t имело вид "справка($\psi\ \varphi \ldots$)", а $\psi =$ "0" — что t не имело такого вида. Оператор "определено(xi)" истинен, если определено значение переменной xi. Оператор "повторение" представляет собой вырожденный перечисляющий оператор, принимающий при каждом выходе на него значение "истина"; он аналогичен оператору repeat в ПРОЛОГе. Оператор "уровень($n_1 \ldots n_k$)" ($n_1, \ldots, n_k$ — логические символы от "0" до "20") истинен, если происходит сканирование задачи, текущий уровень которой равен одному из чисел $n_1, n_2, \ldots, n_k$; оператор "последнийуровень" — если текущий уровень решаемой задачи Z_N равен ее максимальному уровню. Оператор "новый" применяется при сканировании задачи; он принимает значение "истина" в том случае, когда рассматриваемый терм задачи впервые попал в поле зрения системы при данном текущем уровне m_N. Более точно, этот оператор становится ложным лишь в ситуациях повторного просмотра при значениях текущего уровня $U = 0, 1, \ldots, m_N$ временно выпавшего из поля зрения терма, имеющего вес $m_N + 1$. Оператор "новый" позволяет блокировать повторную попытку рассмотрения приема, если изменение окрестности терма не влияет на результаты такого рассмотрения.

Перейдем к описанию серии общелогических операторов языка. Если P — произвольный проверочный оператор, не имеющий выходных переменных, то утверждение "не(P)" является проверочным оператором,

истинным тогда и только тогда, когда ложен P. Если $P_1, \ldots, P_n$ — операторы, то утверждение "и($P_1 \ldots P_n$)" является оператором. Если хотя бы один из операторов $P_1, \ldots, P_n$ реализуется в режиме перечисления, то конъюнкция их также считается реализуемой в режиме перечисления. В стэк перечисляющих операторов при этом заносится не последний перечисляющий оператор P_i, а вся конъюнкция. Аналогичным образом формируется дизъюнкция "или($P_1 \ldots P_n$)" операторов $P_1, \ldots, P_n$. Если каждый из операторов $P_1, \ldots, P_n$ проверочный и не имеет выходных переменных, то дизъюнкция является проверочным оператором. Если каждый оператор $P_i, i = 1, \ldots, n$, имеет непустое множество выходных переменных, то дизъюнкция реализуется в режиме перечисления (даже если все P_i — проверочные). При этом сначала перечисляются значения выходных переменных, определяемые оператором P_1, затем — P_2, и т.д. Дизъюнктивные конструкции, у которых часть операторов P_i имеет пустое множество выходных переменных, а часть — непустое, не рекомендуются, так как тогда наличие либо отсутствие режима перечисления определяется неоднозначно.

Если $P_1, \ldots, P_n$ — операторы, $n > 0$,P_0 - проверочный оператор, $x_1, \ldots, x_k$ — переменные, содержащие все выходные переменные операторов $P_0, P_1, \ldots, P_n$, то утверждение "длялюбого($x_1 \ldots x_k$ если $P_1 \ldots P_n$ то P_0)" является проверочным оператором. К моменту его реализации последняя определенная переменная должна иметь номер, меньший номеров переменных $x_1, \ldots, x_k$. Все свободные переменные оператора P являются его входными переменными. При фиксированных значениях этих переменных цепочка операторов $P_1, \ldots, P_n$ (среди которых могут встречаться перечисляющие операторы) определяет процесс перечисления удовлетворяющих условиям $P_1, \ldots, P_n$ наборов значений их выходных переменных, причем для каждого такого набора вычисляется истинностное значение оператора P_0. Как только здесь возникает значение "ложь", процесс обрывается и P получает значение "ложь". В противном случае, по завершении перечисления, оператор P получает значение "истина", а значения всех переменных $x_1, \ldots, x_k$ снова оказываются не определены. Заметим, что управляющие ходом перечисления операторы "повторение", "обрыв", "сброс(n)" внутри сложных операторов (в частности, в кванторных операторах) не используются. Аналогичным образом определяется оператор P вида "существует($x_1 \ldots x_k$ P_0)", где P_0 — некоторый оператор (возможно, перечисляющий), все выходные переменные которого содержатся среди $x_1, \ldots, x_k$. Как и в случае квантора общности, к моменту реализации P последняя определенная переменная должна иметь номер, меньший номеров переменных $x_1, \ldots, x_k$. (Это правило является общим для всех операторов и операторных выражений со связанными переменными и далее особо не оговаривается. Вообще, при программировании

на ЛОСе предпочтительно вводить новые программные переменные так, чтобы очередной номер переменной был на единицу больше наибольшего из уже использованных номеров.) Если при фиксации значений входных переменных оператора P оператор P_0 оказывается истинным для некоторого набора значений своих выходных переменных, то определяемое им перечисление обрывается, P получает значение "истина", а значения переменных $x_1, \ldots, x_k$ становятся снова не определены.

К числу основных логических операторов присоединен оператор $P =$ "альтернатива($P_0 P_1 P_2$)", где P_0 - проверочный оператор без выходных переменных; P_1, P_2 - некоторые операторы (не обязательно проверочные). Если P_0 получает на некотором наборе значений своих переменных значение "истина", то далее выполняется оператор P_1, иначе — оператор P_2. Здесь возникает возможность появления смешанных проверочно-перечисляющих операторов, так как если один из операторов P_1,P_2 проверочный, а другой — перечисляющий, то выбор фактического режима работы оператора P происходит в зависимости от значений его входных переменных. По аналогии с оператором "альтернатива($P_0 P_1 P_2$)" устроено операторное выражение "вариант($P t_1 t_2$)", где P — проверочный оператор без выходных переменных; t_1 и t_2 — операторные выражения. Значение этого выражения совпадает со значением t_1 при истинном P и со значением t_2 при ложном P.

Для формирования наборов объектов, накапливаемых в процессе просмотра тех или иных структур данных, служат операторные выражения со связанными переменными "перечисление($x_1 \ldots x_k$ P t)" и "выписка($x_1 \ldots x_k$ P t)" (аналоги setof и findall в ПРОЛОГе). Здесь P - некоторый (как правило, перечисляющий) оператор; t — операторное выражение; $x_1, \ldots, x_k$ — переменные, включающие все выходные переменные оператора P. При определении значения этих выражений оператор P перечисляет некоторые наборы значений своих выходных переменных, и для каждого такого набора определяется значение выражения t. В первом случае (оператор "перечисление(...)") найденные указанным образом значения выражения t записываются в формируемый набор подряд, с исключением повторений; во втором случае (оператор "выписка(...)") исключение повторений не выполняется. Операторное выражение "суммавсех($x_1 \ldots x_k$ P t)" осуществляет суммирование значений выражения t (эти значения должны быть числами, т.е. цифрами либо наборами, см. описание представления чисел в разделе 1), возникающих в процессе реализации перечисляющего оператора P так же, как для операторов "перечисление(...)", "выписка(...)". Это выражение используется, например, при принятии решений для интегральной оценки ситуации.

Вычисление значения уже упоминавшегося в первом разделе операторного выражения “справка($\psi\ \varphi\ t_1 \dots t_n$)”, где ψ, φ — логические символы; $t_1, \dots, t_n$ — операторные выражения, осуществляется программой логического символа φ при “типе обращения” ψ к этой программе и исходных значениях $t_1, \dots, t_n$ переменных $x_1, \dots, x_n$. Это выражение используется в тех случаях, когда некоторая процедура, условно обозначаемая логическим символом ψ, естественным образом распадается на множество независимых фрагментов, связанных с различными логическими символами φ.

Оператор “равно($t_1 t_2$)” либо не имеет выходных переменных (если все входящие в него переменные имеют уже определенные на предыдущих этапах значения) и осуществляет проверку равенства значений выражений t_1 и t_2 (равенство наборов понимается как равенство определяемых ими древовидных конструкций), либо имеет выходную переменную t_1, которой присваивается значение выражения t_2.

Для изменения значения переменной x_N используется оператор “замена($x_N\ t$)”, где t — операторное выражение, определяющее заменяющий объект. Сразу заметим, что в фактически исполняемой программе здесь вместо переменной x_N хранится ее символьный номер (логический символ номер $260 + N$), который и сообщается интерпретатору ЛОСа в качестве первого входного данного оператора “замена(. . .)”. Аналогичный оператор “изменение($t_1 t_2$)” позволяет изменять разряд набора, вхождение которого определяется выражением t_1, на объект, являющийся значением выражения t_2. Применение операторов “замена(...)” и “изменение(...)” требует определенной аккуратности, так как после их реализации значения переменных могут измениться таким образом, что истинность предшествующих операторов рассматриваемого фрагмента программы нарушится.Эти операторы оказываются полезны при реализации циклических процедур в тех достаточно редких случаях, когда цикл не удается оформить с помощью перечисляющих операторов.

Возможно обращение к оператору либо операторному выражению, заголовок которых определяется некоторым операторным выражением. Это осуществляется при помощи терма “процедура($t_1\ t_2\ t_3$)”, у которого значением t_1 является указанный заголовок, причем при t_3 равном “0” данный терм реализует обращение к оператору, а при t_3 равном “1” — к операторному выражению. Значением t_2 является набор входных данных соответствующего оператора либо операторного выражения. В случае оператора с выходными переменными позиции набора t_2, соответствующие выходным переменным, заняты логическим символом “неопред”. Эти позиции после обращения к оператору “процедура(. . .)” изменяются на найденные значения выходных переменных.

В тех случаях, когда предпринимается обращение к процедуре, выполнение которой может оказаться неприемлемо трудоемким, используется фиктивный перечисляющий оператор "лимит(t_1)". Этот оператор устанавливает ограничение в t_1 шагов работы интерпретатора (количество таких шагов автоматически фиксируется в специальном регистре интерпретатора и может рассматриваться как машинно-независимые внутренние "часы" решателя) начиная с момента обращения к нему. По истечении этого числа шагов, если все еще не было выхода из следующей за данным оператором ветви программы, реализуется откат к оператору с присвоением ему истинностного значения "ложь". Если после обращения к подпрограмме, которое могло оказаться слишком трудоемким, нужно отменить откат к оператору "лимит(...)", то применяется оператор "Обрыв". Единственное его действие — исключение из стэка перечисляющих операторов последнего элемента (не обязательно установленного оператором "лимит(...)"), без каких-либо откатов.

В заключение подраздела упомянем операторное выражение "типданных(t)", позволяющее различать типы значения выражения t. Если A — логический символ либо символ переменной, то значение p выражения "типданных(t)" равно 0; если A — терм, то p равно 1; если A — вхождение в терм либо в набор, то p равно 2; если A — набор, не являющийся термом, то p есть 4. Значение $p = 3$ зарезервировано для тех случаев, когда A есть ссылка на недопустимый объект (сигнал о наличии ошибки).

2.2.2. Операторы просмотра и преобразования задачи

Для выделения отдельных элементов задачи служит ряд специальных операторных выражений. Так, "цели(t)" имеет своим заголовком список целей задачи, определяемой выражением t; "неизвестные(t)" — цель, перечисляющую неизвестные (если ее нет, то это выражение имеет значение "пустоеслово"); "комментарии(t_1)", "комментариипосылок(t_1)", "комментарииусловия(t_1t_2)", "комментариипосылки(t_1t_2)" - соответствующие списки комментариев (задача определяется здесь выражением t_1, а вхождение рассматриваемого условия либо посылки — выражением t_2); "списокусловий(t)" и "списокпосылок(t)" — списки условий и посылок. Значением выражения "переменные(t)" служит список всех переменных, встречающихся в посылках и условиях задачи, а также среди ее неизвестных. Выражение "максимальныйуровень(t)" позволяет определить максимальный уровень задачи. Для обращения к процедуре решения задачи служат операторные выражения "ответзадачи(t)" и "прямойответ (t)". Выражение t в обоих случаях имеет своим значением некоторую задачу (т.е. набор допустимого вида), причем в первом случае

эта задача начинает решаться с максимальным уровнем, равным текущему уровню решаемой задачи Z_N, а во втором случае - с максимальным уровнем 4. Если перед вычислением значения выражения "ответзадачи(t)" применить оператор "уровеньобращения(k)", то новая задача будет решаться с максимальным уровнем k (установка уровня обращения к задаче имеет одноразовый характер и после обращения к задаче пропадает). Ответ либо логический символ "отказ", найденный при решении задачи, становится значением соответствующего операторного выражения.

Операторы "исходнаязадача(x)", "текущаязадача(x)" присваивают переменной x исходную и текущую задачи цепи задач, а оператор "надзадача(x t)" перечисляет в обратном порядке (т.е. от текущей задачи к исходной) все элементы цепи задач, тип которых равен значению выражения t. Если t имеет значение 0, то последнее ограничение отменяется. Сама текущая задача из перечисления исключается. Оператор "текущийуровень(x)" присваивает переменной x значение текущего уровня текущей задачи. Операторы "посылка($t_1t_2t_3$)" и "условие($t_1t_2t_3$)" используются в следующих двух вариантах:

а) Значением выражения t_3 является задача Z; t_1, t_2 — выходные переменные. В этом случае перечисляются все посылки либо, соответственно, условия задачи Z (порядок перечисления — слева направо), причем значением t_1 становится рассматриваемый терм, а значением t_2 — его вхождение в список посылок либо список условий. Если Z не имеет списка условий, то при реализации оператора "условие(...)" режим перечисления не включается; значением t_1 становится единственное условие задачи, а значением t_2 — его вхождение в задачу.

б) t_1 и t_3 — входные выражения, значения которых суть терм θ и задача Z; t_2 — выходная переменная. Выполняется проверка того, что θ — посылка (условие) задачи Z, и если это так, то t_2 становится равно вхождению θ в соответствующий список либо в саму задачу Z.

В некоторых случаях бывает необходимо осуществить просмотр всех посылок либо условий задачи, имеющих заданный вес; как правило, таковыми оказываются вновь введенные за некоторый период посылки либо условия. Для этого применяются операторы "новаяпосылка($t_1t_2t_3t_4$)", "новоеусловие($t_1t_2t_3t_4$)", реализация которых отличается от случая а) реализации операторов "посылка($t_1t_2t_3$)", "условие($t_1t_2t_3$)" лишь тем, что отбираются термы с весом, равным значению выражения t_4. Операторы "цель(t_1t_2)", "неизвестная(t_1t_2)" допускают два режима реализации, причем в каждом из них значением выражения t_1 является задача Z, не имеющая тип "доказать":

а) t_2 — выходная переменная, и тогда оператор перечисляет значения переменной t_2, являющиеся целями (неизвестными) задачи Z.

б) t_2 определено, и тогда оператор проверяет, что значением выражения t_2 является цель (неизвестная) задачи Z.

Ряд операторов служит для преобразования задачи. Прежде всего, это операторы "заменаусловия($t_1t_2t_3$)" и "заменапосылки($t_1t_2t_3$)" (t_1 — задача, t_2 - вхождение заменяемого терма в соответствующий набор, t_3 - заменяющий терм); "занесениеусловия(t_1t_2)" и "занесениепосылки(t_1t_2)" (t_1 — терм, t_2 — задача, имеющая в первом случае тип "описать"); "удалениеусловия(t_1t_2)" и "удалениепосылки(t_1t_2)" (t_1 — задача, t_2 - вхождение удаляемого терма). Измененный либо новый терм получает в задаче вес 0; если оператор указанного типа является заключительным в фрагменте программы и обращение к этой программе произошло при сканировании задачи, то текущий уровень задачи заменяется на 0 и цикл сканирования повторяется сначала.

Для занесения в соответствующие списки новых комментариев используются операторы "примечание($t\ \theta_1 \dots \theta_n$)" (вводится общий комментарий ($\alpha_1 \dots \alpha_n$) либо, при $n = 1$, общий комментарий α_1 к посылкам задачи, определяемой выражением t; $\alpha_1, \dots, \alpha_n$ — значения выражений $\theta_1, \dots, \theta_n$; α_1 — логический символ), "замечание($t\theta_1 \dots \theta_n$)" (вводится общий комментарий к задаче, определяемой выражением t), "примечпосылки($t\tau\theta_1 \dots \theta_n$)" (вводится комментарий к посылке, определяемой своим вхождением τ в список посылок), "замечусловие($t\ \tau\ \theta_1 \dots \theta_n$)" ($t$ определяет задачу на описание; τ — вхождение в список условий того условия, к которому относится комментарий).

Проверка отсутствия заданного комментария выполняется аналогичными по своей структуре операторами "коммент($t\ \theta_1 \dots \theta_2$)" (отсутствие среди общих комментариев к задаче набора ($\alpha_1 \dots \alpha_n$) либо, при $n = 1$, символа α_1), "комментпосылок ($t\ \theta_1 \dots \theta_n$)", "комментусловия($t\ \tau\ \theta_1 \dots \theta_n$)", "комментпосылки($t\ \tau\ \theta_1 \dots \theta_n$)".

Завершают перечень операторов задач операторы "удалениепримечания($t\ \theta$)", "удалениезамечания($t\ \theta$)", "удалениепримечпосылки($t\ \tau\ \theta$)", "удалениезамечусловия ($t\ \tau\ \theta$)", выполняющие удаление определяемого выражением θ комментария (t определяет задачу, τ — вхождение рассматриваемой посылки либо условия.

2.2.3. Операторы логического представления данных

Начнем с перечисления операторных выражений, предназначенных для работы с логическими конструкциями. Выражения "списокпеременных(t)",

“параметры(t)” определяют, соответственно, набор всех переменных терма, являющегося значением выражения t, и набор всех свободных переменных этого терма. В последнем случае значением выражения t может служить как один терм, так и набор термов. Для образования новых термов используется операторное выражение “запись($t\theta_1 \dots \theta_n$)”; если выражения $t, \theta_1, \dots, \theta_n$ определяют, соответственно, логический символ φ и термы либо символы (т.е. логические символы либо символы переменных) $\tau_1, \dots, \tau_n$, то формируется новый терм $\varphi(\tau_1 \dots \tau_n)$. В тех случаях, когда набор операндов имеет переменную длину, новые термы определяются операторным выражением “сборка($t\ \theta$)”. Здесь t определяет логический символ φ; θ — набор операндов $\tau_1, \dots, \tau_n$ — термов либо символов. Для выделения фрагментов уже построенного терма применяются операторные выражения “подтерм(t)” (t указывает вхождение корня переписываемого подтерма), “набороперандов(t)” (t указывает вхождение первого символа терма $\varphi(\theta_1 \dots \theta_n)$, и значением выражения становится набор термов $\theta_1, \dots, \theta_n$), а также операторные выражения “первыйтерм(t)”, “второйтерм(t)”, “предпоследтерм (t)”, “последнийтерм(t)” для наиболее часто рассматриваемых операндов $\theta_1, \theta_2, \theta_{n-1}, \theta_n$ терма $\varphi(\theta_1 \dots \theta_n)$, вхождение первого символа которого определяется выражением t. Операторные выражения “первыйоперанд(t)”, “второйоперанд (t)”, “предпоследоперанд(t)”, “последнийоперанд(t)” позволяют определить вхождения заголовков указанных операндов $\theta_1, \theta_2, \theta_{n-1}, \theta_n$. Операторные выражения “следоперанд(t_1)” и “предоперанд(t_1)” позволяют находить вхождения операнда некоторой операции, соответственно, непосредственно следующего за вхождением t_1 операнда той же операции либо предшествующего ему. В концевых случаях, при отсутствии следующего либо предыдущего операнда, выдается само значение t_1. Операторное выражение “операндномер(t n)” имеет своим значением вхождение n - го операнда операции, расположенной по вхождению t. Здесь n — символьный номер (логический символ с номером $260+n$). Если операция имеет меньше чем n операндов, то выдается логический символ 0.

Операторное выражение “заменатермов ($t_1t_2t_3$)” определяет результат замены в терме θ набора вхождений ($\alpha_1 \dots \alpha_n$) расположенных слева направо непересекающихся подтермов (при $n = 1$ рассматривается не набор (α_1), а само вхождение α_1) на набор термов либо символов ($\tau_1 \dots \tau_n$). Здесь t_1 определяет θ, t_2 — вхождения, а t_3 — заменяющие термы. Выражение “исключениеоперанда(t_1t_2)” позволяет находить терм $\varphi(\theta_1 \dots \theta_{i-1}\theta_{i+1} \dots \theta_n)$, получающийся из терма $\varphi(\theta_1 \dots \theta_n)$ исключением операнда θ_i (если $n = 2$, то результатом исключения становится оставшийся операнд $\theta_j, j \neq i$); здесь t_1 определяет вхождение первого символа терма $\varphi(\theta_1 \dots \theta_n)$, а t_2 — вхождение операнда θ_i. Если выражение t определяет вхождение символа в некоторый терм θ, то операторное выражение

“базавхождения(t)” имеет своим значением данный терм θ. Для определения переменной с заданным номером применяется выражение “икс(t)”, где t определяет N - й после “0” логический символ; N - номер требуемой переменной. Выражение “кортежпеременных(t_1t_2)” определяет набор $(y_1 \dots y_k)$ переменных, отличных от переменных набора — значения выражения t_2 и имеющий ту же длину, что и набор — значение выражения t_1. Аналогичной цели служит оператор “новаяпеременная(t_1t_2)”, у которого t_1 определяет набор переменных, а выходной переменной t_2 присваивается переменная, не входящая в указанный набор.

Операторы “логсимвол(t)”, “переменная(t)” позволяют распознавать логические символы и переменные; операторы “корень(t)”, “стандарт(t)” — распознавать корневое вхождение в терм и вхождение символа, за которым идет открывающая скобка. Последний оператор бывает полезен, чтобы отличить обычное использование символа операции φ в конструкциях вида $\varphi(\theta_1 \dots \theta_n)$ от использования его в качестве синтаксической константы в описаниях вида термов. При обращении к оператору “символ (t_1t_2)” значением выражения t_1 является вхождение в терм либо в набор. Если t_2 - выходная переменная, то ей присваивается объект, имеющий указанное вхождение; в противном случае оператор проверяет совпадение этого объекта со значением выражения t_2. Аналогично, оператор “заголовок(t_1t_2)” либо присваивает выходной переменной t_2 первый символ терма, являющегося значением t_1, либо, если t_2 определено, проверяет совпадение этого заголовка со значением t_2. Операторы “первыйсимвол (t_1t_2)”, “второйсимвол(t_1t_2)”, “предпоследсимвол(t_1t_2)”, “последнийсимвол(t_1t_2)” определяют, соответственно, заголовки (логические символы либо переменные) операндов $\theta_1, \theta_2, \theta_{n-1}, \theta_n$ подтерма $\varphi(\theta_1 \dots \theta_n)$, вхождение первого символа которого задается выражением t_1; если t_2 — выходная переменная, то ей присваивается найденный заголовок, иначе он сравнивается со значением t_2. Эти операторы часто используются при сканировании задачи для идентификации того или иного подтерма. Оператор “свобвхождение(t)” распознает свободное вхождение переменной в терм; оператор “лексикопредшствует(t_1t_2)” — проверяет лексикографическое предшествование термов. Последний оператор бывает полезен для стандартного упорядочения операндов коммутативных и ассоциативных операций.

Перечислим ряд операторов,используемых для перемещения по вхождениям в терм. Оператор “операнд(t_1t_2)” используется в следующих двух возможных режимах:

а) t_2 определено и имеет значением вхождение α в терм; t_1 — выходная переменная, которой присваивается вхождение той операции, чьим операндом является α. Если такой нет, то оператор ложен.

б) t_1 определено и имеет значением вхождение первого символа подтерма $\varphi(\theta_1 \dots \theta_n)$; $n > 0$. t_2 — выходная переменная, значения которой перечисляют вхождения операндов $\theta_i; i = 1, \dots, n$.

Оператор "новоперанд(t_1 t_2)" имеет своим входным данным t_1 вхождение операнда некоторой операции. Выходная переменная t_2 перечисляет вхождения операндов той же операции, следующие за t_1 (исключая само t_1).

Для просмотра всех внешних операций, а также всех подряд символов некоторого подтерма служит оператор "подчинено($t_1 t_2$)". Существует три возможных режима его реализации:

а) t_1 определено и имеет своим значением вхождение α в некоторый терм. t_2 — выходная переменная, перечисляющая в направлении от α к корню терма все вхождения заголовков содержащих α подтермов (само вхождение α отбрасывается).

б) t_2 определено и имеет своим значением вхождение β заголовка некоторого подтерма; t_1 — выходная переменная, перечисляющая слева направо все вхождения символов в этот подтерм (начиная с заголовка).

в) t_1, t_2 определены и задают вхождения α, β заголовков подтермов τ_1, τ_2. Оператор истинен, если τ_1 лежит внутри τ_2 либо совпадает с τ_2.

Оператор "вхождениетерма($t_1 t_2 t_3$)" имеет входные данные t_1 (терм θ_1) и t_2 (терм θ_2); t_3 - выходная переменная, перечисляющая (слева направо) все вхождения терма θ_2 в терм θ_1. Для просмотра всех антецедентов $f_1, \dots, f_n$ импликативной конструкции "длялюбого($x_1 \dots x_k$ если $f_1 \dots f_n$ то f_0)" применяется оператор "антецедент($t_1 t_2$)", у которого значением t_1 служит указанная конструкция либо вхождение ее заголовка, а выходная переменная t_2 перечисляет вхождения корней утверждений $f_1, \dots, f_n$.

Для сравнения термов, определенных вхождениями своих заголовков, служит оператор "равныетермы($t_1 t_2$)".

Чтобы ускорить поиск в заданном наборе термов A терма, подобного терму t (получающегося из него переобозначением связанных переменных и перестановкой операндов коммутативных операций и симметричных отношений), можно использовать оператор "ключподобия(A t x)". Его выходная переменная x перечисляет вхождения в список A тех термов, которые могут оказаться подобными терму t (сравниваются длины термов, их заголовки и суммы кодов входящих в терм логических символов и переменных; код любой переменной здесь равен 1).

Завершает перечень операторов логических структур данных оператор "результподст($t_1 t_2 t_3 t_4$)". Он допускает следующие два возможных режима реализации:

а) t_1, t_2 определяют набор переменных $(x_1 \dots x_k)$ и набор термов $(\tau_1 \dots \tau_k)$ соответственно; t_3 определяет терм θ. t_4 - выходная переменная, которой присваивается терм — результат подстановки термов $\tau_1, \dots, \tau_k$ вместо переменных $x_1, \dots, x_k$ в терм θ.

б) t_1 определяет набор переменных $(x_1 \dots x_k)$; t_3 и t_4 — термы θ и θ'. t_2 — выходная переменная, которой присваивается такой набор термов $(\tau_1 \dots \tau_k)$, подстановка которого в терм θ вместо переменных $x_1, \dots, x_k$ дает терм θ'; если x_i не входит в θ, то τ_i становится равно логическому символу "пустоеслово". При отсутствии указанного набора оператор ложен.

2.2.4. Операторы сетевого представления данных

Операторы сетевого представления данных связаны с рассмотрением наборов и вхождений в них (в сетевых конструкциях вершины,а иногда и ребра кодируются посредством наборов). Операторное выражение "набор$(t_1 \dots t_n)$" $(n \geq 1)$ позволяет формировать набор значений выражений $t_1, \dots, t_n$. Если выражения $t_1, \dots, t_n$ определяют наборы либо символы (т.е. логические символы либо символы переменных), то выражение "конкатенация$(t_1 \dots t_n)$" определяет конкатенацию этих наборов и символов; логический символ "пустоеслово" при этом рассматривается как набор длины 0. Выражения "суффикс$(t\,\theta)$" и "префикс$(t\,\theta)$" позволяют присоединять к концу (соответственно, к началу) набора, определяемого выражением t, значение выражения θ. Операторное выражение "окончание(t)" осуществляет удаление первого элемента набора (пустой набор этим выражением сохраняется); выражение "вставка$(\ t_1t_2t_3)$" имеет своим значением набор $(\alpha_1 \dots \alpha_{i-1}\beta\alpha_i \dots \alpha_n)$, полученный из набора $(\alpha_1 \dots \alpha_n)$, определяемого выражением t_1, заменой i - го разряда, вхождение которого определяется выражением t_2, на объект β, определяемый выражением t_3. Если значения выражений t_1, t_2 суть набор $(\alpha_1 \dots \alpha_n)$ и набор вхождений в этот набор разрядов $\alpha_{i_1}, \dots, \alpha_{i_s}$; $i_1 < \dots < i_s$ (если $s = 1$, то берется не набор вхождений, а само вхождение), то значением операторного выражения "исключение(t_1t_2)" служит результат удаления указанных разрядов из набора $(\alpha_1 \dots \alpha_n)$. Выражение "заменаразряда$(t_1t_2t_3)$" определяет результат замены в заданном наборе (значение t_1) заданного вхождения разряда (значение t_2) на заданный объект (значение t_3). Если выражения t_1, t_2 определяют наборы α, β, то выражение "пересечениеписков(t_1t_2)" имеет своим значением набор, полученный из α сохранением всех разрядов, встречающихся в β, причем с учетом кратности: число сохраняемых экземпляров одного и того же объекта набора α не превосходит числа его экземпляров в наборе β. Выражение "объединениеписков(t_1t_2)" имеет своим значением набор $\alpha\beta'$, где

β' получено из β удалением всех элементов, входящих в α, а выражение "вычеркивание(t_1t_2)" — набор, получающийся из α в результате непродолжаемой последовательности одновременных вычеркиваний у α и β пар одинаковых элементов (порядок просмотра α — слева направо). Выражение "бланк($t_1t_2t_3$)", где t_1 определяет набор длины k, $k \geq 0$, t_2 — некоторый объект α; t_3 — N -й после "0" логический символ ($N \geq 0$), имеет своим значением набор $(\alpha \dots \alpha)$ длины $N + k$. Наконец, последнее из серии операторных выражений, используемых для построения новых наборов, — выражение "копия(t)", формирующее копию набора (либо терма).

Выражение "буква(t)" имеет своим значением объект, вхождение которого в набор либо терм определяется выражением t. Если значением выражения t_1 является набор $(\alpha_1 \dots \alpha_n)$, а значением t_2 — N - й после "0" логический символ ($N \geq 0$), то значением выражения "левпозиция(t_1t_2)" служит объект α_{N+1}; значением выражения "правпозиция(t_1t_2)" — объект α_{n-N}; значением выражения "окрестность(t_1t_2)" — вхождение $N + 1$ -го разряда в набор $(\alpha_1 \dots \alpha_n)$. В этой же ситуации значением выражения "левыйкрай(t_1)" является вхождение разряда α_1; выражения "правыйкрай(t_1)" — вхождение α_n; выражения "начало(t_1)" — объект α_1 и выражения "конец(t_1)" — объект α_n. Если значением выражения t является вхождение разряда α_i в некоторый набор $(\alpha_1 \dots \alpha_n)$, то выражение "левсосед(t)" определяет вхождение α_{i-1}, а "правсосед(t)" - вхождение α_{i+1} (если разряд крайний и сдвиг невозможен, то сохраняется значение α_i). Оператор "позиция(t_1t_2)" имеет выходную переменную t_1, перечисляющую слева направо все вхождения разрядов в набор, определяемый выражением t_2. Оператор "входит(t_1t_2)", аналогичный предыдущему, имеет два возможных режима реализации:

а) t_2 определено и имеет своим значением набор $(\alpha_1 \dots \alpha_n)$; t_1 — выходная переменная, перечисляющая объекты $\alpha_i (i = 1, \dots, n)$.

б) t_1, t_2 определены и имеют значения α, β; β - набор. В этом случае оператор проверяет вхождение объекта α в набор β.

Для поиска в наборе используются: оператор "разряд($t_1t_2t_3$)", где t_1 определяет набор либо терм α; t_2 — логический символ либо переменную φ; t_3 — выходная переменная, перечисляющая все вхождения φ в α, а также операторы "ключ($t_1t_2\theta$)" и "биключ($t_1t_2t_3\theta$)". У последних двух операторов t_1 определяет набор α; t_2 — логический символ β;t_3 — некоторый объект γ. Выходная переменная θ перечисляет все элементы набора α, имеющие в первом случае вид $(\beta, \dots)$ либо $\beta(\dots)$ либо равные β, а во втором случае — вид $(\beta, \gamma, \dots)$ либо $\beta(\gamma, \dots)$, либо $\beta(\gamma(\dots), \dots)$. Эти операторы часто применяются при поиске комментариев заданного типа, а также при отборе условий либо посылок задачи, имеющих заданный вид. Если выражения

t_2, t_3 определяют наборы $(\alpha_1 \dots \alpha_n)$ и $(\beta_1 \dots \beta_m)$, а выражение t_1 — вхождение разряда $\alpha_i, i \le m$, то операторное выражение "соотвпозиция$(t_1 t_2 t_3)$" определяет вхождение разряда β_i. Аналогично, если t_1, t_2 определяют наборы $(\alpha_1, \dots, \alpha_n)$ и $(\beta_1, \dots, \beta_n)$, а t_3 — некоторый объект β, то выражение "таблзначение$(t_1 t_2 t_3)$" либо имеет своим значением объект β_j, такой, что $\beta = \alpha_j$ и $\beta \neq \alpha_1, \dots, \alpha_{j-1}$, либо, при отсутствии указанного объекта, имеет значение β. Оператор "Таблзначение$(t_1 t_2 t_3 t_4)$" позволяет перечислять все элементы t_4 набора t_2, у которых на соответствующей позиции набора t_1 располагается элемент, равный t_3.

Приведем ряд специальных операторов, используемых для перечисления вхождений в наборы. Если выражения $t_1, \dots, t_n$ имеют своими значениями наборы $(\alpha_{11}, \dots, \alpha_{1m_1}), \dots, (\alpha_{n1}, \dots, \alpha_{nm_n})$ соответственно; $\theta_1, \dots, \theta_n$ — различные переменные, не встречающиеся в $t_1, \dots, t_n$, то оператор "серия$(\theta_1\, t_1 \dots \theta_n\, t_n)$" имеет выходные переменные $\theta_1, \dots, \theta_n$ и перечисляет такие наборы $(\beta_{1i}, \dots, \beta_{ni})$ $(i = 1, \dots, \min(m_1, \dots, m_n))$ их значений, что β_{ji} — вхождение разряда α_{ji} в набор $(\alpha_{j1} \dots\ \alpha_{jm_j})$; $j = 1, \dots, n$. Просмотр наборов "соответствующих" вхождений справа налево выполняется оператором "контрсерия$(\theta_1\, t_1 \dots \theta_n\, t_n)$", причем $t_1, \dots, t_n$ имеют своими значениями вхождения разрядов $\alpha_{1j_1}, \dots, \alpha_{nj_n}$ в некоторые наборы $(\alpha_{11}, \dots, \alpha_{1m_1}), \dots,$ $(\alpha_{n1}, \dots, \alpha_{nm_n})$ и оператор перечисляет наборы $(\beta_{1j_1-i}, \beta_{2j_2-i}, \dots, \beta_{nj_n-i}); i = 0, 1, \dots,$ $\min(j_1 - 1, \dots, j_n - 1)$.

Для просмотра разрядов набора начиная с некоторой фиксированной позиции (слева направо) служит оператор "постпозиция$(t_1 t_2)$". Он допускает два режима реализации:

а) t_1 — выходная переменная; t_2 определяет вхождение разряда α_i в набор $(\alpha_1 \dots \alpha_n)$. Тогда t_1 перечисляет вхождения $\alpha_i, \alpha_{i+1}, \dots, \alpha_n$.

б) t_1, t_2 определены и имеют значения α, β - вхождения в некоторый набор. Оператор проверяет, что вхождение α расположено не раньше, чем β.

Иногда более удобно применять похожий оператор "новпозиция$(t_1 t_2)$", у которого значением t_2 является набор $(\alpha_1 \dots \alpha_n)$ либо вхождение разряда α_i в этот набор, а выходная переменная t_1 перечисляет: в первом случае — все вхождения $\alpha_1, \dots, \alpha_n$, а во втором — $\alpha_{i+1}, \dots, \alpha_n$.

В заключение перечислим ряд проверочных операторов для работы с наборами. Оператор "пересекаются$(t_1 t_2)$" проверяет наличие общего элемента в наборах либо термах (в последнем случае под элементом понимается логический символ либо символ переменной); операторы "равнойдлины$(t_1 t_2)$" и "короче$(t_1 t_2)$" — равенство длин наборов и тот факт, что длина набора t_1 меньше длины набора t_2. Оператор "различны(t)" проверяет попарное отличие друг от друга всех элементов набора. Если значением выражения

t_1 является набор α, а значением выражения t_2 — N-й после “0” логический символ, то оператор “длинатекста(t_1t_2)” истинен, если α имеет длину N, а оператор “длинаменее(t_1t_2)” — если длина α меньше, чем N.

Наконец, оператор “включается(t_1t_2)” проверяет, что набор — значение выражения t_1 получается перестановкой и исключением части разрядов набора — значения выражения t_2.

2.2.5. Арифметические операторы

Арифметические операторы ЛОСа делятся на две группы: для работы с символьным представлением чисел (число N кодируется N-м после “0” логическим символом) и с обычными десятичными представлениями. В последнем случае цифры $0, \ldots, 9$ кодируются логическими символами “0”, ..., “9”; неодноразрядное положительное целое число, имеющее десятичную запись $a_1 \ldots a_n$, кодируется набором логических символов, соответствующих цифрам $a_1, \ldots, a_n$; для представления десятичных дробей внутри этого набора на соответствующей позиции размещается логический символ “,” (специальный символ для запятой); для получения отрицательных чисел в начале набора добавляется логический символ “минус”. Заметим, что все десятичные числа, кроме цифр, кодируются наборами логических символов, а цифры кодируются самими логическими символами.

Для работы с символьным представлением чисел используются операторные выражения “плюссимв(t_1 t_2)”, “вычитсимв(t_1 t_2)”, “умножсимв(t_1 t_2)” (сложение, вычитание и умножение). Заметим, что в ЛОСе логический символ “0” имеет номер 260, и логические символы с номерами от 1 до 259 могут использоваться как символьные представления отрицательных целых чисел от -259 до -1. В интерпретаторе ЛОСа предусмотрена возможность для работы с 65535 логическими символами, т.е. наибольшее допустимое в символьном представлении целое число есть 65275. Обычно символьные представления чисел используются для таких технических целей, как нумерация разрядов наборов, представление весов посылок и условий задач, нумерация переменных, задание координат точек на экране, и т.п. Указанные выше диапазоны изменения символьных представлений чисел для этого вполне достаточны.

Если значениями t_1 и t_2 служат переменные, то значением выражения “вычитсимв (t_1t_2)” служит символьное представление разности номеров этих переменных; если значением t_1 является переменная x, а значением t_2 — символьное представление числа n, то значением выражений “плюссимв(t_1t_2)”, “вычитсимв(t_1t_2)” служит переменная, номер которой

получается, соответственно, увеличением либо уменьшением на n номера переменной x.

Оператор "частноесимв($t_1\ t_2\ t_3\ t_4$)" позволяет находить неполное частное t_3 и остаток t_4 от деления числа t_1 на t_2 (все представления здесь - символьные). Оператор "менее($t_1 t_2$)" истинен, если число t_1 в символьном представлении меньше числа t_2. Для перехода от одного представления чисел к другому служат операторные выражения "симвномер(t)" (логический символ с десятичным номером, определяемым выражением t) и "номерсимв(t)" (десятичный номер логического символа, определяемого выражением t).

Для работы с десятичным представлением служат операторные выражения "минус(t)", "плюс($t_1 t_2$)", "вычитание($t_1 t_2$)", "умножение($t_1 t_2$)", и оператор "меньше ($t_1 t_2$)". Для реализации деления десятичных чисел применяются программы, написанные на ЛОСе; в этих программах можно использовать вспомогательные операторы "блокделения($t_1 t_2 t_3 t_4$)" (t_1, t_2 — делимое и делитель; выходным переменным t_3, t_4 присваиваются неполное частное и остаток, причем оператор рассчитан только на случай, когда t_1 и t_2 не превосходят 65535) и "шагделения($t_1 t_2 t_3$)" (t_1, t_2 - целые неотрицательные; $t_1 \leq 1000$; $1 < t_2 \leq 100$; выходной переменной t_3 присваивается целая часть от деления t_1 на $t_2 + 1$). Здесь приведены лишь базисные операторы ЛОСа, непосредственно реализуемые его интерпретатором; для вычислений в десятичных числах на ЛОСе запрограммирована целая серия дополнительных операторов, которые будут указаны в последующих разделах. Отметим, что вычисления в десятичных представлениях на ЛОСе выполняются с "плавающим" числом разрядов; это число ограничено сверху лишь общими требованиями интерпретатора к длине формируемых наборов и может достигать даже тысячи знаков. Приведенные выше базисные операторы сложения, вычитания, умножения и изменения знака выполняются без округления; программно реализованные на ЛОСе вычислительные операторы выполняют округление с сохранением заданного числа знаков после запятой, причем таким образом, что гарантируется получение нижней либо верхней (по выбору) оценки для точного значения.

Для вычислений в машинных форматах "с плавающей запятой" и "целое со знаком" используются группа операторных выражений "выч(...)" и операторов "Выч(...)". Заметим, что кроме непосредственного представления чисел в этих форматах, можно создавать массивы таких чисел. Манипулирование с массивом осуществляется при помощи ссылки на него, которая формально представляет собой некоторый набор длины 2 (как и представление числа в машинном формате, такой набор несколько отличается от обычного набора ЛОСа, и для работы с ним следует при-

менять только указанные выше операторные выражения и операторы “выч”, “Выч”).

Операторное выражение “выч(число a)” позволяет перейти от обычного десятичного числа a (цифры либо набора цифр и знаков “минус”, “запятая”) к формату “с плавающей запятой”; операторное выражение “выч(целое a)” — к формату “целое со знаком”; операторное выражение “выч(Целое a)” — к формату “длинное целое со знаком”; операторное выражение “выч(комплексное a b)” преобразует в формат “комплексное с плавающей запятой” комплексное число с вещественной частью a и мнимой частью b. Обратные преобразования осуществляются операторным выражением “выч(выход a)”. Здесь a — число в произвольном формате из перечисленных выше. Если оно представлено в формате “с плавающей запятой”, то значением выражения служит пара десятичных чисел $(A_1 A_2)$, такая, что рассматриваемое число равно произведению A_1 на 10 в степени A_2 (по мере возможности, A_2 выбирается равным 0). Для комплексного числа выдается пара таких пар. В прочих случаях значением выражения служит десятичное число.

Чтобы извлекать элементы массива, используется выражение “выч(значение M i)”. Здесь M — ссылка на массив одного из машинных форматов (кроме длинного целого); i — номер элемента массива, представленный в формате “целое со знаком”. Значением выражения является представление в машинном формате i-го элемента массива.

Операции над числами в одинаковых машинных форматах реализуются выражениями “выч(плюс a b)”, “выч(минус a)” (изменение знака), “выч(вычитание a b)”, “выч(умножение a b)”, “выч(дробь a b)” (в случае целых чисел берется неполное частное), “выч(степень a b)” (в случае целых чисел b должно быть неотрицательным), “выч(модуль a)”. Для вычислений в формате “с плавающей запятой” используются также выражения “выч(логарифм a b)” (a — основание логарифма), “выч(натурлог a)” (натуральный логарифм), “выч(синус a)”, “выч(косинус a)”, “выч(тангенс a)”, “выч(котангенс a)”, “выч(арксинус a)”, “выч(арккосинус a)”, “выч(арктангенс a)”, “выч(арккотангенс a)”, “выч(квадркорень a)” (квадратный корень), “выч(эксп a)” (экспонента), “выч(гипсинус a)” (гиперболический синус), “выч(гипкосинус a)”, “выч(гиптангенс a)”, “выч(гипкотангенс a)”.

Операторные выражения “выч(вещественнаячасть a)” и “выч(мнимаячасть a)” определяют вещественную и мнимую части комплексного числа, представляя их в формате “с плавающей запятой”. Выражение “выч(Комплексное a)” переводит в комплексный формат число a, находящееся в формате “с плавающей запятой”. Выражение “выч(Модуль a)”

определяет представленный в формате “с плавающей запятой” модуль комплексного числа a.

Для получения псевдоконстант при построении графиков используются выражения “выч(плюсбеск)” ($2e99$ - плюс-бесконечность), “выч(минусбеск)” ($-2e99$), “выч(нет)” ($1e99$) — указатель на пропуск точки при построении графика).

Копирование представления числа в машинном формате выполняется при помощи выражения “выч(копия a)”.

Чтобы создать (неинициализированный) массив из n чисел в машинном формате, используется оператор “Выч(набор a n x)”. Здесь a — указатель на тип чисел (логический символ “число” — с плавающей запятой; “целое” — целое со знаком). Переменной x присваивается ссылка на созданный массив. Для регистрации в массиве m в качестве элемента с номером n (нумерация начинается с 0) числа a в соответствующем машинном формате применяется оператор “Выч(запись m n a)”.

Для деления с остатком чисел типа “целое со знаком” служит оператор “Выч(деление m n p q)”. Здесь m, n - делимое и делитель; переменной p присваивается неполное частное, q — остаток.

Сравнение чисел осуществляется операторами “Выч(меньше a b)” и “Выч(меньшеилиравно a b)”, “Выч(равно a b)”. Чтобы проверять отличие числа от 0, служит оператор “Выч(0 a)” (истинен, если a не равно 0).

Оператор “Выч(изменение a b)” изменяет старое представление числа a, перенося в него значение числа b.

Чтобы ускорить вычисления по цепочке действий, не требующих условных переходов, предусмотрено создание микропрограммы — набора P (как структуры данных ЛОСа) псевдокоманд; такая микропрограмма выполняется за одно обращение к оператору “Выч(программа P A B)”. Здесь A — набор вход-выходных данных типа “с плавающей запятой”; B — набор вход-выходных данных типа “целое со знаком”. Если вычисления выходят за рамки о.д.з., то оператор ложен. Каждая псевдокоманда представляет собой набор $(Nn_1 \dots n_k)$, где N — номер операции, $n_1, \dots, n_k$ — номера операндов (входных и выходных). $N, n_1, \dots, n_k$ имеют символьный формат. Номера операндов берутся из набора A либо B, соответствующего типу операнда (такой тип однозначно определяется по номеру операции). Можно выходить за рамки наборов A, B, используя номера, большие их длины (но не большие 200). Такие номера рассматриваются как вспомогательные переменные, которым присваиваются значения, используемые в дальнейших вычислениях по микропрограмме. Нумерация элементов наборов A, B начинается с 1.

Рассматриваются следующие типы псевдокоманд (номер операции совпадает с номером в приводимом списке):

1. Сложение в формате “с плавающей запятой”. Первые два операнда — слагаемые, третий — сумма (аналогичное размещение операндов используется и для других операций).

2. Сложение в формате “целое со знаком”.

3. Изменение знака числа в формате “с плавающей запятой”.

4. Изменение знака числа в формате “целое со знаком”.

5. Умножение в формате “с плавающей запятой”.

6. Умножение в формате “целое со знаком”.

7. Деление в формате “с плавающей запятой”.

8. Деление с остатком для формата “целое со знаком”.

9. Возведение в степень в формате “с плавающей запятой”.

10. Экспонента.

11. Возведение в натуральную степень числа типа “целое со знаком”.

12. Модуль в формате “с плавающей запятой”.

13. Модуль в формате “целое со знаком”.

14. Натуральный логарифм.

15. Логарифм — общий случай.

16. Квадратный корень.

17. Синус.

18. Косинус.

19. Тангенс.

20. Котангенс.

21. Секанс.

22. Косеканс.

23. Арксинус.

24. Арккосинус.

25. Арктангенс.

26. Арккотангенс.

27. Гиперболический синус.

28. Гиперболический косинус.

29. Гиперболический тангенс.

30. Гиперболический котангенс.

31. Сигнум.

32. Тождественная операция (присвоение значения выходной переменной).

2.2.6. Операторы интерфейса

Клавиатура, мышь и меню

Для ввода символов с клавиатуры используется оператор “клавиатура(t)”. Здесь t — выходная переменная, которой присваивается логический символ, кодирующий нажатую клавишу. Для того, чтобы определить в процессе написания программы на ЛОСе логический символ, соответствующий той или иной клавише, достаточно, находясь в текстовом редакторе решателя, нажать сначала клавишу F8, а затем — ту клавишу, код которой определяется; при этом с позиции курсора будет прорисовано название соответствующего логического символа. Драйвер клавиатуры, связанный с оператором “клавиатура”, имеет следующие регистры:

а) Регистры больших и малых латинских букв;

б) Регистры больших и малых русских букв;

в) Регистры “левый Ctr” и “правый Ctr”.

Переключение между латинскими и русскими буквами в этом драйвере выполняется при помощи клавиш F11,F12. Нажатие клавиши F11 переводит драйвер в режим русских букв; F12 — в режим латинских букв. Кроме того, имеется оператор “русшрифт(t_1)”, позволяющий переключать драйвер клавиатуры: при t_1 = “1” — на русский шрифт; при t_1 = “0” — на латинский.

При нажатии левой либо правой клавиши мыши оператор “клавиатура(t)” получает логический символ “мышь”. Для уточнения данных, введенных при помощи мыши, после получения указанным образом логического символа “мышь”, используется оператор “мышь($t_1t_2t_3$)”. Его выходные переменные t_1, t_2, t_3 получают в качестве значений, соответственно, указатель нажатой клавиши (логический символ “0” — нажата левая клавиша; “1” — нажата правая клавиша), номер столбца, на котором при нажатии клавиши находился курсор мыши, и номер строки, где он находился. Эти номера имеют символьное представление (т.е. суть логические символы с номерами $260 + N$, где N — номер столбца либо строки).

Экранные операции

Выдача изображений на экран осуществляется при помощи оператора "видео($Mt_1 \dots t_n$)", где M — логический символ, определяющий тип экранной операции; $t_1, \dots, t_n$ - входные данные этой операции и ее выходные переменные. Параметр n меняется в зависимости от M.

Подготовка области экрана к прорисовке изображения выполняется при помощи операции "видео(прямоугольник $t_1t_2t_3t_4t_5$)". Здесь t_1, t_2 — номера столбца и строки для верхнего левого угла прямоугольника; t_3, t_4 — номера столбца и строки для правого нижнего угла (все номера здесь и в рассматриваемых ниже экранных операциях — в символьном представлении). t_5 - логический символ, определяющий цвет прямоугольника. Для указания цвета используются логические символы начиная с "1" до "шестнадцать" : 1 — черный; 2 — синий; 3 — зеленый; 4 — зеленовато-голубой; 5 — красный; 6 — пурпурный; 7 — коричневый; 8 — светло-серый; 9 — темно-серый; 10 — светло-синий; 11 — светло-зеленый; 12 — светло-голубой; 13 — светло-красный; 14 — светло-пурпурный; 15 — желтый; 16 — белый. Если t_5 равно "0", то изображение на экране внутри указанного прямоугольника не изменяется. В любом случае после применения операции "видео(прямоугольник ...)" границы прямоугольника становятся границами той области, внутри которой перечисляемыми ниже текстовыми операциями выполняется выдача текста (в частности, автоматически осуществляется переход к новой строке по достижении правой границы).

Для прорисовки группы отрезков применяется операция "видео(отрезок t_1t_2)". Здесь t_1 — набор $a = (a_1 \dots a_s); a_i = (a_{i1}a_{i2}a_{i3}a_{i4})$, где a_{i1}, a_{i2} - позиция (номера столбца и строки) начала i - го отрезка; a_{i3}, a_{i4} — позиция конца этого отрезка. t_2 — цвет всех выдаваемых на экран отрезков.

Дуга окружности изображается при помощи операции "видео(окружность $t_1 \dots t_9$)". Здесь t_1, t_2 — координаты центра окружности (номера столбца и строки); t_3 — радиус окружности (в символьном представлении); t_4, t_5 — координаты точки, через которую проходит исходный радиус дуги; t_6, t_7 — координаты точки, через которую проходит заключительный радиус дуги; t_8 — направление прорисовки (логический символ "0" — против часовой стрелки; "1" — по часовой стрелке); t_9 — цвет окружности.

Текстовые фрагменты выдаются на экран решателя при помощи его собственного шрифта, в котором каждый символ представляется матрицей размера 8×19. Интерпретатор ЛОСа нетрудно было бы расширить таким образом, чтобы стало доступным использование стандартных шрифтов, однако при этом потребуется существенная коррекция программ формульного редактора, планирующего компоновку формул с учетом геометрических параметров символьных элементов. В интерфейсе решателя предусмотрена возможность изменения шрифта; кроме того, операторы ЛОСа позволяют вводить и использовать различные наборы шрифтов

указанного выше размера. При работе решателя в оперативной памяти создается специальная область (далее называем ее видеокассой), хранящая двоичные матрицы символьных элементов. Изменение содержимого видеокассы выполняется при помощи операции “видео(бланк $t_1 t_2$)”, где t_1 — логический символ, определяющий прорисовываемую букву (соответствие — то же, что для драйвера клавиатуры); t_2 — набор $(a_1 \dots a_{19})$; $a_i = (a_{i1} \dots a_{i8})$ — набор из логических символов “0” и “1”. Этот оператор заносит двоичную матрицу, определяемую набором t_2, на позицию видеокассы, соответствующую логическому символу t_1. Чтобы прочитать матрицу изображения буквы, хранящейся в видеокассе, применяется операция “видео(развертка t_1 t_2)”. Здесь t_1 — логический символ, определяющий букву; выходной переменной t_2 присваивается набор длины 19, состоящий из восьмиэлементных наборов логических символов “0” и “1” — требуемая матрица изображения. При запуске решателя видеокасса загружается из вспомогательного файла; для сохранения ее измененной версии в этом файле служит специальный оператор (см. ниже операторы работы с файлами решателя).

При выдаче на экран фрагмента текста начальная позиция либо задается в операциях явным образом, либо используются координаты из указателя текущей позиции, который хранит некоторые значения номеров текущих столбца и строки. Для занесения в указатель текущей позиции новых значений t_1 и t_2 номеров столбца и строки используется операция “видео(позиция t_1 t_2)”. Для получения хранящихся в указателе номеров используется операция “видео(точка $t_1 t_2$)”. По окончании выдачи текстовой операцией фрагмента текста указатель текущей позиции автоматически устанавливается на первую свободную после фрагмента позицию. Как уже отмечалось выше, при достижении правой границы текущего прямоугольника (т.е. того, который был указан при последнем обращении к операции “видео(прямоугольник ...)”) происходит автоматическая смена строки. Каких-либо автоматических вставок знаков переноса (по крайней мере на уровне базисных операторов) при этом не предусмотрено.

Выдача буквы с заданной позиции выполняется операцией “видео(буква t_1 t_2 t_3 t_4 t_5)”. Здесь t_1, t_2 — номера столбца и строки позиции, с которой выдается буква; t_3 — логический символ, кодирующий выводимую на экран букву; t_4 — цвет фона; t_5 — цвет символа. Если буква выдается с текущей позиции, то можно положить в качестве t_1 логический символ “продолжение”; в качестве t_2 — логический символ “0”.

Для работы с фрагментами текстов введен специальный накопитель, называемый далее буфером текстов. Он представляет собой массив из 2000 ячеек, каждая из которых хранит некоторый экранный символ (фактически — номер этого символа в видеокассе, т.е. число от 0 до 255), а

также цвет фона и цвет символа. Буфер текстов используется для обменов с файлами, хранящими текстовую информацию, а также для временного хранения текстовых фрагментов (например, ранее выданных на экран и требующих в определенных ситуациях повторной выдачи, быть может, с изменением цветовых атрибутов). Большинство из приводимых далее операций, выдающих на экран текстовые фрагменты, автоматически заносят эти фрагменты в заданную область буфера текстов. При ссылке на ячейки буфера текстов они нумеруются числами от 0 до 1999, причем эти номера имеют символьное представление.

Операция "видео (логсимвол $t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7$)" выводит на экран название логического символа t_3. При этом t_1, t_2 суть номера исходных столбца и строки (при выдаче с текущей позиции — "продолжение" и "0"); t_4, t_5 — цвет фона и символов; t_6 — номер первой ячейки буфера, с которой в него заносится название логического символа; t_7 — выходная переменная, которой присваивается номер первой ячейки буфера текстов, идущей после данного названия. Если на экране либо в буфере текстов не хватило места, то оператор ложен. В этом последнем случае прорисовка на экране начала названия логического символа не выполняется.

Для выдачи на экран подтерма заданного терма в скобочной записи используется операция "видео(терм $t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_8$)". Здесь t_1, t_2 — номера столбца и строки исходной позиции (либо логические символы "продолжение", "0"); t_3 — вхождение первого символа выдаваемого на экран подтерма; t_4 — указатель раскраски. Этот указатель представляет собой набор $(a_0, \ldots, a_s)$, где $a_0 = (a_{01}, a_{02})$ - цвет фона и символов, используемые при прорисовке подтерма; $a_i = (a_{i1}, a_{i2}, a_{i3}), i = 1, \ldots, s$ — указывает, что подтерм рассматриваемого подтерма, вхождение первого символа которого есть a_{i1}, следует раскрасить цветом фона a_{i2} и цветом символов a_{i3}. Вхождения $a_{11}, \ldots, a_{s1}$ упорядочены слева направо, причем соответствующие подтермы могут пересекаться (т.е. быть вложенными друг в друга). Возможность раскраски подтермов существенным образом используется в редакторе программ ЛОСа, так как позволяет создавать многоцветную "указку", выделяющую при просмотре сложных логических конструкций цепочку вложенных термов и облегчающую концентрацию внимания и декомпозицию этих конструкций при чтении программ. Значение t_5 используется в тех случаях, когда весь терм не поместился на выделенную область экрана. Тогда для продолжения выдачи оставшейся части терма (например, после нажатия соответствующей клавиши) входной параметр t_5 полагается равным тому вхождению символа, с которого следует продолжать выдачу подтерма. В исходной ситуации, с начала выдачи подтерма, значение t_5 должно быть равно t_3. t_6 — номер исходной ячейки в буфере текстов, начиная с которой в буфер заносятся

выводимые на экран символы. t_7, t_8 — выходные переменные: t_7 — индикатор переноса: если выдача всего подтерма успешно завершена, то он получает значение “0”; иначе — его значением становится вхождение того символа, с которого следует продолжать выдачу на экран. t_8 становится равно номеру первой ячейки в буфере текстов, следующей после заключительного символа подтерма. Если прорисовка терма на экране нежелательна, а требуется лишь зарегистрировать последовательность выводимых на экран символов в буфере текстов и при этом получить некоторую дополнительную информацию о соответствии вхождений в терм позициям буфера текстов, то в указателе раскраски полагается $a_{01} = a_{02}$. В этом случае, если t_1 положить равным номеру некоторой позиции в буфере текстов, то t_7 становится равно вхождению в подтерм логического символа либо переменной, к прорисовке которого относится указанная позиция. Значение t_2 в указанной ситуации игнорируется. Если в буфере текстов не хватило места, то оператор ложен.

Для выдачи набора термов можно использовать операцию “видео(серия $t_1\ t_2\ t_3\ \ t_4\ t_5\ t_6$)”, которая реализуется быстрее, чем последовательное выполнение операций прорисовки отдельных термов. Здесь t_1, t_2 — номера столбца и строки исходной позиции (либо “продолжение”, “0”);t_3, t_4 - цвет фона и символов; t_5 — набор термов. Термы этого набора последовательно прорисовываются на экране начиная с исходной позиции; буфер текстов при этом заполняется начиная с нулевой ячейки. Выходной переменной t_6 присваивается набор той же длины, что t_5. На позиции этого набора, соответствующей позиции некоторого терма в наборе t_5, располагается пара номеров исходной и заключительной ячеек буфера текстов, содержащих текст данного терма.

Для выдачи на экран содержимого заданного отрезка буфера текстов используется операция “видео(слово $t_1\ t_2\ t_3\ t_4\ t_5$)”. Здесь t_1, t_2 — номера столбца и строки исходной позиции на экране (либо “продолжение”, “0”); t_3 - номер ячейки буфера текстов, начиная с которой выдается текст; t_4 — номер ячейки, до которой включительно происходит выдача (если t_4 — логический символ “продолжение”, то выдача происходит вплоть до первой ячейки, хранящей “машинный ноль” либо, если такой нет, до конца буфера текстов). Выходная переменная t_5 является индикатором переноса: “0” — нет переноса, иначе — номер ячейки буфера текстов, с которой следует продолжать выдачу.

Операция “видео(запись $t_1\ t_2\ t_3\ t_4$)” позволяет занести в ячейку буфера текстов, имеющую номер t_1, букву t_2 (т.е. t_2- кодирующий эту букву логический символ)с цветом фона t_3 и цветом символа t_4. В случае $t_1 = 0; t_2 = t_3 =$ “1” в указанную ячейку заносится “машинный ноль”, который воспринимается рядом операций как указатель на конец ис-

пользуемой части буфера текстов. Операция “видео(значение $t_1\ t_2\ t_3\ t_4$)” присваивает выходной переменной t_2 логический символ, обозначающий букву, расположенную в t_1-й ячейке буфера текстов; при этом значения выходных переменных t_3, t_4 становятся равны, соответственно, цвету фона и цвету символа, хранящимся в ячейке. Операция “видео(0)” заполняет весь буфер текстов “машинными нулями”. Операция “видео(пустоеслово $t_1 t_2$)” заполняет “машинными нулями” отрезок буфера текстов начиная с t_1-й ячейки и кончая t_2-й ячейкой. Для изменения цветовых параметров хранящихся в буфере текстов символов (без изменения их изображения на экране) служат операции “видео(актив $t_1\ t_2\ t_3\ t_4$)” и “видео(вхождение $t_1\ t_2\ t_3$)”. У них t_1, t_2 — номера первой и последней ячеек изменяемого отрезка буфера текстов. В первом случае цвет фона всех букв этого отрезка заменяется на t_3, а цвет символов — на t_4; во втором случае изменяется на t_3 только цвет фона.

Для определения границ текущего прямоугольника служит операция “видео(лимит $t_1\ t_2\ t_3\ t_4$)”, присваивающая своим выходным переменным t_1, t_2, t_3, t_4, соответственно, номера столбца и строки верхнего левого угла прямоугольника и номера столбца и строки правого нижнего угла. Сдвиг изображения внутри текущего прямоугольника выполняется операцией “видео(спуск $t_1\ t_2\ t_3$)”. Здесь логический символ t_1 указывает направление сдвига: “префикс” — вниз; “суффикс” — вверх; “правсосед” — вправо; “левсосед” — влево. t_2 — число (в символьном представлении) пикселей, на которые сдвигается изображение; t_3 — цвет чистых полос, возникающих при сдвиге.

Для сохранения текущего экрана в буфере битмэпов используется операция “видео(минус)”; восстановление экрана по содержимому данного буфера выполняется операцией “видео(плюс)”, причем буфер битмэпов после этого обнуливается. Для временного сохранения содержимого буфера текстов введены два дополнительных буфера текстов. Операция “видео(копия t_1)” при t_1 равном “0” копирует буфер текстов в первый дополнительный буфер; при “3” — во второй; при “2” — обменивает содержимое буфера текстов и первого дополнительного буфера; при “1” — копирует первый дополнительный буфер в буфер текстов; при “4” — копирует второй дополнительный буфер в буфер текстов.

В дополнение к буферу текстов, для сохранения текстовых заготовок в оперативной памяти с целью быстрой выдачи их на экран введен так называемый массив текстов. Главным образом, он используется в текст-формульном редакторе. Размеры массива текстов составляют 56000 ячеек; он позволяет сохранять извлекаемые из буфера текстов фрагменты и возвращать сохраненные фрагменты в буфер текстов для различных экранных операций.

Создание массива текстов осуществляется оператором “видео(начало)”; удаление его — оператором “видео(конец)”. Для пересылки в конец массива текстов фрагмента буфера текстов начиная с позиции m и кончая позицией n, используется оператор “видео(образ m n i j)”. Его выходным переменным i, j присваиваются, соответственно, символьные номера первой и последней ячейки массива текстов, отведенные для сохраненного фрагмента. Для обратной пересылки служит оператор “видео(прообраз m n k)”. Здесь m, n — символьные номера первой и последней ячейки фрагмента массива текстов, k — номер ячейки буфера текстов, начиная с которой размещается фрагмент, извлеченный из массива текстов.

Чтобы расчищать массив текстов, служит оператор “видео(сброс m)”, устанавливающий указатель номера первой неиспользованной ячейки этого массива на символьный номер m. Если нужно изменить ранее записанный в массив текстов фрагмент, то применяется оператор “видео(заменавхождения m n k p)”. Символьные номера m, n здесь указывают начало и конец изменяемого фрагмента; заменяющий фрагмент берется из буфера текстов — начиная с ячейки k и кончая ячейкой p. Если $p < k$, то происходит просто исключение фрагмента массива текстов. Если длина заменяющего фрагмента не равна длине заменяемого, то выполняется соответствующий сдвиг всех фрагментов массива текстов, расположенных после измененного. Оператор “видео(вычеркивание m n)” исключает фрагмент массива текстов начиная с позиции m до позиции n.

Оператор “видео(См m n k p)” позволяет определять находящуюся в массиве текстов на позиции m букву n с цветом фона k и цветом символа p. Оператор “видео(посылка m n k p)”, наоборот, заносит на позицию m массива текстов букву n с цветом фона k и цветом символов p.

Для организации поиска в массиве текстов служит оператор “видео(ключ m n k p)”. По заданным начальной и концевой позициям m, n и букве k он перечисляет все номера позиций p, на которых располагается эта буква.

Для прорисовки графиков функций одной переменной используется оператор “видео(график A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9)”. Здесь A_1 — ссылка на массив числовых данных в формате с плавающей запятой (ординаты точек графика); A_2 — длина этого массива (десятичное число); A_3, A_4 — нижняя и верхняя границы отображаемого на экране диапазона значений (формат с плавающей запятой); A_5, A_6 — верхняя и нижняя полосы для графика (символьные числа); A_7 — столбец, начиная с которого рисуется график (символьное число); A_8, A_9 — цвет фона и цвет линии. Масштабирование таково, чтобы нижняя и верхняя границы диапазона значений соответствовали нижней и верхней полосам графика. Прямоугольник,

выделенный для графика, должен иметь количество столбцов, не меньшее A_2.

Для небольших изображений можно непосредственно на ЛОСе создавать битмэпы, прорисовывать их и сохранять в информационных блоках. Первоначально битмэп формируется в виде набора $(A_1, \ldots, A_m)$ наборов $A_i = (B_{i1}, \ldots, B_{in})$; каждое B_{ij} — логический символ "0" либо "1". Здесь $A_1, \ldots, A_m$ — строки, перечисляемые сверху вниз; в каждой строке элементы перечисляются слева направо. Далее битмэп перекодируется в более компактную структуру данных — некоторый набор логических символов (двоичные знаки выписываются последовательно и делятся на отрезки длины 15; каждый тако отрезок образует номер очередного логического символа, уменьшенный на 1). Такие наборы называем кодами двоичных матриц. Перекодировка выполняется оператором "видео(матрица логсимвол a b)", у которого a — исходное представление битмэпа, b — его код. Обратный переход от кода b к явному представлению битмэпа a осуществляется оператором "видео(матрица набор b m n a)", у которого появляются дополнительные параметры m (число строк) и n (число столбцов). Наконец, собственно прорисовка битмэпа по его двоичному коду b осуществляется оператором "видео(матрица видео p q r s b m n)". Здесь p, q — столбец и строка, определяющие верхний левый край прямоугольника, в котором нужно прорисовать двоичную матрицу; r, s — цвет нулевых и единичных элементов; m, n — число строк и столбцов.

Названия логических символов

В структурах данных решателя логические символы представлены своими номерами (от 1 до 65535), или, точнее, четырехбайтными словами, содержащими эти номера. Для определения номера логического символа по его названию либо названия символа по его номеру используется специальная таблица, хранящаяся в файлах вида Vi.lsi; $i = 1, \ldots, 7$. Каждый такой файл рассчитан на хранение названий 10000 символов. Для хранения названия используется 24-байтное слово, так что предельно допустимая длина названия равна 24. В качестве названия допускаются произвольные последовательности русских либо латинских букв, цифр и спецзнаков. Большие и малые буквы в названии различаются. В начале файла размещаются двухбайтные номера всех логических символов, названия которых он хранит, причем эти номера расположены в лексикографическом для соответствующих названий порядке (эти номера занимают ровно 20000 байт. За ними располагаются сами названия, строго по порядку номеров символов. Если для символа еще не введено название, то соответствующие 24 байта заполнены нулями.

Для работы с названиями логических символов служит оператор "словарь($M\ t_1\ \ldots t_n$)". Здесь M — логический символ, определяющий тип выполняемой операции; $t_1, \ldots, t_n$ - набор входных данных либо выходных переменных. Для ввода нового логического символа с заданным названием (точнее, присвоения этого названия некоторому логическому символу, еще не имеющему названия) служит операция "словарь(запись t_1)". Перед обращением к ней в начальном отрезке буфера текстов должно быть сформировано требуемое название, после которого размещается "машинный ноль". Если логический символ с таким названием уже имеется, то оператор ложен; в противном случае выбирается некоторый логический символ, не имевший названия, ему присваивается указанное в буфере текстов название, и выходной переменной t_1 присваивается номер выбранного логического символа. Если не удалось найти символа без названия, то t_1 присваивается "0". Для определения названия заданного логического символа служит операция "словарь(значение t_1)". Если t_1 - логический символ, то в начальный отрезок буфера текстов заносится слово, обозначающее этот символ. При отсутствии названия данного символа первая ячейка буфера текстов обнуливается. Изменение ранее введенного названия логического символа выполняется операцией "словарь(изменение)". Перед обращением к ней в буфер текстов заносится слово вида "A0B0 ...", где A - заменяемое название; B — заменяющее название; 0 — "машинный ноль". Если B уже использовано либо A не использовано, то оператор ложен; в первом случае название A удаляется, но B не вводится. Проверка существования логического символа с заданным названием осуществляется операцией "словарь(проверка t_1)". Если существует логический символ, название которого хранится в начальном отрезке буфера текстов, то выходной переменной t_1 присваивается этот символ, иначе оператор ложен. Для удаления хранящегося в начальном отрезке буфера текстов названия логического символа служит операция "словарь(исключение)".

Наконец, упомянем оператор "кодтекста($t_1\ t_2\ t_3$)", используемый для перевода текстов термов либо наборов термов в сами термы либо наборы термов. Указанный текст размещается в начальном отрезке буфера текстов. Если t_1 есть "0", то этот текст транслируется в терм; если t_1 есть "1", то текст транслируется в набор термов; если t_1 есть "2", то текст транслируется в набор логических символов и термов (т.е. каждый однобуквенный терм рассматривается как символ); если t_1 есть"3", то текст транслируется в единственный логический символ. Если трансляция выполнена успешно, то выходная переменная t_3 получает значение "0", а выходной переменной t_2 присваивается результат трансляции. Если же в тексте обнаружена ошибка, то t_3 указывает на ее тип:

1) t_3 = “2” — после номера переменной идет посторонний символ. Тогда t_2 — ссылка на начало переменной в буфере текстов.

2) t_3 = “3” — номер переменной больше 511. Тогда t_2 — начало переменной в буфере текстов.

3) t_3 = “4” — неправильно набран логический символ; t_2 - его начало в буфере текстов.

4) t_3 = “5” — нехватка места для формирования результата.

5) t_3 = “6” — избыточная правая скобка; t_2 - ссылка на начало текста символа, после которого идет эта скобка.

6) t_3 = “8” — незакрытая левая скобка.

Словарь текстового анализатора

Для работы с текстами естественного языка используется словарь текстового анализатора, хранящий 65535 фрагментов слов (корни, окончания, суффиксы, приставки). Длина одного словарного фрагмента не должна превышать 24 букв. Каждому словарному фрагменту этого словаря (далее называемого внешним словарем) сопоставлен кодирующий его логический символ. При чтении слова оно разбивается на фрагменты, и решателю передаются лишь коды фрагментов.

Для регистрации во внешнем словаре нового словарного фрагмента сначала происходит размещение этого фрагмента в некотором отрезке буфера текстов, и далее выполняется оператор “внешсловарь(запись s m n)”, где m, n — номера начальной и последней ячеек отрезка; s — логический символ, который становится кодом фрагмента. Символ s мог уже являться кодом некоторого словарного фрагмента; в этом случае старая версия фрагмента изменяется во внешнем словаре на новую. Чтобы получить словарный фрагмент, закодированный некоторым логическим символом, применяется оператор “внешсловарь(значение s m p q n)”. У него s — логический символ, кодирующий фрагмент; m — номер ячейки буфера текстов, начиная с которой размещается словарный фрагмент; p,q — цвет фона и символов для размещаемого в буфере текстов фрагмента. Выходной переменной n передается номер первой ячейки буфера текстов после фрагмента. Если символ s не являлся кодом словарного фрагмента, то оператор ложен.

Чтобы исключить из внешнего словаря словарный фрагмент, кодируемый логическим символом s, применяется оператор “внешсловарь(исключение s)”.

Для чтения слова и разбиения его на словарные фрагменты применяется оператор “поиксслова(m s n)”. В качестве входного данного он получает

номер m позиции в буфере текстов, начиная с которого размещается еще не прочитанная часть слова. Выходная переменная s перечисляет логические символы, являющиеся кодами словарных фрагментов, расположенных в буфере текстов начиная с позиции m. При этом переменная n перечисляет номера первых идущих после указанных словарных фрагментов позиций буфера текстов. Окончательный отбор разбиения слова на словарные фрагменты выполняется уже с помощью программы, реализованной на ЛОСе.

Имеется возможность просмотра всех словарных фрагментов внешнего словаря, начинающихся с заданной последовательности букв. Для этого служит оператор "текслово(m n s)". Он получает в качестве входных данных номера m, n первой и последней ячеек буфера текстов, между которыми хранится указанная последовательность букв. Выходная переменная s перечисляет коды словарных фрагментов, начинающихся с данной последовательности. Перечисление происходит в лексикографическом порядке.

Информационные блоки

Логическая и текстовая информация, используемая решателем, может быть размещена в специальным образом организованных файлах. Эти файлы разбиты на группы, называемые далее информационными блоками. Информационный блок хранит древовидную конструкцию, образованную объектами следующих типов:

а) Корневой указатель — каталог. Этот объект фактически представляет собой таблицу ссылок на другие объекты информационного блока, длина которой равна числу логических символов, т.е. 65535. Если на i - й позиции этой таблицы находится 0, то ссылка по логическому символу с номером i из данного указателя считается неопределенной, иначе — определен переход из указателя по i-му символу к некоторому объекту информационного блока.

б) Указатель — список. Этот объект представляет собой набор $((a_1, S_1), \ldots (a_n, S_n))$ пар, имеющих попарно различные первые элементы a_i; каждый такой элемент (называемой меткой перехода в данном указателе) представляет собой либо логический символ, либо неоднобуквенный терм. S_i есть набор ссылок на объекты информационного блока, к которым осуществляется переход по метке a_i.

в) Логический терминал. Этот объект представляет собой набор логических символов, переменных и неоднобуквенных термов (однобуквенные термы при записи их в логический терминал автоматически преобразуются в логические символы либо символы переменных).

г) Текстовый терминал. Этот объект представляет собой просто последовательность байт, кодирующих текст (в зависимости от способа хранения текста, между байтами, кодирующими символы, могут вставляться байты, кодирующие их цветовые атрибуты).

В действительности структура ссылок в информационном блоке "почти" древовидная, так как разрешаются ссылки на один и тот же терминал из различных указателей.

Интерпретатор ЛОСа обеспечивает работу с 16 информационными блоками, имеющими номера от 1 до 16. За ними закреплена определенная функциональная нагрузка, которая будет уточняться далее по мере описания структуры и функционирования решателя. Корневым объектом блоков с номерами 1 и 4 является указатель — список; каждый из этих блоков представляет собой единственный файл: для блока 1 это файл I0000.lsi; для блока 4 - файл R0000.lsi. Остальные блоки имеют своим корневым объектом указатель — каталог и состоят из группы файлов. Все файлы одного и того же информационного блока имеют вид Xijkl.lsi, где X — буква, соответствующая номеру блока; ijkl — восьмеричный номер файла в блоке. Нумерация файлов сплошная и начинается с 0000. Указатель-каталог в информационном блоке может быть лишь корневым; он занимает полностью файл X0000.lsi. Буквенная кодировка информационных блоков (кроме уже указанных 1-го и 4-го) такова: E — 2; H — 3; Q — 5; M — 6; W — 7; P — 8; T — 9; A — 10; B — 11; C — 12; D — 13; F — 14; G — 15; J — 16. Каждый из файлов информационного блока имеет не более 4 Мб; по мере увеличения размеров этих файлов происходит автоматическое разрезание их на две приблизительно одинаковые по размеру части, со сдвигом нумерации последующих за разрезаемым файлов. Все объекты, достижимые из корневого каталога по заданному логическому символу (т.е. "ветвь" этого логического символа в информационном блоке), размещаются в одном и том же файле, так что переходы через указатель — список возможны только в рамках одного файла.

Для работы с информационными блоками используются операторы вида "файл($M\ t_1\ \ldots t_n$)" и "указатель($M\ t_1 \ldots t_n$)". Здесь M — логический символ, определяющий тип выполняемой операции; $t_1, \ldots, t_n$ — набор входных данных либо выходных переменных. Операции применяются к заранее выделенному "активному" информационному блоку. Выделение информационного блока с номером t_1 (номер — в символьном представлении) вместо ранее выделенного осуществляется при помощи операции "файл(актив t_1)". Если информационного блока с таким номером еще не было, то он при этом будет введен (пока без корневого объекта). Проверка наличия информационного блока с номером t_1 выполняется оператором

“файл(проверка t_1)”. Операция “файл (число t_1)” присваивает выходной переменной t_1 номер активного информационного блока.

Для ссылок на объекты активного информационного блока в ЛОСе служат наборы (a_1, a_2), образованные двумя логическими символами a_1, a_2. Эти символы кодируют некоторым образом (подробнее см. в описании интерпретатора ЛОСа) номер файла информационного блока и смещение в данном файле начала рассматриваемого объекта. Ссылкой на корневой объект произвольного информационного блока служит пара (“0”, “3”).

Для перехода по заданной метке в корневом указателе - каталоге служит операция “указатель(логсимвол $t_1\ t_2\ t_3$)”, где t_1 — ссылка на каталог (т.е. набор (“0”, “3”)); t_2 — логический символ, представляющий собой метку перехода. Если переход по данной метке в каталоге не определен, то оператор ложен. Иначе — выходной переменной t_3 присваивается ссылка на объект, к которому имеет место переход по метке t_2.

Для перехода по заданной метке в указателе — списке служит операция “указатель(список $t_1\ t_2\ t_3$)”. Здесь t_1 — ссылка на рассматриваемый указатель-список; t_2 — метка перехода (логический символ либо неоднобуквенный терм). Если в данном указателе-списке не предусмотрен переход по метке t_2, то оператор ложен. Иначе выходной переменной t_3 присваивается набор ссылок на объекты, переход к которым осуществляется по данной метке. Заметим, что порядок ссылок в наборе — обратный, т.е. первой идет ссылка, которая была зарегистрирована (с помощью указываемых далее операций) последней.

Чтобы определить множество всех меток перехода, используемых в указателе- списке, применяется операция “указатель(область $t_1\ t_2$)”. У нее t_1 — ссылка на указатель; выходной переменной t_2 присваивается набор всех меток перехода в данном указателе. Ввод нового указателя (каталога либо списка) осуществляется операцией “указатель(новый $t_1\ t_2\ t_3\ t_4\ t_5$)”. Здесь t_1 — логический символ “логсимвол” (если вводится корневой каталог — в случае пустого информационного блока) либо “список” (если вводится указатель - список). Если вводится корневой указатель, то $t_2 = t_3 =$ “0”; иначе t_2 — ссылка на внешний указатель, а t_3 — метка перехода, по которой в нем регистрируется новый указатель. В случае ввода каталога t_4 игнорируется, а в случае указателя - списка t_4 есть длина в байтах поля, зарезервированного для нового пустого указателя. При превышении данной длины регистрация в указателе новых ссылок связана с переписыванием в файле всего указателя на новой позиции, а до этого регистрация выполняется без переписывания указателя. Для оценки величины t_4 уточним, что хранение n ссылок по одной метке M требует $3n + 3m + 2$ байт, где m — число логических символов и “самостоятельных” (не идущих непосредственно за логическим символом либо

символом переменной) закрывающих скобок в M. Выходной переменной t_5 присваивается ссылка на новый указатель.

Исключение объекта, на который имеется ссылка из некоторого указателя (каталога либо списка), осуществляется операцией "указатель(исключение $t_1\, t_2\, t_3$)". Здесь t_1 - ссылка на данный указатель; t_2 — метка, по которой из него достижим исключаемый объект. Если указатель является каталогом, то t_3 несущественно, иначе t_3 — ссылка на исключаемый объект. Происходит удаление ссылки из указателя по метке t_2 на исключаемый объект. Если этот объект представлял собой указатель-список, то вся его ветвь автоматически удаляется из информационного блока. Сохраняются лишь те ее терминалы, на которые имеются ссылки из указателей, не относящихся к данной ветви. Если исключаемый объект — терминал, то он удаляется из информационного блока лишь при условии, что на него не было других ссылок.

Ссылка на объект может быть извлечена из одного указателя и перенесена в другой указатель. Это выполняется операцией "указатель(перестановка $t_1 t_2 t_3 t_4 t_5$)". Здесь t_1 — ссылка на первый указатель (каталог либо список); t_2 — метка, по которой из него имеется ссылка на переносимый объект. Если первый указатель является каталогом, то t_3 несущественно, иначе t_3 — ссылка на переносимый объект. t_4 есть ссылка на второй указатель (заметим, что он может совпадать с первым); t_5 — метка, по которой переносимый объект требуется зарегистрировать во втором указателе. Если указатели и метки совпадают, то просто происходит изменение порядка ссылок по рассматриваемой метке (новая ссылка заносится в начало списка ссылок). Важно заметить, что оператор применим лишь тогда, когда оба указателя относятся к одному и тому же файлу информационного блока (это гарантируется для блоков с номерами, отличными от 1 и 4, лишь при условии, что оба они относятся к ветви одного и того же логического символа в корневом каталоге). Если это не так, то для перенесения объекта (фактически — ветви информационного блока) требуется создать его копию и удалить оригинал.

Если некоторый терминал информационного блока уже создан и необходимо создать дополнительную ссылку на него, то применяется операция "указатель(терминал $t_1\, t_2\, t_3$)". Здесь t_1 — ссылка на указатель, из которого создается дополнительная ссылка; t_2 — метка этой ссылки; t_3 — ссылка на терминал. Как и в предыдущем случае, новый указатель должен быть расположен в том же файле информационного блока, что и терминал.

Для определения типа объекта информационного блока служит операция "указатель(тип $t_1\, t_2$)". Здесь t_1 - ссылка на объект. Выходной переменной t_2 присваивается логический символ, указывающий тип объекта: "логсим-

вол" - каталог; "список" — указатель-список; "терминал" — текстовый терминал; "терм" — логический терминал.

Если все метки некоторого указателя — списка суть логические символы (как правило, номера в символьном представлении), то для одновременного увеличения либо уменьшения на заданную величину всех его меток, больших или равных заданной, служит операция "указатель(плюссимв $t_1\ t_2\ t_3$)". Здесь t_1 — ссылка на указатель; t_2 — метка, начиная с которой происходит смещение номеров; t_3 — логический символ, определяющий константу сдвига. Если он больше "0" (т.е. его номер больше 260), то номера увеличиваются, иначе — уменьшаются. Эта операция позволяет модифицировать указатель без переписывания его в файле.

Перечислим операции, используемые для чтения хранящихся в файле терминалов. Операция "файл(терм $t_1\ t_2$)" по ссылке t_1 на логический терминал присваивает выходной переменной t_2 набор записанных в этом терминале логических символов, символов переменных и неоднобуквенных термов. Если t_1 — ссылка на текстовый терминал, в котором хранится текст с пропущенными цветовыми атрибутами, то операция "файл(набор $t_1\ t_2\ t_3\ t_4\ t_5$)" переносит этот текст в буфер текстов начиная с ячейки данного буфера, имеющей номер (в символьном представлении) t_4. При этом все символы получают одинаковые цвет фона t_2 и цвет символов t_3. Выходной переменной t_5 присваивается номер последней использованной при чтении ячейки буфера текстов. Если же текстовый терминал хранит текст с явно указанными цветовыми атрибутами, то для его чтения используется операция "файл(слово $t_1\ t_2\ t_3$)", где t_2 — номер начальной ячейки буфера текстов; t_3 — выходная переменная, которой присваивается номер последней использованной ячейки. Если текстовый терминал по ссылке t_1 хранит таблицу видеокассы, то для загрузки ее в видеокассу используется операция "файл(видео t_1)".

Ввод нового терминала осуществляется операцией "файл(запись $t_1\ t_2\ t_3\ t_4\ t_5$)". Здесь t_1 - логический символ, определяющий тип вводимого объекта: "терм" - логический терминал; "слово" — текстовый терминал с явно указанными цветовыми атрибутами; "набор" — текстовый терминал с пропущенными цветовыми атрибутами; "видео" — текстовый терминал, хранящий таблицу видеокассы. Если создается логический терминал, то t_2 — набор логических символов, символов переменных и термов (допускаются и однобуквенные термы, однако в терминал они записываются как символы, и при чтении из терминала восстанавливаются как символы). В остальных случаях значение t_2 несущественно. Содержимое нового текстового терминала извлекается из буфера текстов начиная с нулевой ячейки и кончая указателем конца текста ("машинный нуль") либо концом буфера. t_3 — ссылка на указатель (каталог либо список),

в котором регистрируется новый терминал; t_4 — метка, по которой он регистрируется. Выходной переменной t_5 присваивается ссылка на новый терминал.

Изменение содержимого терминала выполняется при помощи операции "файл (изменение $t_1\ t_2\ t_3\ t_4$)". Здесь t_1 — логический символ, определяющий тип терминала таким же образом, как в операции "файл(запись ...)"; если изменяется логический терминал, то t_2 задает новый набор символов и термов, иначе t_2 несущественно. Как и при вводе терминала, информация для изменения текстовых терминалов берется из начального отрезка буфера текстов. t_3 — ссылка на изменяемый терминал. Выходной переменной t_4 присваивается ссылка на измененный терминал. Так как изменение терминала связано с переписыванием его новой версии в конце файла (старая версия снабжается специальной пометкой, причем ее поле "портится" - используется для хранения ссылки на новую версию), то при последующей работе с терминалом следует использовать только ссылку t_4.

При изменении указателей и терминалов, хранящихся в информационном блоке, старые их версии (снабженные специальными пометками) накапливаются и приводят к дополнительному увеличению размеров файла. Кроме того, так как переход к новой версии объекта осуществляется по цепочке ссылок, хранящихся в старых его версиях, может несколько замедляться работа программ. Поэтому предусмотрена процедура "уплотнения" информационного блока, переписывающая измененные его файлы без старых версий объектов и выполняющая необходимую коррекцию всех ссылок (смещений в файле) из указателей. Эта же процедура, при усмотрении превышающего 450 Кбайт файла информационного блока (кроме блоков 1 и 4), осуществляет разрезание его на две примерно равные части, с увеличением на 1 номеров последующих файлов данного блока. Обращение к указанной процедуре выполняется операцией "файл(уплотнение)". После выполнения ее информационный блок перестает быть активным. Если t_1 — ссылка на некоторый объект информационного блока (кроме корневого каталога), который был изменен, и таким образом содержит ссылку на новую его версию, та — ссылку на ее новую версию, и т.д., то ссылка на концевой объект данной цепочки (т.е. на "реальную" текущую версию объекта) присваивается операцией "файл(ссылка $t_1\ t_2$)" выходной переменной t_2. Операция "файл(выписка t_1)" присваивает выходной переменной t_1 набор номеров всех информационных блоков, для которых, возможно, требуется уплотнение (т.е. в некотором файле блока имеется хотя бы одна "отключенная" версия объекта).

Блок программ ЛОСа

Программа решателя, записанная на ЛОСе, хранится в группе файлов, называемой далее блоком программ. Эти файлы имеют вид Lijkm, где i,j,k,m — восьмеричные цифры. Файл L0000 хранит каталог программ — в нем для каждого логического символа, имеющего свою программу, указана ссылка на корневой фрагмент данной программы. Такая ссылка состоит из номера файла Lijkm (отличного от L0000) и смещения в этом файле начала фрагмента (более подробно формат данных в файлах блока программ приведен в описании интерпретатора ЛОСа). Все фрагменты программы одного и того же логического символа хранятся в одном файле блока программ. Так как программы языка ЛОС реализуются интерпретатором, то существенно облегчено изменение программой своих собственных фрагментов непосредственно в процессе работы решателя. Разумеется, здесь должны соблюдаться определенные ограничения на изменение тех фрагментов программ, которые на текущий момент "активизированы", т.е. начато их выполнение и в стэках интерпретатора хранятся значения программных переменных, связанных с этими фрагментами. Однако, эти фрагменты продублированы в оперативной памяти, так что даже их при определенных условиях можно изменить. Например, реализованный на ЛОСе программный редактор позволяет изменять все без исключения фрагменты программ, в том числе свои собственные (последние изменения требуют особой аккуратности, во избежание необратимой порчи блока программ). Для обеспечения возможности восстановления файлов решателя при аварийных ситуациях в его интерфейсе предусмотрена возможность копирования полного комплекта его файлов в резервную директорию. Перед копированием выполняется проверка корректности структур данных в этих файлах (как в блоке программ, так и в информационных блоках); при обнаружении нарушений происходит выход в операционную систему, а копирование не выполняется.

Для работы с фрагментами программ в ЛОСе используется оператор "прогфайл ($M\ t_1 \ldots t_n$)". Здесь M — логический символ, определяющий тип выполняемой операции; $t_1, \ldots, t_n$ — набор входных данных либо выходных переменных. Ссылка на фрагмент программы в блоке программ представляет собой пару логических символов. Более подробно структура таких ссылок будет указана ниже, при описании интерпретатора (она несущественна при использовании приводимых далее операций). Операция "прогфайл(логсимвол $t_1\ t_2\ t_3$)" позволяет по заданному логическому символу t_1 находить ссылку (t_2, t_3) на начало корневого фрагмента программы символа t_1; если такой программы нет, то оператор ложен. Операция "прогфайл(значение $t_1\ t_2\ t_3$)" по ссылке (t_1, t_2) на фрагмент в блоке программ присваивает выходной переменной t_3 набор операторов этого фрагмента (каждый оператор, в том числе односимвольный, представляется термом). Операторы "ветвь N" и "иначе N" в данном наборе

представлены как “ветвь(a_1a_2)”; “иначе(a_1a_2)”, где (a_1, a_2) — ссылка на тот фрагмент, переход к которому определяется оператором.

Изменение ранее введенного фрагмента программы либо ввод нового фрагмента выполняются операцией “прогфайл(запись $t_1t_2t_3t_4t_5t_6$)”. Здесь t_4 — набор операторов (термов), образующих новый фрагмент. В нем могут встречаться в указанном выше формате ссылки на другие фрагменты программы (напомним, что повторные ссылки на один и тот же фрагмент программы не разрешаются, так что те фрагменты, на которые имеются ссылки из нового фрагмента, либо должны быть непосредственными подфрагментами заменяемого фрагмента, либо должны быть заранее “отключены” при помощи операции “прогфайл(вычеркивание ...)”; см. ниже). Кроме того, допускаются термы вида “ветвь(m)”; “иначе(m)”, у которых m — логическиий символ, представляющий собой метку недоопределенного перехода из фрагмента. Такие недоопределенные переходы, после записи их в блок программ, восстанавливаются в том же виде операцией “прогфайл(значение ...)”. Если новый фрагмент заносится в качестве корня программы некоторого логического символа f (возможно, уже имевшейся до этого), то t_1 есть символ “0”; $t_2 = f$; значение t_3 несущественно. Иначе t_1 равно “1”; (t_2, t_3) есть ссылка на вхождение оператора “ветвь” либо “иначе” в тот внешний фрагмент, который через данный оператор должен ссылаться на новый фрагмент. Под ссылкой на вхождение оператора перехода “ветвь” либо “иначе” в некоторый фрагмент здесь понимается пара (A_1, A_2), у которой A_1 — ссылка на фрагмент (пара логических символов); A_2 — номер (начиная с 1) данного оператора перехода в фрагменте. Если новый фрагмент F заносится вместо некоторого другого фрагмента G, то G и все ветви фрагмента G, не упомянутые в F, удаляются. По окончании регистрации в блоке программ нового фрагмента выходным переменным t_5, t_6 присваивается ссылка на этот фрагмент.

Если фрагмент F и всю его ветвь требуется временно отключить для последующего использования в другом месте программы, то применяется операция “прогфайл (вычеркивание $t_1\,t_2\,t_3\,t_4$)”. Если указанный фрагмент F является корнем программы некоторого логического символа f, то t_1 есть “0”; $t_2 = f$; t_3 — несущественно. Иначе t_1 равно “1”; (t_2, t_3) — ссылка на вхождение оператора “ветвь” либо “иначе” во внешний фрагмент, по которому происходит переход к фрагменту F. Если t_1 есть “0”, то из каталога программ исключается ссылка по символу f; иначе — ссылка из внешнего фрагмента на F заменяется меткой t_4 недоопределенного перехода (логическим символом). В обоих случаях исключаются только ссылки на фрагмент F, а сам он и вся его ветвь сохраняются в блоке программ без изменений.

Для удаления ветви ранее отключенного фрагмента программы используется операция “прогфайл(исключение $t_1\ t_2$)”. Здесь (t_1, t_2) — ссылка на данный фрагмент. Этот фрагмент и все достижимые из него фрагменты снабжаются в блоке программ специальными пометками. Аналогичным образом, операция “прогфайл(сброс t_1)” уничтожает всю программу логического символа t_1. Операция “прогфайл(корень $t_1\ t_2\ t_3$)” подключает ранее отключенную ветвь фрагмента, ссылкой на который является пара (t_2, t_3), в качестве программы логического символа t_1 (ранее имевшаяся программа этого символа удаляется).

При замене фрагмента программы на новую его версию старая версия снабжается специальной пометкой, но сохраняется в файле блока программ. Для того, чтобы периодически “расчищать” блок программ от накапливающихся в нем неиспользуемых отрезков, используется операция “прогфайл(уплотнение)”. Она инициирует перезапись фрагментов программ без сохранения указанных отрезков, с соответствующей коррекцией ссылок между фрагментами. Так как при этом полностью нарушается корректность “старых” ссылок из регистров интерпретатора и из хранящихся в оперативной памяти копий фрагментов программы на фрагменты в блоке программ, то после указанного уплотнения блока программ происходит автоматический перезапуск решателя (это выглядит как возвращение к исходному меню). Заметим, что процедуре уплотнения подвергаются не все файлы блока программ, а лишь те из них, которые были изменены после последнего уплотнения. Как и при уплотнении информационных блоков, происходит разрезание пополам тех файлов, которые превысили 450 Кбайт. Заметим, что процедура уплотнения сначала предпринимает проверку корректности блока программ, и лишь после этого начинает собственно уплотнение. При обнаружении ошибки эта процедура выдает значение “истина” (в отличие от процедуры уплотнения информационных блоков), а в случае успешного уплотнения — значение “ложь”.

Если произошло изменение программы некоторого логического символа, причем перезапуск решателя нежелателен (например, в цикле логического вывода и автоматической генерации приемов), то для устранения рассогласования между копиями фрагментов программы, хранящимися в оперативной памяти, и измененными фрагментами в блоке программ, используется операция “прогфайл (компонента t_1)”. Эта операция удаляет из оперативной памяти все копии фрагментов программы логического символа t_1 и восстанавливает в копии каталога программ, хранящейся в оперативной памяти, ссылку на корневой фрагмент этой программы по блоку программ. Операция “прогфайл(пересмотр)” уничтожает вообще все хранящиеся в оперативной памяти решателя копии фрагментов программ и реализует его перезапуск.

Приведем в заключение ряд достаточно редко используемых операций над блоком программ. Операция “прогфайл(позиция $t_1\,t_2\,t_3\,t_4\,t_5\,t_6$)” по ссылке (t_1, t_2) на фрагмент программы; набору t_3 операторов этого фрагмента и вхождению t_4 в этот набор оператора “ветвь” либо “иначе” присваивает выходным переменным t_5, t_6 ссылку на вхождение данного оператора в рассматриваемый фрагмент (см. выше формат таких ссылок при описании операции “прогфайл(запись ...)”). Эта операция сохранилась от старой версии интерпретатора, в которой формат ссылок был иным; требуемую ссылку теперь легко получить простым подсчетом количества операторов перехода, предшествующих данному. Операция “прогфайл(продолжение $t_1\ t_2\ t_3$)” используется для поиска недоопределенных переходов. У нее t_1 - логический символ, в программе которого ищутся такие переходы. Находится первый такой переход “ветвь А” либо “иначе А”; здесь А - логический символ, являющийся меткой недоопределенного перехода. Определяется путь $(F_1, \ldots, F_s)$ от корневого фрагмента F_s программы символа t_1 к тому фрагменту F_1, в котором найден указанный переход (т.е. каждый фрагмент F_{i+1} имеет ссылку на F_i; $i = s-1, \ldots, 1$). Выходной переменной t_2 присваивается метка А; переменной t_3 — набор $(a_1, \ldots, a_s)$ пар логических символов — ссылок на фрагменты $F_1, \ldots, F_s$. Операция “прогфайл(имя $t_1\ t_2\ t_3$)” позволяет по заданным логическому символу t_1 и оператору t_2 (терму) находить первый фрагмент программы символа t_1, содержащий данный оператор. При этом выходной переменной t_3 присваивается такой же набор ссылок, определяющих путь к найденному фрагменту, как и для предыдущей операции.

Операции трассировки

Для отладки программ языка ЛОС используется программа логического символа “прерывание”, написанная на ЛОСе. Эта программа реализована как программа фиктивного операторного выражения (именно, однобуквенного терма “прерывание”), причем данное операторное выражение в явном виде в программах ЛОСа не встречается, а обращение к указанной программе выполняется интерпретатором автоматически — при обнаружении тех или иных ошибочных действий либо при выполнении условий обрыва, заданных используемым режимом трассировки. Программа символа “прерывание” реализует обычные операции отладчика — позволяет устанавливать требуемый режим трассировки, просматривать программу текущего и внешних операторов, определять значения программных переменных, и т.п. Более подробно об этой программе будет рассказано в специальном разделе книги. Для использования в отладчике был введен ряд базисных операций ЛОСа, объединенных в операторе “трассировка($M\,t_1 \ldots t_n$)”. Эти операции существенным образом используют специфику применяемого интерпретатора ЛОСа, поэтому их

описание будет приведено в разделе, посвященном программе отладчика ЛОСа.

3. Редактор программ ЛОСа

3.1. Перечень названий операторов ЛОСа

Для поиска нужного оператора ЛОСа создано оглавление, в котором базисные и основные реализованные на ЛОСе операторы и операторные выражения ЛОСа упорядочены по своему назначению. Это оглавление достижимо из главного меню ЛОСа при нажатии клавиши “о” (кир.); соответствующий пункт находится в правом нижнем углу. В него также можно войти из редактора программ ЛОСа (см. ниже). Через концевой пункт оглавления осуществляется выход (нажатием клавиши “курсор вправо”) в справочную информацию о логическом символе, являющемся заголовком соответствующего оператора либо операторного выражения. Интерфейс работы с такой справочной информацией уже был описан выше. Заметим, что в данном случае появляется возможность перехода к просмотру корневого фрагмента программы текущего логического символа — для этого достаточно нажать клавишу “Home”. Для возвращения из просмотра программы нажимается “End”.

Оглавление операторов ЛОСа можно пополнять самостоятельно, используя для этого следующие действия. После создания программы для нового оператора либо операторного выражения следует найти подходящий раздел оглавления либо ввести новый такой раздел; в нем создать концевой пункт с краткой характеристикой действий, выполняемых новой программой. Далее - нажать клавишу “Enter”. Текст пункта окажется перерисованным в верхней части экрана; под ним будет проведена горизонтальная линия. После повторного нажатия “Enter” под данной линией возникнет курсор текстового редактора, с помощью которого следует ввести название нового оператора (операторного выражения). После завершения ввода (еще одно “Enter”) нажимается “курсор влево”, и новый концевой пункт создан.

3.2. Интерфейс редактора программ

Как уже говорилось выше, программа на ЛОСе представляет собой дерево фрагментов, переходы между которыми выполняются с помощью операторов перехода “ветвь N”, “иначе N”. Каждый фрагмент является последовательностью операторов ЛОСа. Редактор программ ЛОСа осуществляет постраничный показ фрагментов программы, так что каждый

фрагмент целиком умещается на экране. Никаких прокруток изображения на экране при этом не предусмотрено. Если фрагмент слишком большой и целиком на экране не помещается, то он разрезается на части, между которыми имеются линейные переходы: в конце очередной части помещаются операторы "ветвь N", "продолжение", где N — номер перехода к следующей части. Операторы размещаются в программе, отделенные друг от друга пробелами; никаких других разделяющих знаков быть не должно. Их последовательность не форматируется; при автоматической перерисовке операторы идут друг за другом с промежутками ровно в один пробел. Если строка заканчивается, то текст продолжается с начала следующей строки без каких-либо пробелов и знаков переноса. Разумеется, такой стиль изображения программы отличается от общепринятого. Однако, плотный режим размещения операторов имеет свои достоинства: удается разместить больший объем информации на меньшем пространстве, что упрощает анализ контекста при написании программы и ее чтении. Решающим фактором, сделавшим плотный режим размещения операторов практически приемлемым и даже удобным, оказалась специальная многоцветная указка, которая, по существу, заменяет форматирование, позволяя концентрировать внимание на нужном участке текста и определяя архитектуру его включения в целый текст. Разумеется, возможно создать более традиционные способы изображения ЛОС-программы на экране — если они обеспечат большую эффективность программирования, чем настоящая версия. Впрочем, уже сейчас подавляющее большинство приемов записывается не на ЛОСе, а на языке значительно более высокого уровня — ГЕНОЛОГе. Поэтому более перспективным направлением, чем совершенствование интерфейса редактора программ ЛОСа, представляется (естественное для логической системы) развитие средств автоматического создания ЛОС — программ.

3.2.1. Вход в редактор программ ЛОСа

Вход в редактор программ ЛОСа через главное меню обычно осуществляется одним из двух способов — с помощью явного указания логического символа, программу которого нужно просматривать либо редактировать, либо через оглавление программ ЛОСа, в котором представлены основные блоки системы или большие вспомогательные процедуры общего назначения.

Для входа в корневой фрагмент программы логического символа A нужно либо нажать клавишу "п", либо нажать левую клавишу мыши, переведя ее курсор в окно главного меню "Просмотр программы логического символа". В обоих случаях в синей рамке, расположенной внутри этого окна, появится курсор текстового редактора. Далее нужно набрать

название символа A и нажать “Enter”. При наборе последовательности букв, не являющейся названием логического символа, синяя рамка снова становится пустой, и набор следует повторить либо нажать “Esc”. При правильном наборе символа на экране появится изображение корневого фрагмента программы символа A. Если такой программы еще не было создано, то в верхней части экрана будет перерисовано название символа A, а остальная часть экрана будет пустой.

Для входа в оглавление основных программ ЛОСа из главного меню нужно либо нажать клавишу “л”, либо переместить курсор мыши в окно “Оглавление программ” и нажать левую клавишу мыши. После этого на экране возникает подменю оглавления программ, которое рассматривалось перед последним выходом из этого оглавления. Неконцевые пункты оглавления программ соответствуют основным блокам либо вспомогательным процедурам системы, или подблокам таких блоков и процедур. Фактически ветвь оглавления программ является вынесенной в отдельную структуру данных иерархической системой комментариев к программе. Концевые пункты этой ветви жестко связаны с конкретными точками в программе; выйдя на такой пункт и нажав клавишу “курсор вправо”, можно оказаться в соответствующей контрольной точке программы. Эта точка будет обозначена в прорисованном на экране фрагменте программы выделенным синим цветом фиктивным оператором “прием(N)”; N - последовательность цифр, образующая номер контрольной точки. Нумерация контрольных точек — своя для каждого логического символа. Для возвращения в оглавление из просмотра фрагмента программы следует нажать “End”. Для перехода к просмотру надфрагмента либо подфрагментов, либо для изменения фрагмента нужно выйти из режима просмотра подтермов операторов фрагмента, в котором мы оказываемся после нажатия “курсор вправо”, нажав клавишу “о”. Заметим, что возвращение в оглавление может быть осуществлено из любого фрагмента просматриваемой программы (причем в тот концевой пункт оглавления, из которого был сделан переход к рассмотрению программы). Однако, для такого возвращения нужно сначала восстановить режим просмотра подтермов операторов (снова нажатием клавиши “о”). Способ регистрации в оглавлении программ ЛОСа новых контрольных точек описывается ниже.

В принципе, возможен еще один способ входа в просмотр фрагментов программы — через оглавление операторов ЛОСа. Это оглавление (как уже говорилось выше) достижимо из главного меню по клавише “о” (выбор пункта в правом нижнем углу экрана). После перехода в концевой пункт такого оглавления и нажатия клавиши “курсор вправо” появляется справочная информация о логическом символе — заголовке оператора.

При нажатии “Home” здесь осуществляется переход к просмотру корневого фрагмента программы данного логического символа (но уже не в режим просмотра подтермов операторов, а в обычный режим просмотра). После просмотра и редактирования этого либо других фрагментов той же программы возможно возвращение в оглавление операторов — по нажатии клавиши “End”.

Наконец, имеется возможность входа в просмотр программы приема, заданного на ГЕНОЛО́Ге — об этом будет сказано в разделе, посвященном редактору ГЕНОЛОГа.

3.2.2. Просмотр фрагментов программы

Перемещение по дереву фрагментов программы

При просмотре фрагмента программы переходы из этого фрагмента в подфрагменты занумерованы числами $1, 2, \ldots$. Каждый такой переход представляет собой оператор “ветвь N” либо “иначе N”, где N — номер перехода. Один из этих переходов (“текущий”) выделен желтым цветом. Используя клавиши “курсор вправо” (к следующему переходу) и “курсор влево” (к предыдущему переходу), можно выбрать нужный текущий переход. После этого нажатие клавиши “курсор вниз” переводит в просмотр подфрагмента, к которому ведет данный переход. Последующее нажатие клавиши “курсор вверх” вызовет возвращение к исходному фрагменту.

Если клавиша “курсор вверх” нажимается в корневом фрагменте, то происходит возвращение в главное меню (именно, в режим ввода логического символа, программу которого требуется просмотреть). Нажатие клавиши “PageUp” из произвольного фрагмента программы логического символа также позволяет (причем за один шаг) попасть в указанный режим главного меню. Обычно эти возможности используются для перехода к просмотру программы другого логического символа.

Если вход в редактор программ ЛОСа имел место через указание логического символа в главном меню, а не из какого-либо оглавления или из редактора ГЕНОЛОГа, то нажатие клавиши “End” при просмотре любого фрагмента программы переводит в корневой фрагмент этой программы.

Для перемещения по дереву фрагментов программы с помощью мыши следует подвести курсор мыши к нужному оператору перехода в подфрагмент и нажать правую кнопку (нажатие левой кнопки переведет в режим просмотра подтермов). Чтобы вернуться из подфрагмента в надфрагмент, следует перевести курсор мыши в зону под текстом программы и также нажать правую кнопку.

Режим просмотра подтермов операторов фрагмента программы

Многие операторы ЛОСа представляют собой сложные многоэтажные логические конструкции, текст которых занимает иногда значительную часть экрана. Для того, чтобы облегчить концентрацию внимания на отдельном фрагменте такого оператора и, более того, сделать видимой его “архитектуру”, используется специальная многоцветная указка — режим просмотра подтермов операторов. Вход в режим просмотра подтермов обеспечивается нажатием клавиши “о” (кириллица). При этом пропадает выделение желтым цветом текущего перехода к подфрагменту, но первый оператор фрагмента программы окрашивается в голубой цвет. Используя клавиши “курсор вправо” и “курсор влево”, можно последовательно выделять все операторы текущего фрагмента.

После выбора отдельного, представляющего интерес, оператора, можно выделить новым (отличным от голубого) цветом один из его корневых операндов. Для этого достаточно нажать сначала на клавишу “курсор вниз”, и далее клавишами “курсор вправо” и “курсор влево” обеспечить выбор требуемого корневого операнда. Повторяя эту операцию для уже выделенного на некотором уровне операнда (снова нажимая клавишу “курсор вниз”, и т.д.), можно войти в просмотр подоперандов этого операнда. Чтобы избежать смешения цветов, они чередуются в соответствии с некоторым фиксированным списком цветов (по достижении последнего в списке цвета снова используется первый).

Нажатие клавиши “курсор вверх” возвращает из просмотра подоперандов к внешней операции. Если уже достигнут уровень просмотра всего оператора в целом, то нажатие данной клавиши игнорируется.

Для возвращения из режима просмотра подтермов операторов к обычному режиму просмотра фрагмента программы достаточно повторно нажать клавишу “о”.

Можно войти в режим просмотра подтермов, переведя курсор мыши к корню представляющего интерес подтерма и нажав левую кнопку мыши. Если нужно перейти к выделению другого подтерма - операция повторяется. Для выхода из режим просмотра подтермов можно перевести курсор мыши в зону под текстом программы и нажать левую кнопку.

Если при выделенном текущем подтерме, заголовком которого служит логический символ, нажать клавишу F3, то на экране появится справочная информация об этом логическом символе (интерфейс работы с ней уже был описан). Выход из просмотра этой информации — по любой клавише, кроме используемых для перелистывания и редактирования справочных текстов (например, по нажатию пробела).

Если нужно перейти к программе логического символа, выделенного при просмотре подтермов в текущем фрагменте программы, то нажимается

клавиша “c”. После этого появляется корневой фрагмент программы данного логического символа. Из этого фрагмента, далее, можно повторно перейти указанным образом к новому фрагменту, и т.д. По этой цепочке переходов можно возвращаться, нажимая клавишу “End” (при каждом возвращении устанавливается корневой фрагмент программы, а не тот, из которого имел место переход; восстановлению исходного фрагмента для первого элемента цепочки здесь иногда помогает нажатие клавиши F8 - например, если переход к нему был из оглавления программ ЛОСа либо из редактора ГЕНОЛОГа).

Справочная информация о логических символах

Чтобы при просмотре программы получить информацию о том, что делает тот или иной оператор, или что означает некоторое понятие, с которым работает прием, можно пользоваться справочной информацией о логических символах. Она размещена в 3-м информационном блоке, и фактически представляет собой записную книжку, в которой собрана вся необходимая техническая информация о логических символах — описания операторов, смысл применяемых в задачах обозначений, пояснения к использованию целей и комментариев задач, заголовком которых является данный логический символ, и т.п. По мере ввода новых символов и новых операторов, эту записную книжку следует регулярно пополнять новыми данными. Интерфейс просмотра и редактирования справочной информации о логическом символе уже описан выше в подразделе “Разное” раздела “Общие операторы интерфейса”. Перечислим возможные способы обращения к этой информации из просмотра фрагмента программы.

Если требуется получить информацию о логическом символе, являющемся заголовком данной программы, то нажимается клавиша F3. Если нужна информация о каком-либо другом логическом символе, то следует нажать клавишу F2. Далее возникает окно диалога, в котором следует набрать название нужного символа и нажать “Enter”.

Для получения справочной информации о логическом символе, встречающемся в текущем фрагменте программы, можно подвести курсор мыши к этому символу и нажать левую кнопку мыши. Тогда произойдет переход в режим просмотра подтермов операторов, в котором текущим окажется подтерм с корнем в выбранном логическом символе. Далее нажимается правая кнопка мыши, и на экране возникает справочная информация о символе. Для выхода из ее просмотра достаточно нажать любую кнопку мыши (кроме нажатия этой кнопки на пунктах меню $\rightarrow$, $\leftarrow$, которое вызовет перелистывание справочной информации). После этого можно либо убрать режим просмотра подтермов, выведя курсор мыши за рамки текста программы и нажав левую кнопку, либо перевести курсор к

другому представляющему интерес логическому символу и нажать левую кнопку, чтобы соответствующий подтерм стал текущим.

Кроме справочной информации о логических символах, при просмотре фрагмента программы можно войти в оглавление операторов ЛОСа. Для этого достаточно нажать “Ctr-л”. Напомним, что из главного меню к этому же оглавлению переход происходил по нажатию клавиши “о”.

Поиск заданного терма в ветви программы логического символа

Для того, чтобы найти все вхождения в операторы текущей ветви программы заданного терма либо логического символа, следует нажать клавишу F4 и далее набрать в окне диалога текстовым редактором требуемый терм либо логический символ. После нажатия “Enter” будет либо прорисовано многоцветной указкой первое найденное вхождение, либо (если таких вхождений в ветви вообще нет) возникнет пустой экран; по нажатию любой клавиши от него - возвращение в просмотр фрагмента. При поиске вхождений, кроме текущего фрагмента программы, просматриваются все достижимые из него фрагменты. Здесь используется принцип “сначала вглубь”, причем непосредственные подфрагменты любого фрагмента упорядочиваются “от начала к концу” (т.е. по убыванию локальных номеров).

Для перехода к следующему вхождению искомого терма (символа) повторно нажимается F4. Если найдено представляющее интерес вхождение, то можно (как обычно, нажатием клавиши “о”) выйти из режима просмотра подтермов, автоматически установленного при поиске, и выполнить необходимые операции (например, по редактированию программы). При этом установка на поиск заданного терма (символа) сохраняется, однако при повторном нажатии F4 будет выполняться поиск только в той ветви, в которой это нажатие (первое в возобновленном цикле поиска) было осуществлено. Поэтому перед возобновлением цикла поиска следует сначала выйти на корень необходимой ветви программы.

Если после очередного нажатия F4 возникает пустой экран, то поиск завершен. Только после этого установка на заданный терм (символ) сбрасывается. Далее при нажатии любой клавиши - переход в обычный режим просмотра фрагментов программы.

Последовательный просмотр программ логических символов

Можно последовательно просматривать программы всех логических символов, начиная с заданного — в соответствии с нумерацией логических символов. Для перехода к корню программы очередного логического символа (первого после текущего, для которого создана программа) нажимается клавиша “ш”. Кроме этого простейшего средства полного просмотра

программ “вручную”, имеется возможность автоматического просмотра всех программ системы, связанная с использованием специального оператора “смпрог”. При полном просмотре всех программ и входящих в них операторов, ему передается информация о каждом текущем подоператоре. В зависимости от того, как будет запрограммирован оператор “смпрог”, при этом может происходить поиск всех мест в программе, удовлетворяющих заданному условию, и изменение их по заданному принципу. Кроме того, будет выполняться фоновый (не зависящий от оператора “смпрог”) поиск типовых ошибок в программах. Наличие такого режима автоматического просмотра сделало пока излишним развитие других средств поиска и контроля в программах ЛОСа; оно позволило обращаться со всем многообразием программ системы как с единым целым и вводить в них при необходимости любые изменения (в том числе, связанные с развитием языка ЛОС). Подробнее о режиме автоматического просмотра программы будет сказано ниже, после изложения необходимой для изменения оператора “смпрог” техники редактирования программ.

3.2.3. Редактирование текущего фрагмента программы

Для начала редактирования текущего просматриваемого (в обычном режиме) фрагмента программы следует нажать клавишу “р” (кир.). При этом в правом верхнем углу экрана появляется красный прямоугольник, предупреждающий о том, что в случае нажатия клавиши “Enter” новая версия текста фрагмента будет немедленно записана в файлы блока программ, а старая пропадет (в этом случае, все же, при необходимости можно будет извлечь из резервной директории SLCOPY все файлы ранее сохраненной в ней версии решателя). Редактирование осуществляется текстовым редактором.

Для завершения редактирования нажимается клавиша “Enter”; для отмены изменений фрагмента — “Esc”. Чтобы изменения фрагмента были сохранены, в тексте не должно иметься незавершенных или неправильно набранных операторов. Попытка сохранить такой некорректный текст приводит обычно к появлению сообщения об ошибке, прорисовываемого внутри красного прямоугольника в правом верхнем углу. Курсор текстового редактора при этом перемещается либо в начало текста, либо к той точке, около которой произошла ошибка. После устранения ошибки попытка сохранения текста фрагмента может быть повторена. Предусмотрены следующие типы сообщений об ошибках:

1) “Незакрытая левая скобка” — число левых скобок в тексте больше числа правых. Курсор перемещается в начало текста.

2) “Избыточная правая скобка” — число правых скобок в некоторой точке превысило число предшествующих левых. Курсор перемещается к этой точке.

3) “Ошибка в логическом символе” — при наборе логического символа допущена ошибка либо произошла склейка идущих подряд названий символов или переменных. Курсор перемещается к началу неправильно набранного символа.

4) “Одинаковые метки” — несколько различных операторов перехода из фрагмента имеют один и тот же номер перехода. Курсор перемещается в начало текста.

5) “Недопустимая переменная” — номер переменной больше 512. Курсор перемещается в начало записи этой переменной.

Ссылки из фрагмента на подфрагменты нумеруются при редактировании произвольным образом натуральными числами. Впоследствии, при перерисовках фрагмента, они будут автоматически переобозначены на $1, 2, 3, \ldots$. Если изменяется старая версия фрагмента, то исключение оператора перехода, ссылающегося на подфрагмент, либо изменение номера его ссылки на новый, не использовавшийся ранее в фрагменте, приведут к полной потере всей ветви этого оператора.

Если в редактируемом фрагменте возникли новые номера перехода, то после нажатия клавиши “Enter” и регистрации в файлах блока программ новой версии фрагмента на экране появляется в верхнем левом углу один из новых номеров перехода (первый при движении от конца фрагмента к началу), а остальная часть экрана расчищается (остается только курсор текстового редактора). Это — приглашение к вводу подфрагмента с указанным номером. От него можно отказаться нажатием клавиши “Esc”, а можно воспользоваться, введя новый подфрагмент. Этот подфрагмент, в свою очередь, может содержать ссылки на подфрагменты, и тогда снова появится приглашение к вводу последнего из них, и т.п. В результате, если не нажимать “Esc”, будут по принципу “сначала вглубь” перечислены все ссылки на новые подфрагменты. Здесь важно не использовать дважды одного и того же номера ссылки (в процессе перечисления новых подфрагментов нумерация ссылок - глобальная, но по завершении перечисления эти номера теряются и вводится локальная нумерация).

Если при наборе подфрагментов нажималась клавиша “Esc”, то останутся недоопределенные переходы. Лучше всего доопределить их сразу, по завершении указанного выше перечисления подфрагментов. Для создания фрагмента по недоопределенному переходу, следует нажать на этом переходе клавишу “курсор вниз” (экран расчистится) и войти в его редактирование повторным нажатием “p”. Вообще, предпочтительнее

не нажимать при перечислении подфрагментов клавишу “Esc”, а вводить какой-либо простой оператор для последующего возвращения к этому подфрагменту и фактического его набора (например, оператор “продолжение”).

Исходный фрагмент программы любого оператора, справочника либо операторного выражения должен начинаться с оператора “метка(икс(N))”, где N — символьный номер первой программной переменной, не определенной при обращении и не являющейся выходной переменной. Этот оператор должен размещаться сразу после операторов “программа” либо “обращение(t)”, указывающих тип обращения к программе. В случае сканирования задачи (т.е. после оператора “решить”) оператор “метка(икс(...))” не используется.

Чтобы получить справочную информацию о логическом символе непосредственно в процессе редактирования, нажимается клавиша F4 и в диалоговом окне вводится название этого символа. После просмотра информации восстанавливается изображение на момент прерывания редактирования. Если при наборе текста нужно ввести некоторый символьный номер N, больший двадцати (номера от 0 до 20 обозначены логическими символами “0”, ..., “9”, “десять”, ..., “двадцать”), то вводится запись “логсимвол(N)”. По окончании набора текста, после нажатия “Enter”, эта запись будет автоматически переведена в требуемый символьный номер, и впоследствии в данном месте программы он будет прорисовываться в явном виде. Если нужно вставить в программу логический символ — код клавиатуры, то нажимается клавиша F8, после чего нажимается та клавиша, для которой нужен код. Этот код будет прорисован автоматически, начиная с текущей позиции курсора. Если нужен код для режима латинских букв, то предварительно нажимается F12; для возвращения к режиму кириллицы нажимается F11.

Для копирования части фрагмента программы в другие фрагменты можно использовать буфер текстового редактора: войти в режим редактирования фрагмента, содержащего нужный текст (клавиша “p”); занести этот текст в буфер; выйти из режима редактирования, не изменяя программы (клавиша “Esc”); войти в редактирование того фрагмента, в котором нужно вставить копию, и извлечь ее из буфера текстового редактора. При этом, однако, не будут скопированы те фрагменты, к которым из выделенного текста имелись переходы — их придется создавать повторно. Чтобы скопировать всю ветвь программы, предусмотрена специальная операция (см. ниже).

Если в процессе набора программы возникла необходимость получить информацию, недоступную из текстового редактора, то следует донабрать

(произвольным образом, но со строгим соблюдением синтаксических правил для операторов со связанными переменными) текущий оператор, сохранить набранную часть фрагмента нажатием “Enter”, выйти в просмотр необходимых данных, а затем вернуться в прерванное редактирование.

При редактировании программы в блоке программ обычно возникают “пустоты” — неиспользуемые остатки предыдущих версий фрагментов программ. Эти пустоты рекомендуется периодически устранять, выбирая в главном меню пункт “Уплотнение измененных файлов”. Кроме исключения неиспользуемых фрагментов, операция уплотнения осуществляет контроль за корректностью общей структуры программ. Если по причине ошибки или машинного сбоя в блоке программ обнаруживается дефект, то при выборе указанного пункта либо пункта “Сохранение копий файлов” (тоже осуществляющего данную проверку) произойдет выход из программы в операционную систему. В этом случае следует воспользоваться сохраненной в резервной директории SLCOPY копией программы. Такую копию следует постоянно обновлять выбором в главном меню пункта “Сохранение копий файлов”. Рекомендуемый режим работы — регулярный выбор пунктов “Уплотнение...”, “Сохранение...” после любых изменений в программе и информационных блоках.

3.2.4. Сдвиг номеров программных переменных

При программировании на ЛОСе следует придерживаться принципа последовательной нумерации новых переменных — это связано в первую очередь с логикой откатов, при которых сбрасываются значения всех переменных, номера которых больше некоторого, а также с экономией объема глобального стэка интерпретатора ЛОСа. Однако, иногда возникает необходимость при доработке программы вставлять внутри нее операторы, вычисляющие те или иные дополнительные значения. Чтобы ввести для этих значений программные переменные, не нарушая принципа последовательной нумерации, приходится выполнять сдвиг (в данной ситуации - увеличение) номеров всех программных переменных, определяемых после точки вставки оператора, на заданную величину. Это осуществляется следующей последовательностью действий:

а) Войти в режим просмотра подтермов операторов и выделить оператор, после которого (включая его самого) требуется выполнить сдвиг нумерации переменных.

б) Нажать клавишу “п”. Тогда в левом верхнем углу экрана возникает курсор текстового редактора. Если нужно увеличить номера переменных, большие или равные n, на величину m, то набирается терм “плюс(хn m)” (буква “х” — кир.). Если нужно уменьшить номера переменных,

большие или равные n, на величину m, то набирается терм “минус(xn m)”. В обоих случаях после нажатия “Enter” выполняется указанный сдвиг (он затрагивает не только данный фрагмент, но и все подфрагменты, достижимые из выбранного оператора).

3.2.5. Операции с ветвями программы

Чтобы скопировать всю ветвь некоторого фрагмента программы, из обычного просмотра этого фрагмента нажимается клавиша “Insert” — это приводит к регистрации ссылки на фрагмент в специальном буфере редактора программ. Затем предпринимается переход к тому фрагменту программы, всю ветвь которого следует заменить на всю ветвь исходного (учтенного в буфере) фрагмента. Эта ветвь НЕ ДОЛЖНА находиться внутри заменяющей ветви либо содержать ее. Как правило, такую ветвь сначала приходится специально создавать — состоящей из единственного фрагмента с единственным оператором “продолжение” — ответвляя в нужном месте переход к ней по “ветвь” либо “иначе”. Если заменяемая ветвь относится к программе другого логического символа, то выход в главное меню для перехода к программе этого символа должен происходить только по нажатии клавиши “PageUp” — иначе ссылка в буфере на заменяющую ветвь будет утеряна. После того, как на экране возник корневой фрагмент заменяемой ветви, последовательно нажимаются клавиши “o” (кир.) — для перехода в режим просмотра подтермов — и “к” (кир.) — собственно для копирования ветви.

Для удаления ветви программы, переход к которой из текущего фрагмента программы определяется выделенным желтым цветом оператором перехода, достаточно нажать клавишу “Ctr-Del”. Эту операцию можно применять лишь однократно; для повторного удаления с ее помощью следует сначала выйти из просмотра программы текущего логического символа.

Для удаления всей программы логического символа следует перейти в корневой фрагмент этой программы и нажать “Ctr-F7”.

Для удаления сразу нескольких ветвей программы, к которым из текущего фрагмента имеются переходы, можно войти в режим редактирования этого фрагмента и исключить операторы перехода, ссылающиеся на указанные ветви. По завершении редактирования (клавиша “Enter”) эти ветви будут полностью удалены.

Если ветвь программы была удалена с помощью “Ctr-Del” либо “Ctr-F7”, то ее можно восстановить в качестве корневой ветви всей программы данного логического символа — достаточно в корневом фрагменте программы

нажать “Ctr-F6”. Такое нажатие должно быть выполнено до выхода из программы текущего логического символа. Впрочем, для удаления части программы, находящейся выше некоторого фрагмента, более удобным оказалось использовать операцию слияния двух последовательных фрагментов (см. ниже).

Если фрагмент программы оказался чрезмерно большим (например, после вставки в него дополнительных операторов), то его можно разрезать на две части. Для этого следует войти в режим просмотра подтермов, выделить тот оператор, который должен стать первым оператором второй части, и нажать клавишу “р” (кир.). К концу первой части фрагмента будут добавлены оператор “ветвь N”, ссылающийся на вторую часть, а также оператор “продолжение” (переход ко второй части будет выполнен после “отражения” от этого завершающего оператора).

Если два фрагмента программы выполняются последовательно — в том смысле, что первый из них завершается оператором “ветвь N”, ссылающимся на второй, и оператором “продолжение”, — то можно выполнить слияние их в один общий фрагмент. Эта операция в точности обратна описанной выше операции разрезания фрагмента. Разумеется, применять ее следует так, чтобы результат слияния умещался целиком на экране. В принципе, появление фрагментов, не помещающихся на экране целиком, допустимо (они иногда возникают при работе компилятора ГЕНОЛОГа). Однако, для удобства работы их лучше разрезать на части. Во всяком случае, это необходимо делать перед редактированием такого фрагмента.

3.2.6. Обнаружение ошибок в программе

Предусмотрена возможность автоматического контроля правильности программы логического символа, включающего обнаружение простейших ошибок — несоответствия числа операндов операторов и операторных выражений их арности, использование значения переменной, которое еще не было определено, попытка повторно определить уже определенное значение, и т.п. Для выполнения такого контроля следует войти в просмотр корневого фрагмента программы и нажать клавишу “п”. При наличии ошибок будут выданы соответствующие сообщения (при этом произойдет переход в режим просмотра подтермов и будет выделена точка ошибки — для последующего ее исправления нужно будет сначала нажатием клавиши “о” (кир.) перейти в обычный просмотр фрагмента). При отсутствии ошибок, по окончании анализа программы, произойдет перерисовка корневого фрагмента.

Если контролируется программа нового оператора либо операторного выражения, причем после первого оператора ее корневого фрагмента,

уточняющего тип обращения, не идет “иначе”, то процедура контроля автоматически вводит программу справочника “арность”, указывающую на число операндов этого оператора либо операторного выражения. Для перечисляющих операторов вводится также программа справочника “комментпосылок”. При определении арности здесь используется оператор “метка(икс(N))”, размещаемый в начале контролируемой программы.

Возможна серийная проверка программ логических символов, начиная с текущего логического символа. Для запуска этой проверки нажимается клавиша “П”. Остановка серийной проверки осуществляется нажатием любой клавиши. Для просмотра всех программ системы следует начинать с логического символа “метка”, имеющего номер 1. При просмотре перерисовки фрагмента программы на экране не происходит; меняются только заголовки просматриваемых программ в верхней части экрана.

При контроле программы одного логического символа (клавиша “п”) либо при серийном контроле (клавиша “П”) возможно выполнение дополнительных действий, определяемых программой оператора “смпрог(a b c d e)”. К этому оператору происходят обращения для каждого текущего просматриваемого вхождения в операторы фрагментов программ. При этом значением переменной a становится логический символ, к которому относится фрагмент программы; значением переменной b — набор $(F_1, \ldots, F_n)$, определяющий путь от корня программы символа a к текущему просматриваемому фрагменту программы. Каждое F_i есть тройка (F_{i1}, F_{i2}, F_{i3}), у которой (F_{i1}, F_{i2}) - ссылка на фрагмент программы;F_{i3} — символьный номер перехода от этого фрагмента к подфрагменту, соответствующему тройке F_{i-1}. Последний элемент набора b определяет переход от корня программы символа a; первый элемент - переход к текущему фрагменту. Значением переменной c служит набор троек $(G_1, \ldots, G_{n+1})$; $G_i = (G_{i1}, G_{i2}, G_{i3})$. Эти тройки описывают фрагменты программы, расположенные на пути, определяемом набором b; G_{n+1} соответствует корню программы символа a, а G_1 — текущему фрагменту. G_{i1} есть сам фрагмент программы, представленный набором термов в зоне задач; G_{i2} — набор дополнительных логических условий на программные переменные, встречающиеся в операторах фрагмента G_{i1}. G_{i3} — набор информационных элементов, характеризующих фрагмент G_{i1}. Для указания корневого фрагмента здесь используется элемент “корень”; для указания вхождения v в фрагмент оператора перехода, ссылающегося на подфрагмент текущего пути — элемент (позиция v). Входной переменной d оператора “смпрог(...)” присваивается вхождение текущего оператора в список операторов текущего фрагмента; переменной e — вхождение текущего логического символа в текущем операторе.

Если оператор “смпрог” оказывается истинным в некоторой точке просмотра, то просмотр обрывается, и многоцветной указкой выделяется текущее вхождение в текущий оператор текущего фрагмента. Для продолжения просмотра после этого (кроме случаев сообщения о найденной ошибке) следует нажать клавишу “ш”. Следует учитывать, что при выходе из режима многоцветной указки цикл просмотра сбрасывается, и тогда нужно запускать его с текущего логического символа заново.

Оператор “смпрог” можно использовать для поиска точек в программе, удовлетворяющих заданным условиям. Эти условия, с помощью редактора программ, записываются в начале программы данного оператора (предварительно следует удалить тождественно ложный оператор, обычно вставляемый в начало программы “смпрог” для предотвращения ее срабатываний в режиме общей проверки; обычно таким оператором является “равно(0 1)”). В найденной точке можно изменить программу вручную и возобновить поиск, начиная с текущего логического символа. Возможна реализация цикла автоматического изменения всей программы системы — для этого в операторе “смпрог” нужно реализовать процедуру модификации фрагмента или целой ветви программы, и заменить старый фрагмент (ветвь) на новый, используя специальные операторы для работы с фрагментами программ, описанные в разделе, посвященном программной реализации редактора программ.

3.2.7. Сопровождение справочной информацией избранных точек в программе

Чтобы сопровождать избранные точки программ ЛОСа необходимыми пояснениями, а также чтобы быстро можно было находить нужную процедуру и находить в этой процедуре конкретную точку, создано специальное оглавление, в котором перечислены основные процедуры системы, реализованные на ЛОСе. Об этом оглавлении уже говорилось вкратце выше (при описании способов входа в просмотр программ); вход в него — через пункт “Оглавление программ” главного меню. Концевые пункты данного оглавления суть комментарии к избранным точкам программы. Для входа в просмотр такой точки достаточно нажать в концевом пункте клавишу “курсор вправо”. После этого возникает текст фрагмента, в котором синим цветом выделен оператор “прием(i)” — он отмечает нужно место в программе. Здесь имеет место режим просмотра подтермов операторов. Далее можно произвольным образом перемещаться по фрагментам программы (не выходя в главное меню) и менять режим просмотра; если в произвольной точке восстановить режим просмотра подтермов и нажать “End”, то будет восстановлен исходный концевой пункт оглавления программ.

Если при просмотре фрагментов программы после перехода к ним из оглавления программ (в обычном режиме просмотра) нужно вернуться к тому фрагменту, с которого был начат этот просмотр, то нажимается F8. Произойдет возвращение к просмотру исходного оператора "прием(i)", причем восстановится режим просмотра подтермов.

Если нужно связать комментарий с какой-либо точкой программы, то следует войти в режим просмотра подтермов, выделить тот оператор A, к которому относится этот комментарий, и нажать клавишу "PageDown". После этого на экране возникнет некоторая страница оглавления программ ЛОСа. Перемещаясь по этому оглавлению, а при необходимости — создавая новые его подразделы, следует перейти в то меню, которое связано с описанием рассматриваемой части программы. Далее нужно ввести новый концевой пункт этого меню (клавиша "к" либо "К"), и текстом этого пункта сделать требуемый комментарий. По окончании ввода текста (нажатие клавиши "Enter") перед оператором A будет вставлен фиктивный оператор "прием(N)", ссылающийся на введенный концевой пункт оглавления программ.

Если вход в программу происходил не из оглавления программ, то для получения информации оглавления программ, относящейся к выбранной ее точке, следует выделить в этой точке многоцветной указкой какой-либо оператор и нажать "End". Тогда будет найден первый предшествующий выделенной позиции оператор "прием(N)", и осуществится переход к его концевому пункту оглавления программ.

Чтобы удалить концевой пункт оглавления программ и одновременно удалить ссылку "прием(i)" в программе, соответствующую этому пункту, достаточно нажать "Ctr-Del" при выделенном концевом пункте оглавления программ.

3.2.8. Дополнительные возможности интерфейса редактора программ

Находясь в режиме просмотра фрагмента программы, можно выполнить ряд вспомогательных операций. Прежде всего, именно из этого режима осуществляется вход в создание и редактирование шаблонов диалоговых блоков. При нажатии клавиши "д" возникает оглавление диалоговых блоков, в котором можно либо выбрать для просмотра и редактирования ранее созданный шаблон блока диалога ("курсор вправо" на выбранном пункте; для возвращения в оглавление из редактирования шаблона — "Esc"), либо создать новый шаблон. Последнее осуществляется созданием нового концевого пункта и входом в него — так же, как и для ранее созданного пункта. Интерфейс редактирования шаблона блока диалога

был описан выше. Для удаления шаблона достаточно удалить (Ctr-Del) соответствующий концевой пункт оглавления.

Кроме оглавления шаблонов блоков диалога, из режима просмотра фрагмента программы можно перейти в ряд других справочных оглавлений, которые бывает нужно использовать при редактировании программ. Подробнее об этих оглавлениях будет сказано в последующих разделах книги.

4. Отладчик ЛОСа

4.1. Семантическая трассировка решения задачи

Семантическая трассировка представляет собой пошаговый просмотр процесса решения задачи с отображением на экране сообщений о применении приемов, сопровождаемых пояснениями. Такая трассировка позволяет понять способ решения задачи “в целом” и локализовать моменты, требующие специальной доработки.

Способы запуска решения задачи уже были описаны выше в разделе “Общая схема функционирования решателя”. Для запуска решения без трассировки нажимается клавиша “о”; для запуска решения с семантической трассировкой нажимается “р”, причем предварительно уточняется режим трассировки. Это делается с помощью диалогового блока, активизируемого нажатием клавиши “т”. Выбор левой кнопкой мыши пунктов этого блока включает либо выключает соответствующие установки на трассировку. Для обычного просмотра рекомендуется режим с выключенным пунктом “ручной выбор входа в подпроцесс”; включение этого пункта происходит в тех случаях, когда при решении задачи встречается обращение к чрезвычайно трудоемкой вспомогательной задаче, не выводимое на экран. Такое включение не обязательно выполнять с самого начала трассировки: обращение к диалоговому блоку выбора режима трассировки возможно на каждом шаге. Другие пункты этого диалогового блока таковы:

1) “Пропуск обращений к проверочным операторам” - обычно выключен. Его включение приведет к тому, что перестанут появляться сообщения об обращениях к процедурам проверки различных вспомогательных утверждений.

2) “Пропуск шагов удаления посылок и условий” - обычно включен. Его выключение приведет к тому, что будут явно указываться шаги удаления ненужных условий и посылок задачи.

3) “Пропуск изменений сопровождающих условий” - обычно включен. Его выключение приведет к тому, что будут отображаться изменения условий на область допустимых значений, сопровождающие основные шаги решения.

4) “Пропуск входа в любые подпроцессы” — обычно выключен. Сообщения об обращениях к наиболее существенным для решения задачи вспомогательным задачам и процедурам при его выключении автоматически выводятся на экран (хотя такие обращения означают не срабатывание приема, а только начало попытки его применения). Если этот пункт включен, то данные сообщения пропадут, а останутся лишь сообщения о фактически сработавших приемах.

5) “Просмотр шагов изменения комментариев” — обычно выключен. Изменения технических пометок — так называемых комментариев образуют достаточно интенсивный поток, вывод которого на экран существенно замедлит трассировку.

6) “Пропуск регистрации условий на о.д.з.” — обычно включен. При его выключении более подробно будет отображен процесс первоначального ввода условий на область допустимых значений, сопровождающих условия и посылки задачи.

Чтобы перейти из режима семантической трассировки в просмотр текущей ситуации с помощью отладчика ЛОСа, достаточно нажать клавишу “ф”. Тогда на экране появится текст текущего исполняемого фрагмента программы, вплоть до текущего оператора (пока не реализованного), который будет выделен малиновым цветом. Дальнейшие действия — согласно инструкции по отладчику ЛОСа (см. ниже). Для возвращения в просмотр текущего кадра семантической трассировки и продолжения ее достаточно нажать клавишу “р” либо клавишу “з”.

4.2. Установка режимов технической трассировки перед запуском решения

Если нужно отладить прием, реализованный на ГЕНОЛÓГе, то сначала следует создать либо найти в задачнике такую задачу, где он должен сработать. После этого можно создать установку на прерывание при попытке применения данного приема и запустить решение задачи до обнаружения такой попытки. Для этого (из просмотра задачи в задачнике) нажимается клавиша “г” или выбирается пункт “Генолог” меню выбора режима трассировки, расположенного в верхней части экрана. Тогда возникает некоторый пункт оглавления базы приемов ГЕНОЛОГа. В этом оглавлении находится нужный пункт, и внутри группы приемов этого

пункта находится нужный прием (для этого служат клавиши “курсор вверх” — “курсор вниз”). Далее нажимается клавиша “Ctr-Enter”, которая одновременно создает установку на прерывание и запускает решение задачи.

Если в процессе решения произойдет выход на начальный отрезок программы выбранного приема (именно, на используемый для указания этого начала фиктивный оператор “контрольприема(A_1 A_2 A_3)”; здесь A_1, A_2, A_3 — логические символы и термы, образующие стандартную техническую ссылку на прием), то включится отладчик ЛОСа, и на экране будет отображен фрагмент программы с выделенным малиновым цветом оператором, который должен будет выполняться непосредственно на следующем шаге (этот оператор следует за оператором “контрольприема($A_1A_2A_3$)”). При этом устанавливается пошаговый режим трассировки (см. ниже).

Если выбранный прием применяется при сканировании задачи либо при работе пакетного нормализатора (см. описание ГЕНОЛОГа), то можно установить прерывание при фактическом применении этого приема. Для этого вместо клавиши “Ctr-Enter” нажимается клавиша “Enter”.

Если нужно проверить, применяется ли при решении задачи некоторый оператор либо операторное выражение, реализованные с помощью программы на ЛОСе, и проследить по шагам ход выполнения этой программы, то перед запуском решения задачи нажимается клавиша “Л”. Тогда в левой нижней части экрана возникает надпись “Логический символ:” и появляется курсор текстового редактора. Выполняется набор необходимого логического символа и нажимается “Enter”. Если было набрано слово, не являющееся названием логического символа, то оно пропадает с экрана и курсор возвращается в исходное положение — для повторного набора. Иначе — одновременно создается установка на прерывание при обращении к программе указанного логического символа и запускается процесс решения задачи. По достижении первого оператора корневого фрагмента программы выбранного символа включается отладчик ЛОСа; сам этот оператор прорисован малиновым цветом и еще не выполнен. Для дальнейших действий автоматически устанавливается пошаговый режим трассировки.

Если нужно проследить моменты выхода к точкам программы, для которых созданы соответствующие концевые пункты оглавления программ, то перед запуском решения задачи нажимается клавиша “л”. Тогда возникает некоторый пункт оглавления программ. Если выбрать нужный концевой пункт этого оглавления и нажать на нем клавишу “курсор вправо”, то одновременно инициируется прерывание при выходе на соответствующую контрольную точку программы (напомним, что эта точка представляет

собой фиктивный оператор “прием(N)”; N — номер контрольной точки внутри программы данного логического символа). Как и в предыдущих случаях, по достижении контрольной точки включается отладчик ЛОСа и устанавливается пошаговый режим трассировки.

При отладке бывает полезно применение счетчика шагов работы интерпретатора ЛОСа. Оно дает возможность при повторных запусках решения задачи идентифицировать (например, методом деления отрезка пополам) момент ошибочного действия. Это средство применяется, впрочем, достаточно редко, так как подавляющее большинство ошибок обнаруживаются по недопустимым типам входных данных операторов ЛОСа и немедленно выводятся на экран. Однако, в некоторых случаях появление заведомо ошибочных элементов структуры данных не выявляется средствами общего контроля, и анализ текущей информации не позволяет объяснить их происхождение. В этих случаях и используется счетчик шагов. При трассировке на уровне отладчика ЛОСа номер текущего шага работы интерпретатора выводится на экран в правом верхнем углу. Этот номер позволяет примерно определить диапазон, в котором следует искать момент появления ошибки. Если при повторном запуске решения задачи нажать клавишу “Ш” и ввести текстовым редактором в появившемся окне диалога (после надписи “Число операторов:”) номер шага, соответствующего началу диапазона поиска ошибки, то по нажатии “Enter” будет одновременно установлено прерывание по достижении заданного шага и начато решение задачи. По достижении нужного шага включается отладчик ЛОСа и устанавливается пошаговый режим.

Кроме установок на автоматическое прерывание процесса решения, можно в любой момент прервать этот процесс вручную. Для этого достаточно нажать клавишу “Break”. В результате появится текст текущего исполняемого фрагмента программы, в котором текущий (еще не выполненный) оператор выделен малиновым цветом. Для работы в отладчике ЛОСа устанавливается пошаговый режим. Заметим, что нажатие клавиши “Break” не позволяет входить в просмотр функционирования программ отладчика.

Наконец, самый простой (но сравнительно трудоемкий) способ вызвать прерывание работы программы и обращение к отладчику ЛОСа в нужной точке программы — вставить в этой точке оператор “трассировка(стоп 0)”. Этот оператор вызывает немедленный вход в отладчик ЛОСа с пошаговым режимом, не выполняя никаких других действий (он всегда истинен). По завершении анализа текущего шага отладчиком и продолжении трассировки будет выполняться следующий за ним оператор. Единственное необходимое условие его корректной работы — наличие после него в программе хоть какого-либо другого оператора. Заметим, что помещение данного оператора внутри процедур интерфейса и даже процедур самого

отладчика приводит к тому же эффекту, так что он делает возможной отладку с помощью отладчика ЛОСа тех участков программы отладчика, в которых этот оператор не будет немедленно повторно выполняться при попытке обращения к отладчику.

4.3. Просмотр информации о моменте прерывания

Собственно отладчик ЛОСа представляет собой программу логического символа “прерывание”, которая активизируется на тех шагах работы системы, для которых либо оказывается истинным заранее установленное условие прерывания, либо нажимается клавиша “Break”, либо обнаруживается попытка реализации оператора ЛОСа для недопустимых входных данных. Эта программа позволяет просмотреть информацию о текущем состоянии системы и изменить режим трассировки, в том числе ввести новые установки на прерывание. При выходе из нее продолжается прерванное решение задачи. На момент входа в программу “прерывание” счетчик шагов интерпретатора ЛОСа отключается, так что работа с отладчиком никак не сказывается на соответствии номеров шагов действиям решателя. В процессе работы с отладчиком ЛОСа можно восстановить исходное изображение, возникшее на момент обращения к нему при прерывании, нажав клавишу “р” (кир.).

4.3.1. Просмотр программы

При входе в отладчик ЛОСа на экране прорисовывается некоторый фрагмент программы, в котором малиновым цветом выделен текущий (пока не выполненный) оператор. Этот фрагмент, вообще говоря, не является корневым; можно просмотреть все его надфрагменты в дереве программы текущего логического символа, используя клавиши “Home” (шаг по направлению к корню программы) и “End” (шаг от корня программы к фрагменту с текущим оператором). С указанными фрагментами связан общий набор значений программных переменных текущей ситуации, возникшей при выполнении текущей программы; эти значения сохраняются в некотором кадре глобального стэка интерпретатора ЛОСа. Заметим, что из режима трассировки возможен просмотр только линейной цепочки фрагментов, от корня программы до фрагмента с текущим оператором; выход на боковые ветви программы не предусмотрен.

При просмотре фрагментов программы выдается только начальный отрезок фрагмента до оператора, выделенного малиновым цветом (это либо текущий оператор, либо оператор перехода к подфрагменту). Чтобы просмотреть все операторы фрагмента, нажимается клавиша “ф”. Чтобы убрать с экрана лишние (идущие после выделенного малиновым цветом)

операторы, нажимается клавиша “o”. В левой верхней части экрана размещен текст “Программа A”, где A — заголовок программы (выделяется синим цветом). В правой верхней части экрана размещен текст “шаг № N”, где N — номер текущего шага работы интерпретатора ЛОСа. Этот шаг не является абсолютным; он отсчитывается только с момента запуска решения извлеченной из задачника задачи, что делает его неизменным при повторных запусках и позволяет использовать для локализации событий, происходящих при решении. Под текстом фрагмента программы проведена горизонтальная черта, отделяющая область экрана, в которой можно получать дополнительную информацию о текущем шаге.

Кадр глобального стэка интерпретатора ЛОСа сохраняет информацию о текущих значениях программных переменных, возникающих при реализации программы некоторого оператора либо операторного выражения, либо при сканировании некоторой задачи. При обращении от программы к подпрограмме создается новый стэковый кадр, для сохранения значений программных переменных этой подпрограммы. Таким образом, возникает цепочка стэковых кадров, в которых сохраняются значения программных переменных как для текущей программы, так и для всех ее надпрограмм. Самая “верхняя” программа в этой цепочке - сканирование исходной фиктивной задачи на исследование; от нее имело место обращение к сканированию задачи, извлеченной из задачника; далее — к некоторой вспомогательной задаче либо оператору, и т.д. Возможен просмотр всей цепочки “надпрограмм” текущей реализуемой программы с помощью отладчика ЛОСа. Для этого служат клавиши “PageUp” и “PageDown”. Первая из них переводит от просмотра программы к ее надпрограмме; вторая - от программы к подпрограмме (все это — вдоль линейной цепочки обращений, сложившейся на текущий момент). При переходах прорисовываются отрезки соответствующих фрагментов программы вплоть до того оператора, от которого имело место обращение к подпрограмме. На каждом уровне цепочки обращений возможен просмотр значений программных переменных данного уровня. Так как значения этих переменных при выполнении программы на ЛОСе часто сохраняются неизменными (значение однажды определенной переменной обычно меняется только при откатах), то становится возможен анализ значительной части “предыстории” текущей ситуации.

Использование пар клавиш “Page Up — Page Down” и “Home - End” делает просмотр текущей ситуации двумерным: можно варьировать фрагмент программы по цепочке переходов внутри программы одного логического символа, а также варьировать саму рассматриваемую программу по цепочке обращений от одной программы к другой.

При просмотре программы отладчиком, как и при просмотре ее редактором программ, возможен вход в режим просмотра подтермов операторов (многоцветная указка). Этот режим упрощает чтение и понимание сложных логических операторов, а также используется при установке прерываний (см. ниже). Для входа в режим просмотра подтермов операторов следует нажать клавишу “курсор вниз” — первый из операторов текущего фрагмента выделяется синим цветом. Далее клавиши “курсор вправо” — “курсор влево” позволяют выбрать нужный оператор фрагмента; клавиша “курсор вниз” — войти в просмотр операндов выделенной операции; клавиши “курсор вправо” — “курсор влево” — выбрать нужный операнд. Для возвращения от операнда к внешней операции и выхода из режима просмотра подтермов нажимается клавиша “курсор вверх”.

Для получения информации о логическом символе, выделенном при просмотре подтермов оператора, нажимается клавиша F3.

В режиме просмотра фрагмента программы (без выделения подтермов операторов) возможен обычный доступ к справочной информации о логических символах: нажатие клавиши F3 приводит к выдаче информации о логическом символе — заголовке текущей программы; нажатие клавиши F2 — к появлению курсора текстового редактора, с помощью которого набирается название логического символа, о котором требуется получить информацию.

4.3.2. Просмотр цепи задач

Для входа в просмотр цепи задач (текущей задачи и всех ее надзадач) нажимается клавиша “з”. При просмотре применяются клавиши “курсор вверх” — “курсор вниз”; “PageUp” — “PageDown”; “Ctr-PageUp” (начало цепи задач); “Ctr - PageDown” (конец цепи задач, т.е. текущая задача); “Ctr - курсор вверх” (к началу предыдущей задачи); “Ctr — курсор вниз” (к началу следующей задачи). Заметим, что хотя внешне просмотр цепи задач из отладчика ЛОСа ничем не отличается от просмотра ее при трассировке по шагам решения задачи, нажатие клавиши “Enter” в первом случае не приводит к продолжению процесса решения задачи.

Для возвращения в основной интерфейс отладчика ЛОСа из интерфейса просмотра цепи задач нажимается клавиша “ф”.

4.3.3. Просмотр значений программных переменных

Для просмотра различной информации о текущей ситуации используется область экрана, расположенная под горизонтальной чертой, отделяющей текст фрагмента программы. Если эта область недостаточна, то можно

вообще убрать с экрана текст фрагмента программы, нажав клавишу “Delete” (этот текст восстанавливается нажатием клавиш “о” — до текущего оператора, либо “ф” — полный текст фрагмента). Если нужно убрать текст программы, сохранив информацию, размещенную в нижней части экрана, то вместо “Delete” нажимается F5. В нижней части экрана данные выдаются в режиме автоматической прокрутки вверх (обратная прокрутка невозможна, и для повторного просмотра удаленных с экрана объектов их следует востребовать заново). Возможна ручная прокрутка вверх содержимого нижней части экрана (при сохраняющемся в верхней части тексте фрагмента программы) с помощью клавиши “курсор вниз”.

Для просмотра значения программной переменной “хi” следует нажать клавишу “х” (кириллица) — появятся буква “х” и курсор текстового редактора, и далее набрать номер i в обычной десятичной записи.

Если значением программной переменной “хi” является терм, то он будет выдан в стандартной либо в скобочной записи в зависимости от ранее установленного режима просмотра термов. Клавиша “с” переводит в скобочный режим, клавиша “м” (кир.) — в режим стандартной математической записи.

Если значением переменной “хi” служит набор, не являющийся термом, то будут последовательно выданы все его разряды. Здесь логические символы и символы переменных указываются явно (последние — в виде, согласованном либо со скобочной записью, т.е как буква “х” с номером, либо в виде, согласованном со стандартной математической записью — как латинская буква, быть может, с индексом). Элемент набора, представляющий собой терм, обозначается посредством записи “тj”, где j — номер, закрепляемый за данным термом на период просмотра элементов текущей структуры данных (эти номера сбрасываются при смене текущего просматриваемого фрагмента клавишами PageUp,PageDown, Home,End). Элемент набора, представляющий собой набор, отличный от терма, обозначается посредством записи “нj”, где j - номер, закрепляемый за указанным набором на период просмотра структуры данных. Элемент набора, представляющий собой вхождение в некоторый набор либо в терм, обозначается посредством “вj”, где j — номер, закрепляемый за данным вхождением на период просмотра. Нумерация объектов всех указанных типов — общая.

Если значением переменной “хi” служит вхождение в терм либо в набор, то будет прорисован, соответственно, этот терм либо набор, в котором подтерм либо разряд, находящийся по рассматриваемому вхождению, выделен синим цветом.

Объекты “нj”, “тj”, “вj”, введенные при просмотре набора, сами могут быть отображены на экране в одном из указанных выше видов. Для этого следует лишь нажать, соответственно, клавишу “н”, “т” либо “в” и после этого набрать номер j объекта (набор номера завершается нажатием клавиши “Enter”).

Для большей наглядности можно изображать набор в виде последовательности строк, каждая из которых в указанном выше формате представляет один разряд набора. При этом предусмотрена выдача лишь тех элементов набора, которые не являются логическими символами либо переменными (символьные элементы доступны при обычном просмотре набора, и здесь они пропускаются). Если рассматриваемый набор является значением переменной “хi”, то нажимается клавиша “Х” (кир.) и вводится номер i; если же набор обозначен как “нi”, то нажимается клавиша “Н” (кир.) и вводится номер i. Возможно отображение в указанном виде лишь тех элементов набора, которые начинаются с заданного логического символа S. Для этого после набора номера i в окне текстового редактора помещается название символа S (отделенное от номера пробелом).

Если требуется одновременно выдать на экран в скобочной записи все элементы набора термов “нi”, то нажимается клавиша “а” (кир.), после чего вводится текстовым редактором номер i.

По-видимому, для просмотра сложных структур данных наиболее удобным является использование “сквозного режима”. В этом режиме используется все пространство экрана (текст фрагмента программы временно удаляется); каждый разряд набора прорисовывается, после указания его номера, на отдельной строке либо группе строк, причем возможна обычная прокрутка (клавиши “курсор вверх-вниз” и “PageUp — PageDown”). Синим цветом выделяется номер текущего разряда набора (смена этого номера — клавишами “курсор вверх - вниз”). Если текущий разряд сам является набором, то для его просмотра нажимается клавиша “курсор вправо”, после чего он прорисовывается на экране в указанном поразрядном виде. Таким образом можно дойти до атомарных объектов, не являющихся наборами. Возвращение от элемента к внешнему набору, а также выход из указанного режима обеспечиваются нажатиями клавиши “курсор влево”. Номера атомарных (не являющихся наборами) разрядов завершаются круглой скобкой; номера разрядов, являющихся наборами — точкой. При отображении разряда, являющегося набором, используются обычные сокращения: “т” означает терм, “н” — набор, “в” — вхождение. Эти сокращения, однако, не сопровождаются номерами, излишними при возможности выбора объекта для просмотра с помощью курсора. Чтобы просмотреть в сквозном режиме набор, являющийся значением

переменной “хi”, нажимается клавиша “К” (кир.) и вводится номер i. Если просматривается набор “нi”, то нажимается “к” (кир.) и вводится i.

Если задача сопровождается чертежом, то возможна его прорисовка в отладчике ЛОСа. Нажатие клавиши “ч” приводит к его прорисовке в верхней части экрана; клавиши “Ч” — к прорисовке чертежа под ранее выведенной в нижней части экрана информацией. После вывода чертежа возможен обычный просмотр элементов текущей структуры данных, размещаемых под чертежом.

Если используются как стандартная математическая, так и скобочная записи термов, то для установления связи между обозачениями переменных в этих способах записи используется переходная таблица (она перечисляет пары “обозначение переменной задачи в стандартной записи — обозначение этой переменной в скобочной записи”). Для вызова на экран переходной таблицы нажимается клавиша “п”.

4.3.4. Сообщение об ошибке в программе

Если интерпретатор ЛОСа обнаруживает попытку реализации некоторого оператора с недопустимыми для него типами входных данных, то он инициирует вход в отладчик ЛОСа и выдачу сообщения об ошибке: в верхней левой части пустого экрана возникает сообщение “Некорректное обращение к оператору”. При входе в просмотр текущего фрагмента программы (нажатие клавиши “ф”) будет прорисована часть этого фрагмента, предшествующая текущему оператору (выделенному малиновым цветом). Анализ окрестности этого оператора обычно позволяет выявить описку в программе. Возможны также обращения к отладчику ЛОСа при обнаружении интерпретатором переполнения различных ресурсов. В этих случаях выдаются следующие сообщения об ошибке:

1) “Переполнение глобального стэка” — означает, что имеется чрезмерно длинная цепочка обращений от программы к подпрограмме, так что суммарный объем кадров глобального стэка, описывающих значения программных переменных всех этих программ, превысил предельно допустимую величину. Часто это происходит при “зацикливании” в рекурсивных обращениях.

2) “Переполнение буфера текстов” — означает, что предпринята попытка создать настолько большой терм или набор, что он превысил предельно допустимую величину.

3) “Переполнение зоны задач” — означает, что вся зона задач заполнена объектами, введенными решателем, и места для дальнейшей его работы не хватает. Так как величина зоны задач достаточно велика, и обычный

(нормальный) режим работы системы — при практически пустой зоне задач, появление этого сообщения обычно связано с каким-либо зацикливанием программы. Появляется это сообщение чрезвычайно редко.

4) “Исчерпание переменных либо переполнение при арифметическом действии” — обычно означает, что ведено слишком много символов переменных и запрос на ввод нового такого символа не может быть удовлетворен.

5) “Превышение допустимого сброса” — означает попытку применения оператора сброса, выводящую за рамки действия текущей программы (число указанных для сброса перечисляющих операторов в рамках программы больше фактически имеющегося).

Кроме перечисленных сообщений, может появиться сообщение “Нет программы логического символа S” — если предпринята попытка обратиться к выполнению отсутствующей программы этого символа.

4.3.5. Выход в базу приемов, реализованных на ГЕНОЛОГе

Если выход в отладчик ЛОСа произошел при семантической трассировке после срабатывания некоторого приема, то нажатие клавиши “б” приводит к просмотру описания примененного приема на ГЕНОЛОГе — если был применен прием, запрограммированный на ГЕНОЛОГе, либо к просмотру концевого пункта оглавления программ ЛОСа, — если прием запрограммирован непосредственно на ЛОСе, и данному концевому пункту соответствует последняя пройденная перед срабатыванием контрольная точка “прием(N)”. При технической трассировке, если просматривается некоторый фрагмент программы (текущий либо некоторый его надфрагмент, достижимый из текущего с помощью клавиши “PageUp”), то нажатие клавиши “б” также приводит к просмотру описания соответствующего этому фрагменту приема ГЕНОЛОГа либо концевого пункта оглавления программ ЛОСа — однако, лишь при условии, что текущий оператор рассматриваемого фрагмента расположен после оператора “контрольприема(...)”, указывающего на прием ГЕНОЛОГа, либо оператора “прием(N)”, указывающего на концевой пункт оглавления ЛОСа.

Выйти в оглавление ГЕНОЛОГа из отладчика ЛОСа можно и безотносительно к срабатыванию текущего приема. Для этого следует нажать клавишу “г”, после чего появляется некоторый пункт оглавления базы приемов. По этому оглавлению можно перемещаться без каких-либо ограничений, входя в просмотр любых приемов ГЕНОЛОГа — так же, как и при просмотре этих приемов из главного меню. Заблокированы лишь те операции просмотра приемов, которые приводят к изменению приемов.

Если в оглавлении базы приемов был выбран некоторый концевой пункт и произошел вход в просмотр приема, закрепленного за данным пунктом (смена таких приемов, если их несколько, обеспечивается клавишами “курсор вверх — курсор вниз”), то возможна установка прерывания при попытке применить данный прием. Для ее ввода достаточно нажать клавишу “Ctr-Enter”. Автоматически будет возобновлен процесс решения задачи, и выход в отладчик произойдет при обращении к оператору “контрольприема(...)”, с которого начинается программа данного приема. Здесь устанавливается режим пошаговой трассировки и прорисовывается текущий фрагмент программы с выделенным малиновым цветом оператором, непосредственно следующим за оператором “контрольприема(...)”. Для выявления собственно момента применения данного приема далее можно установить прерывание при выполнении завершающей части его программы (выбрав один из заключительных ее операторов, например, оператор “пересмотр”).

4.3.6. Техническая трассировка

При отладке программы обычно используются такие режимы ее выполнения (называемые технической трассировкой), при которых процесс дробится на отдельные небольшие фрагменты, после каждого из которых осуществляется выход в отладчик ЛОСа. Принцип дробления можно либо задать один раз на весь период трассировки, переходя после этого к очередному шагу нажатием клавиши “Enter”, либо каждый раз устанавливать очередное прерывание процесса специально.

Перечислим используемые в отладчике ЛОСа режимы технической трассировки и способы установки прерываний процесса. Сразу заметим, что для отмены любых установок на прерывания служит клавиша “0”. После нажатия на нее, если не создать новых установок на прерывания и нажать “Enter”, то решение задачи будет продолжено без выдачи каких-либо пояснений на экране вплоть до завершения (ответа либо отказа).

1) Трассировка с самым мелким дроблением процесса — так называемая пошаговая трассировка. Для установки этого режима из отладчика ЛОСа достаточно нажать клавишу “1” (кроме того, пошаговый режим устанавливается автоматически после целого ряда прерываний, выводящих в отладчик). После этого каждое нажатие клавиши “Enter” будет вызывать выполнение единственного шага - реализацию оператора (корневого либо расположенного внутри некоторого составного оператора) либо вычисление невырожденного операторного выражения. При пошаговом режиме трассировки можно входить в просмотр шагов выполнения составных операторов (дизъюнкций, конъюнкций, кванторов и т.п.).

2) Следующий режим — пооператорная трассировка. В этом режиме за один шаг выполняется один корневой оператор программы — как простой, так и составной. Для установки режима следует выбрать, используя клавиши “PageUp — PageDown”, ту программу, в которой предполагается выполнять трассировку, и нажать клавишу “2”. Будут отслеживаться только корневые операторы, которые выполняются в “кадре” этой программы, без выхода в выполнение подоператоров и вспомогательных задач. По окончании выполнения программы трассировка должна быть переустановлена - иначе следующее прерывание произойдет в каком-то достаточно случайном месте. Эта трассировка при отладке является наиболее употребительной.

3) Иногда бывает нужно отслеживать моменты обращения к программе заданного логического символа A. Чтобы установить прерывание при обращении к этой программе, следует сначала нажать клавишу “л” — в результате появится курсор текстового редактора, — а затем набрать текстовым редактором название символа A. После этого, при нажатии “Enter”, установка оказывается введенной, хотя продолжение процесса реализации программы еще не включается. При запуске программы (снова нажатием “Enter”) отладчик обеспечит выход в просмотр первого оператора начального фрагмента программы символа A, как только произойдет обращение к этой программе. Установка на данное прерывание сохраняется до установки другого прерывания либо отмены прерываний, так что последовательные нажатия “Enter” позволят проследить цепочку указанных обращений.

4) При просмотре отладчиком некоторого фрагмента программы (текущего фрагмента либо его надфрагмента, достигнутого при помощи “Home” либо “PageUp”) возможен ввод установки на прерывание при переходе к его подфрагменту. Для этого следует войти в режим просмотра операторов фрагмента (нажатием клавиши “курсор вниз”; выбрать требуемый оператор перехода к подфрагменту (клавиши “курсор вправо — курсор влево”) и нажать клавишу “Ctr-Enter”. Нажатие этой клавиши автоматически запускает процесс продолжения выполнения программы; при выходе на указанный подфрагмент произойдет прерывание. Заметим, что прерывание будет иметь место, даже если выход на подфрагмент произошел не из того “кадра” реализации программы, где была введена установка на прерывание, а из какого-либо совсем другого “кадра” реализации той же программы. Если требуется, чтобы прерывание произошло лишь при условии, что обращение к подфрагменту имело место в текущем “кадре” реализации программы, то после выбора оператора перехода нажимается “Enter”. В этом случае завершение выполнения программы

без выхода на выбранный ее подфрагмент приведет к прерыванию по ее завершении.

5) При просмотре отладчиком некоторого фрагмента программы (текущего фрагмента либо какого-либо его надфрагмента, достигнутого при помощи клавиш “Home” либо “PageUp”) возможен ввод установки на прерывание перед выполнением какого-либо оператора этого фрагмента (не обязательно корневого; возможна установка прерывания перед выполнением подоператора составного оператора). Для этого следует войти в режим просмотра операторов фрагмента (“курсор вниз”); выбрать требуемый оператор (сначала клавишами “курсор вправо — курсор влево” выйти на нужный терм фрагмента, а при необходимости использовать клавишу “курсор вниз” и указанные выше клавиши для выделения подтерма) и нажать “Ctr-Enter”. При этом автоматически запускается продолжение действий по программе, и при выходе на указанный оператор (непосредственно перед его реализацией) происходит прерывание. Заметим, что прерывание произойдет, даже если “кадр” реализации программы, в котором было установлено прерывание, отличается от “кадра” реализации этой же программы, в котором имело место обращение к оператору. Чтобы ввести установку на прерывания только в том же самом “кадре” реализации программы, следует нажать “Enter”.

Использование установок на прерывания из пунктов 3,4,5 позволяет при отладке свободно перемещаться по программам различных логических символов и анализировать ситуации, возникающие на моменты выхода в различные их точки.

6) Чтобы использовать при отладке счетчик шагов работы интерпретатора ЛОСа, можно установить прерывание по достижении заданного числа шагов. Для этого нажимается клавиша “ш” — появляется курсор текстового редактора, — и набирается номер нужного шага. При наборе номера следует учитывать расположенный в правом верхнем углу экрана (если просматривается фрагмент программы) номер текущего шага. Как уже говорилось выше, при обращениях к отладчику счетчик шагов отключается, так что процесс отладки не изменяет выраженных в числе шагов интерпретатора “координат” событий, происходящих при решении задачи. После набора номера требуемого шага (как обычно, завершаемого нажатием “Enter”) установка на прерывание введена; для продолжения цикла выполнения программы еще раз нажимается “Enter”.

Если решается задача, извлеченная из задачника, то можно сохранить номер текущего шага и при повторных запусках ее решения выходить в отладчик по достижении шага с данным номером. Для сохранения номера следует нажать клавишу “щ”. Чтобы повторно запустить задачу с прерыванием по достижении данного шага, следует нажать клавишу “а”

при просмотре условия задачи в задачнике — это нажатие одновременно запускает решение и вводит установку на прерывание.

7) Если произошло прерывание с выходом в отладчик ЛОСа, отличное от обычного шага семантической трассировки, то для возвращения в режим семантической трассировки следует нажать клавишу “пробел”, после чего нажать “Enter”. Прерывание произойдет при первом же применении приема — в режиме сканирования задачи либо при реализации пакетного оператора ГЕНОЛОГа (только для особо крупных операторов, выделяемых специальной опцией в своем описании). Если вместо клавиши “пробел” нажать клавишу “й”, то прерывание произойдет при первом же применении приема на уровне сканирования текущей задачи — выполнение пакетных операторов при этом игнорируется.

8) Если нужно ввести прерывание при выходе из текущего “кадра” реализации программы (выбираемого с помощью “PageUp, PageDown”), то нажимаются клавиши “4” и “Enter”. При ликвидации этого кадра и возвращении во внешнюю программу произойдет выход в отладчик и установится пошаговый режим трассировки.

9) Для ввода прерывания при изменении оператором “замена” в текущем кадре реализации программы значения переменной “xi”, если этим значением служит терм, нажимается клавиша “7” - появляются буква “з” и курсор текстового редактора, — и далее набирается символьный (!) номер i. При каждом изменении указанной переменной будет возникать текст “значение переменной xi: A заменяется на B”.

10) Если требуется установить прерывание при изменении разрядов набора, обозначенного при просмотре элементов текущей структуры данных посредством “нi”, то нажимается клавиша “и” - появляется курсор текстового редактора, — и вводится номер i в обычном формате десятичного числа. Прерывание произойдет непосредственно перед выполнением оператора, изменяющего указанный набор. Если представляющий интерес набор является значением программной переменной “xi”, для установки прерывания при изменении его разрядов нажимается клавиша “И” и вводится номер i.

11) Из отладчика можно выйти в оглавление программ и установить прерывание по достижении заданной в этом оглавлении контрольной точки. Для этого сначала нажимается “Ctr-r” - появляется некоторое меню оглавления программ. После выбора в оглавлении нужного концевого пункта, следует нажать на нем клавишу “курсор вправо”. Это приведет к установке прерывания по выходу на соответствующую контрольную точку “прием(N)” и возобновлению процесса решения задачи.

12) Прерывание перед выполнением заданного оператора фрагмента программы (см. пункт 5) можно сопровождать списком дополнительных условий, а также обеспечивать автоматическую выдачу на экран при таком прерывании значений заданных программных переменных. Для входа в просмотр и редактирование списка дополнительных условий следует нажать при просмотре фрагмента программы клавишу “у” (кир.). При этом включается редактор списков, отображающий ранее введенные условия. Окно его располагается непосредственно под горизонтальной чертой, завершающей текст фрагмента программы. Список условий разбивается на группы; первым элементом каждой группы служит некоторый ключевой терм (см. ниже), после которого размещаются сопровождающие его информационные элементы. Используются ключевые термы следующих типов:

а) “терм(A)”. A есть программное выражение, определяющее объект для сравнения. Следующий элемент списка есть терм, который должен совпадать с этим объектом.

б) “заголовок(xi A)”. Значением программной переменной “xi” является логический символ A либо набор или терм, начинающийся с логического символа A.

в) “входит(A xi)”. Значением переменной “xi” является терм либо набор, содержащий логический символ либо символ переменной A.

г) “корень”. Текущий терм задачи должен совпадать с следующим элементом данного списка установок.

д) “См(A)”. A — программная переменная “xi” либо выражение “буква(xi)”. Если значением A служит терм либо логический символ, то при прерывании предпринимается выдача этого значения на экран.

В редакторе списков используется следующий интерфейс. Для набор очередного элемента списка формульным редактором служит клавиша “Enter”; для набора элемента текстовым редактором - клавиша “т” (кир.). Для изменения текущего элемента формульным редактором нажимается “Ctr-ф”; для изменения его текстовым редактором — “Ctr-т”. Завершается просмотр и редактирование списка нажатием клавиши “курсор влево”. После этого, как обычно вводится установка на прерывание при обращении к некоторому оператору текущего фрагмента программы. Данное прерывание будет происходить всякий раз перед обращением к выбранному оператору, как только окажутся выполнены дополнительные ограничения согласно списку. Для сброса списка дополнительных установок на прерывание нажимается клавиша “Ctr-у” (кир.).

13) В заключение отметим еще одну исключительно важную возможность, предоставляемую отладчиком ЛОСа. При автоматическом решении серии

задач из задачника используется так называемый режим “внутреннего перезапуска” — после каждой задачи интерпретатор повторно инициирует выполнение программы логической системы, обнулив все ее регистры и массивы в оперативной памяти. Решение серии задач невозможно отменить, даже используя предоставляемые операционной системой возможности по экстренному выходу из программы — при повторном запуске оно возобновится с той точки, где было прервано. Единственным способом отмены этого режима служит выход в отладчик ЛОСа с помощью клавиши “Break” и нажатие в отладчике клавиши “Ctr-z”, которое, собственно, и отменяет серийное решение задач. Далее можно вернуться в главное меню, например, нажатием клавиши “Esc” (возвращение в главное меню без нажатия “Ctr-z” просто вызовет переход к следующей задаче серии).

4.4. Тестирование программы оператора, операторного выражения либо справочника

Для тестирования программы нового оператора, операторного выражения либо справочника лучше всего воспользоваться реальным контекстом, в котором она должна применяться — приемом решения задачи, либо приемом пакетного оператора, либо блоком интерфейса системы, и т.п. Однако, можно обратиться к этому оператору и непосредственно. Для этого служит программа логического символа “пуск”, которую можно запускать из просмотра программы в редакторе программ ЛОСа нажатием клавиши “Ctr-Enter”. Эта программа имеет единственную входную переменную x1, значением которой служит исходная задача. Войдя в редактирование программы символа “пуск”, нужно набрать обращение к тестируемому оператору (программному выражению, справочнику). Сначала программируется построение входных объектов для тестируемой программе, затем размещается само обращение, после которого нужно поместить хотя бы один оператор — чтобы до выхода из программы “пуск” иметь возможность просмотра значений выходных переменных, полученных после обращения. Для удобства создания входных объектов и просмотра результатов обращения, если необходимо, в программу “пуск” можно включить упоминавшиеся выше вспомогательные операторы интерфейса. По завершении редактирования программы “пуск” следует нажать “Ctr-Enter”. Тогда инициализируется работа только что набранной программы “пуск”, причем сразу же включается отладчик ЛОСа с режимом пошаговой трассировки. Далее с помощью отладчика ЛОСа можно предпринять анализ выполнения тестируемой программы. По завершении программы “пуск” происходит возвращение в главное меню.

Список литературы

[1] Подколзин А. С., “Введение в логические процессы. Представление задач в решателе”, *Интеллектуальные системы. Теория и приложения*, **29**:2 (2025), 5–138.

[2] Подколзин А. С., “Введение в логические процессы. Общая схема функционирования решателя”, *Интеллектуальные системы. Теория и приложения*, **29**:3 (2025), 6–52.

[3] Подколзин А. С., *Компьютерное моделирование логических процессов. Том 1. Архитектура и языки решателя задач.*, Физматлит, Москва, 2008, 1024 с.

[4] Подколзин А. С., *Компьютерное моделирование логических процессов. Том 2. Опыт обучения компьютерного решателя задач: логические приемы, алгебра множеств, комбинаторика и элементарная алгебра.*, ВИНИТИ РАН, Москва, 2015, 1153 с.

[5] Подколзин А. С., *Компьютерное моделирование логических процессов. Том 3. Опыт обучения компьютерного решателя задач: математический анализ, дифференциальные уравнения и элементарная геометрия.*, ВИНИТИ РАН, Москва, 2015, 1320 с.

[6] Подколзин А. С., *Компьютерное моделирование логических процессов. Том 4. Опыт обучения компьютерного решателя задач: аналитическая геометрия, линейная алгебра, теория вероятностей, комплексный анализ и другие разделы.*, ВИНИТИ РАН, Москва, 2017, 969 с.

[7] Подколзин А. С., *Компьютерное моделирование логических процессов. Том 5. Опыт обучения компьютерного решателя задач: элементарные физика и химия, шахматы.*, ВИНИТИ РАН, Москва, 2019, 938 с.

[8] Подколзин А. С., *Компьютерное моделирование логических процессов. Том 6. Опыт обучения компьютерного решателя задач: понимание естественного языка и анализ рисунков.*, ВИНИТИ РАН, Москва, 2019, 757 с.

[9] Подколзин А. С., *Компьютерное моделирование логических процессов. Том 7. Автоматическое создание приемов логической системы: классификация приемов решателя, логический ассемблер, компилятор спецификаций, создание тестовых приемов и доводка приемов.*, ВИНИТИ РАН, Москва, 2021, 739 с.

[10] Подколзин А. С., *Компьютерное моделирование логических процессов. Том 8. Автоматическое создание приемов логической системы: база теорем, характеризация теорем, создание спецификаций приемов.*, ВИНИТИ РАН, Москва, 2021, 515 с.

[11] Подколзин А. С., *Компьютерное моделирование логических процессов. Том 9. Автоматическое создание приемов логической системы: логический вывод в базе теорем.*, ВИНИТИ РАН, Москва, 2022, 1494 с.

Introduction to logical processes. The algorithmic language LDS
Podkolzin A.S.

This article describes the algorithmic language LDS (logical description of situations). It describes the language's basic operators, the LDS program editor, and demonstrates how to debug LDS programs.

Keywords: mathematical problem solver, logical processes, logical language, logical formalization of problems.

References

[1] Podkolzin A. S., "Introduction to Logical Processes. Representation of Problems in the Solver", *Intelligent Systems. Theory and Applications*, **29**:2 (2025), 5–138 (In Russian)

[2] Podkolzin A. S., "Introduction to Logical Processes. General diagram of the Solver's functioning.", *Intelligent Systems. Theory and Applications*, **29**:3 (2025), 6–52 (In Russian)

[3] Podkolzin A. S., *Computer modeling of logical processes. Volume 1. Architecture and languages of the problem solver.*, Fizmatlit, Moscow, 2008, 1024 pp.

[4] Podkolzin A. S., *Computer modeling of logical processes. Volume 2. Experience in training a computer problem solver: logical techniques, set algebra, combinatorics and elementary algebra.*, VINITI RAS, Moscow, 2015, 1153 pp.

[5] Podkolzin A. S., *Computer modeling of logical processes. Volume 3. Experience in teaching computer problem solver: mathematical analysis, differential equations and elementary geometry.*, VINITI RAS, Moscow, 2015, 1320 pp.

[6] Podkolzin A. S., *Computer modeling of logical processes. Volume 4. Experience in teaching computer problem solver: analytical geometry, linear algebra, probability theory, complex analysis and other topics.*, VINITI RAS, Moscow, 2017, 969 pp.

[7] Podkolzin A. S., *Computer modeling of logical processes. Volume 5. Experience in teaching a computer problem solver: elementary physics and chemistry, chess.*, VINITI RAS, Moscow, 2019, 938 pp.

[8] Podkolzin A. S., *Computer modeling of logical processes. Volume 6. Experience in training a computer problem solver: natural language understanding and image analysis.*, VINITI RAS, Moscow, 2019, 757 pp.

[9] Podkolzin A. S., *Computer modeling of logical processes. Volume 7. Automatic creation of logic system techniques: classification of solver techniques, logic assembler, specification compiler, creation of test techniques and refinement of techniques.*, VINITI RAS, Moscow, 2021, 739 pp.

[10] Podkolzin A. S., *Computer modeling of logical processes. Volume 8. Automatic creation of logical system techniques: theorem base, characterization of theorems, creation of technique specifications.*, VINITI RAS, Moscow, 2021, 515 pp.

[11] Podkolzin A. S., *Computer modeling of logical processes. Volume 9. Automatic creation of logical system techniques: logical inference in the theorem base.*, VINITI RAS, Moscow, 2022, 1494 pp.

Часть 2
Специальные вопросы теории интеллектуальных систем

Функции для аппроксимации объемов газа и нефти в области фазовых переходов

Е. В. Колдоба[1]

Рассматриваются фазовые переходы в многокомпонентных растворах нефти. Предложены зависимости для молярных объемов газовой и жидкой фаз, учитывающие особенности поведения растворов — изменения концентраций на фазовых переходах. Функции позволяют с высокой точностью аппроксимировать реальные зависимости, строить аналитические модели с фазовыми переходами в широких диапазонах давлений и температур.

Ключевые слова: углеводородные растворы, фазовое равновесие, математическое моделирование, термодинамика.

1. Введение

Для проектирования и мониторинга нефтяных месторождений создаются сложные гидродинамические модели, и один из важных этапов — это задание PVT-свойств пластовых многофазных флюидов (термин «флюид» используется как обобщённое название жидких и газовых фаз) [1, 2].

Природные газы и нефть — это многокомпонентные растворы (смеси). В растворах фазовые переходы имеют ряд особенностей: переход начинается при одном давлении и заканчивается при другом, и в его процессе непрерывно меняется компонентный состав фаз. Эти особенности значительно усложняют расчеты фазового равновесия, и аппроксимации молярных объемов могут упростить и даже улучшить результаты. Для моделирования фильтрации с фазовыми переходами существуют модели разной степени точности и сложности. Самые популярные — это модель черной нефти (black-oil) и композиционная модель [3, 4].

В модели черной нефти многокомпонентность сводится к двум псевдокомпонентам: «нефтяной» и «газовый». При расчетах используются измеренные табличные PVT-значения, которые из-за экспериментальных ошибок необходимо численно обрабатывать перед применением. Для этого необходимо знать вид этих зависимостей, например, использовать формулы, предложенные в данной работе.

Композиционная модель содержит значительно большее количество компонент, поэтому точнее описывает фазовые переходы в растворах. В

[1] *Колдоба Елена Валентиновна* — доцент каф. вычислительной механики мех.-мат. ф-та МГУ, e-mail: elenakoldoba@mail.ru.

Koldoba Elena Valentinovna — Associate Professor, Lomonosov Moscow State University, Faculty of Mechanics and Mathematics, Chair of Computational Mechanics.

ней задаются концентрации компонент и их свойства, и по ним рассчитываются PVT-свойства. Так как в нефти содержится сотни и даже тысячи компонент, то их группируют по свойствам и вместо реальных компонент получают укрупненные псевдокомпоненты (или фракции), количество которых становится значительно меньше. Тем не менее свойства расчетных флюидов иногда существенно отличаются от свойств реальной нефти, поэтому и эта модель содержит большое количество настраиваемых параметров и библиотеку возможных зависимостей, задающих свойства растворов. В ряде случаев расчеты требуют столь значительных вычислительных ресурсов, что результаты теряют смысл для проектирования и мониторинга в режиме реального времени. Аналитические приближения в таких случаях являются единственно возможным вариантом для снижения вычислительных затрат и минимизации ошибок расчета вследствие большого объема обрабатываемой информации.

Аппроксимации объемов также могут быть актуальны для создания прокси-моделей месторождений, которые в силу упрощённого описания, позволяют проводить расчёты значительно быстрее, чем традиционные гидродинамические модели.

2. Вычисление молярных объемов на фазовых переходах в композиционной модели

Пусть заданный N-компонентный раствор с полной молярной концентрацией (долей) $\{z_i\}$ находится в двухфазном состоянии, разделившись на газ с концентрацией $\{y_i\}$ и жидкость с концентрацией $\{x_i\}$, где i — номер компоненты, $i = 1, 2, \ldots, N$. Для молярных концентраций выполняются следующие условия:

$$\sum_{i=1}^{N} z_i = 1, \quad \sum_{i=1}^{N} x_i = 1, \quad \sum_{i=1}^{N} y_i = 1.$$

Для расчетов фазового равновесия используют уравнения состояния, задающие связь температуры T, давления P, молярного объема V и молярных концентраций в фазах. В современных гидродинамических симуляторах для расчета нефтяных и газовых месторождений используются различные уравнения состояния Ван-дер-Ваальсовского типа, одно из самых популярных среди них это уравнение Пенга-Робинсона [5]:

$$P = \frac{RT}{V - b} - \frac{a}{V^2 + 2bV - b^2} \tag{1}$$

где R — газовая постоянная; a, b – параметры, зависящие от критической температуры T_{ci}, критического давления P_{ci}, ацентрического фактора ω_i

каждого i-ого компонента и их концентраций. Процедуры расчета a, b для растворов хорошо известны.

Когда рассчитано фазовое равновесие и вычислены $\{y_i\}$ и $\{x_i\}$, то по уравнению состояния (1) можно вычислить молярные объемы газа V_G и жидкости V_L:

$$V_G = V_G(P, T, \{y_i(P, T, \{z_i\})\}),$$
$$V_L = V_L(P, T, \{x_i(P, T, \{z_i\})\}).$$

Обозначим эти объемы как W_G и W_L, чтобы подчеркнуть, что эти объемы вычислены для конкретного раствора с концентрацией $\{z_i\}$. С учетом того, что в данном случае $\{y_i\} = \{y_i(P, T, \{z_i\})\}$ и $\{x_i\} = \{x_i(P, T, \{z_i\})\}$, имеем следующие соотношения:

$$W_G \equiv V_G(P, T, \{z_i\}), \quad W_L \equiv V_L(P, T, \{z_i\}).$$

3. Аппроксимации молярных объемов газовой и жидкой фазы на фазовых переходах

Пусть заданы абсолютная температура $T = Const$ и полная концентрация раствора $\{z_i\}$. В данной работе для аппроксимации функций W_G и W_L предложены функции следующего вида:

$$W_G(P, T) = \frac{\beta RT}{P} + b_G, \quad W_L = \frac{\alpha RT}{P + P_*} + b_L, \tag{2}$$

где $R = 8.314462$дж/(моль К) — универсальная газовая постоянная; b_G, β — параметры аппроксимации для газовой фазы; b_L, α, P_* — параметры для жидкой фазы. T — температура в Кельвинах, P — давление в барах, W_G, W_L, b_G, b_L — объемы в кубических метрах, α, β — безрахмерные величины.

Функции (2), аппроксимирующие объемы газовой и жидкой фаз, выбирались похожими, так чтобы они переходили одна в другую в критической точке раствора, в которой различия между фазами исчезают. Предполагалось, что подобный подход может оказаться интересным для моделирования газовых конденсатов или околокритических флюидов — случаев особенно трудных для моделирования. Однако проведенные исследования показали применимость формул (2) в значительно большем диапазоне давлений и температур.

Для вычисления параметров функций (2) необходимы данные о флюиде, они берутся либо из композиционной модели, либо из экспериментальных измерений. В данной работе аппроксимации молярных объемов проводились для ряда многокомпонентных растворов на основе данных,

полученных по композиционной модели гидродинамического симулятора t-Navigator. Один из примеров приведен ниже.

4. Пример аппроксимации молярных объемов

Рассмотрим углеводородный раствор, состоящий из 6 псевдокомпонент (или 6 фракций), лабораторные данные о котором представлены в таблице 1. Для каждого нефтяного пласта, состав которого известен, всегда

	z_i	$T_{ci}, ^oK$	P_{ci}, bar	ω_i
$C_1C_2 - N_2$	0.389665	199.83	45.976	0.015
$C_3 - C_4$	0.080687	397.01	39.126	0.171
$C_5 - C_6$	0.059501	483.91	32.981	0.254
$C_7 - C_{13}$	0.241202	615.447	27.055	0.431
$C_{14} - C_{20}$	0.108817	793.728	16.926	0.752
C_{21+}	0.120128	919.579	9.171	1.243

Таблица 1.

строятся фазовые диаграммы «давление-температура», чтобы определить какие сложности возникнут при моделировании, какой точности модель необходимо использовать. Важнейшей характеристикой является соотношение температуры пласта T и критической температуры раствора $T_{c,mix}$. Например:

– если $T << T_{c,mix}$, то можно использовать модель черной нефти;

– если $T < T_{c,mix}$, то используется модель летучей нефти;

– если $T \approx T_{c,mix}$, то это область околокритического флюида;

– если $T > T_{c,mix}$, то это область ретроградной конденсации (газовые конденсаты).

Далее температура будет в градусах Цельсия, чтобы подчеркнуть важность полученных результатов для реальных месторождений.

Фазовая диаграмма «давление-температура» данной смеси, построенная с помощью симулятора t-Navigator, представлена на Рис. 1. Кривая кипения и кривая конденсации ограничивают двухфазную область $L + G$ и сходятся в критической точке данного раствора C ($P_{c,mix} = 123.04\ bar,\ t_{c,mix} = 439.35^oC$). Выше кривой кипения находится однофазная жидкая область L, ниже кривой конденсации находится однофазная газовая область G. При фиксированной температуре фазовый переход начинается при одном давлении P_{start} и заканчивается при другом P_{end}, и между ними находится двухфазная область. Так при $t = 0^oC$ (здесь и далее температура в градусах Цельсия) фазовый переход начинается

при $P = 65\ bar$ и заканчивается при $P = 1\ bar$. Для разных температур границы области фазовых переходов представлены в таблице 2. Температуры 0, 430, 500^oC выбирались так, чтобы продемонстрировать применимость предлагаемых аппроксимаций для модели черной нефти, для околокритического флюида и для ретроградной области.

t^oC	0^oC	430^oC	500^oC
P_{start}, bar	65	126.8	83.5
P_{end}, bar	1	2.7	13.8

Таблица 2.

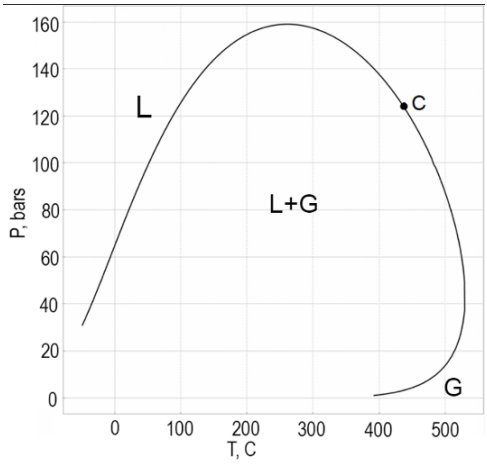


Рис. 1. Фазовая диаграмма «давление-температура» данного раствора. C — критическая точка раствора.

При каждой температуре для трех давлений рассчитывались молярные объемы по «точной» композиционной модели, затем вычислялись параметры аппроксимаций. Результаты представлены в таблице 3.

Полученные по формулам (2) аппроксимации молярных объемов сравнивались с «точными» значениями. Максимальные относительные погрешности аппроксимации молярных объемов газовой и жидкой фаз при разных температурах представлены в таблице 4.

На рисунках 2, 3, 4 сплошными линиями нанесены «точные» значения молярных объемов W_G и W_L (отмечены цифрой 1), а пунктирными линиями — аппроксимирующие их кривые (отмечены цифрой 2). Графики «точных» значений и аппроксимирующие их кривые практически сливаются, что говорит о том, что предлагаемые функции (2) правильно описывают вид зависимости.

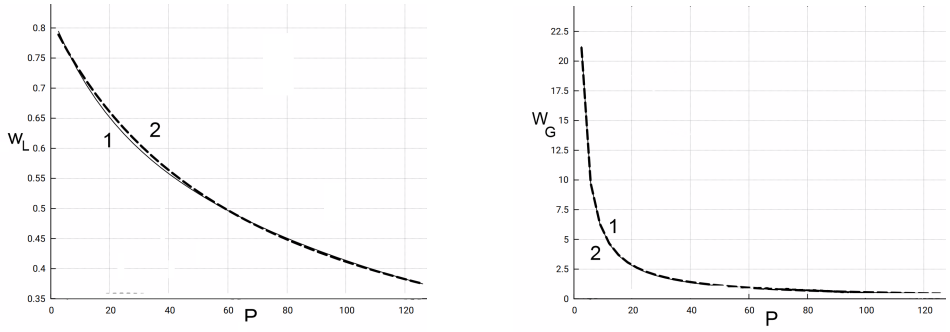


Рис. 2. Зависимость от давления молярных объемов жидкой (а) и газовой (б) фазы при $t = 430^oC$

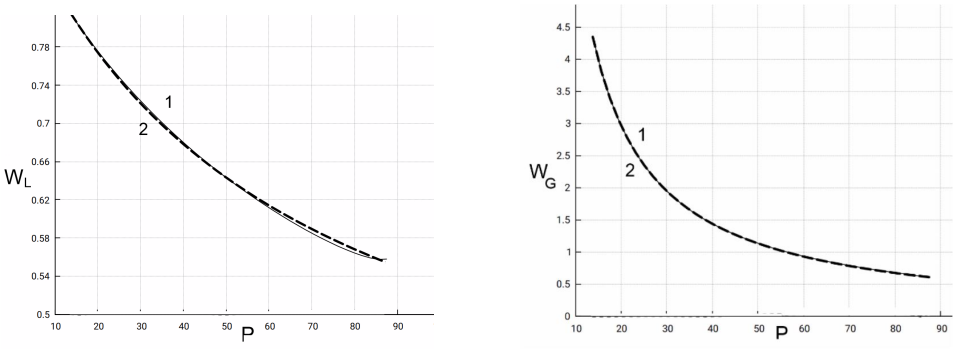


Рис. 3. Зависимость от давления молярных объемов жидкой (а) и газовой (б) фазы при $t = 500^oC$

Графики на рисунках 2, 3 демонстрируют, что аппроксимации с высокой точностью передают зависимости объемов для околокритического флюида ($t = 430^oC$) и в области ретроградных явлений ($t = 500^oC$) — в областях наиболее сложных для численного моделирования.

Графики на рисунках 4 демонстрируют, что и при температурах, далеких от критической точки раствора, формулы (2) также хорошо передают зависимости молярных объемов: для газовой фазы очень хорошо во всем диапазоне фазового перехода и немного хуже для жидкой фазы в окрестности точки вхождения в двухфазную область.

5. Результаты

Предложенные функции молярных объемов фаз с хорошей точностью передают реальные зависимости для фиксированного состава во всей области фазовых переходов.

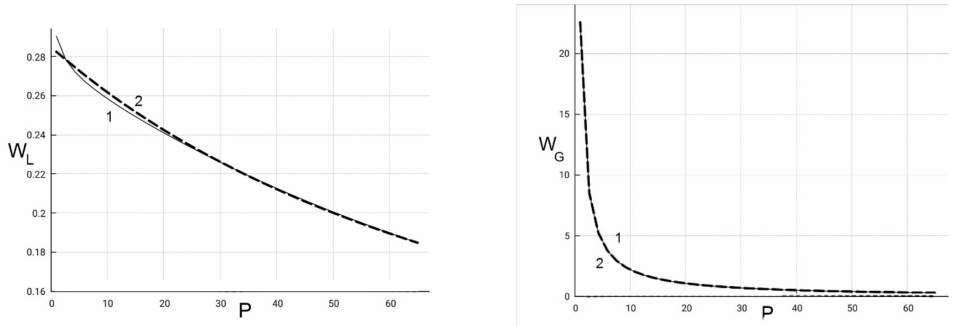


Рис. 4. Зависимость от давления молярных объемов жидкой (а) и газовой (б) фазы при $t = 0^oC$

	b_G	β	b_L	α	P_*
$t = 0^oC$	0.785625	0.975963	-0.045357	0.137398	67.86224
$t = 430^oC$	0.605515	0.953059	-0.096679	0.291150	60.53591
$t = 500^oC$	1.100905	0.996608	-0.074652	0.032193	98.89928

Таблица 3.

Результаты представляют интерес для создания упрощенных моделей фильтрации с фазовыми переходами [6], для обработки экспериментальных данных, а также для создания прокси-моделей пластов на месторождениях нефти и газа.

Так как предложенные аналитические выражения передают интегральные характеристики флюидов и не содержат большого количества настраиваемых параметров, то они могут значительно сократить время расчета термодинамической части моделирования.

t^oC	0^oC	430^oC	500^oC
$\Delta(max)_L$	2.78%	1.46%	0.81%
$\Delta(max)_G$	0.51%	3.64%	0.91%

Таблица 4.

Список литературы

[1] Whitson C. H, Brulé M. R., *Phase behavior*, SPE, Richardson, Texas, 2000, 240 pp.

[2] Ющенко Т.С., Брусиловский А.И., "Новый инженерный метод создания и адаптации PVT-модели природной газоконденсатной смеси", *Научно-технический сборник «ВЕСТИ ГАЗОВОЙ НАУКИ»*, **4**:24 (2015), 14–20.

[3] Mydland S., Whitson C. H., Carlsen M. L., et al., "Black-oil and compositional reservoir simulation of gas-based EOR in tight unconventionals", *URTeC 2765. In: Unconventional Resources Technology Conference*, 2020, 1–30.

[4] Лобанова О. А., Индрупский И. М., "Особенности реализации алгоритмов композиционного моделирования в современных гидродинамических симуляторах", *SOCAR Proceedings*, **3** (2023), 120–130.

[5] Брусиловский А. И., *Фазовые превращения при разработке месторождений нефти и газа*, Грааль, Москва, 2002, 575 с.

[6] Колдоба Е.В., "Эффективный термодинамически согласованный подход для численного моделирования процессов вытеснения нефти", *Математическое моделирование*, **10** (2009), 7–12.

Functions for approximating gas and oil volumes in phase transition region

Koldoba E.V.

Phase transitions in multicomponent oil solutions are considered. Dependencies for the molar volumes of the gas and liquid phases are proposed. The formulas take into account the behavioral characteristics of solutions, nnamely, changes in concentration during phase transitions. They enable highly accurate approximation of real dependencies and the construction of analytical models with phase transitions over wide ranges of pressures and temperatures.

Keywords: hydrocarbon solutions, phase equilibrium, mathematical modeling, thermodynamics.

References

[1] Whitson C. H, Brulé M. R., *Phase behavior*, SPE, Richardson, Texas, 2000, 240 pp.

[2] Yushchenko T. S., Brusilovskiy A. I., "A new engineering method for creating and adapting a PVT model of a natural gas condensate mixture", *Scientific and technical collection "NEWS OF GAS SCIENCE"*, **4**:24 (2015), 14–20 (In Russian)

[3] Mydland S., Whitson C. H., Carlsen M. L., et al., "Black-oil and compositional reservoir simulation of gas-based EOR in tight unconventionals", *URTeC 2765. In: Unconventional Resources Technology Conference*, 2020, 1–30

[4] Lobanova O. A., Indrupskiy I. M., "Features of the implementation of compositional modeling algorithms in modern hydrodynamic simulators", *SOCAR Proceedings*, **3** (2023), 120–130 (In Russian)

[5] Brusilovskiy A.I., *Phase transformations in the development of oil and gas fields*, Graal, Moscow, 2002 (In Russian), 575 pp.

[6] Koldoba E.V., "An efficient thermodynamically consistent approach for numerical modeling of oil displacement processes", *Matematicheskoe Modelirovanie*, **10** (2009), 7–12 (In Russian)

Часть 3
Математические модели

Сложность задачи о существовании сюръективного гомоморфизма на смешанно-ориентированные рефлексивные циклы

Н. П. Корчагин[1]

Для графа $\mathcal{H}$ в задаче Surj-Hom($\mathcal{H}$) по данному графу $\mathcal{G}$ требуется определить, существует ли сюръективный гомоморфизм из $\mathcal{G}$ на $\mathcal{H}$. В данной работе мы изучаем сложность задачи Surj-Hom для графов, которые получаются из неориентированных циклов добавлением ориентации некоторым рёбрам, и в которых каждая вершина содержит петлю.

Мы рассматриваем задачу Surj-Hom как частный случай массовой задачи сюръективного удовлетворения ограничениям SCSP. Мы вводим свойство, которое позволяет определять сложность SCSP с помощью сведения. Мы используем это свойство и определяем сложность Surj-Hom для всех рассматриваемых циклов, кроме трёх циклов длины 4, 5 и 6.

Ключевые слова: сюръективный гомоморфизм графов, сложность вычислений, удовлетворение ограничениям, полиморфизм.

1. Введение

Для графа $\mathcal{H}$ задача о существовании сюръективного гомоморфизма Surj-Hom($\mathcal{H}$) это массовая задача, в которой по данному графу $\mathcal{G}$ требуется определить, существует ли сюръективное отображение из вершин $\mathcal{G}$ на вершины $\mathcal{H}$, которое сохраняет отношение смежности в графе.

Массовые задачи, связанные с сюръективными гомоморфизмами графов, привлекли внимание исследователей ещё в конце прошлого века, и с тех пор стали объектом множества работ [1, 3]. Несмотря на то, что определение сложности подобных задач представляет в первую очередь теоретический интерес, задача Surj-Hom имеет практические корни, поскольку тесно связана с алгоритмами поиска различных свойств в графе [4]. Несложно заметить, что эта задача лежит в NP. Более того, существует предположение, что она всегда либо решается за полиномиальное

[1] *Корчагин Никита Павлович* — выпускник аспирантуры каф. математической теории интеллектуальных систем мех.-мат. ф-та МГУ, e-mail: kolkor92@gmail.com.

Korchagin Nikita Pavlovich. — postgraduate student, Lomonosov Moscow State University, Faculty of Mechanics and Mathematics, Chair of Mathematical Theory of Intellectual Systems.

время, либо является NP-полной [11, 18]. В связи с этим, большой интерес представляет вопрос определения сложности Surj-Hom($\mathcal{H}$) для всех графов $\mathcal{H}$. Для некоторых классов графов сложность задачи известна. Так, эта задача решается за полиномиальное время, если каждая связная компонента $\mathcal{H}$ – полный граф с петлями или полный двудольный граф без петель [6]. Задача является NP-полной для связных графов, содержащих ровно две петли у несмежных вершин [7]. В случае, если $\mathcal{H}$ – дерево с петлями, задача Surj-Hom($\mathcal{H}$) решается за полиномиальное время, если вершины с петлями образуют полный подграф, и является NP-полной иначе [8]. Однако, полная классификация сложности задачи остается недостижимой.

Одним из самых простых классов графов, для которых сложность Surj-Hom ещё не определена, являются *рефлексивные циклы* – графы, которые получаются из неориентированных циклов путём добавления ориентации некоторым рёбрам, и в которых каждая вершина содержит петлю. Известно, что Surj-Hom является NP-полной для неориентированного рефлексивного цикла длины 4 [2]. Также известна сложность задачи для всех рефлексивных циклов, содержащих три вершины [9]. В предыдущих работах автора была определена сложность задачи для всех неориентированных рефлексивных циклов [10] и рефлексивных циклов, в которых каждое ребро ориентировано ровно в одну сторону [16], кроме нескольких циклов длины 5, 6. В данной статье же речь пойдет о *смешанно-ориентированных рефлексивных циклах*, в которых часть рёбер ориентирована в одну сторону, а часть – в две.

1.1. Основные результаты

Главным результатом данной статьи является доказательство того, что для любого смешанно-ориентированного рефлексивного цикла $\mathcal{C}$, содержащего больше трёх вершин и не изоморфного ни одному из графов $\mathcal{C}_4$ (см. Рис. 1, а), $\mathcal{C}_5$ (Рис. 1, б) и $\mathcal{C}_6$ (Рис. 1, в), задача Surj-Hom($\mathcal{C}$) является NP-трудной:

Теорема 1. *Пусть $\mathcal{C}$ – смешанно-ориентированный рефлексивный цикл, содержащий больше трёх вершин и не изоморфный $\mathcal{C}_4, \mathcal{C}_5$ или $\mathcal{C}_6$. Тогда задача* Surj-Hom($\mathcal{C}$) *является* NP*-трудной.*

Для графов $\mathcal{C}_4, \mathcal{C}_5, \mathcal{C}_6$ сложность остаётся неизвестной. Заметим, что вместе с уже полученными результатами [9, 2, 16, 10], данная теорема позволяет описать сложность задачи Surj-Hom для всех рефлексивных циклов кроме конечного числа графов:

Теорема 2. *Пусть $\mathcal{C}$ – рефлексивный цикл, не изоморфный ни одному из циклов, изображённых на Рис. 1. Если $\mathcal{C}$ изоморфен одному из*

циклов, изображённых на Рис. 2, то задача Surj-Hom($\mathcal{C}$) *решается за полиномиальное время. Иначе она является* NP*-полной.*

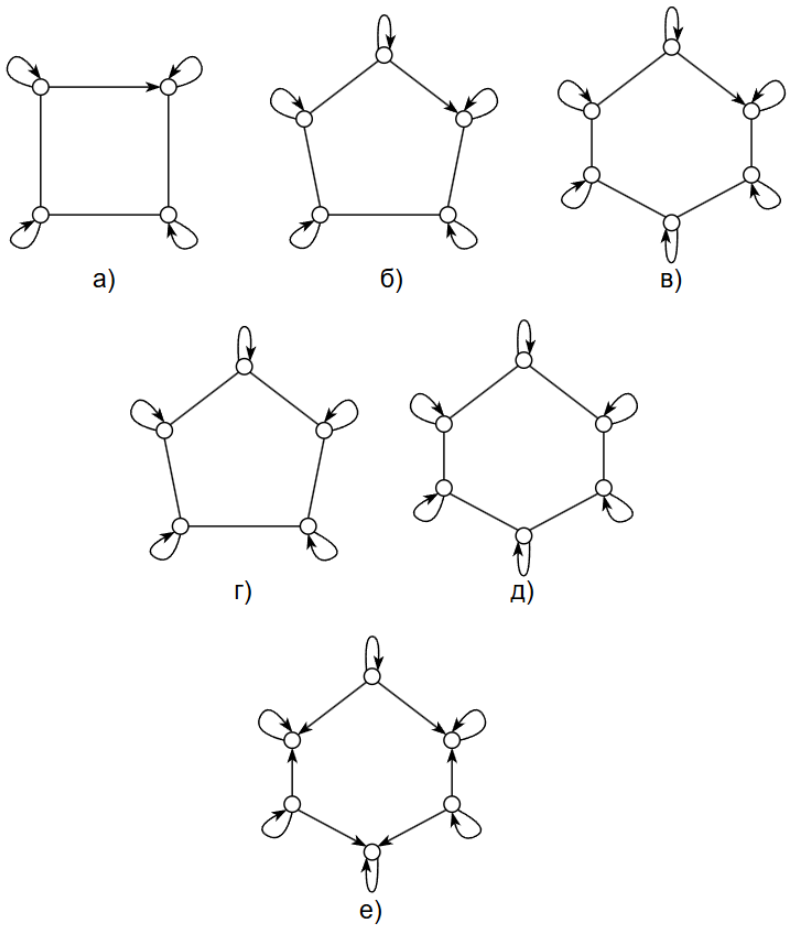


Рис. 1. Циклы, для которых ещё не определена сложность Surj-Hom.

Примечательно, что все циклы, для которых сложность задачи не определена, имеют длину от 4 до 6.

Дальнейшее изложение структурировано следующим образом. В разделе 2 приводятся основные понятия, которые используются в статье. В разделе 3.1 рассматриваются смешанно-ориентированные рефлексивные циклы с большим количеством ориентированных рёбер. Раздел 3.2 посвящен циклам с маленьким числом ориентированных рёбер. Наконец, в разделе 3.3 подробнее рассматриваются циклы $\mathcal{C}_4, \mathcal{C}_5, \mathcal{C}_6$, для которых не удалось определить сложность задачи Surj-Hom.

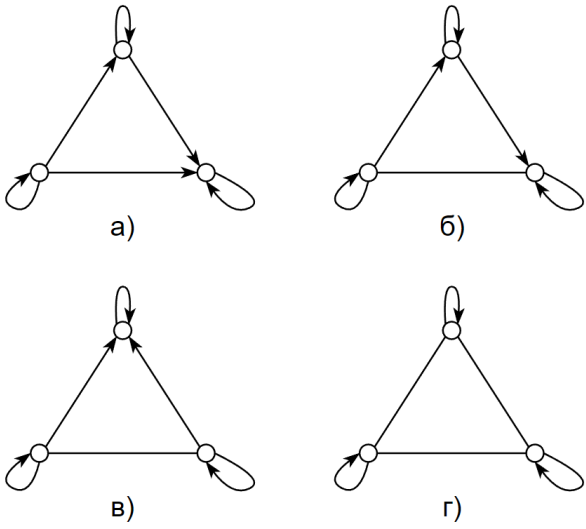


Рис. 2. Циклы, для которых Surj-Hom лежит в P

2. Основные понятия

2.1. Графы

Формально определим понятия, введённые выше. *Графом* $\mathcal{H} = (V, E)$ будем называть пару из множества V и множества $E \subseteq V \times V$ упорядоченных пар из V. Множество V будем называть *множеством вершин*, а множество E – *множеством рёбер*. Двуместное отношение E будем также называть *отношением смежности* $\mathcal{H}$. Для вершин $v, w \in V$ пару $(v, w) \in E$ будем называть *ребром из* v *в* w и обозначать как $v \to w$ или $w \leftarrow v$. Будем говорить, что в $\mathcal{H}$ *существует ориентированное ребро из* v *в* w, если $(v, w) \in E$ и $(w, v) \notin E$. В противном случае будем говорить, что существует *неориентированное ребро между* v *и* w. Будем обозначать такие рёбра как $v \leftrightarrow w$. *Петлёй* в $\mathcal{H}$ будем называть ребро вида $(v, v), v \in V$. Для удобства изложения далее в тексте мы часто будем отождествлять граф и отношение смежности на множестве его вершин.

Граф $\mathcal{H} = (V, E)$ называется *строго ориентированным*, если все его рёбра, кроме петель, ориентированные, и *неориентированным*, если все его рёбра неориентированны. Граф будем называть *смешанно-ориентированным*, если он содержит как ориентированные рёбра, так и неориентированные рёбра, не являющиеся петлями. Граф называется *рефлексивным*, если каждая вершина в нём содержит петлю.

Через V^k при $k \geq 1$ будем называть множество всех наборов вида $\overline{v} = (v^1, \ldots, v^k)$, где $v^i \in V$. Элементы V^k будем называть *векторами*,

для вектора $\overline{v} \in V^k$ через v^i будем обозначать i-ую компоненту этого вектора. Будем говорить, что существует ребро $\overline{v}\, e\, \overline{w}, e \in \{\leftarrow, \rightarrow, \leftrightarrow\}$, если для каждого $i \in \{1, \ldots, k\}$ верно $v^i\, e\, w^i$.

Определим *ориентированный путь π длины s* как последовательность вида $v_0\, e_0\, v_1\, e_1 \ldots v_{s-1}\, e_{s-1}\, v_s$, где $v_i \in V$ и $e_i \in \{\leftarrow, \rightarrow, \leftrightarrow\}$ – ориентации рёбер такие, что в $\mathcal{H}$ существует ребро $v_i\, e_i\, v_{i+1}$. Для $k \geq 1$ ориентированный путь Π длины s на V^k определяется аналогичным образом.

2.2. Смешанно-ориентированные циклы

Определим цикл формально: *цикл $\mathcal{C}$ длины n* – это граф с множеством вершин $Z_n = \{0, 1, \ldots, n-1\}, n \geq 3$, в котором для каждой вершины $v \in Z_n$ есть ребро $v\, e\, (v+1 \pmod n)$, $e \in \{\leftarrow, \rightarrow, \leftrightarrow\}$, и нет других рёбер кроме, может быть, петель.

Рассмотрим смешанно-ориентированный рефлексивный цикл $\mathcal{C}$ длины n. Пусть v, w – его вершины. Ориентированный путь из v в w, в котором все рёбра ориентированы как $\leftrightarrow$, будем обозначать как $v \leftrightarrow \cdots \leftrightarrow w$. Аналогично, через $v \rightarrow \cdots \rightarrow w$ и $v \leftarrow \cdots \leftarrow w$ будем обозначать ориентированные пути из v в w, в которых все рёбра ориентированы как $\rightarrow$ и $\leftarrow$ соответственно.

Подмножество вершин $V \subseteq Z_n$ будем называть *неориентированной компонентой*, если для любых $v, w \in V$ существует ориентированный путь вида $v \leftrightarrow \cdots \leftrightarrow w$. Несложно заметить, что отношение принадлежности одной и той же неориентированной компоненте это отношение эквивалентности. Отсюда, множество вершин Z_n разбивается на $m \leq n$ неориентированных компонент, соединённых между собой ориентированными рёбрами. Заметим, что количество неориентированных компонент в $\mathcal{C}$ равно количеству ориентированных рёбер.

Пусть в $\mathcal{C}$ содержится m неориентированных компонент. Обозначим их как $V_0, \ldots, V_{m-1}$, обходя граф по часовой стрелке, начиная с произвольной компоненты. Строго ориентированный цикл $\mathcal{C}^s$ с m вершинами будем называть *остовным циклом для $\mathcal{C}$*, если для любых $i, j \in Z_m, i \neq j$ верно

$$i \rightarrow j \iff \exists v \in V_i, w \in V_j : v \rightarrow w.$$

Иными словами, остовный цикл получается из оригинального путем стягивания неориентированных компонент в одну вершину. В этом случае между неориентированными компонентами $\mathcal{C}$ и вершинами $\mathcal{C}^s$ устанавливается взаимно-однозначное соответствие. Будем говорить, что вершина i цикла $\mathcal{C}^s$ и компонента V_i цикла $\mathcal{C}$ *соответствуют* друг другу.

Пример 2.1. *Рассмотрим смешанно-ориентированный рефлексивный цикл длины четыре, содержащий три ориентированных ребра (см. Рис.*

3, а). Этот граф содержит три неориентированные компоненты $V_0 = \{0, 3\}$, $V_1 = \{1\}$ *и* $V_2 = \{2\}$. *Его остовный цикл содержит три вершины* $0, 1, 2$ *и изображен на Рис. 3, б.*

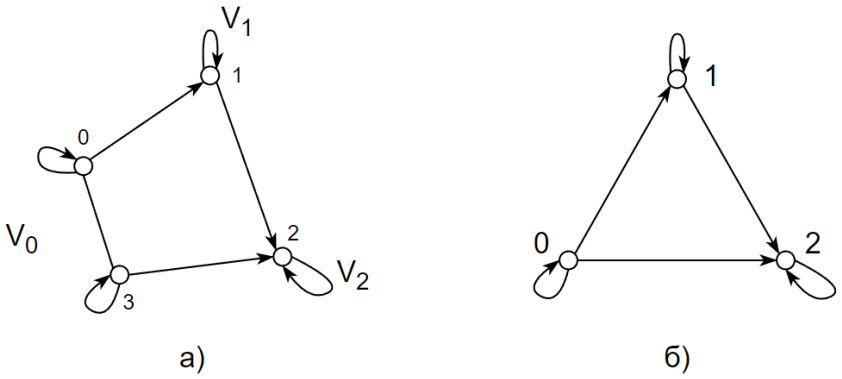


Рис. 3. Смешанно-ориентированный цикл с четырьмя вершинами и его остовной цикл.

2.3. Задача о существовании гомоморфизма графа

Рассмотрим граф $\mathcal{H} = (V, E)$ с множеством вершин V и множеством рёбер E и граф $\mathcal{G} = (V', E')$ с вершинами V' и рёбрами E'. *Гомоморфизм* графа $\mathcal{G}$ на граф $\mathcal{H}$ – это отображение $f : V' \to V$ такое, что для любого ребра $(v, w) \in E'$ верно $(f(v), f(w)) \in E$. Гомоморфизм f из $\mathcal{G}$ на $\mathcal{H}$ будем называть *сюръективным*, если f сюръективно. Для фиксированного графа $\mathcal{H}$ задача о существовании сюръективного гомоморфизма Surj-Hom($\mathcal{H}$) – это массовая задача, в которой по данному графу $\mathcal{G}$ требуется проверить, существует ли сюръективный гомоморфизм из $\mathcal{G}$ на $\mathcal{H}$.

Задача о существовании гомоморфизма является частным случаем более общей задачи удовлетворения ограничениям.

2.4. Задача удовлетворения ограничениям

В задаче об удовлетворении ограничениям *Constraint Satisfaction Problem* (CSP) необходимо определить, существует ли подстановка, удовлетворяющая определённому набору ограничений. Определим эту задачу формально.

Пусть A – конечное множество (*область значений*), Γ – конечное множество отношений на A. *Конъюнктивной формулой на множестве* Γ будем называть формулу, в которую входят отношения из Γ, свободные

переменные из X, конъюнкции и равенства, где X – множество переменных. *Решением конъюнктивной формулы* $\mathcal{I}$ будем называть подстановку $f : X \to A$ в переменные этой формулы, которая выполняет её. Решение f будем называть сюръективным, если $f(X) = A$. Тогда задача удовлетворения ограничениям, также обозначаемая как CSP(Γ) – это задача, в которой по набору переменных $X = \{x_1, \ldots, x_n\}$ и формуле $\mathcal{I}$:

$$\mathcal{I} : R_1(x_{i_{1,1}}, \ldots, x_{i_{1,n_1}}) \wedge \cdots \wedge R_s(x_{i_{s,1}}, \ldots, x_{i_{s,n_s}}),$$

где $R_1, \ldots, R_s$ – отношения из Γ и $i_{1,1}, \ldots, i_{1,n_1}, \ldots, i_{s,1}, \ldots, i_{s,n_s} \in \{1, \ldots, n\}$, необходимо определить, выполнима ли она. Данная формула вместе с набором переменных называется *экземпляром* $\mathcal{I}$ задачи CSP(Γ), а решение $f : X \to A$ формулы $\mathcal{I}$ – *решением* экземпляра.

Функция p от n переменных называется *полиморфизмом* m-местного отношения R, если для любых n наборов $(a_1^1, \ldots, a_1^m), \ldots, (a_n^1, \ldots, a_n^m)$ из R верно, что набор $(p(a_1^1, \ldots, a_n^1), \ldots, p(a_1^m, \ldots, a_n^m))$ тоже из R (говорят также, что p *сохраняет* R). p – полиморфизм множества отношений Γ, если p – полиморфизм каждого отношения из Γ. Множество полиморфизмов Γ обозначается как Pol(Γ). Множество сюръективных полиморфизмов Γ обозначается как SPol(Γ).

Отметим важнейшее свойство полиморфизмов графов, которое следует из их определения. Пусть $\mathcal{H} = (V, E)$ – граф, p – k-местный полиморфизм $\mathcal{H}$ и существует ориентированный путь Π в V^k вида:

$$\Pi = \overline{v_0}\, e_0\, \overline{v_1}\, e_1 \ldots \overline{v_{s-1}}\, e_{s-1}\, \overline{v_s}.$$

Тогда образы элементов этого пути образуют путь π в V:

$$\pi = p(\overline{v_0})\, e_0\, p(\overline{v_1})\, e_1 \ldots p(\overline{v_{s-1}})\, e_{s-1}\, p(\overline{v_s}),$$

ориентации рёбер в котором совпадают с ориентациями рёбер в Π. Иными словами, полиморфизмы $\mathcal{H}$ сохраняют ориентации рёбер в ориентированных путях.

Функция f от n переменных называется *существенно-унарной*, если она существенно зависит не более, чем от одной переменной. Иными словами, f – существенно унарна тогда и только тогда, когда существуют $i \in \{1, \ldots, n\}$ и g – функция от одной переменной такие, что $f(x_1, \ldots, x_i, \ldots, x_n) = g(x_i)$.

Пусть f – n-местная функция, определенная на множестве A. Элементы вида $(a, \ldots, a), a \in A$ будем называть *диагональными*, а множество $\{f(a, \ldots, a) \mid a \in A\}$ – *диагональю функции* f.

В задаче *Surjective Constraint Satisfaction Problem* (SCSP) помимо выполнимости формулы также требуется сюръективность решения. Эта задача определяется так. Пусть Γ – конечное множество отношений на

множестве A. В задаче SCSP(Γ) по набору переменных $X = \{x_1, \ldots, x_n\}$ и данной формуле $\mathcal{I}$:

$$\mathcal{I} : R_1(x_{i_{1,1}}, \ldots, x_{i_{1,n_1}}) \wedge \cdots \wedge R_s(x_{i_{s,1}}, \ldots, x_{i_{s,n_s}}),$$

где $R_1, \ldots, R_s$ – отношения из Γ и $i_{1,1}, \ldots, i_{1,n_1}, \ldots, i_{s,1}, \ldots, i_{s,n_s} \in \{1, \ldots, n\}$, необходимо определить, существует ли решение $f : X \to A$ формулы $\mathcal{I}$ такое, что $f(X) = A$.

Несложно убедиться, что для графа $\mathcal{H}$ задача Surj-Hom($\mathcal{H}$) эквивалентна SCSP(E), где E – бинарное отношение смежности $\mathcal{H}$. В самом деле, пусть множество вершин графа имеет вид $V = \{v_1, \ldots, v_m\}$. Возьмём произвольный граф $\mathcal{G} = (V', E')$, где $V' = \{w_1, \ldots, w_n\}$ и $E' = \{(w_{i_1}, w_{j_1}), (w_{i_2}, w_{j_2}) \ldots, (w_{i_h}, w_{j_h})\}$, где $i_1, j_1, i_2, j_2, \ldots, i_h, j_h \in \{1, \ldots, n\}$. Возьмём множество переменных $X = \{x_1, \ldots, x_n\}$ и построим конъюнктивную формулу $\mathcal{I}$ над E следующего вида:

$$\mathcal{I} : E(x_{i_1}, x_{j_1}) \wedge E(x_{i_2}, x_{j_2}) \wedge \cdots \wedge E(x_{i_h}, x_{j_h}).$$

В получившейся формуле каждой переменной x_s из X соответствует вершина w_s из $\mathcal{G}$, а каждому вхождению ограничения $E(x_{i_l}, x_{j_l})$ в $\mathcal{I}$ – ребро (w_{i_l}, w_{j_l}). Тогда каждый гомоморфизм $f : V' \to V$ задаёт подстановку $g : X \to V$: для $i \in \{1, \ldots, n\}$ положим $g(x_i) = f(w_i)$. При этом по построению полученная подстановка будет решением $\mathcal{I}$, и если гомоморфизм f сюръективен, то и g сюръективна. Аналогично, каждое сюръективное решение $g : X \to V$ формулы $\mathcal{I}$ задаёт отображение $f : V' \to V$, которое также будет являться сюръективным гомоморфизмом из $\mathcal{G}$ на $\mathcal{H}$. Иными словами, полученный экземпляр Surj-Hom($\mathcal{H}$) имеет решение тогда и только тогда, когда построенный экземпляр SCSP(E) имеет решение.

В отличие от задачи Surj-Hom($\mathcal{H}$), задача удовлетворения ограничениям имеет классификацию сложности, которая была впервые сформулирована в [11], и позже независимо доказана в [12, 13]. Эта классификация формулируется в терминах полиморфизмов: если для набора отношений Γ на множестве A существует полиморфизм $p \in \mathrm{Pol}(\Gamma)$ такой, что $\forall x, y \in A : p(y, x, \ldots, x) = p(x, y, x, \ldots, x) = \cdots = p(x, x, \ldots, x, y)$, то CSP($\Gamma$) решается за полиномиальное время; иначе, она является NP-полной. Также известно, что если Γ – набор отношений на множестве A, то SCSP(Γ) можно свести к CSP($\Gamma \cup \{x = d \mid d \in A\}$)[5]. Более того, если все полиморфизмы Γ являются существенно-унарными, то SCSP(Γ) является NP-полной [14]. Несмотря на это, было показано, что сложность SCSP нельзя определить только с помощью полиморфизмов [15].

2.5. Свойство наследования сюръективности

В дальнейшем изложении мы будем активно пользоваться свойством, которое для набора отношений Γ позволяет связать сложность задачи SCSP(Γ) со структурой его полиморфизмов. Пусть Γ – конечное множество отношений на множестве A, $|A| > 1$. Сперва сформулируем два вспомогательных свойства. Пусть $k \geq 1$, $p_1, \ldots, p_k$ – набор трёхместных полиморфизмов Γ.

- Набор $p_1, \ldots, p_k$ является *совместно сюръективным*, если для каждого элемента $a \in A$ существуют $i \in \{1, \ldots, k\}, b_1, b_2, b_3 \in A$ такие, что $p_i(b_1, b_2, b_3) = a$.

- Набор $p_1, \ldots, p_k$ *диагонально согласован*, если для всех $i, j \in \{1, \ldots, k\}$ и всех $a \in A$ верно $p_i(a, a, a) = p_j(a, a, a)$.

Будем говорить, что *полиморфизмы* Γ *наследуют сюръективность*, если в каждом совместно сюръективном диагонально согласованном наборе трёхместных полиморфизмов Γ найдётся сюръективная функция. Иными словами, это свойство требует, чтобы в каждом наборе трёхместных полиморфизмов Γ, который принимает в совокупности все значения из A и совпадает на диагонали, найдётся сюръективная функция.

Сформулируем ещё одно вспомогательное свойство. Пусть R – двуместное отношение из Γ, $p_1, \ldots, p_k$ – набор его трёхместных полиморфизмов. Будем говорить, что набор $p_1, \ldots, p_k$ *имитирует проекции*, если для любой пары $i_1, i_2 \in \{1, \ldots, k\}, i_1 \neq i_2$, выполняется ограничение $R(p_{i_1}(\overline{a}), p_{i_2}(\overline{b}))$, где $\overline{a}, \overline{b}$ – векторы из A^3 такие, что для всех $h, l \in \{1, 2, 3\}$ верно $R(a^h, b^l)$. В общем случае это свойство формулируется следующим образом: набор трёхместных полиморфимзов Γ *имитирует проекции*, если для каждых:

- n-местного отношения $R \in \Gamma$,

- отображения $h : \{1, \ldots, n\} \to \{1, \ldots, k\}$ и

- набора векторов $\overline{a_1}, \ldots, \overline{a_n} \in A^3$ такого, что для всех $j_1, \ldots, j_k \in \{1, 2, 3\}$ верно $R(a_1^{j_{h(1)}}, a_2^{j_{h(2)}}, \ldots, a_n^{j_{h(n)}})$,

выполняется $R(p_{h(1)}(\overline{a_1}), p_{h(2)}(\overline{a_2}), \ldots, p_{h(n)}(\overline{a_n}))$.

Будем говорить, что *полиморфизмы* Γ *наследуют сюръективность в слабой форме*, если в каждом совместно сюръективном диагонально согласованном наборе его трёхместных полиморфизмов, который имитирует проекции, найдётся сюръективная функция. Верно, что если полиморфизмы Γ наследуют сюръективность, то они наследуют сюръективность в слабой форме. Выполняется следующая теорема:

Теорема 3. *[16] Пусть* Γ *– конечное множество отношений на множестве* A, $|A| > 1$. *Пусть выполняются следующие условия:*

- *Все сюръективные полиморфизмы* Γ *являются существенно унарными.*
- *Полиморфизмы* Γ *наследуют сюръективность в слабой форме.*

Тогда задача SCSP(Γ) *является* NP*-полной.*

Свойство наследования сюръективности было впервые описано в [10] и позже обобщено до слабой формы в [16]. В этих статьях свойство используется для анализа сложности задачи Surj-Hom в классах неориентированных и строго ориентированных рефлексивных циклов. Данное свойство представляет интерес, поскольку позволяет анализировать сложность SCSP(Γ) исключительно посредством полиморфизмов Γ. При этом существуют отношения, полиморфизмы которых наследуют сюръективность, но для которых задача SCSP лежит в P. К ним относится, например, отношение смежности цикла, изображённого на Рис. 3, б [16]. Тем не менее, свойство наследования сюръективности остается крайне сильным инструментом для анализа сложности SCSP для целых классов отношений.

3. Сложность задачи для смешанно ориентированных графов

В данном разделе рассматривается сложность задачи Surj-Hom для смешанно-ориентированных рефлексивных циклов. Раздел 3.1 посвящен циклам $\mathcal{C}$, для остовных циклов $\mathcal{C}^s$ которых задача Surj-Hom($\mathcal{C}^s$) является NP-трудной. В разделе 3.2 рассматриваются циклы $\mathcal{C}$, у которых нет остовных циклов или для которых не определена NP-трудность Surj-Hom($\mathcal{C}^s$). Наконец, в разделе 3.3 рассматриваются циклы $\mathcal{C}_4$, $\mathcal{C}_5$ и $\mathcal{C}_6$ (см. Рис. 1, а, б, в), для них обосновывается, почему к ним не применимы схемы определения сложности Surj-Hom, которые используются для остальных циклов в данной статье.

3.1. Случай большого числа ориентированных рёбер

Рассмотрим n-местное отношение R на множестве A, $|A| > 1$. Пусть S – n-местное отношение на множестве $B, |B| > 1$. Будем говорить, что R *сюръективно интерпретирует* S, если существуют отображения $\varphi : A \to B$, $\psi : B \to A$ и $n + k$-местное отношение Q, задающееся конъюнктивной формулой над R, для которых выполняются следующие условия:

1) $\forall b \in B : \varphi(\psi(b)) = b$.

2) $\forall (b_1, \ldots, b_n) \in S \; \exists c_1, \ldots, c_k \in A$:

$$(\psi(b_1), \ldots, \psi(b_n), c_1, \ldots, c_k) \in Q$$

и $\{\psi(b_1), \ldots, \psi(b_n), c_1, \ldots, c_k\} = \varphi^{-1}(\{b_1, \ldots, b_n\})$.

3) $\forall (a_1, \ldots, a_n, c_1, \ldots, c_k) \in Q$:

$$(\varphi(a_1), \ldots, \varphi(a_n)) \in S,$$

и $\varphi(\{a_1, \ldots, a_n\}) = \varphi(\{a_1, \ldots, a_n, c_1, \ldots, c_k\})$.

Иными словами, первые n переменных отношения Q моделируют отношение S. Согласно первому условию, отображение φ разбивает A на блоки, соответствующие разным элементам B, а отображение ψ сопоставляет каждому элементу B элемент A из его блока. Согласно второму условию, образы любого набора $(b_1, \ldots, b_n)$ из S можно дополнить элементами A до набора из Q, причём выбранные элементы будут полностью покрывать блоки, соответствующие $b_1, \ldots, b_n$. Наконец, третье условие гласит, что для любого набора из Q образы первых n членов этого набора образуют набор из S, а последние k членов лежат в тех же блоках, что и первые n.

Пример 3.1. *Рассмотрим смешанно-ориентированный рефлексивный цикл $\mathcal{C}$ длины 4 и его остовный цикл $\mathcal{C}^s$ из Примера 2.1 (см. Рис. 3). Рассмотрим отображения $\varphi : Z_4 \to Z_3$ и $\psi : Z_3 \to Z_4$:*

$$\varphi(x) = i, \text{ если } x \in V_i$$
$$\psi(x) = x$$

и 6-местное отношение Q задающееся формулой (см. Рис. 4):

$$Q(x_1, x_2, y_1, y_2, y_3, y_4) = \mathcal{C}(x_1, y_1) \wedge \mathcal{C}(y_1, x_1) \wedge \mathcal{C}(y_1, y_2) \wedge \mathcal{C}(y_2, y_1) \wedge$$
$$\wedge \mathcal{C}(y_2, y_3) \wedge \mathcal{C}(y_3, y_4) \wedge \mathcal{C}(y_4, y_3) \wedge \mathcal{C}(y_4, x_2) \wedge \mathcal{C}(x_2, y_4).$$

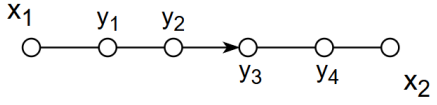


Рис. 4.

Покажем, что эти отображения вместе с формулой удовлетворяют условиям из определения сюръективного интерпретирования. Несложно

проверить, что $\varphi(\psi(x)) = x$. *Теперь, возьмём ребро* $(b_1, b_2) \in \mathcal{C}^s$. *В зависимости от значений* b_1 *и* b_2 *выберем элементы* c_1, c_2 *(см. Табл. 1),* c_3 *и* c_4 *(см. Табл. 2):*

b_1	b_2	c_1	c_2
$\neq 0$	$\neq 1$	b_1	b_1
$\neq 0$	1	b_1	b_1
0	$\neq 1$	3	3
0	1	3	0

Табл. 1:

b_1	b_2	c_3	c_4
$\neq 1$	$\neq 0$	b_2	b_2
1	$\neq 0$	b_2	b_2
$\neq 1$	0	3	3
1	0	0	3

Табл. 2:

Тогда набор $(\psi(b_1), \psi(b_2), c_1, c_2, c_3, c_4)$ *будет удовлетворять* Q *и* $\{\psi(b_1), \psi(b_2), c_1, c_2, c_3, c_4\} = \varphi^{-1}(\{b_1, b_2\})$, *то есть второе условие будет выполняться. Наконец, возьмём произвольный набор* $(a_1, a_2, c_1, c_2, c_3, c_4) \in Q$. *Нетрудно заметить, что* c_1, c_2 *лежат в одной неориентированной компоненте с* a_1, *а* c_3, c_4 *в одной компоненте с* a_2. *Значит,* $\varphi(\{a_1, a_2\}) = \varphi(\{a_1, a_2, c_1, c_2, c_3, c_4\})$. *А так как выполняется* $\mathcal{C}(c_2, c_3)$, *то верно* $(\varphi(a_1), \varphi(a_2)) \in \mathcal{C}^s$. *Значит третье свойство тоже выполняется и* $\mathcal{C}$ *сюръективно интерпретирует* $\mathcal{C}^s$.

Теорема 4. *Пусть* R *–* n*-местное отношение на множестве* A. *Пусть* S *–* n*-местное отношение на множестве* B *такое, что* R *сюръективно интерпретирует* S. *Тогда* SCSP(S) *полиномиально сводится к* SCSP(R).

Доказательство. Рассмотрим отображения $\varphi : A \to B$, $\psi : B \to A$ и $n + k$-местное отношение Q, задающееся конъюнктивной формулой над R, которые удовлетворяют условиям из определения сюръективного интерпретирования. Возьмём экземпляр задачи SCSP(S), задающийся набором переменных $X = \{x_1, \ldots, x_l\}$ и формулой $\mathcal{I}$:

$$\mathcal{I} : S(x_{i_{1,1}}, \ldots, x_{i_{1,n}}) \wedge \cdots \wedge S(x_{i_{h,1}}, \ldots, x_{i_{h,n}}),$$

где $i_{j,s} \in \{1, \ldots, l\}$, $j \in \{1, \ldots, h\}$, $s \in \{1, \ldots, n\}$. Построим формулу $\mathcal{J}$ над Q с множеством переменных $X' = \{x_1, \ldots, x_l, y_{1,1}, \ldots, y_{1,k}, \ldots, y_{h,1}, \ldots, y_{h,k}\}$:

$$\mathcal{J} : Q(x_{i_{1,1}}, \ldots, x_{i_{1,n}}, y_{1,1}, \ldots, y_{1,k}) \wedge \cdots \wedge Q(x_{i_{h,1}}, \ldots, x_{i_{h,n}}, y_{h,1}, \ldots, y_{h,k}).$$

Покажем, что $\mathcal{J}$ имеет сюръективное решение тогда и только тогда, когда $\mathcal{I}$ имеет сюръективное решение.

Пусть $\mathcal{I}$ обладает сюръективным решением $f : X \to B$. Построим подстановку $g : X' \to A$ в переменные $\mathcal{J}$ следующим образом:

- Для x_i положим $g(x_i) = \psi(f(x_i))$.

- Для $y_{j,l}$ рассмотрим j-ое вхождение Q в $\mathcal{J}$:

$$Q(x_{i_{j,1}}, \ldots, x_{i_{j,n}}, y_{j,1}, \ldots, y_{j,k}).$$

Так как f – решение $\mathcal{I}$, то $(f(x_{i_{j,1}}), \ldots, f(x_{i_{j,n}})) \in S$. Тогда по второму условию сюръективного интерпретирования существуют $c_{j,1}, \ldots, c_{j,k} \in A$ такие, что верно

$$Q(\psi(f(x_{i_{j,1}})), \ldots, \psi(f(x_{i_{j,n}})), c_{j,1}, \ldots, c_{j,k})$$

и

$$\{\psi(f(x_{i_{j,1}})), \ldots, \psi(f(x_{i_{j,n}})), c_{j,1}, \ldots, c_{j,k}\} = \varphi^{-1}(\{f(x_{i_{j,1}}), \ldots, f(x_{i_{j,n}})\}).$$

Положим $g(y_{j,l}) = c_{j,l}$.

Несложно заметить, что по построению g будет решением $\mathcal{J}$. Покажем, что оно также будет сюръективным. Пусть какое-то значение $a \in A$ не принимается на g. Рассмотрим $\varphi(a) = b$. Поскольку f сюръективно, то для некоторых $j \in \{1, \ldots, h\}, s \in \{1, \ldots, n\}$ верно $f(x_{i_{j,s}}) = b$. Но по построению g имеем $a \in \varphi^{-1}(b) \subseteq \{g(x_{i_{j,1}}), \ldots, g(x_{i_{j,n}}), g(y_{j,1}), \ldots, g(y_{j,k})\}$ – противоречие. Значит, g – это сюръективное решение.

Теперь, пусть у $\mathcal{J}$ существует сюръективное решение $g : X' \to A$. Построим подстановку $f : X \to B$ следующим образом: для $x_{i_{j,s}}$ положим $f(x_{i_{j,s}}) = \varphi(g(x_{i_{j,s}}))$. Несложно заметить, что полученная подстановка будет решением, поскольку по третьему условию сюръективного интерпретирования из выполнения

$$Q(g(x_{i_{j,1}}), \ldots, g(x_{i_{j,n}}), g(y_{j,1}), \ldots, g(y_{j,k}))$$

следует

$$(\varphi(g(x_{i_{j,1}})), \ldots, \varphi(g(x_{i_{j,n}}))) \in S.$$

Покажем, что это решение будет сюръективным. Пусть f не принимает значение $b \in B$. Рассмотрим $a = \psi(b)$. Из первого условия сюръективного интерпретирования следует $\varphi(a) = b$. Поскольку g сюръективно, то a принимается на некотором $y_{j,l}$ (если a принимается на одном из $x_{i_{j,s}}$, то по построению f имеем $f(x_{i_{j,s}}) = \varphi(a) = b$). Но тогда по третьему свойству сюръективного интерпретирования получается, что

$$b \in \varphi(\{g(x_{i_{j,1}}), \ldots, g(x_{i_{j,n}}), g(y_{j,1}), \ldots, g(y_{j,k})\}) = f(\{x_{i_{j,1}}, \ldots, x_{i_{j,n}}\}).$$

Получили противоречие. Значит, f – сюръективное решение.

Итак, по экземпляру задачи SCSP(S) из множества переменных X и формулы $\mathcal{I}$ мы за полиномиальное время построили экземпляр SCSP(R) из множества переменных X' и формулы $\mathcal{J}$ такой, что $\mathcal{I}$ имеет сюръективное решение тогда и только тогда, когда $\mathcal{J}$ имеет сюръективное решение. Это полиномиально сводит SCSP(S) к SCSP(R). □

Заметим, что из Теоремы 4 не следует, что если отношение R сюръективно интерпретирует отношение S, то задачи SCSP(R) и SCSP(S) эквивалентны. Так, отношение смежности цикла $\mathcal{C}$ длины 4 из Примера 3.1 сюръективно интерпретирует отношение смежности его остовного цикла $\mathcal{C}^s$. При этом Surj-Hom($\mathcal{C}^s$) лежит в P [9], но, как будет показано в разделе 3.2.4, Surj-Hom($\mathcal{C}$) является NP-трудной.

Следствие 1. *Пусть $\mathcal{C}$ – смешанно ориентированный рефлексивный цикл, содержащий n вершин и m неориентированных компонент, $m \geq 3$. Пусть $\mathcal{C}^s$ – остовный цикл для $\mathcal{C}$. Если задача* Surj-Hom($\mathcal{C}^s$) *является* NP*-трудной, то и задача* Surj-Hom($\mathcal{C}$) *является* NP*-трудной.*

Доказательство. Покажем, что отношение смежности $\mathcal{C}$ сюръективно интерпретирует $\mathcal{C}^s$. Так как в $\mathcal{C}$ содержится m неориентированных компонент, то $\mathcal{C}^s$ содержит m вершин. Обозначим неориентированные компоненты $\mathcal{C}$ как $V_0, \ldots, V_{m-1}$, обходя цикл по часовой стрелке, начиная с произвольной вершины. Обозначим вершины $\mathcal{C}^s$ как $\{0, \ldots, m-1\}$ так, чтобы каждой вершине $i \in Z_m$ соответствовала компонента V_i. Выберем в каждой компоненте V_i произвольную вершину v_i. Пусть неориентированные компоненты $\mathcal{C}$ содержат не больше k вершин. Рассмотрим отображения $\varphi : Z_n \to Z_m$ и $\psi : Z_m \to Z_n$, которые задаются следующим образом:

$$\varphi(v) = i, \text{если } v \in V_i,$$
$$\psi(i) = v_i$$

и $4k+2$-местное отношение Q, задающееся конъюнктивной формулой над $\mathcal{C}$ (см. Рис. 5):

$$\begin{aligned} Q(x_1, x_2, y_1, \ldots, y_{4k}) = \mathcal{C}(x_1, y_1) \wedge \mathcal{C}(y_1, x_1) \wedge \mathcal{C}(y_2, y_1) \wedge \mathcal{C}(y_1, y_2) \wedge \cdots \wedge \\ \wedge \mathcal{C}(y_{2k-1}, y_{2k}) \wedge \mathcal{C}(y_{2k}, y_{2k-1}) \wedge \mathcal{C}(y_{2k}, y_{2k+1}) \wedge \\ \wedge \mathcal{C}(y_{2k+1}, y_{2k+2}) \wedge \mathcal{C}(y_{2k+2}, y_{2k+1}) \wedge \cdots \wedge \\ \wedge \mathcal{C}(y_{4k-1}, y_{4k}) \wedge \mathcal{C}(y_{4k}, y_{4k-1}) \wedge \mathcal{C}(y_{4k}, x_2) \wedge \mathcal{C}(x_2, y_{4k}). \end{aligned}$$

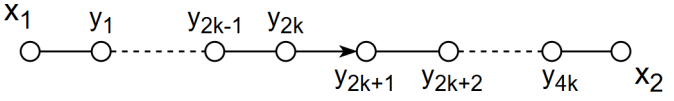


Рис. 5.

Проверим, что полученные отображения и формула удовлетворяют условиям из определения сюръективной интерпретации. Так как для каждой вершины $i \in Z_m$ верно $v_i \in V_i$, то $\varphi(\psi(x)) = x$, и первое условие выполняется. По построению в любом наборе $(a_1, a_2, c_1, \ldots, c_{4k}) \in Q$

элементы $c_1, \ldots, c_{2k}$ лежат в одной неориентированной компоненте с a_1, а $c_{2k+1}, \ldots, c_{4k}$ – в одной компоненте с a_2. При этом, так как формула Q содержит ограничение $\mathcal{C}(y_{2k}, y_{2k+1})$, то для любых $(b_1, b_2) \in \mathcal{C}^s$ можно выбрать $c_1, \ldots, c_{4k} \in Z_n$ такие, что $(\psi(b_1), \psi(b_2), c_1, \ldots, c_{4k}) \in Q$ и $\{\psi(b_1), c_1, \ldots, c_{2k}\} = \varphi^{-1}(\{b_1\})$, $\{\psi(b_2), c_{2k+1}, \ldots, c_{4k}\} = \varphi^{-1}(\{b_2\})$, откуда второе условие выполняется.

Теперь, возьмём произвольный набор $(a_1, a_2, c_1, \ldots, c_{4k}) \in Q$. Поскольку $c_1, \ldots, c_{2k}$ лежат в одной неориентированной компоненте с a_1, а $c_{2k+1}, \ldots, c_{4k}$ – с a_2, то $\{\varphi(a_1)\} = \varphi(\{a_1, c_1, \ldots, c_{2k}\})$ и $\{\varphi(a_2)\} = \varphi(\{a_2, c_{2k+1}, \ldots, c_{4k}\})$, а так как $(c_{2k}, c_{2k+1}) \in \mathcal{C}$, то $(\varphi(a_1), \varphi(a_2)) \in \mathcal{C}^s$. Значит, третье свойство также выполняется, и $\mathcal{C}$ сюръективно интерпретирует $\mathcal{C}^s$. Тогда, Surj-Hom($\mathcal{C}^s$) полиномиально сводится к Surj-Hom($\mathcal{C}$). А так как Surj-Hom($\mathcal{C}^s$) является NP-трудной, то и Surj-Hom($\mathcal{C}$) также является NP-трудной. □

3.2. Случай циклов с маленьким числом ориентированных рёбер

В этом разделе мы рассмотрим смешанно-ориентированные рефлексивные циклы, для которых мы ещё не определили сложность задачи Surj-Hom. Это циклы $\mathcal{C}$, к которым не применимо Следствие 1. К ним относятся циклы, у которых не существует остовных циклов $\mathcal{C}^s$, для которых задача Surj-Hom($\mathcal{C}^s$) решается за полиномиальное время или для которых сложность Surj-Hom($\mathcal{C}^s$) не известна. Будем рассматривать такие графы относительно взаимного расположения в них ориентированных рёбер (см. Рис. 6, пунктиром обозначены последовательности из нескольких неориентированных рёбер):

- Будем говорить, что смешанно-ориентированный цикл *имеет тип* A, если он содержит ровно одно ориентированное ребро.
- Цикл *имеет тип* B, если он содержит два ориентированных ребра, две неориентированные компоненты V_0, V_1 и одно ориентированное ребро выходит из V_0 в V_1, а одно – из V_1 в V_0.
- Цикл *имеет тип* C, если он содержит два ориентированных ребра, две неориентированные компоненты V_0, V_1 и все ориентированные рёбра выходят из V_0 в V_1.
- Цикл *имеет тип* D, если он содержит три ориентированных ребра, три неориентированные компоненты V_0, V_1, V_2 и ориентированные рёбра выходят из V_0 в V_1, из V_1 в V_2 и из V_0 в V_2.

- Цикл *имеет тип* E, если он содержит шесть ориентированных рёбер, шесть неориентированных компонент $V_0, V_1, \ldots, V_5$ и ориентированные рёбра выходят из V_1 в V_0 и V_2, из V_3 в V_2 и V_4 и из V_5 в V_4 и V_0.

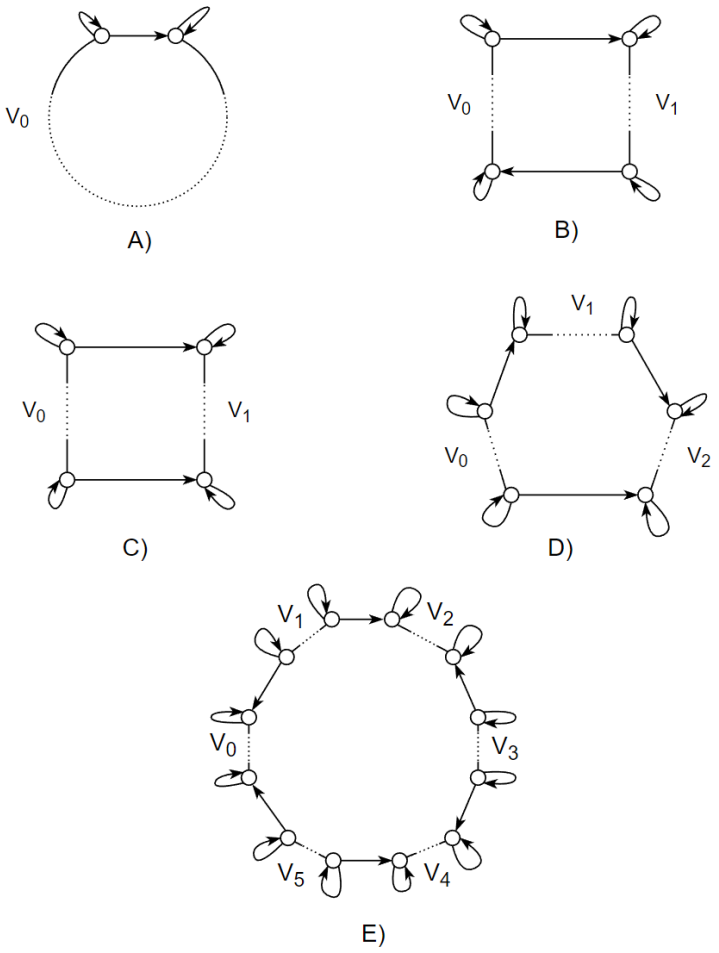


Рис. 6. Типы циклов, к которым не применимо Следствие 1.

Перед исследованием сложности задачи Surj-Hom для данных циклов мы сформулируем два важных утверждения о смешанно-ориентированных рефлексивных циклах и их полиморфизмах. Первое утверждение естественным образом следует из определения полиморфизмов графов.

Утверждение 1. *Пусть* $\mathcal{C}$ *– смешанно-ориентированный рефлексивный цикл,* $p \in \mathrm{Pol}(\mathcal{C})$*. Пусть* $\overline{a}$ *и* $\overline{b}$ *– векторы, которые лежат в одной*

неориентированной компоненте. Тогда $p(\overline{a})$ *и* $p(\overline{b})$ *тоже лежат в одной неориентированной компоненте.*

Второе утверждение является частным случаем Теоремы 1.1 из [17]:

Утверждение 2. *Пусть* $\mathcal{C}$ *– смешанно-ориентированный рефлексивный цикл, содержащий больше трёх вершин. Тогда сюръективные полиморфизмы* $\mathcal{C}$ *являются существенно унарными.*

Для определения сложности задачи Surj-Hom мы будем использовать свойство наследования сюръективности, сформулированное в разделе 2.5. Как мы показали в Утверждении 2, сюръективные полиморфизмы всех циклов, которые мы будем рассматривать в этом разделе, являются существенно-унарными. Значит, для доказательства NP-трудности задачи с помощью Теоремы 3 нам всего лишь достаточно доказать, что полиморфизмы циклов наследуют сюръективность.

3.2.1. Циклы типа A

Рассмотрим цикл типа A. Этот граф состоит из единственной неориентированной компоненты и содержит одно ориентированное ребро.

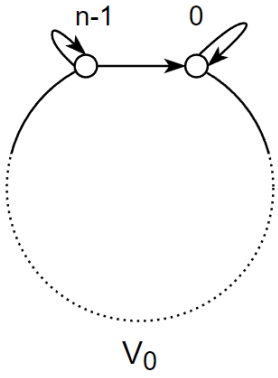


Рис. 7. Цикл типа A

Доказательство того, что полиморфизмы циклов типа A наследуют сюръективность, во многом схоже с аналогичным доказательством для неориентированных циклов из [10]. Тем не менее, наличие ориентированного ребра в циклах, которые мы рассматриваем, не позволяет нам полностью скопировать доказательство. В связи с этим, мы сформулируем следующее утверждение, опираясь на Леммы 1, 2, 10 из [10], а то, что полиморфизмы циклов наследуют сюръективность, мы докажем независимо.

Утверждение 3. *Пусть $\mathcal{C}$ – смешанно-ориентированный рефлексивный цикл типа A, содержащий n вершин, $n > 3$. Тогда:*

1) *Для любого вектора $\overline{a} \in Z_n^3$ существует вершина $r \in Z_n$ такая, что существует ориентированный путь вида $\overline{a} \to \cdots \to (r,r,r)$ или $(r,r,r) \to \cdots \to \overline{a}$ длины $l \le c, c = \lceil \frac{\lfloor \frac{2n}{3} \rfloor}{2} \rceil$.*

2) *Пусть p – полиморфизм $\mathcal{C}$, который принимает значения $s, s+1 \pmod n, \ldots, s + \lfloor \frac{n}{2} \rfloor + 1 \pmod n$ и существует вершина r такая, что $p(r,\ldots,r) = s$. Тогда p сюръективен.*

Доказательство. Обозначим неориентированный рефлексивный цикл длины n как $\mathcal{C}'$.

1. Пронумеруем вершины $\mathcal{C}$, обходя цикл по часовой стрелке так, чтобы единственное ориентированное ребро имело вид $n-1 \to 0$ (см. Рис. 7). Согласно Лемме 10 из [10], можно выбрать вершину $r \in Z_n$ такую, что в $\mathcal{C}'$ существуют пути из неориентированных рёбер от r до a_1, a_2 и a_3 длины меньшей или равной c. Пусть $r \in \{0,1,\ldots,c\}$. В этом случае в цикле $\mathcal{C}$ существуют аналогичные ориентированные пути, в которых все рёбра ориентированы как $\leftarrow$. Теперь, пусть $r \in \{c+1, c+2, \ldots, n-1\}$. В этом случае в $\mathcal{C}$ есть пути из r в компоненты $\overline{a}$ длины не более c, в которых все рёбра имеют ориентацию $\to$. Значит, в $\mathcal{C}$ есть ориентированный путь вида $(r,r,r) \to \cdots \to \overline{a}$ или $(r,r,r) \leftarrow \cdots \leftarrow \overline{a}$ длины $l \le c$.

2. Возьмём вектор $\overline{a}$ такой, что $p(\overline{a}) = s + \lfloor \frac{n}{2} \rfloor + 1 \pmod n$. Обозначим $\overline{r} = (r,r,r)$. Согласно Леммам 1, 2 из [10] в $\mathcal{C}'$ существует путь между $\overline{a}$ и $\overline{r}$ из неориентированных рёбер длины $l \le \lfloor \frac{n}{2} \rfloor$. Значит, если $r \in \{0,1,\ldots,\lfloor \frac{n}{2} \rfloor\}$, то в $\mathcal{C}$ есть ориентированный путь вида $\overline{r} \leftarrow \cdots \leftarrow \overline{a}$ длины l. А если $r \in \{\lfloor \frac{n}{2} \rfloor + 1, \lfloor \frac{n}{2} \rfloor + 2, \ldots, n-1\}$, то в $\mathcal{C}$ есть путь вида $\overline{r} \to \cdots \to \overline{a}$ длины l. Образы элементов этого пути проходят через все значения $s + \lfloor \frac{n}{2} \rfloor + 1 \pmod n, s + \lfloor \frac{n}{2} \rfloor + 2 \pmod n, \ldots, s$, поскольку иначе этот путь проходил бы через все $s, s+1 \pmod n, \ldots, s + \lfloor \frac{n}{2} \rfloor + 1 \pmod n$ и содержал как минимум $\lfloor \frac{n}{2} \rfloor + 1$ элементов. Значит, p сюръективен.

□

Утверждение 4. *Пусть $\mathcal{C}$ – смешанно-ориентированный рефлексивный цикл типа A, содержащий больше шести вершин. Тогда полиморфизмы $\mathcal{C}$ наследуют сюръективность.*

Доказательство. Пусть $\mathcal{C}$ содержит n вершин, $n > 6$. Без ограничения общности пронумеруем вершины этого графа так, чтобы единственное ориентированное ребро имело вид $n-1 \to 0$. Рассмотрим набор трёхместных полиморфизмов $p_1, \ldots, p_k$, который является совместно сюръективным и диагонально согласованным. Покажем, что среди них найдётся

сюръективная функция. Возьмём $i, j \in \{1, \dots, k\}$, векторы $\overline{a}, \overline{b} \in Z_n^3$ такие, что $p_i(\overline{a}) = n-1$, $p_j(\overline{b}) = 0$.

По Утверждению 3.1 можно выбрать элемент $r \in Z_n$ такой, что существует ориентированный путь вида $\overline{b} \to \dots \to (r,r,r)$ или $(r,r,r) \to \dots \to \overline{b}$ длины $l \le c = \lceil \frac{\lfloor \frac{2n}{3} \rfloor}{2} \rceil$. Это значит, что $p_j(r,r,r) \in \{0,1,\dots,c\}$ или $p_j(r,r,r) \in \{n-c, n-c+1, \dots, n-1\}$. Рассмотрим первый случай. Существует путь П из вектора (r,r,r) в $\overline{a}$ из неориентированных рёбер. Поскольку полиморфизмы из набора совпадают на диагонали, то $p_i(r,r,r) \in \{0,\dots,c\}$, откуда образы элементов П принимают на p_i все значения из $\{c, c+1, \dots, n-1\}$. Значит, p_i принимает как минимум $n-c$ значений. Аналогично, пусть $p_j(r,r,r) \in \{n-c, n-c+1, \dots, n-1\}$. Рассмотрим произвольный путь между (r,r,r) и $\overline{b}$ из неориентированных рёбер. Образы его элементов принимают на p_j все $n-c+1$ значений из $\{0,1,\dots,n-c\}$. Если $n \ne 8$, то из $n > 6$ имеем $n - c > \lfloor \frac{n}{2} \rfloor + 1$, откуда по Утверждению 3.2 одна из функций p_i и p_j сюръективна.

Теперь, пусть $n = 8$. В этом случае $c = 3$. Пусть существует $s \in Z_8$ такое, что функции из набора принимают на (s,s,s) значение из $\{0,1,2\}$. Рассмотрим произвольный путь вида $(s,s,s) \leftrightarrow \dots \leftrightarrow \overline{a}$. Поскольку каждое ребро этого пути ориентировано как $\leftrightarrow$ и $p_i(\overline{a}) = 7$, то образы элементов этого пути принимают на p_i все значения из $\{2,3,\dots,7\}$ – всего 6 значений, откуда по Утверждению 3.2 функция p_i сюръективна. Аналогично, если существует s такое, что $p_1, \dots, p_k$ принимают на (s,s,s) значение из $\{5,6,7\}$, то p_j – сюръективна.

Тогда, пусть функции из набора принимают на диагонали только значения из $\{3,4\}$. Заметим, что для произвольных вектора $\overline{v} \in Z_8^3$ и элемента $s \in Z_8$ кратчайший ориентированный путь из (s,s,s) в $\overline{v}$ имеет длину 4 тогда и только тогда, когда $s \in \{v^1 + 4 \pmod 8, v^2 + 4 \pmod 8, v^3 + 4 \pmod 8\}$ (в противном случае длина этого пути будет не больше трёх). Обозначим через d количество диагональных элементов, которые $p_1, \dots, p_k$ отображают в 3, тогда в 4 отображаются $8-d$ элементов. Возьмём произвольный элемент $r \in Z_8$ такой, что $p_i(r,r,r) = 3$. При этом, так как $p_i(\overline{a}) = 7$, то любой ориентированный путь из (r,r,r) в $\overline{a}$ должен иметь длину не меньше 4. Это значит, что $d \le 3$. Аналогично рассматривая диагональные элементы, которые отображаются в 4 и $p_j(\overline{b})$, получаем $8-d \le 3$, то есть $d \ge 5$ – противоречие. Значит, на диагонали принимаются не только $\{3,4\}$ и в наборе найдётся сюръективная функция.

□

Из Утверждений 2, 4 и Теоремы 3 следует:

Утверждение 5. *Пусть $\mathcal{C}$ – смешанно-ориентированный рефлексивный цикл типа A, содержащий больше шести вершин. Тогда* Surj-Hom($\mathcal{C}$) *является* NP*-трудной.*

Циклы типа A длины $4, 5, 6$ более детально рассматриваются в разделе 3.3.

3.2.2. Циклы типа B

Рассмотрим цикл $\mathcal{C}$ типа B. Обозначим множество вершин $\mathcal{C}$ как V. Обозначим неориентированные компоненты цикла как V_0 и V_1 (см. Рис. 8).

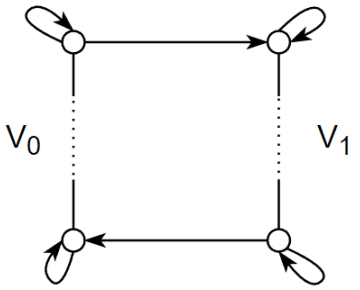


Рис. 8. Цикл типа B

Утверждение 6. *Пусть $\mathcal{C}$ – смешанно-ориентированный рефлексивный цикл типа B. Тогда полиморфизмы $\mathcal{C}$ наследуют сюръективность.*

Доказательство. Рассмотрим набор полиморфизмов $\mathcal{C}$ $p_1, \ldots, p_k$, которые удовлетворяют условиям наследования сюръективности. Покажем, что среди них найдётся сюръективная функция.

Пусть существует $a \in V$ такое, что полиморфизмы из набора принимают на векторе $\overline{a} = (a, a, a)$ значение $v \in V_0$ (случай $v \in V_1$ разбирается аналогично). Возьмём $i \in \{1, \ldots, k\}$, вектор $\overline{b} \in V^3$ такие, что $p_i(\overline{b}) \in V_1$. Существует ориентированный путь вида:

$$\overline{a} \to \cdots \to \overline{b} \to \cdots \to \overline{a}.$$

Поскольку на этом пути первый и последний элементы совпадают, все рёбра ориентированы как $\to$ и полиморфизм p_i принимает на нём значения и из V_0, и из V_1, то по определению полиморфизма образы элементов этого пути на p_i проходят через все значения из V, откуда p_i – сюръективная функция. □

Таким образом, с помощью Утверждений 2, 6 и Теоремы 3 мы доказали:

Утверждение 7. *Пусть $\mathcal{C}$ – смешанно-ориентированный рефлексивный цикл типа B, содержащий больше трёх вершин. Тогда* Surj-Hom($\mathcal{C}$) *является* NP*-трудной.*

3.2.3. Циклы типа C

Рассмотрим смешанно-ориентированный рефлексивный цикл типа C. Обозначим множество его вершин как V, $|V| = n$. Обозначим его неориентированные компоненты как V_0 и V_1 так, чтобы ориентированные рёбра выходили из V_0 в V_1. Пусть $|V_0| = n_0$ и $|V_1| = n_1$. Обозначим вершины V_0, инцидентные ориентированным рёбрам, как $0, 0'$ и вершины V_1 – как $1, 1'$ так, чтобы ориентированные рёбра имели вид $0 \to 1$ и $0' \to 1'$ (см. Рис 9).

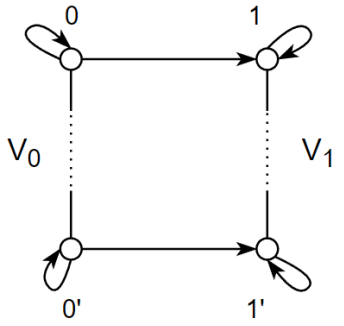


Рис. 9. Цикл типа C.

Утверждение 8. *Пусть $\mathcal{C}$ – смешанно-ориентированный рефлексивный цикл типа C, содержащий больше трёх вершин. Тогда полиморфизмы $\mathcal{C}$ наследуют сюръективность.*

Доказательство. Рассмотрим набор трёхместных полиморфизмов $\mathcal{C}$ $p_1, \ldots, p_k$, которые удовлетворяют условиям наследования сюръективности. Покажем, что среди них найдётся сюръективная функция.

Рассмотрим векторы $\overline{0} = (0, 0, 0)$ и $\overline{1} = (1, 1, 1)$. Рассмотрим $i \in \{1, \ldots, k\}$, вектор $\overline{a} \in V^3$ такие, что $p_i(\overline{a}) \in V_0$. Поскольку существует ориентированный путь вида $\overline{0} \to \cdots \to \overline{a}$, то $p_i(\overline{0}) \in V_0$ (иначе пути вида $p(\overline{0}) \to \cdots \to p(\overline{a})$ не существовало бы), а так как полиморфизмы из набора совпадают на диагонали, то для каждого $j \in \{1, \ldots, k\}$ верно $p_j(\overline{0}) \in V_0$. Аналогично, $p_j(\overline{1}) \in V_1$. А так как в $\mathcal{C}$ есть ребро $\overline{0} \to \overline{1}$ и нет ребра $\overline{0} \leftarrow \overline{1}$, то либо $p_j(\overline{0}) = 0$ и $p_j(\overline{1}) = 1$, либо $p_j(\overline{0}) = 0'$ и $p_j(\overline{1}) = 1'$. Без ограничения общности положим $p_j(\overline{0}) = 0$ и $p_j(\overline{1}) = 1$.

Пусть $n_0 \geq n_1$ (случай $n_1 \geq n_0$ рассматривается аналогично). Возьмём $i \in \{1, \ldots, k\}$, вектор $\overline{a} \in V^3$ такие, что $p_i(\overline{a}) = 0'$. Покажем, что p_i – сюръ-

ективная функция. Несложно заметить, что для каждой вершины v можно выбрать вершину $w \in \{0, 1, 0', 1'\}$ такую, что между v и w существует путь из неориентированных рёбер, содержащий не больше $\lfloor \frac{\max(n_0,n_1)-1}{2} \rfloor$ рёбер. Это значит, что можно выбрать вектор $\overline{b} \in \{0, 1, 0', 1'\}^3$ такой, что существует ориентированный путь Π вида $\overline{a} \leftrightarrow \cdots \leftrightarrow \overline{b}$ длины $l \leq \lfloor \frac{\max(n_0,n_1)-1}{2} \rfloor = \lfloor \frac{n_0-1}{2} \rfloor$. По Утверждению 1 $p_i(\overline{b}) \in V_0$. Тогда можно выбрать векторы $\overline{c} \in V_0^3, \overline{d} \in V_1^3$ такие, что существуют рёбра $\overline{c} \to \overline{d}$ и $\overline{c} \to \overline{b} \to \overline{d}$. Из того, что $p_i(\overline{0}) = 0$, $p_i(\overline{1}) = 1$, по Утверждению 1 имеем $p_i(\overline{c}) \in \{0, 0'\}$, $p_i(\overline{d}) \in \{1, 1'\}$, откуда $p_i(\overline{b}) \in \{0, 0'\}$. Но любой путь из 0 в $0'$ из неориентированных рёбер должен иметь длину не меньше $n_0 - 1$, откуда $p_i(\overline{b}) = 0'$ и $p_i(\overline{c}) = 0', p_i(\overline{d}) = 1'$.

Существуют путь Π_1 из векторов V_0^3 вида $\overline{0} \leftrightarrow \cdots \leftrightarrow \overline{c}$ и путь Π_2 из векторов V_1^3 вида $\overline{1} \leftrightarrow \cdots \leftrightarrow \overline{d}$. Поскольку $p_i(\overline{0}) = 0$, $p_i(\overline{c}) = 0'$ и все рёбра Π_1 ориентированы как $\leftrightarrow$, то образы элементов Π_1 на p_i принимают все значения из V_0. Аналогично, элементы Π_2 принимают на p_i все значения из V_1. Значит, p_i – сюръективная функция. □

Из Утверждений 2, 8 и Теоремы 3 следует:

Утверждение 9. *Пусть $\mathcal{C}$ – смешанно-ориентированный рефлексивный цикл типа C, содержащий больше трёх вершин. Тогда* Surj-Hom($\mathcal{C}$) *является* NP*-трудной.*

3.2.4. Циклы типа D

Рассмотрим смешанно-ориентированный рефлексивный цикл типа D. Обозначим множество его вершин как $V, |V| = n > 3$. Обозначим его неориентированные компоненты как V_0, V_1, V_2 так, чтобы ориентированные рёбра выходили из V_0 в V_1, из V_1 в V_2 и из V_0 в V_2 (см. Рис. 10).

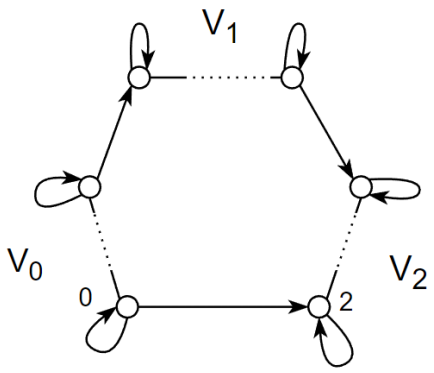


Рис. 10. Цикл типа D

Утверждение 10. *Пусть $\mathcal{C}$ – смешанно-ориентированный рефлексивный цикл типа D. Тогда полиморфизмы $\mathcal{C}$ наследуют сюръективность.*

Доказательство. Рассмотрим набор трёхместных полиморфизмов $p_1, \dots, p_k$, удовлетворяющих условиям наследования сюръективности. Докажем, что среди них найдётся сюръективная функция.

Обозначим вершины, инцидентные ориентированному ребру из V_0 в V_2, как 0 и 2 (см. Рис. 10). Положим $\overline{0} = (0,0,0)$, $\overline{2} = (2,2,2)$. Возьмём $i \in \{1, \dots, k\}$, вектор $\overline{a} \in V^3$ такие, что $p_i(\overline{a}) \in V_0$. Существует ориентированный путь вида $\overline{0} \to \dots \to \overline{a}$, откуда $p_i(\overline{0}) \in V_0$ (иначе существовал бы ориентированный путь из элемента V_1 или V_2 в элемент V_0, в котором все рёбра ориентированы как $\to$, что невозможно). А так как полиморфизмы совпадают на диагонали, то для любого $j \in \{1, \dots, k\}$ верно $p_j(\overline{0}) \in V_0$. Аналогично, $p_j(\overline{2}) \in V_2$. А так как существует ребро $\overline{0} \to \overline{2}$, то $p_j(\overline{0}) = 0$ и $p_j(\overline{2}) = 2$.

Рассмотрим $i \in \{1, \dots, k\}$, вектор $\overline{b} \in V^3$ такие, что $p_i(\overline{b}) \in V_1$. Существует ориентированный путь вида:

$$\overline{0} \to \dots \to \overline{b} \to \dots \to \overline{2}.$$

Поскольку $p_i(\overline{0}) = 0$, $p_i(\overline{2}) = 2$ и все рёбра пути ориентированы как $\to$, то образы элементов этого пути принимают на p_i либо только значения из V_0 и V_2, либо все значения из V. А так как $p_i(\overline{b}) \in V_1$, то p_i принимает все значения из V, то есть она сюръективна. □

Заметим, что так как смешанно-ориентированные графы по определению содержат неориентированные рёбра, то смешанно-ориентированные циклы типа D содержат не меньше четырёх вершин. Значит, из Утверждений 2, 10 и Теоремы 3 следует:

Утверждение 11. *Пусть $\mathcal{C}$ – смешанно-ориентированный рефлексивный цикл типа D. Тогда* Surj-Hom($\mathcal{C}$) *является* NP*-трудной.*

3.2.5. Циклы типа E

Рассмотрим смешанно-ориентированный рефлексивный цикл $\mathcal{C}$ типа E. Обозначим неориентированные компоненты $\mathcal{C}$ как $V_0, V_1, \dots, V_5$ так, чтобы ориентированные рёбра выходили из V_1 в V_0, V_2, из V_3 в V_2, V_4 и из V_5 в V_4, V_0. Обозначим множество вершин $\mathcal{C}$ как $V, |V| = n > 6$. Обозначим вершины, инцидентные ориентированным рёбрам $\mathcal{C}$, как $0, 1, \dots, 5, 0', 1', \dots, 5'$ образом, изображённым на Рис. 11. Заметим, что для некоторых $i \in \{0, 1, \dots, 5\}$ неориентированные компоненты V_i могут содержать всего одну вершину. В этом случае вершины i и i' будут совпадать. При этом как минимум одна неориентированная компонента всегда

будет содержать больше одной вершины. Множество вершин $V_0 \cup V_2 \cup V_4$ будем обозначать как V_{in}, а множество $V_1 \cup V_3 \cup V_5$ – как V_{out}.

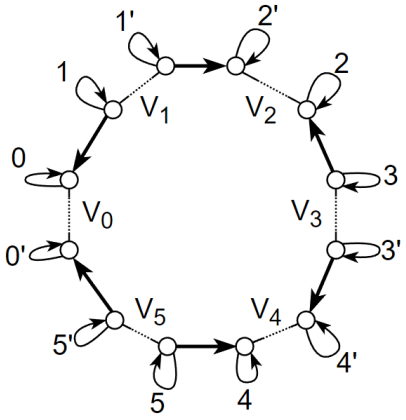


Рис. 11. Цикл типа E.

Полиморфизмы циклов типа E по структуре похожи на полиморфизмы строго ориентированных циклов, которые рассматриваются в [16]. В частности, следующее утверждение формулируется аналогично Утверждению 4 из этой статьи:

Утверждение 12. *Пусть $\mathcal{C}$ – смешанно-ориентированный рефлексивный цикл типа E, p – его трёхместный полиморфизм.*

1) *Пусть $\overline{a} \in V^3$ – вектор такой, что $p(\overline{a}) \in V_i$, $i \in \{0, 2, 4\}$. Тогда существует вектор $\overline{b} \in V_{in}^3$ такой, что $p(\overline{b}) \in V_i$.*

2) *Пусть $\overline{a} \in V^3$ – вектор такой, что $p(\overline{a}) \in V_i$, $i \in \{1, 3, 5\}$. Тогда существует вектор $\overline{b} \in V_{out}^3$ такой, что $p(\overline{b}) \in V_i$.*

3) *Для любого вектора $\overline{a} \in V^3$ существует вершина $w \in V$ такая, что есть ориентированный путь из (w, w, w) в $\overline{a}$, содержащий не более двух ориентированных рёбер.*

4) *Пусть $i, j \in Z_6$, $a, b \in V$ такие, что $p(a, \ldots, a) \in V_i$, $p(b, \ldots, b) \in V_j$. Тогда полиморфизм p принимает на диагонали значения из всех $V_i, V_{i+1 \,(\mathrm{mod}\, 6)}, \ldots, V_j$ или из всех $V_j, V_{j+1 \,(\mathrm{mod}\, 6)}, \ldots, V_i$.*

Доказательство.

1. Выберем вектор $\overline{b} \in V_{in}^3$ такой, что существует ориентированный путь вида $\overline{b} \leftarrow \cdots \leftarrow \overline{a}$. Пусть $p(\overline{b}) \in V_{out}$. Тогда образы этого пути составляют ориентированный путь из элемента V_{out} в элемент V_{in}, в

котором все рёбра ориентированы как $\leftarrow$, что невозможно. Значит, $p(\overline{b}) \in V_{in}$.

2. Доказывается аналогично 1.

3. Поскольку вектор $\overline{a}$ трёхместный, то существует $i \in \{0, \ldots, 5\}$ такое, что ни одна из a^1, a^2, a^3 не лежит в V_i. Положим $i' = i + 3 \pmod 6$. Пусть $V_{i'} \subseteq V_{in}$ (случай $V'_i \subseteq V_{out}$ разбирается аналогично). Возьмём произвольную $w \in V_{i'}$ Тогда для каждой $j \in \{1, 2, 3\}$ можно выбрать вершины v_1, v_2, v_3, v_4 такие, что существует ориентированный путь из w в a^j, содержащий не более двух ориентированный рёбер, следующего вида:

$$w \leftrightarrow \cdots \leftrightarrow v_1 \leftarrow v_2 \leftrightarrow \cdots \leftrightarrow v_3 \rightarrow v_4 \leftrightarrow \cdots \leftrightarrow a^j.$$

А это значит, что в $\mathcal{C}$ есть ориентированный путь из (w, w, w) в $\overline{a}$, который содержит не больше двух ориентированных рёбер.

4. Существует ориентированный путь из (a, a, a) в (b, b, b), состоящий только из диагональных элементов. Образы членов этого пути принимают на p значения из всех $V_i, V_{i+1 \pmod 6}, \ldots, V_j$ или из всех $V_j, V_{j+1 \pmod 6}, \ldots, V_i$. □

Утверждение 13. *Пусть $\mathcal{C}$ – смешанно-ориентированный рефлексивный цикл типа E, p – его полиморфизм. Пусть p принимает значения как минимум из пяти различных неориентированных компонент. Тогда p сюръективен.*

Доказательство. Сперва покажем, что если p принимает значения из V_0, V_2 и V_4, то он принимает все значения из V_{out}. Пусть p не принимает значение $v \in V_1$ (случаи $v \in V_3$ и $v \in V_5$ разбираются аналогично). По Утверждению 12.1 существуют векторы $\overline{a}, \overline{b} \in V_{in}^3$ такие, что $p(\overline{a}) \in V_0$ и $p(\overline{b}) \in V_2$. Можно выбрать векторы $\overline{v_1}, \overline{v_2}, \overline{v_3}, \overline{v_4} \in V^3$ такие, что существует ориентированный путь из $\overline{a}$ в $\overline{b}$, содержащий всего два ориентированных ребра вида:

$$\overline{a} \leftrightarrow \cdots \leftrightarrow \overline{v_1} \leftarrow \overline{v_2} \leftrightarrow \cdots \leftrightarrow \overline{v_3} \rightarrow \overline{v_4} \leftrightarrow \cdots \leftrightarrow \overline{b}.$$

Образы элементов этого пути принимают на p все значения из V_1, поскольку любой путь из элемента V_0 в элемент V_2, не проходящий через V_1, будет содержать как минимум 4 ориентированных ребра.

Аналогично показывается, что если p принимет значения из V_1, V_3 и V_5, то он принимает все значения из V_{in}. Значит, если полиморфизм $\mathcal{C}$ принимает значения из всех шести неориентированных компонент, то он сюръективен. Теперь, пусть p принимает значения из всех неориентированных компонент кроме, может быть, $V_i \subseteq V_{out}$ (случай $V_i \in V_{in}$ рассматривается аналогично). Выше показали, что тогда эта функция принимает все значения из V_{out}, откуда p сюръективна.

□

Утверждение 14. *Пусть $\mathcal{C}$ – смешанно-ориентированный рефлексивный цикл типа E, $p_1, \ldots, p_k$ – набор его полиморфизмов, удовлетворяющий условиям наследования сюръективности. Тогда:*

1) *$p_1, \ldots, p_k$ принимают на диагонали значения как минимум из двух неориентированных компонент.*

2) *Если на диагонали принимаются значения хотя бы из трёх неориентированных компонент, то среди $p_1, \ldots, p_k$ найдётся сюръективная функция.*

Доказательство.

1. Пусть на диагонали принимается значение s. Без ограничения общности положим $s \in V_0$. Рассмотрим $i \in \{1, \ldots, k\}$, вектор $\overline{a} \in V^3$ такие, что $p_i(\overline{a}) \in V_3$. По Утверждению 12.3 существует элемент w такой, что есть ориентированный путь из (w, w, w) до $\overline{a}$, который содержит не больше двух ориентированных рёбер. А это значит, что $p_i(w, w, w) \notin V_0$. Так как полиморфизмы совпадают на диагонали, то все $p_1, \ldots, p_k$ принимают на диагонали как минимум одно значение из V_0 и одно не из V_0.

2. Без ограничения общности положим, что на диагонали принимается значение из V_0. В этом случае по Утверждению 12.4 на диагонали также принимаются значения из V_1 и V_2, из V_5 и V_4 или из V_1 и V_5. Рассмотрим первый вариант (остальные доказываются аналогично). Рассмотрим $i \in \{1, \ldots, k\}$, вектор $\overline{a} \in V^3$ такие, что $p_i(\overline{a}) \in V_4$. Рассмотрим произвольный элемент $s \in V$ такой, что $p_i(s, s, s) \in V_1$. Рассмотрим произвольный ориентированный путь из (s, s, s) в $\overline{a}$. Образы элементов этого пути принимают на p значения из V_3 или V_5. Значит, p_i принимает значения из как минимум пяти неориентированных компонент и, по Утверждению 13, она сюръективна. □

Утверждение 15. *Пусть $\mathcal{C}$ – смешанно-ориентированный рефлексивный цикл типа E. Тогда полиморфизмы $\mathcal{C}$ наследуют сюръективность.*

Доказательство. Рассмотрим набор трёхместных полиморфизмов $p_1, \ldots, p_k$, удовлетворяющий условиям наследования сюръективности. Докажем, что среди них найдётся сюръективная функция.

По Утверждению 14.2 если $p_1, \ldots, p_k$ принимают на диагонали значения из трёх неориентированных компонент, то один из этих полиморфизмов будет сюръективным. Тогда, по Утверждению 14.1 достаточно рассмотреть случай, когда эти функции принимают на диагонали значения только из двух неориентированных компонент. По Утверждению 12.4 эти компоненты смежны. Без ограничения общности положим, что $p_1, \ldots, p_k$ принимают на диагонали только значения из V_0, V_1.

Рассмотрим $i \in \{1, \dots, k\}$, вектор $\overline{a} \in V^3$ такие, что $p_i(\overline{a}) \in V_3$. По Утверждению 12.2 выберем вектор $\overline{a'} \in V^3_{out}$ такой, что $p_i(\overline{a'}) \in V_3$. Возьмём вершину $s \in V$ такую, что $p_i(s,s,s) \in V_0$. Любой ориентированный путь из (s,s,s) в $\overline{a'}$ должен содержать не меньше трёх ориентированных рёбер. Поскольку для любых двух векторов $\overline{v}, \overline{w} \in V^3_{out}$ существует ориентированный путь из $\overline{v}$ в $\overline{w}$, содержащий не более двух ориентированных рёбер, то из $\overline{a} \in V^3_{out}$ следует $(s,s,s) \notin V_{out}$, то есть $s \in V_{in}$. Это значит, что для любых $j \in \{1, \dots, k\}$ и $s' \in V_{out}$ верно $p_j(s',s',s') \in V_1$. Аналогично рассматривая $h \in \{1, \dots, k\}$ и вектор $\overline{b} \in V^3$ такие, что $p_h(\overline{b}) \in V_4$, показываем, что для всех $s' \in V_{in}$ верно $p_j(s',s',s') \in V_0$. Более того, отсюда по Утверждению 1 для любого $j \in Z_6$ и любых $\overline{v} \in V^3_j, i \in \{1, \dots k\}$ верно $p_i(\overline{v}) \in V_0$, если $j \in \{0,2,4\}$ и $p_i(\overline{v}) \in V_1$, если $j \in \{1,3,5\}$.

Пусть $|V_3| > 1$ (случай $|V_4| > 1$ рассматривается аналогично). Рассмотрим $i \in \{1, \dots, k\}$, $\overline{a} \in V^3$ такие, что $p_i(\overline{a}) = 3'$. Существует вектор $\overline{a'} \in V_{in} \cup V_1 \cup V_5 \cup \{3, 3'\}$ такой, что есть путь из $\overline{a}$ в $\overline{a'}$ из неориентированных рёбер длины $l \leq \lfloor \frac{|V_3|}{2} \rfloor$. По Утверждению 1 $p_i(\overline{a'}) \in V_3$. Можно выбрать векторы $\overline{b} \in \{0, 0'\}^3$, $\overline{v_1}, \dots \overline{v_4} \in V^3$ такие, что существует ориентированный путь Π с тремя ориентированными рёбрами следующего вида:

$$\Pi : \overline{b} \leftarrow \overline{v_1} \leftrightarrow \cdots \leftrightarrow \overline{v_2} \rightarrow \overline{v_3} \leftrightarrow \cdots \leftrightarrow \overline{v_4} \leftarrow \overline{a'}.$$

Так как $p_i(\overline{b}) \in V_0$ и путь до $\overline{v_4}$ содержит всего два ориентированных ребра, то $p_i(\overline{v_4}) \notin V_3$, а так как $p_i(\overline{a'}) \in V_3$, то $p_i(\overline{a'}) \in \{3, 3'\}$. Поскольку $p_i(\overline{a}) = 3'$ и пути из неориентированных рёбер из $3'$ в 3 длины $l \leq \lfloor \frac{|V_3|}{2} \rfloor$ не существует, то $p_i(\overline{a'}) = 3'$. А так как $|V_3| > 1$, то это значит, что $p_i(\overline{v_4}) \in V_4$ и, значит, $p_i(\overline{v_1}) \in V_5$. Получили, что p_i принимает значения из всех неориентированных компонент кроме, может быть, V_2. Значит, по Утверждению 13 эта функция сюръективна.

Теперь, пусть $|V_3| = |V_4| = 1$, тогда положим $V_3 = \{3\}$ и $V_4 = \{4\}$. В этом случае в $\mathcal{C}$ есть рёбра $2 \leftarrow 3 \rightarrow 4 \leftarrow 5$. Пусть $|V_5| > 1$ (случай $|V_2| > 1$ разбирается аналогично). Возьмём $i \in \{1, \dots, k\}$, $\overline{a} \in V^3_{in}$ такие, что $p_i(\overline{a}) = 4$. Можно выбрать векторы $\overline{b} \in \{1, 1'\}^3$, $\overline{v_1}, \overline{v_2}, \overline{v_3} \in V^3$ такие, что существует ориентированный путь Π с тремя ориентированными рёбрами следующего вида:

$$\Pi : \overline{b} \rightarrow \overline{v_1} \leftrightarrow \cdots \leftrightarrow \overline{v_2} \leftarrow \overline{v_3} \rightarrow \overline{a}.$$

Заметим, что так как путь до $\overline{v_3}$ содержит ровно два ориентированных ребра, $p_i(\overline{b}) \in V_1$ и есть ребро $\overline{v_3} \rightarrow \overline{a}$, то $p_i(\overline{v_3}) \in V_3 \cup V_5$. Но если $p_i(\overline{v_3}) \in V_5$, то по построению Π имеем $p_i(\overline{v_2}) \in V_0$. Но тогда из существования рёбер $\overline{v_2} \leftarrow \overline{v_3} \rightarrow \overline{a}$ следует $|V_5| = 1$ – противоречие. Значит, $p_i(\overline{v_3}) \in V_3$. А это значит, что $p_i(\overline{v_2}) \in V_2$, и p_i принимает значения из пяти неориентированных компонент $V_0, V_1, \dots, V_4$. Значит, p_i сюръективная.

Наконец, пусть все компоненты V_2, V_3, V_4, V_5 содержат по одной вершине, тогда положим $V_2 = \{2\}$, $V_5 = \{5\}$. В этом случае $|V_0| > 1$ или $|V_1| > 1$. Рассмотрим первый случай (второй рассматривается аналогично). Возьмём $i \in \{1, \dots, k\}$, вектор $\overline{a} \in V_{in}^3$ такие, что $p_i(\overline{a}) = 4$. Можно выбрать вектор $\overline{a'} \in \{0, 2, 4\}^3$, лежащий в одной неориентированной компоненте с $\overline{a}$, для него по Утверждению 1 верно $p_i(\overline{a'}) = 4$. Также можно выбрать векторы $\overline{v_1} \in \{1, 1'\}^3$, $\overline{v_2}, \overline{v_3} \in V^3$ такие, что существует ориентированный путь П, в котором все рёбра ориентированные, следующего вида:

$$\overline{v_1} \to \overline{v_2} \leftarrow \overline{v_3} \to \overline{a'}.$$

Поскольку $p_i(\overline{v_1}) \in V_1$, $p_i(\overline{a'}) \in V_4$, то образы элементов П принимают на p_i все значения из V_0 и 5 или 2 и 3. Но поскольку $|V_0| > 1$, то любой путь, проходящий через все элементы из V_0 и 5 имел бы длину не меньше четырёх. Значит, p_i принимает значения $2, 3$. Получили, что p_i принимает значения из пяти различных неориентированных компонент $V_0, \dots, V_4$, откуда p_i – сюръективная функция. □

Из Утверждений 2, 15 и Теоремы 3 следует:

Утверждение 16. *Пусть $\mathcal{C}$ – смешанно-ориентированный рефлексивный цикл типа E. Тогда* Surj-Hom($\mathcal{C}$) *является* NP*-трудной.*

Наконец, Следствие 1 и Утверждения 5, 7, 9, 11 и 16 доказывают Теорему 1, сформулированную в разделе 1.1:

Теорема 1. *Пусть $\mathcal{C}$ – смешанно-ориентированный рефлексивный цикл, содержащий больше трёх вершин и не изоморфный $\mathcal{C}_4, \mathcal{C}_5$ или $\mathcal{C}_6$. Тогда задача* Surj-Hom($\mathcal{C}$) *является* NP*-трудной.*

3.3. Циклы без свойства наследования сюръективности

В этом разделе мы рассмотрим смешанно-ориентированные рефлексивные циклы, для которых не известна сложность задачи Surj-Hom. К ним относятся циклы типа A длины $n \in \{4, 5, 6\}$ (см. Рис. 12). Эти циклы не обладают остовными циклами, поэтому Следствие 1 к ним не применимо. Мы явно продемонстрируем, что подобные циклы не обладают свойством наследования сюръективности в слабой форме (что покажет, что Теорема 3 к ним также неприменима).

Утверждение 17. *Пусть $\mathcal{C}$ – смешанно-ориентированный рефлексивный цикл типа A, содержащий 4 вершины. Тогда полиморфизмы $\mathcal{C}$ не наследуют сюръективность в слабой форме.*

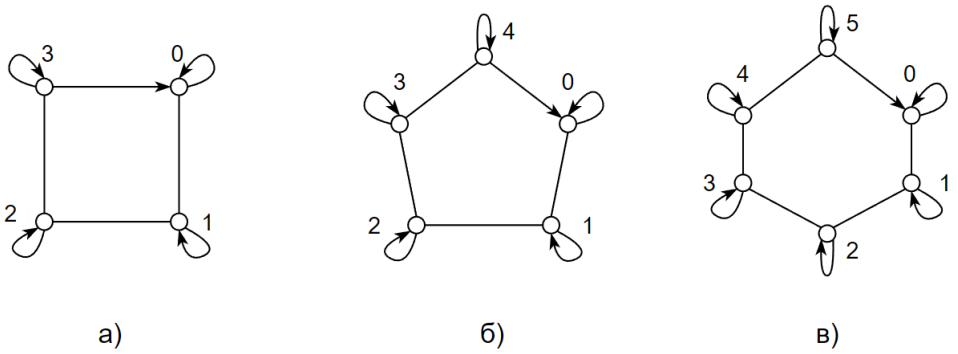


Рис. 12. Циклы типа A длины $n = 4, 5, 6$

Доказательство. Рассмотрим трёхместные функции p_1, p_2, устроенные следующим образом:

$$p_1(x,y,z) = \begin{cases} 0, & \text{если } (x,y,z) = (0,3,2), \\ 2, & \text{если } (x,y,z) \in \{(1,1,1),(2,2,2)\}, \\ 1, & \text{иначе.} \end{cases}$$

$$p_2(x,y,z) = \begin{cases} 3, & \text{если } (x,y,z) = (1,2,3), \\ 1, & \text{если } (x,y,z) \in \{(0,0,0),(3,3,3)\}, \\ 2, & \text{иначе.} \end{cases}$$

Эти функции являются полиморфизмами $\mathcal{C}$, совместно сюръективны и диагонально согласованы. Покажем, что они также имитируют проекции.

Возьмём произвольное ограничение из формулировки свойства имитирования проекций. Оно будет иметь следующий вид:

$$\mathcal{C}(p_i(\overline{a}), p_j(\overline{b})),$$

где $i, j \in \{1,2\}, i \neq j$ и для любых $h, l \in \{1,2,3\}$ верно $\mathcal{C}(a^h, b^l)$. Пусть вектор $\overline{a}$ диагональный (случай диагональности вектора $\overline{b}$ рассматривается аналогично). Обозначим $\overline{a} = (a,a,a)$. Поскольку p_i и p_j совпадают на диагонали, то ограничение $\mathcal{C}(p_i(\overline{a}), p_j(\overline{b}))$ равносильно $\mathcal{C}(p_j(\overline{a}), p_j(\overline{b}))$, где для каждого $h \in \{1,2,3\}$ верно $\mathcal{C}(p_j(a, b^h))$. А это ограничение выполняется, поскольку p_j – полиморфизм.

Итак, пусть ни один из векторов $\overline{a}, \overline{b}$ не диагональный. Если элементы $\overline{a}$ принимают три разных значения, то существует ровно одно b такое, что выполняется $\mathcal{C}(a^i, b), i \in \{1,2,3\}$, то есть $\overline{b}$ диагональный. Аналогично, если $\overline{b}$ содержит три разных элемента, то $\overline{a}$ диагональный. Если же $\overline{a}$ и $\overline{b}$ принимают не более двух разных значений, то по построению $p(\overline{a}), p(\overline{b}) \in \{1,2\}$, откуда $\mathcal{C}(p_i(\overline{a}), p_j(\overline{b}))$ выполняется.

Итак, мы предъявили набор из двух функций, которые удовлетворяют условиям наследования сюръективности в слабой форме, но ни одна из них не является сюръективной, откуда $\mathcal{C}$ не обладает этим свойством. $\square$

Утверждение 18. *Пусть $\mathcal{C}$ – смешанно-ориентированный рефлексивный цикл типа A, содержащий 5 вершин. Тогда полиморфизмы $\mathcal{C}$ не наследуют сюръективность в слабой форме.*

Доказательство. Рассмотрим трёхместные функции p_1, p_2, устроенные следующим образом:

$$p_1(x,y,z) = \begin{cases} 0, \text{ если} (x,y,z) = (0,1,3), \\ 1, \text{ если } (x,y,z) \neq (0,1,3) \text{ и есть ребро} \\ \qquad\qquad \text{между } (x,y,z) \text{ и } (0,1,3), \\ 2, \text{ иначе.} \end{cases}$$

$$p_2(x,y,z) = \begin{cases} 4, \text{ если} (x,y,z) = (0,2,4), \\ 3, \text{ если } (x,y,z) \neq (0,2,4) \text{ и есть ребро} \\ \qquad\qquad \text{между } (x,y,z) \text{ и } (0,2,4), \\ 2, \text{ иначе.} \end{cases}$$

Заметим, что p_1 и p_2 принимают на диагонали только значение 2. Несложно заметить, что эти функции являются полиморфизмами $\mathcal{C}$, совместно сюръективны и диагонально согласованы. Покажем, что они имитируют проекции.

Возьмём произвольное ограничение из формулировки свойства имитирования проекций. Оно будет иметь следующий вид:

$$\mathcal{C}(p_i(\overline{a}), p_j(\overline{b})),$$

где $i, j \in \{1,2\}, i \neq j$ и $\overline{a}, \overline{b} \in Z_5^3$. Аналогично случаю из предыдущего утверждения, достаточно рассмотреть случаи, когда векторы $\overline{a}$, $\overline{b}$ содержат ровно по два разных элемента. Заметим, что в этом случае из того, что для любых $h, l \in \{1,2,3\}$ выполняется условие $\mathcal{C}(a^h, b^l)$, следует, что $\{a^1, a^2, a^3\} = \{b^1, b^2, b^3\}$. По построению функций p_1 и p_2 выбранное условие может не выполняться только в том случае, если $\{p_i(\overline{a}), p_j(\overline{b})\} = \{1,3\}$. Без ограничения общности положим $i = 1, j = 2$ и $p_1(\overline{a}) = 1, p_2(\overline{b}) = 3$. Так как $\overline{a}$ содержит ровно два разных элемента и существует ребро между $\overline{a}$ и $(0,1,3)$, то $\overline{a} \in \{1,2\}^3$ или $\overline{a} \in \{0,4\}^3$. Аналогично, $\overline{b} \in \{0,1\}^3$ или $\overline{b} \in \{3,4\}^3$. Иными словами, если $\{p(\overline{a}), p(\overline{b})\} = \{1,3\}$, то $\{a^1, a^2, a^3\} \neq \{b^1, b^2, b^3\}$. Значит, ограничение из определения имитирования проекций выполняется, то есть функции p_1 и p_2 удовлетворяют условиям наследования сюръективности в слабой форме, но ни одна из них не является сюръективной. $\square$

Утверждение 19. *Пусть $\mathcal{C}$ – смешанно-ориентированный рефлексивный цикл типа A, содержащий 6 вершин. Тогда полиморфизмы $\mathcal{C}$ не наследуют сюръективность в слабой форме.*

Доказательство. Рассмотрим трёхместные функции p_1, p_2, устроенные следующим образом:

$$p_1(x,y,z) = \begin{cases} 0, \text{ если } (x,y,z) = (0,4,2), \\ 1, \text{ если } (x,y,z) \neq (0,4,2) \text{ и есть ребро} \\ \qquad\qquad \text{между } (x,y,z) \text{ и } (0,4,2), \\ 3, \text{ если } (x,y,z) \in \{(1,1,1),(3,3,3),(5,5,5)\}, \\ 2, \text{ иначе.} \end{cases}$$

$$p_2(x,y,z) = \begin{cases} 5, \text{ если } (x,y,z) = (1,3,5), \\ 4, \text{ если } (x,y,z) \neq (1,3,5) \text{ и есть ребро} \\ \qquad\qquad \text{между } (x,y,z) \text{ и } (1,3,5), \\ 2, \text{ если } (x,y,z) \in \{(0,0,0),(2,2,2),(4,4,4)\}, \\ 3, \text{ иначе.} \end{cases}$$

Заметим, что любой ориентированный путь между $(0,4,2)$ и любым набором из $\{(1,1,1),(3,3,3),(5,5,5)\}$ будет иметь длину $l \geq 3$ и p_1 – полиморфизм $\mathcal{C}$. Аналогично, p_2 – также полиморфизм $\mathcal{C}$. Эти функции совместно сюръективны и диагонально согласованы. Покажем, что они имитируют проекции.

Возьмём произвольное ограничение из формулировки свойства имитирования проекций. Оно будет иметь следующий вид:

$$\mathcal{C}(p_i(\overline{a}), p_j(\overline{b})),$$

$i,j \in \{1,2\}, i \neq j$ и $\overline{a}, \overline{b} \in Z_6^3$. Аналогично случаю из Утверждения 16, достаточно рассмотреть векторы $\overline{a}, \overline{b}$, которые содержат ровно по два разных элемента. Заметим, что так как для любых $h, l \in \{1,2,3\}$ верно $\mathcal{C}(a^h, b^l)$, то все a^1, a^2, a^3 и b^1, b^2, b^3 смежные между собой. Также заметим, что по построению p_1 и p_2 если $p_1(\overline{v}) \in \{0,1\}$ или $p_2(\overline{v}) \in \{5,4\}$, то среди v^1, v^2, v^3 всегда найдутся несмежные вершины. Значит, $p_i(\overline{a}), p_j(\overline{b}) \in \{2,3\}$ и это ограничение выполняется. Отсюда, функции p_1 и p_2 удовлетворяют всем трём условиям, но ни одна из них не является сюръективной. □

Таким образом, данные циклы не обладают свойством наследования сюръективности в слабой форме. Заметим, что это не значит, что задача Surj-Hom для них решается за полиномиальное время: так, неориентированный рефлексивный цикл $\mathcal{C}_4$ длины 4 тоже не обладает этим свойством, но для него задача является NP-трудной [2]. Тем не менее, для анализа

сложности задачи для этих циклов требуются новые инструменты, не описанные в этой статье.

The complexity of the graph homomorphism problem on mixed reflexive cycles

Korchagin N.P.

The surjective graph homomorphism problem Surj-Hom($\mathcal{H}$) is a problem of deciding whether a given graph allows vertex-surjective homomorphism to a fixed graph $\mathcal{H}$. In this paper we study the Surj-Hom problem for cyclic graphs which are obtained from undirected cycles by assigning direction to some edges and in which each vertex contains a loop.

We explore the Surj-Hom problem in its conjunction with the surjective constraint satisfaction problem SCSP. We define a property which allows to obtain the complexity of the SCSP problem for some predicates via reduction. We implement this property to determine the complexity of the Surj-Hom problem for all desired cycles except for three cycles with 4, 5 and 6 vertices.

Keywords: surjective graph homomorphism, computational complexity, constraint satisfaction, polymorphism

References

[1] Feder T. and Hell P., "List Homomorphisms to Reflexive Graphs", *Journal of Combinatorial Theory, Series B*, **72**:2 (1998), 236-250.

[2] Martin B. and Paulusma D., "The computational complexity of disconnected cut and 2K2-partition", *Journal of Combinatorial Theory, Series B*, **111** (2015), 17-37.

[3] Vikas N., "Computational complexity of compaction to irreflexive cycles", *Journal of Computer and System Sciences*, **68**:3 (2004), 473-496.

[4] Hell P. and Nesetril J., *Graphs and Homomorphisms*, Oxford University Press, 2004.

[5] Bodirsky M. and Kara J. and Martin B., "The Complexity of Surjective Homomorphism Problems – a Survey", *Computing Research Repository - CORR*, **160**:12 (2011), 1680-1690.

[6] Focke J. and Goldberg L. and Živný S., "The Complexity of Counting Surjective Homomorphisms and Compactions", *SIAM Journal on Discrete Mathematics*, **33**:2 (2019), 1006-1043.

[7] Golovach P.A. and Johnson M. and Martin B. and Paulusma D. and Stewart A., "Surjective H-colouring : new hardness results.", *Computability.*, **8**:1 (2019), 27-42.

[8] Golovach P. A. and Paulusma D. and Song J., "Computing vertex-surjective homomorphisms to partially reflexive trees", *Theoretical Computer Science*, **457** (2012), 86-100.

[9] Larose, B. and Martin, B. and Paulusma, D., "Surjective H-Colouring over Reflexive Digraphs", *ACM Trans. Comput. Theory*, **11**:1 (2018).

[10] Korchagin N. P., "Complexity of surjective homomorphism problem to reflexive cycles", *Intelligent Systems. Theory and Applications*, **27**:4 (2023), 40–61 (In Russian).

[11] Feder T. and Vardi M. Y., "The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory", *SIAM Journal on Computing*, **28**:1 (1998), 57-104.

[12] Zhuk D., "A Proof of the CSP Dichotomy Conjecture", *J. ACM*, **67**:5 (2020).

[13] Bulatov A., "A Dichotomy Theorem for Nonuniform CSPs", *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, 2017, 319-330.

[14] Chen H., "An algebraic hardness criterion for surjective constraint satisfaction", *Algebra universalis*, **72** (2014), 393-401.

[15] Zhuk D., "No-Rainbow Problem and the Surjective Constraint Satisfaction Problem", *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2021, 1-7.

[16] Korchagin N. P., "The complexity of the graph homomorphism problem on directed reflexive cycles", *Discrete Mathematics and Applications*, **37**:4 (2025), 54–88 (In Russian).

[17] Larivière I. and Larose B. and Pullas D., "Surjective polymorphisms of directed reflexive cycles", *Algebra universalis*, **85** (2023).

[18] Bulatov A. and Jeavons P. and Krokhin A., "Classifying the Complexity of Constraints Using Finite Algebras", *SIAM Journal on Computing*, **34**:3 (2005), 720-742.

Аппроксимационная полнота линейных дефинитных автоматов

И. В. Молдованов[1]

У линейных дефинитных автоматов выходные сигналы в каждый момент зависят лишь от ограниченного числа последних входных значений.

В работе исследуется вопрос функциональной полноты относительно оператора аппроксимационного замыкания для класса линейных дефинитных автоматов над полем из двух элементов. Для обозначенного множества автоматов получен критерий полноты, сформулированный в виде системы предполных классов.

Ключевые слова: аппроксимационное замыкание, линейные автоматы, дефинитные автоматы.

1. Введение

Линейные автоматы имеют важное значение для вычислительной техники, систем связи, теории кодирования и при построении генераторов псевдослучайных последовательностей. В работах [1, 2] было проведено исследование линейных автоматов над конечными полями с точки зрения функциональной полноты: были описаны A-предполные и K-предполные классы относительно соответствующих операторов замыкания. Отдельный интерес представляет изучение полноты не во всём классе линейных автоматов, а в его значимых подклассах, определяемых дополнительными ограничениями на поведение автомата. Такой подкласс составляют, в частности, линейные дефинитные автоматы, у которых выходные сигналы в каждый момент зависят лишь от ограниченного числа последних входных значений. В настоящей работе рассматривается вопрос полноты в классе линейных дефинитных автоматов над полем из двух элементов относительно оператора аппроксимационного замыкания.

2. Определения

Рассмотрим E_2 — конечное поле, состоящее из двух элементов $\{0,\ 1\}$. Конечный инициальный автомат над E_2 с n входами и одним выходом преобразует бесконечные входные последовательности $\alpha_1, \ldots, \alpha_n$ в бесконечную

[1] *Молдованов Илья Владимирович* — аспирант каф. математической теории интеллектуальных систем мех.-мат. ф-та МГУ, e-mail: jamfr-d@mail.ru.

Moldovanov Ilia Vladimirovich — graduate student, Lomonosov Moscow State University, Faculty of Mechanics and Mathematics, Chair of Mathematical Theory of Intellectual Systems.

последовательность β. В момент времени t по вектору $(\alpha_1(t), \ldots, \alpha_n(t))$, составленному из соответствующих элементов входных последовательностей, автомат производит элемент $\beta(t)$ выходной последовательности. Положим R_2 — множество формальных степенных рядов, построенных по последовательностям элементов из E_2, т.е. $R_2 = \{\sum\limits_{t=0}^{\infty} x(t)\xi^t \mid x(t) \in E_2\}$. Обозначим $\alpha_1(\xi), \ldots, \alpha_n(\xi)$ — степенные ряды, построенные по входным последовательностям автомата, и $\beta(\xi)$ — ряд, соответствующий выходной последовательности. В этом смысле можно говорить, что автомат задает некоторое отображение $R_2^n \to R_2$. Далее в работе в первую очередь будем использовать данное представление автоматов.
Замыкание некоторого множества автоматов по операциям суперпозиции [3] (переименование переменных, отождествление переменных и подстановка) будем называть Σ-замыканием и обозначать $S(M)$.

3. Линейные дефинитные автоматы

Определение дефинитного автомата представлено в [4]. Дефинитным называется автомат, для которого найдется $t \in \mathbb{N}$ такое, что каждое входное слово длины t переводит автомат из любого состояния в одно и то же состояние, зависящее от этого входного слова. Данный класс совпадает с замыканием по операциям суперпозиции системы, состоящей из задержки с заданным начальным состоянием и универсальной «истинностной» ограниченно-детерминированной функции.

К линейным автоматам относятся задержки с заданным начальным состоянием $\xi x_1 + b,\ b = 0, 1$ и сумматор $x_1 + x_2$ [5]. Обозначим данные автоматы $F_\xi(x_1), F_{\xi,1}(x_1)$ и $F_+(x_1, x_2)$, соответственно. При замыкании данного множества по операциям композиции получаем класс линейных автоматов [1].

Сформулируем определение *линейного дефинитного* автомата. Автомат, содержащийся в замыкании по операциям суперпозиции системы $\{F_\xi(x_1), F_{\xi,1}(x_1), F_+(x_1, x_2)\}$, назовем *линейным дефинитным* автоматом. Обозначим данный класс LD_2. Отметим, что LD_2 не совпадает с пересечением классов линейных и дефинитных автоматов, а является подклассом данного множества. *Линейные дефинитные* автоматы, в том смысле, в котором они определены в данной работе, например, не содержат функцию, реализующую последовательность 1^∞, которая содержится в пересечении классов линейных и дефинитных автоматов.

Охарактеризуем вид отображения $R_2^n \to R_2$, которое задается произвольным линейным дефинитным автоматом с n входами. $E_2[\xi]$ будем обозначать множество полиномов по ξ с коэффициентами из E_2.

Теорема 1. *Функция $f : R_2^n \to R_2$ принадлежит классу LD_2 тогда и только тогда, когда существуют многочлены $u_0, u_1, \ldots, u_n \in E_2[\xi]$ такие, что для любых $\alpha_1, \alpha_2, \ldots, \alpha_n \in R_2$ выполнено равенство:*

$$f(\alpha_1, \alpha_2, \ldots, \alpha_n) = \sum_{i=1}^{n} u_i(\xi)\alpha_i + u_0(\xi). \tag{1}$$

Доказательство. Для доказательства прямого утверждения покажем, что любой линейный дефинитный автомат представим в виде (1).
Проведем индукцию по построению автомата. Базу индукции реализуют

$$F_+(\alpha_1, \alpha_2) = 1 \cdot \alpha_1 + 1 \cdot \alpha_2 + 0,$$
$$F_\xi(\alpha_1) = \xi \cdot \alpha_1,\ F_{\xi,1}(\alpha_1) = \xi \cdot \alpha_1 + 1.$$

Покажем, что представление (1) сохраняется при применении операций суперпозиции к автоматам вида $f(x_1, \ldots, x_n) = \sum\limits_{i=1}^{n} u_i(\xi)x_i + u_0(\xi)$.
Для автомата $f(x'_1, \ldots, x'_n)$, получаемого из $f(x_1, \ldots, x_n)$ переименованием переменных с $x_1, \ldots, x_n$ на $x'_1, \ldots, x'_n$, соответственно, имеем:

$$f(x'_1, \ldots, x'_n) = \sum_{i=1}^{n} u_i(\xi)x'_i + u_0(\xi).$$

Рассмотрим операцию отождествления переменных. Пусть (без ограничения общности) для автомата $g(x_1, \ldots, x_{n-1})$ выполнено, что он получается из автомата $f(x_1, \ldots, x_{n-1}, x_n)$ отождествлением переменных x_{n-1} и x_n:

$$\begin{aligned} g(x_1, \ldots, x_{n-1}) &= f(x_1, \ldots, x_{n-1}, x_{n-1}) = \\ &= \sum_{i=1}^{n-2} u_i(\xi)x_i + u_{n-1}(\xi)x_{n-1} + u_n(\xi)x_{n-1} + u_0(\xi) = \\ &= \sum_{i=1}^{n-2} u_i(\xi)x_i + (u_{n-1}(\xi) + u_n(\xi))x_{n-1} + u_0(\xi) = \\ &= \sum_{i=1}^{n-1} u'_i(\xi)x_i + u'_0(\xi). \end{aligned}$$

Рассмотрим операцию подстановки. Пусть (без ограничения общности) для автомата $g(x_1, \ldots, x_{n+k-1})$ выполнено, что он получается из автомата

$f(x_1, \ldots, x_{n-1}, x_n)$ подстановкой $h(x'_1, \ldots, x'_k)$ вместо переменной x_n:

$$\begin{aligned} g(x_1, \ldots, x_{n-1}, x'_1, \ldots, x'_k) &= f(x_1, \ldots, x_{n-1}, h(x'_1, x'_2, \ldots, x'_k)) = \\ &= \sum_{i=1}^{n-1} u_i(\xi) x_i + u_n(\xi) \left(\sum_{j=1}^{k} u'_j(\xi) x'_j + u'_0(\xi) \right) + u_0(\xi) = \\ &= \sum_{i=1}^{n-1} u_i(\xi) x_i + \sum_{j=1}^{k} u_n(\xi) u'_j(\xi) x'_j + u_n(\xi) u'_0(\xi) + u_0(\xi) = \\ &= \sum_{i=1}^{n-1} u''_i(\xi) x_i + \sum_{j=1}^{k} u'''_j(\xi) x'_j + u''_0(\xi). \end{aligned}$$

Следовательно, любой автомат, полученный при помощи операций суперпозиции из элементов $\{F_+, F_\xi, F_{\xi,1}\}$, будет также иметь вид (1).

Докажем вторую часть утверждения. Рассмотрим произвольную функцию $f(x_1, \ldots, x_n) : R_2^n \to R_2$, для которой справедливо представление (1), и приведем линейный дефинитный автомат, который ее реализует.

Рассмотрим подстановку

$$F_1(x_1) = F_{\xi,1}(F_+(x_1, x_1)) = \xi \cdot 0 + 1 \in S(\{F_{\xi,1}(x_1), F_+(x_1, x_1)\}).$$

Следовательно, автомат, реализующий константную последовательность 10^∞, является линейным дефинитным.

Индукцией по степени многочлена, покажем, что для произвольного многочлена $u(\xi) \in E_2[\xi],\ u(\xi) = \sum\limits_{i=0}^{n} a_i \xi^i$, найдется одноместный линейный дефинитный автомат, который реализует умножение на данный многочлен.

В качестве базы индукции рассмотрим многочлен нулевой степени $u(\xi) = a_0$.

- Случай $a_0 = 0$: рисунок 1.

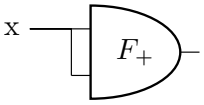


Рис. 1. Автомат F_0

- Случай $a_0 = 1$: рисунок 2.

Пусть для произвольного многочлена степени $n',\ 0 \le n' < n$ существует линейный дефинитный автомат, который реализует умножение на данный многочлен. Рассмотрим произвольный многочлен $u(\xi),\ \deg(u(\xi)) = n$.

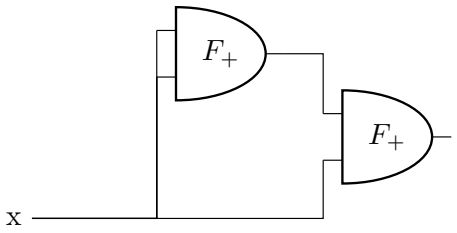


Рис. 2. Автомат $F_{\cdot 1}$

Имеем $u'(\xi) = \xi^n + u(\xi)$ для некоторого $u'(\xi) \in E_2[\xi],\ \deg(u'(\xi)) < n$. По предположению индукции существует линейный дефинитный автомат $F_{\cdot u'(\xi)}$, который реализует умножение на многочлен $u'(\xi)$, следовательно автомат $F_{\cdot u(\xi)}$, представленный на рисунке 3, реализует умножение на многочлен $u(\xi)$.

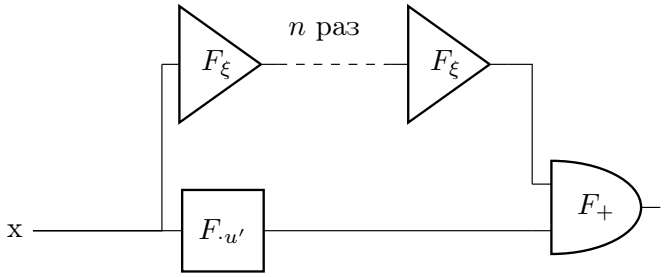


Рис. 3. Автомат $F_{\cdot u(\xi)}$

Для произвольного многочлена $u(\xi) \in E_2[\xi],\ u(\xi) = \sum\limits_{i=0}^{n} a_i \xi^i$ на рисунке 4 представлен автомат, реализующий сложение с данным многочленом.

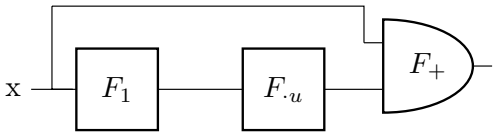


Рис. 4. Автомат $F_{+u(\xi)}$

Приведем автомат, реализующий функцию $f(x_1, x_2, \ldots, x_n)$, для которой справедливо $f(\alpha_1, \ldots, \alpha_n) = \sum\limits_{i=1}^{n} u_i(\xi)\alpha_i + u_0(\xi),\ \forall \alpha_1, \alpha_2, \ldots, \alpha_n \in R_2$. Опи-

санный автомат представлен на рисунке 5.

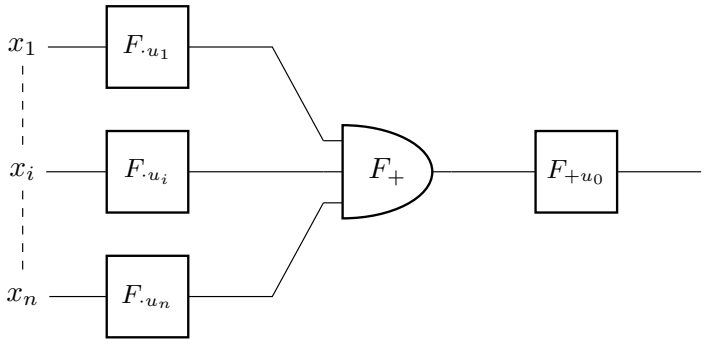


Рис. 5. Автомат $f(x_1, \ldots, x_n)$

□

4. A-полнота

Введем оператор A-замыкания [6] для класса линейных дефинитных автоматов LD_2.

Пусть $\alpha \in R_2$ — формальный степенной ряд, $\alpha = \sum\limits_{t=0}^{\infty} \alpha(t)\xi^t,\ \alpha(t) \in E_2$. Обозначим $\alpha|_{\bar{t}} = \sum\limits_{t=0}^{\bar{t}-1} \alpha(t)\xi^t$. Для автоматов $f(x_1, \ldots, x_n),\ g(x_1, \ldots, x_n) \in LD_2$ будем говорить, что они $\bar{t}$-эквивалентны, если для любых $\alpha_1, \ldots, \alpha_n \in R_2$ имеем $f(\alpha_1, \ldots, \alpha_n)|_{\bar{t}} = g(\alpha_1, \ldots, \alpha_n)|_{\bar{t}}$. Для некоторого $f \in LD_2$ и $\bar{t} \in \mathbb{Z}_+$ будем обозначать $E(f, \bar{t})$ множество всех $\bar{t}$-эквивалентных автомату f, с точностью до переименования переменных и удаления фиктивных переменных, линейных дефинитных автоматов.

Для некоторого множества $M \subseteq LD_2$ будем говорить, что автомат f *A-выразим* через множество M, если для любого $t,\ t \in \mathbb{Z}_+$ найдется автомат $f^{(t)},\ f^{(t)} \in S(M)$, что f t-эквивалентен $f^{(t)}$. Множество всех автоматов A-выразимых через $M,\ M \subseteq LD_2$ будем обозначать $A(M)$.

Лемма 1. *Пусть $M \subseteq LD_2$ и $S(M) = M$. Если существует $T \in \mathbb{Z}_+,\ T < \infty$, такое что для произвольного $f \in M$ имеем $E(f, T) \subseteq M$, то $A(M) = M$.*

Доказательство. Пусть произвольный автомат $g \in A(M)$. По определению A-выразимости, существует $f \in S(M)$ такой, что $g \in E(f, T)$. По условию $S(M) = M$, следовательно, $f \in M$ и $E(f, T) \subseteq M$. Имеем $g \in M$, что завершает доказательство леммы. □

Данная лемма позволяет упростить доказательство A-замкнутости для таких подмножеств LD_2, принадлежность которым для произвольного автомата $f \in LD_2$ можно определить по его поведению на словах фиксированной длины T.

Определение 1. *Определим семейство H_A подмножеств LD_2 следующим образом:*

$$T_0 = \{f(\dots) \mid f \in LD_2,\ f = \sum_{i=1}^{n} u_i(\xi)x_i + u_0(\xi)\ u\ \sum_{i=1}^{n} u_i(0)\cdot 0 + u_0(0) = 0\},$$

$$T_1 = \{f(\dots) \mid f \in LD_2,\ f = \sum_{i=1}^{n} u_i(\xi)x_i + u_0(\xi)\ u\ \sum_{i=1}^{n} u_i(0)\cdot 1 + u_0(0) = 1\},$$

$$V_1 = \{f(\dots) \mid f \in LD_2,\ f = \sum_{i=1}^{n} u_i(\xi)x_i + u_0(\xi)\ u \text{ среди чисел } u_i(0),$$
$$i = 1, \dots, n, \text{ не более одного отличного от нуля}\},$$

$$V_2 = \{f(\dots) \mid f \in LD_2,\ f = \sum_{i=1}^{n} u_i(\xi)x_i + u_0(\xi)\ u\ \sum_{i=1}^{n} u_i(0) = 1\},$$

$$U = \{f(\dots) \mid f \in LD_2,\ f = \sum_{i=1}^{n} u_i(\xi)x_i + u_0(\xi),\ u_i(\xi) = \sum_{j=0}^{\deg(u_i(\xi))} a_{j,i}\xi^j$$
$$u\ a_{1,i} = 0,\ \forall i = 1, \dots, n\}.$$

Классы T_0, T_1, V_1, V_2 определяют поведение автоматов только в начальный момент времени. Дадим неформальное определение каждого из классов. Множества $T_b,\ b \in \{0, 1\}$ можно описать как классы автоматов, которые в начальный момент времени сохраняют соответствующий элемент b поля E_2. Классы V_1 и V_2 содержат автоматы, которые в начальный момент времени существенно зависят не более чем от одного своего входа или от нечетного числа своих входов, соответственно.

Лемма 2. *Множества $T_0, T_1, V_1, V_2 \in H_A$ являются A-замкнутыми.*

Доказательство. В начальный момент времени для произвольного линейного дефинитного автомата, зависящего от n входов, для любых $\alpha_1, \dots, \alpha_n \in R_2$ имеем

$$f(\alpha_1, \dots, \alpha_n)|_1 = \sum_{i=1}^{n} u_i(0)\alpha_i(0) + u_0(0).$$

Следовательно, в начальный момент времени линейным дефинитным автоматом, зависящим от n входов, реализуется некоторая линейная булева

функция от n переменных. Данное отображение $E_2^n \to E_2$, порождаемое автоматом f, обозначим $f^{(0)}$. Покажем, что операции отождествления и подстановки для автоматов индуцируют соответствующие операции суперпозиции для реализуемых ими в начальный момент времени булевых функций.

Операция отождествления. Пусть, без ограничения общности, автомат $g(x_1, \ldots, x_{n-1})$ получается из автомата $f(x_1, \ldots, x_n)$ отождествлением переменных x_{n-1} и x_n, имеем

$$g^{(0)}(x_1^{(0)}, \ldots, x_{n-1}^{(0)}) = f^{(0)}(x_1^{(0)}, \ldots, x_{n-1}^{(0)}, x_{n-1}^{(0)}) = \\ = \sum_{i=1}^{n-2} u_i(0) x_i^{(0)} + (u_{n-1}(0) + u_n(0))\, x_{n-1}^{(0)} + u_0(0).$$

Операция подстановки. Пусть, без ограничения общности, автомат $g(x_1, \ldots, x_{n-1}, y_1, \ldots, y_k)$ получается из автомата $f(x_1, \ldots, x_n)$ подстановкой $h(y_1, \ldots, y_k)$ вместо переменной x_n, имеем

$$g^{(0)}(x_1^{(0)}, x_2^{(0)}, \ldots, x_{n-1}^{(0)}, y_1^{(0)}, y_2^{(0)}, \ldots, y_k^{(0)}) = \\ = f^{(0)}(x_1^{(0)}, x_2^{(0)}, \ldots, x_{n-1}^{(0)}, h(y_1^{(0)}, y_2^{(0)}, \ldots, y_k^{(0)})) = \\ = \sum_{i=1}^{n-1} u_i(0) x_i^{(0)} + u_n(0) \cdot \left(\sum_{j=1}^{k} u_j'(0) y_j^{(0)} + u_0'(0) \right) + u_0(0).$$

Покажем, что для каждого класса T_0, T_1, V_1, V_2 реализуемые ими в начальный момент булевы функции принадлежат некоторому замкнутому подклассу линейных функций [7]. Обозначим L_b — множество всех линейных функций, сохраняющих константу b, $b = 0, 1$, L_u — все линейные функции от одной переменной, L_s — все самодвойственные линейные функции.

- T_b, $b = 0, 1$. По определению для любого $f \in T_b$, $b = 0, 1$ имеем $f^{(0)}(b, \ldots, b) = b$, $b = 0, 1$, следовательно, $f^{(0)} \in L_b$, $b = 0, 1$.
- V_1: По определению для любого $f \in V_1$ функция $f^{(0)}$ существенно зависит не более чем от одной своей переменной, следовательно, $f^{(0)} \in L_u$.
- V_2: По определению для любого $f \in V_2$ функция $f^{(0)}$ существенно зависит от нечетного числа своих переменных, что соответствует определению линейной самодвойственной функции, следовательно, $f^{(0)} \in L_s$.

Было показано, что для любого Θ, $\Theta \in \{T_0, T_1, V_1, V_2\}$, имеем $S(\Theta) = \Theta$. Каждый из перечисленных классов определяет выход автомата только в

начальный момент времени, следовательно, по лемме 1, имеем $A(\Theta) = \Theta,\ \Theta \in \{T_0, T_1, V_1, V_2\}$. □

Класс U содержит автоматы, выход которых в первый момент времени не зависит от входа в начальный момент.

Лемма 3. *Множество $U \in H_A$ является A-замкнутым.*

Доказательство. Покажем, что существенные операции суперпозиции не выводят за пределы класса U.

Операция отождествления. Пусть, без ограничения общности, автомат $g(x_1, \ldots, x_{n-1})$ получается из автомата $f(x_1, \ldots, x_n)$ отождествлением переменных x_{n-1} и x_n, имеем

$$\begin{aligned} g(x_1, \ldots, x_{n-1}) &= f(x_1, \ldots, x_{n-1}, x_{n-1}) = \\ &= \sum_{i=1}^{n-2} u_i(\xi) x_i + (u_{n-1}(\xi) + u_n(\xi))\, x_{n-1} + u_0(\xi) = \\ &= \sum_{i=1}^{n-2} u_i(\xi) x_i + \left((u_{n-1}(0) + u_n(0)) + \xi^2 \left(\bar{u}_{n-1}(\xi) + \bar{u}_n(\xi)\right)\right) x_{n-1} + u_0(\xi). \end{aligned}$$

Операция подстановки. Пусть, без ограничения общности, автомат $g(x_1, \ldots, x_{n-1}, y_1, \ldots, y_k)$ получается из автомата $f(x_1, \ldots, x_n)$ подстановкой $h(y_1, \ldots, y_k)$ вместо переменной x_n, имеем

$$\begin{aligned} g(x_1, x_2, \ldots, x_{n-1}, y_1, y_2, \ldots, y_k) &= f(x_1, x_2, \ldots, x_{n-1}, h(y_1, y_2, \ldots, y_k)) = \\ &= \sum_{i=1}^{n-1} u_i(\xi) x_i + u_n(\xi) \left(\sum_{j=1}^{k} u'_j(\xi) y_j + u'_0(\xi) \right) + u_0(\xi) = \\ &= \sum_{i=1}^{n-1} u_i(\xi) x_i + \left(u_n(0) + \xi^2 \bar{u}_n(\xi)\right) \left(\sum_{j=1}^{k} \left(u'_j(0) + \xi^2 \bar{u}'_j(\xi)\right) y_j \right) + u''_0(\xi) = \\ &= \sum_{i=1}^{n-1} u_i(\xi) x_i + \sum_{j=1}^{k} \left(u''_j(0) + \xi^2 \bar{u}''_j(\xi)\right) y_j + u''_0(\xi). \end{aligned}$$

Было показано, что $S(U) = U$. Класс U определяет выход автомата только в первый момент времени, следовательно, по лемме 1, имеем $A(U) = U$.

□

Сформулируем основное утверждение.

Теорема 2. *Пусть $M \subset LD_2$, тогда $A(M) = LD_2 \iff M \not\subseteq \Theta,\ \forall \Theta \in H_A$.*

Доказательство. Прямое утверждение следует из A-замкнутости классов $T_0,\ T_1,\ V_1,\ V_2,\ U$, каждый из которых не совпадает с LD_2.

Докажем обратное утверждение. Для каждого класса $\Theta \in H_A$ в M существует хотя бы один автомат f_Θ, такой что $f_\Theta \notin \Theta$. Введем соответствующие обозначения: $f_{T_0}, f_{T_1}, f_{V_1}, f_{V_2}, f_U$, которые будем ассоциировать с некоторым автоматом, содержащимся в соответствующем множестве $M \setminus \Theta,\ \Theta \in H_A$.

Покажем, что в $A(M)$ существуют автоматы $f_{\omega_b}(x_1),\ b = 0, 1$, которые 2-эквивалентны константам и в начальный момент времени принимают значения 0 и 1 соответственно:

$$f_{\omega_b}(x_1) = \xi^2 \bar{u}_{1,b}(\xi)x_1 + (b + \xi \bar{u}_{0,b}(\xi)),\ b = 0, 1. \tag{2}$$

Рассмотрим автомат

$$f_{V_2}(x_1, \ldots, x_n) = \sum_{i=1}^{n} u_i(\xi)x_i + u_0(\xi),\ \sum_{i=1}^{n} u_i(0) = 0,$$

следовательно,

$$f_c(x_1) = f_{V_2}(x_1, \ldots, x_1) = \xi u_1'(\xi)x_1 + (a_0 + \xi u_0'(\xi)).$$

В зависимости от значения коэффициента a_0, положим:

- $a_0 = 0$: $f_0(x_1) = f_c(x_1)$ и $f_1(x_1) = f_{T_0}(f_c(x_1), \ldots, f_c(x_1))$,
- $a_0 = 1$: $f_0(x_1) = f_{T_1}(f_c(x_1), \ldots, f_c(x_1))$ и $f_1(x_1) = f_c(x_1)$.

Искомые автоматы могут быть получены при помощи подстановки:

$$f_{\omega_b}(x_1) = f_b(f_0(x_1)) = \xi^2 \bar{u}_{1,b}(\xi)x_1 + (b + \xi \bar{u}_{0,b}(\xi)).$$

Покажем, что сумматор от трех переменных содержится в A-замыкании множества M. Рассмотрим автомат $f_{V_1}(x_1, x_2, \ldots, x_n)$. По определению, в начальный момент времени он существенно зависит хотя бы от двух своих переменных. Без ограничения общности будем считать, что x_1 и x_2 содержатся среди этих переменных. Отождествим остальные переменные f_{V_1} и получим автомат:

$$f(x_1, x_2, x) = f_{V_1}(x_1, x_2, x, \ldots, x) = (1 + \xi u_1(\xi))x_1 + (1 + \xi u_2(\xi))x_2 + f'(x).$$

Используя подстановку $f(f(x, x_1, x), f(x_2, x, x), x)$, получим автомат

$$h(x_1, x_2, x) = (1 + \xi u_1(\xi))(1 + \xi u_2(\xi))x_1 + (1 + \xi u_1(\xi))(1 + \xi u_2(\xi))x_2 + \\ + h'(x) = (1 + \xi u'(\xi))x_1 + (1 + \xi u'(\xi))x_2 + h'(x).$$

Индукцией по $s \in \mathbb{Z}_+$ покажем, что для любого $s \in \mathbb{Z}_+$ автомат

$$h_s(x_1, x_2, x) = (1 + \xi u'(\xi))^{2^s} x_1 + (1 + \xi u'(\xi))^{2^s} x_2 + h'_s(x),$$

принадлежит $A(M)$ некоторого $h'_s(x)$. База индукции

$$h_0(x_1, x_2, x) = h(x_1, x_2, x) = (1 + \xi u'(\xi))^{2^0} x_1 + (1 + \xi u'(\xi))^{2^0} x_2 + h'(x).$$

Для доказательства шага индукции рассмотрим подстановку

$$\begin{aligned} &h_{s+1}(x_1, x_2, x) = h_s(h_s(x_1, x_2, x), x, x) = \\ &= (1 + \xi u'(\xi))^{2^s} (1 + \xi u'(\xi))^{2^s} x_1 + (1 + \xi u'(\xi))^{2^s} (1 + \xi u'(\xi))^{2^s} x_2 + \\ &+ h'_{s+1}(x) = (1 + \xi u'(\xi))^{2^{s+1}} x_1 + (1 + \xi u'(\xi))^{2^{s+1}} x_2 + h'_{s+1}(x). \end{aligned}$$

Воспользуемся данным утверждением, и для некоторого $s \in \mathbb{N}$ рассмотрим подстановку

$$\begin{aligned} &f_s(x_1, x_2, x_3, x) = h_s(h_s(x_1, x_2, x), x_3, x) = (1 + \xi u'(\xi))^{2^{s+1}} x_1 + \\ &+ (1 + \xi u'(\xi))^{2^{s+1}} x_2 + (1 + \xi u'(\xi))^{2^s} x_3 + (1 + \xi u'(\xi))^{2^s} f'_s(x) + f'_s(x). \end{aligned}$$

Учитывая, что $(a + b)^2 = a^2 + b^2$ в поле характеристики два, получаем:

$$\begin{aligned} &f_s(x_1, x_2, x_3, x) = x_1 + x_2 + x_3 + \\ &+ \xi^{2^{s+1}} u''(\xi) x_1 + \xi^{2^{s+1}} u''(\xi) x_2 + \xi^{2^s} \tilde{u}(\xi) x_3 + \xi^{2^s} f'_s(x). \end{aligned} \tag{3}$$

Утверждение (3) верно для любого $s \in \mathbb{N}$, следовательно сумматор от трех переменных $F_+(x_1, x_2, x_3) \in A(M)$.

Докажем, что для любого $s \in \mathbb{N}$ автомат $g_s(x_1)$, s-эквивалентный задержке с нулевым начальным состоянием, принадлежит $A(M)$. Рассмотрим автомат

$$f_U(x_1, \ldots, x_n) = \sum_{i=1}^{n} u_i(\xi) x_i + u_0(\xi),\ u_i(\xi) = \sum_{j=0}^{\deg(u_i(\xi))} a_{j,i} \xi^j.$$

$f_U \notin U$, следовательно, $a_{1,i} = 1$ для некоторого i, $1 \le i \le n$. Без ограничения общности будем считать, что $a_{1,1} = 1$. Используя операцию отождествления переменных, получим:

$$f_U(x_1, x_2, \ldots, x_2) = g(x_1, x_2) = u_1(\xi) x_1 + u'_2(\xi) x_2 + u_0(\xi).$$

Воспользуемся (2) и рассмотрим подстановку:

$$g'(x_1) = g(x_1, f_{\omega_0}(x_1)) = (a + \xi + \xi^2 u'_1(\xi)) x_1 + (b + c\xi + \xi^2 u'_0(\xi)).$$

В зависимости от значения коэффициентов a и b, используя (3), положим:

- $a = 0,\ b = 0 : g_1(x_1) = g'(x_1)$,
- $a = 0,\ b = 1 : g_1(x_1) = g'(x_1) + f_{\omega_0}(x_1) + f_{\omega_1}(x_1)$,
- $a = 1,\ b = 0 : g_1(x_1) = g'(x_1) + x_1 + f_{\omega_0}(x_1)$,
- $a = 1,\ b = 1 : g_1(x_1) = g'(x_1) + x_1 + f_{\omega_1}(x_1)$.

Следовательно:

$$g_1(x_1) = (\xi + \xi^2 u''_{1,1}(\xi))x_1 + (c\xi + \xi^2 u''_{0,1}(\xi)).$$

База индукции для $s = 2$ реализуется подстановкой

- $c = 0 : g_2(x_1) = g_1(x_1)$,
- $c = 1 : g_2(x_1) = g_1(x_1) + g_1(f_{\omega_0}(x_1)) + g_1(f_{\omega_1}(x_1))$.

Докажем шаг индукции для $g_{s+1}(x_1)$, имеем:

$$g_s(x_1) = (\xi + a\xi^s + \xi^{s+1} u_{1,s}(\xi))x_1 + (b\xi^s + \xi^{s+1} u_{0,s}(\xi)),$$

тогда

$$\tilde{g}_s(x_1) = \underbrace{g_s(g_s(\dots g_s(x_1)\dots))}_{s \text{ раз}} = (\xi^s + \xi^{s+1}\tilde{u}_1(\xi))x_1 + \xi^s \tilde{u}_0(\xi).$$

В зависимости от значения коэффициентов a и b, построим автомат $g_{s+1}(x_1)$:

- $a = 0,\ b = 0$: $g_{s+1}(x_1) = g_s(x_1)$,
- $a = 0,\ b = 1$: $g_{s+1}(x_1) = g_s(x_1) + \tilde{g}_s(f_{\omega_0}(x_1)) + \tilde{g}_s(f_{\omega_1}(x_1))$,
- $a = 1,\ b = 0$: $g_{s+1}(x_1) = g_s(x_1) + \tilde{g}_s(x_1) + \tilde{g}_s(f_{\omega_0}(x_1))$,
- $a = 1,\ b = 1$: $g_{s+1}(x_1) = g_s(x_1) + \tilde{g}_s(x_1) + \tilde{g}_s(f_{\omega_1}(x_1))$.

Так как утверждение верно для любого $s \in \mathbb{N}$, то задержка с нулевым начальным состоянием принадлежит $A(M)$.

Покажем, что константы 0 и 1 принадлежат $A(M)$. Рассмотрим, полученные в (2), автоматы

$$f_{\omega_b}(x_1) = \xi^2 \bar{u}_{1,b}(\xi)x_1 + (b + \xi\bar{u}_{0,b}(\xi)).$$

Покажем, что для любого $s \in \mathbb{N}$ автомат

$$f_{b,s}(x_1) = \xi^s \bar{u}_{1,b,s}(\xi)x_1 + (b + \xi^s \bar{u}_{0,b,s}(\xi)) \in A(M).$$

Положим $f_{b,1}(x_1) = f_{\omega_b}(x_1)$. Пусть

$$f_{b,s}(x_1) = \xi^s \left(a + \xi\bar{u}_{1,b,s}(\xi)\right) x_1 + \left(b + \xi^s(c + \xi\bar{u}_{0,b,s})(\xi)\right),$$

тогда

- $a = 0$: $f_{b,s+1}(x_1) = f_{b,s}(x_1) + \xi^s f_{\omega_0}(x_1) + \xi^s f_{\omega_c}(x_1)$,
- $a = 1$: $f_{b,s+1}(x_1) = f_{b,s}(x_1) + \xi^s x_1 + \xi^s f_{\omega_c}(x_1)$,

Так как утверждение верно для любого $s \in \mathbb{N}$, то константы 0 и 1 принадлежат $A(M)$.
Имеем $x_1 + x_2 = x_1 + x_2 + 0$, следовательно в $A(M)$ содержится полная система $\{x_1 + x_2, \xi x_1, 1\}$. □

Следствие 1. *Множество замкнутых классов H_A является приведенной критериальной системой относительно оператора A-замыкания в классе линейных дефинитных автоматов.*

Доказательство. Для каждого класса $\Theta \in H_A$ приведем набор автоматов, которые отличают Θ от любого другого класса $\Theta' \in H_A \setminus \{\Theta\}$.

$$\bar{T}_0 = \{0,\ x_1 + x_2,\ \xi x_1\},$$
$$\bar{T}_1 = \{1,\ x_1 + x_2 + x_3,\ \xi x_1 + 1\},$$
$$\bar{V}_1 = \{0,\ 1,\ \xi x_1\},$$
$$\bar{V}_2 = \{x_1 + 1,\ x_1 + x_2 + x_3,\ (1 + \xi)x_1\}$$
$$\bar{U} = \{1,\ x_1 + x_2\}.$$

□

5. Заключение

В данной работе для множества линейных дефинитных автоматов была получена система предполных классов относительно оператора A-замыкания. Дальнейшие исследования по данной теме могут заключаться в исследовании полноты множества относительно оператора Σ-замыкания.

В заключение автор выражает особую признательность Часовских А. А. за научное руководство.

Список литературы

[1] Часовских А. А., “О полноте в классе линейных автоматов”, *Математические вопросы кибернетики*, **3**, Наука, М., 1991, 140–166.

[2] Часовских А. А., “Проблема полноты в классах линейных автоматов”, *Интеллектуальные системы. Теория и приложения*, **22**:2 (2018), 151–154.

[3] Кудрявцев В. Б., Алешин С. В., Подколзин А. С., *Введение в теорию автоматов*, Наука, М., 1985, 320 с.

[4] Алешин С. В., Бабин Д. Н., Часовских А. А., "Автоматы: полнота, выразимость, применение", *Математические вопросы кибернетики*, **22**, ФИЗМАТЛИТ, М., 2024, 223–275.

[5] Гилл А., *Линейные последовательностные машины*, Наука, М., 1974, 288 с.

[6] Буевич В. А., "Об алгоритмической неразрешимости распознавания A-полноты для ограниченно-детерминированных функций", *Матем. заметки*, **11**:6 (1972), 687–697.

[7] Яблонский С. В., Гаврилов Г. П., Кудрявцев В. Б., *Функции алгебры логики и классы Поста*, Наука, М., 1966, 120 с.

Approximation Completeness of Linear Definite Automata

Moldovanov I.V.

In linear definite automata, the output signals at each moment depend only on a bounded number of the most recent input values.

This paper studies functional completeness with respect to the operator of approximation closure for the class of linear definite automata over the two-element field. For this class of automata, a completeness criterion is obtained, formulated in terms of a system of precomplete classes.

Keywords: approximation closure, linear automata, definite automata.

References

[1] Chasovskikh A. A., "On completeness in the class of linear automata", *Mathematical Problems of Cybernetics*, **3**, Nauka, Moscow, 1991, 140–166 (In Russian).

[2] Chasovskikh A. A., "The completeness problem in classes of linear automata", *Intelligent Systems. Theory and Applications*, **22**:2 (2018), 151–154 (In Russian).

[3] Kudryavtsev V. B., Aleshin S. V., Podkolzin A. S., *Introduction to Automata Theory*, Nauka, Moscow, 1985 (In Russian), 320 pp.

[4] Aleshin S. V., Babin D. N., Chasovskikh A. A., "Automata: completeness, expressibility, applications", *Mathematical Problems of Cybernetics*, **22**, Fizmatlit, Moscow, 2024, 223–275 (In Russian).

[5] A. Gill, *Linear sequential circuits: Analysis, Synthesis, and Applications*, McGraw-Hill, New York, 1967, 215 pp.

[6] Buevich V. A., "On the algorithmic undecidability of A-completeness for the boundedly determinate functions", *Mathematical Notes of the Academy of Sciences of the USSR*, **11** (1972), 417–421.

[7] S. V. Yablonskii, G. P. Gavrilov, V. B. Kudryavtsev, *Functions of Boolean algebra and Post classes*, Nauka, Moscow, 1966 (In Russian), 120 pp.

О мощности порождающих множеств в классе автоматов с линейной функцией выходов

Ф. Р. Юсупов[1]

Аннотация: Класс конечных автоматов конечно порожден по операциям композиции [1]. Важными его подклассами являются классы автоматов с линейными функциями выходов. Набор операций над автоматами, состоящий из подстановки переменных, подстановки автоматов и обратной связи, мы называем ограниченной композицией. По операциям ограниченной композиции класс одноместных автоматов замкнут. В настоящей работе показано, что в классе одноместных конечных автоматов с линейными функциями выходов и операциями ограниченной композиции отсутствуют конечные полные множества, но автоматы из этого класса, реализуемые схемами с не более чем одной задержкой, порождают этот класс.

Ключевые слова: конечные автоматы, операции композиции для автоматов, конечные автоматы с линейными выходами

1. Определения и преамбула

Мы используем определение инициального конечного автомата из работы [1].

Определение 1. *Конечный автомат* $\mathfrak{A}$ *– это шестерка* $(A, B, Q, \Phi, \Psi, \overline{q}_0)$, *где:*
$A = E_2^n$ - *входной алфавит*
$B = E_2^m$ - *выходной алфавит*
$Q = E_2^s$ - *алфавит состояний*
$\Phi : A \times Q \to Q$ - *функция переходов*
$\Psi : A \times Q \to B$ - *функция выходов*
$\overline{q}_0 \in \{0,1\}^s$ - *начальное состояние.*

Конечный автомат $\mathfrak{A}$, заданный шестеркой из определения 1, имеет n входов, m выходов и функционирует в дискретном времени t, $t = 0, 1, \ldots$. Полагаем: $\overline{q}(0) = \overline{q}_0$. Если в момент времени t заданы значения вектора состояния $\overline{q}(t)$ и входов автомата $\overline{x}(t) = (x_1(t), x_2(t), ..., x_n(t))$,

[1] *Юсупов Феликс Ренатович* — аспирант каф. математической теории интеллектуальных систем мех.-мат. ф-та МГУ, e-mail: fel.99@list.ru

Yusupov Feliks Renatovich — graduate student, Lomonosov Moscow State University, Faculty of Mechanics and Mathematics, Chair of Mathematical Theory of Intellectual Systems.

то значения вектора состояний $\overline{q}(t+1)$ в момент $t+1$ и выходов $\overline{y}(t) = (y_1(t), y_2(t), ..., y_m(t))$ в момент t определяются, соответственно, равенствами:

$$\overline{q}(t+1) = \Phi(\overline{x}(t), \overline{q}(t)), \tag{1}$$

$$\overline{y}(t) = \Psi(\overline{x}(t), \overline{q}(t)). \tag{2}$$

Конечный автомат может быть задан схемой из элементов $\vee, \wedge, \neg, q^0$(задержка с начальным состоянием 0), q^1 (задержка с начальным состоянием 1), множество которых обозначим M^0.

Определение 2. *Схемой из элементов множества M^0 называется ориентированный граф, полустепени захода вершин которого не превосходят двух, каждой вершине которого приписан:*
- элемент множества $X = \{x_i | i = 1, 2, \dots\}$ (входная переменная), если полустепень захода вершины 0,
- один из элементов множества $\{\neg, q^0, q^1\}$, если полустепень захода вершины 1,
- конъюнкция $\wedge$ либо дизъюнкция $\vee$, если полустепень захода вершины 2, некоторым вершинам приписана переменная из множества выходных переменных $Y = \{y_i | i = 1, 2, \dots\}$, причем, разным вершинам – разные выходные переменные, а любой контур графа проходит через задержку. Вершины графа, которым приписаны входные (выходные) переменные, мы называем входами (выходами) схемы, а каждый выход схемы выделяем еще исходящим из него полуребром.

Заметим, что мы не нумеруем входы элементов, так как перестановка входов не изменяет функции $\wedge$ и $\vee$.

Функционирование схемы определяется во времени. Начальные состояния задержек составляют вектор $\overline{q}(0)$. Если в момент времени t определены состояния задержек $\overline{q}(t)$ и заданы значения входных переменных $\overline{x}(t)$, то значения на выходах элементов $\vee, \wedge, \neg$ и на входах задержек определяются также, как и в [2] для схемы из функциональных элементов с входами $\overline{x}(t), \overline{q}(t)$ и с выходами схемы, к которым добавлены выходы элементов, соединенных с входами задержек. Набор значений, получаемых на выходах схемы в момент t, составляет вектор $\overline{y}(t)$, а состоянием схемы в момент $t+1$ является набор значений входов задержек $\overline{q}(t+1)$.

Таким образом, для некоторых отображений Φ и Ψ, $\Phi : E_2^n \times E_2^s \to E_2^s$, $\Psi : E_2^n \times E_2^s \to E_2^m$, имеют место равенства (1), (2), и схема из элементов множества M^0 задает некоторый конечный автомат $\mathfrak{A} = (E_2^n, E_2^s, E_2^m, \Phi, \Psi, \overline{q}(0))$.

С другой стороны, каждый автомат $\mathfrak{A} = (E_2^n, E_2^m, E_2^s, \Phi, \Psi, \overline{q}(0))$ можно реализовать схемой, как показано на рисунке 1.

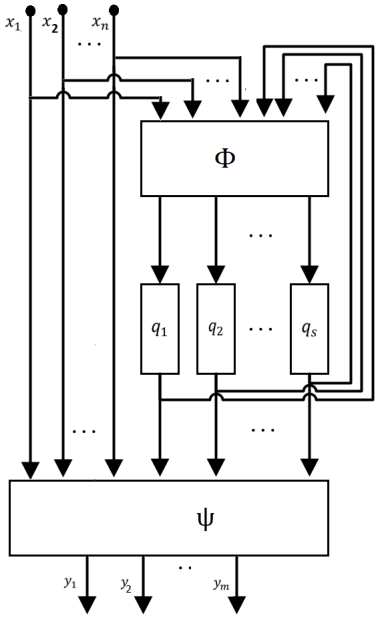


Рис. 1. Схема автомата $\mathfrak{A} = (E_2^n, E_2^s, E_2^m, \Phi, \Psi, \overline{q}(0))$.

Определим два набора операций над автоматами, используя схемы автоматов: операции ограниченной композиции и операции пополненной композиции. Пусть автоматы $\mathfrak{A}_1$, $\mathfrak{A}_2$ заданные шестерками $(A_i, B_i, Q_i, \Phi_i, \Psi_i, \overline{q}_{i,0})$, где $A_i = E_2^{n_i}$, $B_i = E_2^{m_i}$, $Q_i = E_2^{s_i}$, $i = 1, 2$, реализуются, соответственно, схемами S_1 и S_2, изображенными на рисунке 2.

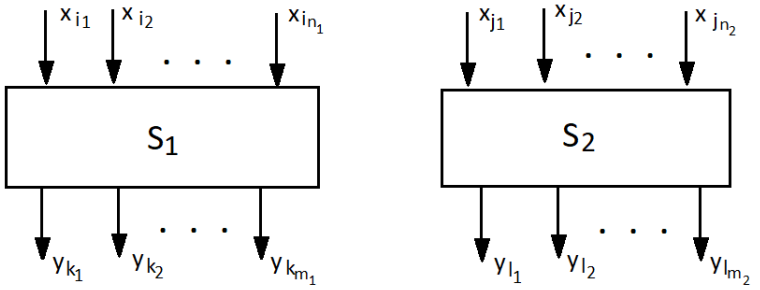


Рис. 2. Схемы автоматов $\mathfrak{A}_1$, $\mathfrak{A}_2$.

Определение 3. *Предположим, что* $\{i_1, \ldots, i_{n_1}\} \cap \{j_1, \ldots, j_{n_2}\} = \emptyset$ *и* $\{k_1, \ldots, k_{m_1}\} \cap \{l_1, \ldots, l_{m_2}\} = \emptyset$.

Операции ограниченной композиции (K_c).

1) Подстановка переменных.*[3] Пусть σ – взаимнооднозначное отображение индексов переменных $i_1, i_2, \ldots, i_{n_1}$ на множество индексов $i'_1, i'_2, \ldots, i'_{n_1}$. Заменяя в схеме S_1 переменные $x_{i_1}, x_{i_2}, \ldots, x_{i_{n_1}}$ на переменные $x_{i'_1}, x_{i'_2}, \ldots, x_{i'_{n_1}}$, получим схему автомата $\mathfrak{A}_3$, изображенную на рисунке 3.*

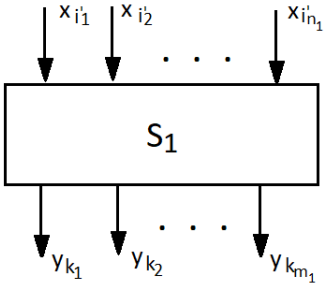


Рис. 3. Схема автомата $\mathfrak{A}_3$, полученного из $\mathfrak{A}_1$ подстановкой переменных.

Нетрудно видеть, что операция подстановки переменных включает операции переименования переменных, перестановки переменных и отождествления переменных.

2) Подстановка автоматов. *Пусть количество входов n_1 автомата $\mathfrak{A}_1$ не меньше количества выходов m_2 автомата $\mathfrak{A}_2$. Тогда, соединяя дугами выход y_{l_r} схемы S_2 с каждым элементом схемы S_1, который соединен дугой с входом x_{i_r}, $r = 1, \ldots, m_2$, и удаляя все вершины x_{i_r}, $r = 1, \ldots, m_2$, вместе с дугами, выходящими из этих вершин, получим схему S_4 автомата $\mathfrak{A}_4$. Входами схемы S_4 являются все входы S_2 и входы x_{m_2+1}, $x_{m_2+2}, \ldots$, x_{n_1} схемы S_1, а ее выходами – выходы схемы S_1. Автомат $\mathfrak{A}_4$ получен подстановкой автомата $\mathfrak{A}_2$ на автомат $\mathfrak{A}_1$. Схема S_4 представлена на рисунке 4. Жирные линии на рисунке указывают на возможность наличия нескольких дуг.*

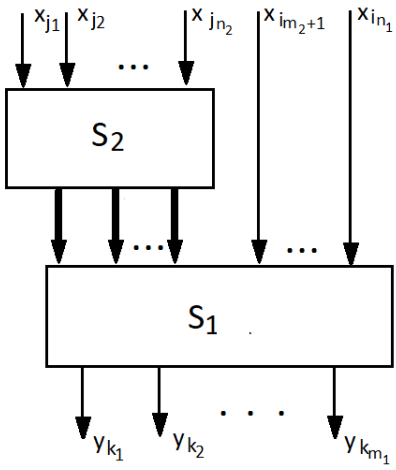


Рис. 4. Схема автомата $\mathfrak{A}_4$, полученного подстановкой автомата $\mathfrak{A}_2$ на автомат $\mathfrak{A}_1$.

Отметим, что использование операции подстановки переменных снимает ограничение на выбор входов схемы S_1 для операции подстановки автоматов.

3) Обратная связь. *Если каждый ориентированный путь от входа x_{n_1} схемы S_1 к ее выходу y_{k_i} проходит через задержку, то к S_1 добавим дуги из вершины y_{k_i}, в вершины, соединенные дугами с x_{n_1}. При этом удалим вершину x_{n_1} и ведущие из нее дуги. Получим схему S_5 нового автомата $\mathfrak{A}_5$, изображенную на рисунке 5, на котором жирной линией обозначена возможность наличия нескольких дуг. Будем говорить, что автомат $\mathfrak{A}_5$ получен применением операции обратной связи к автомату $\mathfrak{A}_1$.*

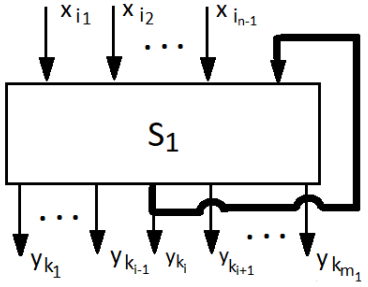


Рис. 5. Схема автомата $\mathfrak{A}_5$, полученного применением операции обратной связи к автомату $\mathfrak{A}_1$.

Использование операции подстановки переменных снимает ограничение на выбор входа схемы S_1 для операции обратной связи.

Добавляя к операциям ограниченной композиции, операцию объединения двух автоматов, получаем операции *пополненной композиции*:
4) *Объединение автоматов.* Автоматы $\mathfrak{A}_1$ и $\mathfrak{A}_2$ могут быть объединены в один автомат $\mathfrak{A}_6$. При этом входами схемы S_6, реализующей $\mathfrak{A}_6$, является объединение входов S_1 и S_2, а выходами S_6 – объединение выходов S_1 и S_2 (рисунок 6).

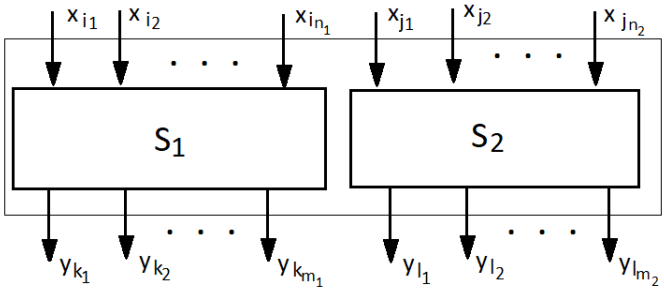


Рис. 6. Схема автомата $\mathfrak{A}_6$, полученного объединением автоматов $\mathfrak{A}_1$ и $\mathfrak{A}_2$.

Для автомата с линейной функцией выходов найдутся такие матрицы из $GF(2)$ T и R, где $T = (t_{ij})_{m\times n}, R = (r_{ij})_{m\times s}$, что для его функции выходов выполнено:

$$\Psi(\overline{x}, \overline{q}) = T\overline{x}^* + R\overline{q}^*.$$

Через $P_{F.D.L.}$ обозначим множество всех конечных автоматов с линейной функцией выходов.

Обозначим через $K_c(M)$ множество всех автоматов, полученных из множества $M, M \subseteq P_{F.D.L.}$, с использованием операций ограниченной композиции, через $K_s(M)$ обозначим множество автоматов, полученных из множества M при помощи операций пополненной композиции.

Далее, с использованием введенных определений, будут сформулированы и доказаны результаты этой работы.

2. Результаты и их доказательства

Из последних двух определений вытекает утверждение:

Утверждение 1. $K_c(M) \subset K_s(M)$.

Нетрудно видеть, что для класса $P_{F.D.L}$ имеет место следующая лемма.

Лемма 1. *Автоматы с одним выходом образуют замкнутый класс по операциям ограниченной композиции.*

Имеет место:

Утверждение 2. *В классе конечных автоматов с операциями пополненной композиции автоматы из $P_{F.D.L}$ выражаются через элементы конечного множества M^0.*

Доказательство. Для заданного автомата с функциями переходов Φ и выходов Ψ с использованием операций пополненной композиции из элементов множества M^0 можно построить автомат $(A, B, Q, \Phi, \Psi, \overline{q}_0)$ со схемой, представленной на рисунке 1 [1]. □

Конечный автомат $(A, B, Q, \Phi, \Psi, \overline{q}_0)$ реализует ограниченно-детерминированную функцию [1], преобразующую последовательности из A^∞ в последовательности из B^∞. Мы не будем различать автоматы, реализующие одну и ту же ограниченно-детерминированную функцию. При этом, автомат будем отождествлять с этой функцией.

Далее рассмотрим класс автоматов $P^1_{F.D.L.}$ с одним линейным выходом. Покажем, что в классе $P^1_{F.D.L}$ отсутствует конечный базис по операциям ограниченной композиции. Имеет место:

Теорема 1. $\forall M, M \subset P^1_{F.D.L}, |M| < \infty$, *выполнено:* $K_c(M) \neq P^1_{F.D.L}$.

Доказательство. Возьмем произвольное $M \subseteq P^1_{F.D.L}, |M| < \infty$.

$$M = \{f_1, ..., f_s\}, f_i = f_i(x_{i,1}, ..., x_{i,n_i}),$$

$$n = \max\{n_i | i = \overline{1, s}\}.$$

Рассмотрим автомат $f(x_1, \ldots, x_{n+1})$, получаемый подстановкой конъюнкции переменных $x_1, \ldots, x_{n+1}$ на задержку с начальным состоянием 0,

$$f(x_1, \ldots, x_{n+1}) = q^0(x_1 \wedge \cdots \wedge x_{n+1}).$$

Имеем: $f(x_1, \ldots, x_{n+1}) \in P^1_{F.D.L}$.

Для выхода этого автомата в момент времени 1 имеет место равенство: $y(1) = x_1(0) \wedge x_2(0) \wedge ... \wedge x_{n+1}(0)$. Далее покажем, что $f \notin K_c(M)$. Рассмотрим выход у автомата из M. Пусть этот автомат имеет n' входов, $n' \leq n$. Он задается системой канонических уравнений [1]:

$$\begin{cases} \overline{q}(0) = \overline{q}_0, \\ \overline{q}(t+1) = \Phi(\overline{x}(t), \overline{q}(t)), \\ y(t) = \psi(\overline{x}(t), \overline{q}(t)). \end{cases}$$

В момент времени $t = 0$: $y(0) = \psi(\overline{x}(0), \overline{q}(0)) = \psi(\overline{x}(0), \overline{q}_0)$
В момент времени $t = 1$: $y(1) = \psi(\overline{x}(1), \overline{q}(1)) = \psi(\overline{x}(1), \Phi(\overline{x}(0), \overline{q}(0)))$
$\overline{q}(0)$ - это вектор из констант. $\overline{x}(0) = (x_1(0), \ldots, x_{n'}(0))$. В то время как Φ может быть любым булевским оператором. Распишем подробнее: пусть рассматриваемый автомат реализуется схемой с l задержками. Тогда: $\overline{q}(t) = (q_1(t), ..., q_l(t))$, $\Phi = (\varphi_1, ..., \varphi_l)$. Возвращаясь к $y(1)$:
$y(1) = \psi(x_1(1), \ldots, x_{n'}(1), \varphi_1(x_1(0), \ldots, x_{n'}(0), q_1(0), \ldots, q_l(0)),$
$\varphi_2(x_1(0), \ldots, x_{n'}(0), q_1(0), \ldots, q_l(0)), \ldots, \varphi_l(x_1(0), \ldots, x_{n'}(0), q_1(0), \ldots, q_l(0)))$.

Далее
$y(1) = \psi(x_1(1), \ldots, x'_n(1), \Pi_1, \Pi_2, \ldots, \Pi_l)$, где Π_i - полином Жегалкина для $\varphi_i(x_1(0), \ldots, x'_n(0), q_1(0), \ldots, q_l(0))$

Функция ψ линейна относительно всех своих переменных, поэтому:
$y(1) = c_1\Pi_1 + c_2\Pi_2 + \cdots + c_l\Pi_l + d_1x_1(1) + d_2x_2(1) + \cdots + d_{n'}x_{n'}(1)$.

Для степени Π_i по компонентам вектора $\overline{x}(0)$ имеем: $\deg_{\overline{x}(0)}\Pi_i \leq n'$. Так как степени у каждого полинома Жегалкина Π_i не превосходят n', следовательно: $\deg_{\overline{x}(0)} y(1) \leq n'$.

То есть, мы показали, что для выхода в момент $t = 1$ автомата из M выполнено: $deg_{\overline{x}_0} y(1) \leq n$.
Далее несложно показать, что если взять две функции f_1 и f_2 из $P_{F.D.L}$, степени выходов которых по $\overline{x}(0)$ в момент времени $t = 1$ не более n, то, применяя однократно операции 1-3 из определения операций $K_c(M)$, получаем функции с таким же свойством. Следовательно, $f(x_1, \ldots, x_{n+1}) \notin K_c(M)$. □

Мы показали, что класс $P^1_{F.D.L}$ не является конечнопорожденным по операциям ограниченной композиции.

Далее, введем множество M_0, состоящее из сумматора по модулю 2, константы 1 и всех автоматов, получаемыми подстановкой булевской функции на задержку (рисунок 7, где φ - булевская функция)

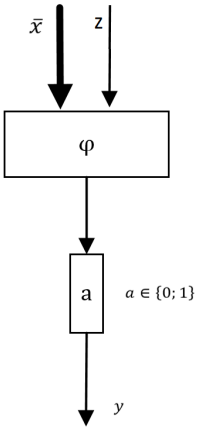


Рис. 7. Автоматы, содержащиеся в M_0.

Теорема 2. $P^1_{F.D.L.} = K_c(M_0)$.

Доказательство. Докажем теорему индукцией по числу задержек в схеме автомата. Пусть $f \in P^1_{F.D.L}$.
1) Если задержек 0, то f - линейная функция алгебры логики, которую нетрудно получить, используя операции K_c из сумматора и константы 1.
2) Если задержка одна, то рассмотрим каноническую схему этого автомата (см. рисунок 8).

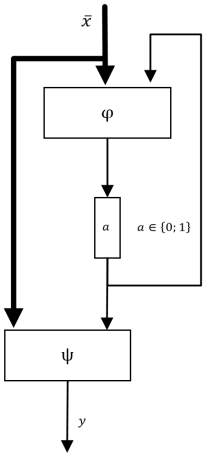
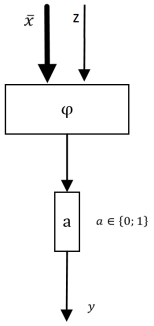


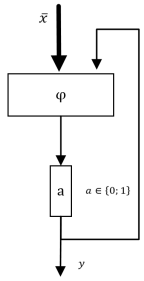
Рис. 8. Каноническая схема автомата с одной задержкой.

Автомат, схема которого представлена на рисунке 8, строится с использованием операций ограниченной композиции из элементов множества

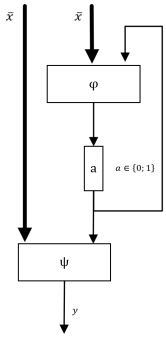
M_0. На рисунке 9 представлена последовательность построения этого автомата из автоматов множества M_0. Жирные линии для соединений схемы, возникающие при использовании операций подстановки и обратной связи, в дальнейшем не прорисовываем.



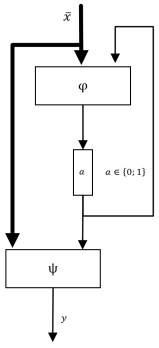
Шаг 1: строим схему для функции φ и подстановляем ее на задержку.



Шаг 2: используем операцию обратной связи.



Шаг 3: строим функцию ψ и используем подстановку.



Шаг 4: отождествляем соответствующие переменные.

Рис. 9. Построение автомата с одной задержкой.

Мы построили канонический автомат с одной задержкой, используя операции сокращенной композиции. Получили базу индукции. Далее рассмотрим **индукционное предположение:** Если автомат f из $P^1_{F.D.L.}$ можно реализовать схемой, содержащей не более $s, s \geq 1$, задержек, то $f \in K_c(M_0)$.

Индукционный переход: Пусть схема S содержит $s+1$ задержку. Тогда автомат f, реализуемый схемой S, можно задать системой канонических уравнений:

$$\begin{cases} q_0 - \text{вектор начальных состояний задержек,} \\ q_i(t+1) = \varphi_i(\overline{x}(t), q_1(t), q_2(t), ..., q_{s+1}(t)), i = 1, ..., s+1 \\ y(t) = \psi(\overline{x}(t), q_1(t), q_2(t), ..., q_{s+1}(t)). \end{cases}$$

Для заданного k, $k \in \{1, 2..., s+1\}$, по предположению индукции, в $K_c(M_0)$ содержатся автоматы $h_j\,(\overline{x}, z)$, заданные следующими системами канонических уравнений:

$$\begin{cases} q_i(t+1) = \varphi_i(\overline{x}(t), q_1(t), q_2(t), ..., q_{k-1}(t), z, q_{k+1}(t), ..., q_{s+1}(t)), \\ i = 1, ..., k-1, k+1, ..., s+1, \\ y(t) = q_j(t). \end{cases}$$

$j = 1, ..., k-1, k+1, ..., s+1.$

Операциями ограниченной композиции из автоматов $h_j\,(\overline{x}, z)$, $j = 1, ..., k-1, k+1, ..., s+1$, и автомата $q^a\,(\varphi_k\,(\overline{x}, x_{n+1}, \ldots, x_{n+s}, z))$, содержащегося в M_0, построим автомат, изображенный на рисунке 10.

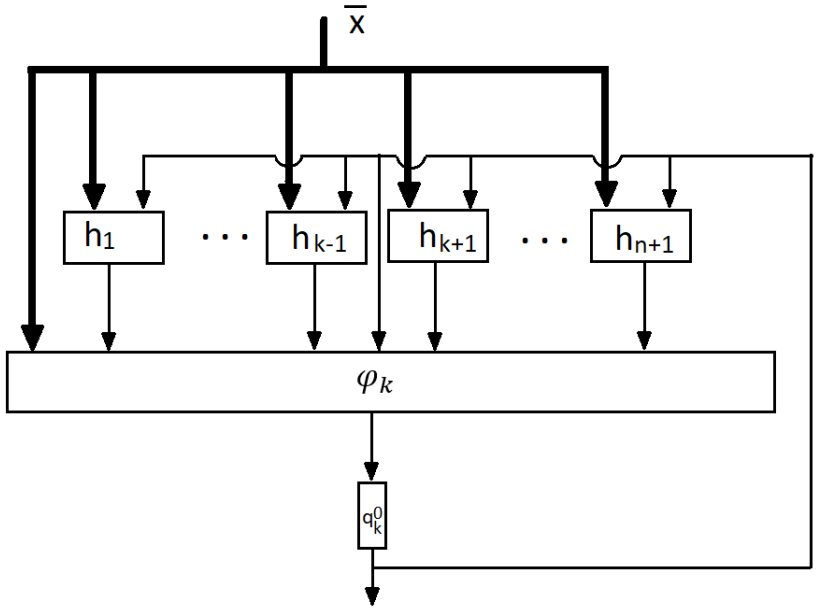


Рис. 10. Схема автомата с выходом $q_k(t)$ в момент времени t.

Здесь $a = q_k^0$ – начальное состояние задержки q_k.

Нетрудно видеть, что на выходе схемы, изображенной на рисунке 10, в момент времени t получаем $q_k(t)$. Обозначим функцию, реализуемую

этой схемой $f_k(\overline{x})$. Схема, изображенная на рисунке 11, реализует автомат f, который содержится в $K_c(M_0)$.

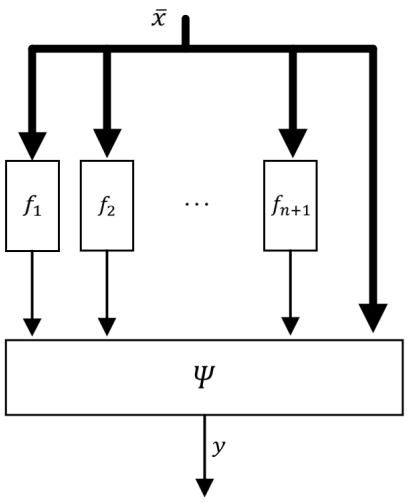


Рис. 11. Схема автомата f.

□

3. Заключение

В работе рассматривается класс автоматов с линейными выходами и два набора операций композиции над автоматами: операции ограниченной композиции и операции пополненной композиции. Доказана теорема об отсутствии конечной полной системы по операциям ограниченной композиции для класса конечных автоматов с одним линейным выходом, найдено содержательное бесконечное множество автоматов, порождающее этот класс по операциям ограниченной композиции.

Автор выражает благодарность профессору кафедры МаТИС механико-математического факультета МГУ имени М.В. Ломоносова Часовских А.А. за научное руководство.

Список литературы

[1] Кудрявцев В.Б., Алешин С.В., Подколзин А.С., *Введение в теорию автоматов*, «Наука», Москва, 1985, 320 pp.

[2] Лупанов О.Б., *Курс лекций по дискретной математике*, Механико-математический факультет МГУ им. М.В. Ломоносова, Москва, 2006, 38 pp.

[3] Яблонский С.В., Лупанов О.Б., *Дискретная математика и математические вопросы кибернетики*, **1**, Наука, Москва, 1974, 313 pp.

On the cardinality of generating sets in the class of finite automata with a linear output function

Yusupov F.R.

Abstract: The class of finite automata is finitely generated by composition operations [1]. An important subclass of this class are classes of automata with linear output functions. A set of operations on automata consisting of variable substitution, automata substitution, and feedback is called bounded composition. The class of unary automata is closed under bounded composition operations. In this paper, we show that the class of unary finite automata with linear output functions and bounded composition operations lacks finite complete sets. However, automata from this class, implemented by circuits with at most one delay, generate this class.

Keywords: finite automata, composition operations for automata, finite automata with linear outputs.

References

[1] Кудрявцев В.Б., Алешин С.В., Подколзин А.С., *Введение в теорию автоматов*, «Наука», Москва, 1985, 320 pp.

[2] Лупанов О.Б., *Курс лекций по дискретной математике*, Механико-математический факультет МГУ им. М.В. Ломоносова, Москва, 2006, 38 pp.

[3] Яблонский С.В., Лупанов О.Б., *Дискретная математика и математические вопросы кибернетики*, **1**, Наука, Москва, 1974, 313 pp.

Часть 4
Доклады семинаров

Доклады семинара «Теория автоматов»

Во втором полугодии 2024 года на научном семинаре «Теория автоматов» под руководством профессора Эльяра Эльдаровича Гасанова состоялось 8 докладов.

25 сентября 2024 года

Моделирование динамики SARS-CoV-2 в клеточных линиях

доц. Д. В. Алексеев, доц. А. В. Галатенко, доц. Е. Н. Князев,
ст. н. с. С. А. Нерсисян, доц. В. М. Староверов, проф. А. Г. Тоневицкий

Математическое моделирование вирусной динамики в клеточных линиях представляет интерес в силу нескольких причин: модели получаются более простыми из-за отсутствия адаптивного иммунного ответа и клеток врожденного иммунитета, имеется возможность измерения существенно большего количества параметров по сравнению со случаем in vivo, сами измерения оказываются менее шумными. В докладе рассматривается модель из шести интегро-дифференциальных уравнений, описывающих динамику числа здоровых, информированных и зараженных клеток, а также числа вирусных частиц, цитокинов и внутриклеточной вирусной РНК при заражении клеточных линий вирусом SARS-CoV-2. Модель с достаточно высокой точностью воспроизводит экспериментальные данные для штаммов Дельта и Омикрон на двух клеточных линиях: Caco-2 (эпителий кишечника человека) и Calu-3 (эпителий легких). В терминах модели разница между штаммами объясняется тремя параметрами: скоростью проникновения вирусов в клетку, скоростью выработки цитокинов (то есть врожденным иммунным ответом) в зараженных клетках, а также устойчивостью к заражению клеток, информированных цитокинами. Выявленные различия могут быть обоснованы с биологической точки зрения.

9 октября 2024 года

GF(2)-операции над формальными языками

асп. Е. А. Сажнева

В докладе рассматриваются GF(2)-операции над формальными языками — варианты классической конкатенации и звёздочки Клини, в определениях которых операции булевой логики (дизъюнкция и конъюнкция) заменены операциями в двухэлементном поле (конъюнкция и ис-

ключающее ИЛИ). Показывается замкнутость и незамкнутость различных семейств языков относительно этих операций. Для регулярных языков устанавливается замкнутость и приводятся точные нижние оценки сложности по числу состояний для этих операций при различных размерах алфавитов. Также показывается замкнутость класса языков, распознаваемых перестановочными автоматами, относительно операции GF(2)-конкатенация и незамкнутость этого класса относительно операции GF(2)-звёздочка. Кроме того, показывается, что языки, распознаваемые клеточными автоматами, не замкнуты относительно ни одной из операций, но замкнуты относительно GF(2)-конкатенации с регулярными языками.

Также доказывается незамкнутость языков, определяемых линейными, однозначными и однозначными линейными грамматиками, относительно GF(2)-конкатенации с множеством $\{\varepsilon, d\}$ с обеих сторон. В то же время для LR(k)-грамматик устанавливается замкнутость относительно GF(2)-конкатенации с регулярным множеством справа, но не слева.

Наконец, определяется класс формальных грамматик, основанных на GF(2)-операциях, и показывается, что он имеет ту же вычислительную сложность, что и обычные грамматики с объединением и конкатенацией.

16 октября 2024 года

О связи между теориями автоматов в многообразиях, алгебр с операторами и полигонов над полугруппами

доц. В. Л. Усольцев

В докладе рассматривается эквивалентность понятий автомата в многообразии алгебр и алгебры с операторами. Вводится понятие полигона (над полугруппой) в многообразии алгебр и показывается, что оно также эквивалентно понятию автомата в многообразии. Тем самым, возникает возможность использования хорошо разработанного аппарата универсальной алгебры (в частности, теории алгебр с операторами) и аппарата теории полигонов в многообразиях при исследовании автоматов в многообразиях.

23 октября 2024 года

Траектории автоматных систем

ст. н. с. Н. Ю. Волков

Предложено новое понятие автоматной системы, обобщающее все известные в дискретной математике вычислительные модели. Фактически,

автоматная система — это произвольный конечный автомат, или группа автоматов, с, вообще говоря, бесконечной внешней памятью. Т. е., это автомат, преобразующий произвольное множество X. Состояние такой модели в каждый момент времени задаётся парой (q, x), где q — состояние автомата в данный момент, а x — значение вычисляемого системой параметра в этот момент.

Введён специальный класс графов, задающих вычислительные системы. Показано, что каждая автоматная система является вычислительной системой, и наоборот, каждая вычислительная система является автоматной системой.

Рассмотрен случай, когда автоматная система распадается на некоторое количество более простых автоматных систем. Введены операции декомпозиции автоматной системы на компоненты и обратная операция объединения автоматных систем. Показано, что в результате произвольного сокращения траекторий автоматной системы получается траектория, которая также может быть реализована некоей автоматной системой.

На множестве автоматных (вычислительных) систем введены несколько отношений эквивалентности. Полностью изучена связь всех этих видов эквивалентности. Получен критерий сильной эквивалентности вычислительных систем.

Показано, что сложные автоматные системы, состоящие из конечного числа автоматов, сводятся к обычным автоматным системам, состоящим из одного автомата.

Фактически, автоматные системы — это общая модель для описания всех известных и перспективных вычислителей. Теория автоматных систем даёт новый взгляд на понятие алгоритма.

6 ноября 2024 года

О методах машинного обучения для работы с естественнонаучными данными графового типа

проф. А. А. Часовских, ст. н. с. В. С. Половников, доц. Г. В. Боков

В докладе будут представлены результаты, полученные в рамках совместного научного проекта механико-математического, химического факультетов и факультета биоинженерии и биоинформатики МГУ, поддержанного грантом междисциплинарной научно-образовательной школы «Мозг, когнитивные системы, искусственный интеллект» МГУ. Будет рассказано о задаче поиска группы симметрии молекул, задаче поиска оптимального положения иона металла на поверхности молекулы, задаче предсказания отклонения дипольного момента молекулы, задаче предсказания константы устойчивости молекулярных комплексов металлов,

задаче представления поверхностей макромолекул (белков и нуклеиновых кислот), задаче предсказания участков связывания белков, задаче синтеза белков, связывающихся с целевым белком.

13 ноября 2024 года

Оценки длин минимальных тестов для аргументов функций при подстановке констант, алгебраических операциях и сдвигах

инж. Г. В. Антюфеев

В докладе будет представлено содержание диссертации на соискание ученой степени кандидата физико-математических наук по специальности 1.2.3. Теоретическая информатика, кибернетика.

Схема из функциональных элементов реализует булеву функцию. На входах схемы возникают неисправности, которые влияют на аргументы реализуемой функции и могут приводить к тому, что схема начинает реализовывать отличную от исходной функцию. Подавая на входы схемы множество двоичных векторов и анализируя соответствующие значения реализуемой функции, выясняется, была ли неисправность на входах или нет. Такое множество двоичных векторов называется проверяющим тестом. Если же множество двоичных векторов позволяет конкретизировать неисправность, то оно называется диагностическим тестом. Естественным образом возникает задача поиска минимальных тестов для различных источников неисправностей, а также получения оценок функции Шеннона. Функция Шеннона отражает зависимость размера минимального теста самой труднотестируемой булевой функции от числа переменных этой функции. В диссертации получены результаты для константного источника неисправностей и его модификации. Константный источник неисправностей является исторически первым источником, исследуемым в теории тестов. Также в диссертации получены результаты для различных модификаций мультипликативных и сдвиговых неисправностей. Все исследуемые неисправности могут представлять практический интерес при разработке и тестировании микроэлектронных устройств.

20 ноября 2024 года

Вычисления с частичными оракулами

асп. А. М. Хайруллин

Работа нацелена на изучение различных способов вычисления с частичным оракулом. В отличие от вычислений с тотальным оракулом,

различные модели вычислений с частичным оракулом порождают различные классы функций.

Даны три различных определения вычисления с частичным оракулом. Показана неэквивалентность этих трех подходов вычисления, а также то, что множества функций, вычислимых с частичным оракулом согласно этим определениям, строго вложены друг в друга.

Рассмотрены сводимости, порождаемые различными способами вычислений с частичным оракулом. Доказана импликация сводимостей, соответствующих различным определениям вычислений с оракулом, и выявлена связь этих сводимостей с сводимостью по перечислению.

Получен критерий вычислимости с частичным оракулом согласно первому определению, обобщающий критерий для вычислений с тотальным оракулом, а также связь этих трех определений с моделью частично-рекурсивных функций. Выявлены три класса оракулов, которые порождают классы функций, для которых обобщения определения перечислимых множеств различны.

27 ноября 2024 года

Градиентная маска и некоторые свойства процедур обучения нейронных сетей

асп. Л. Цзян

Современное развитие искусственного интеллекта и машинного обучения значительно связано с прогрессом в области глубоких нейронных сетей (ГНС), которые черпают вдохновение в работе человеческого мозга. Эти сети добились выдающихся результатов в таких областях, как компьютерное зрение, обработка естественного языка, автономное вождение и других, что привело к революции в современных технологиях. Однако, несмотря на успех, проблема обобщающей способности ГНС, то есть их способность работать с новыми данными, остаётся важной и активно исследуемой. Обобщение зависит от множества факторов, включая архитектуру сети, методы обучения и свойства обучающей выборки.

В данной работе исследуется проблема обобщения в ГНС и предлагается новый метод её улучшения через механизм «градиентной маски», вдохновлённый биологическими принципами, в частности механизмом нейронного торможения. В рамках исследования разработаны два подхода к маскированию градиентов: первый основан на латеральном торможении, а второй реализован на уровне весов нейронной сети. Оба метода продемонстрировали значительное улучшение обобщающей способности ГНС.

Доклады семинара «Теория автоматов»

В первом полугодии 2025 года на научном семинаре «Теория автоматов» под руководством профессора Эльяра Эльдаровича Гасанова состоялось 14 докладов.

19 февраля 2025 года

Задачи машинного обучения в электронной торговле и транспортных технологиях

м. н. с. А. П. Соколов

В докладе будут рассмотрены некоторые сервисы электронной торговли и транспортных технологий, в которых применяются методы машинного обучения. Будут рассмотрены актуальные постановки задач, по которым проводятся исследования в рамках службы развития ML технологий Яндекса, подходы к решению этих задач и ближайшие планы.

26 февраля 2025 года

Классы линейных 2-адических автоматов с сумматором

асп. М. Э. Калашников

Для линейных автоматов известны результаты о выразимости линейных автоматных функций через системы, содержащие сумматор. Также известен критерий вхождения сумматора в замкнутый класс линейных функций.

В докладе будет рассказано о линейных 2-адических автоматах и некоторых предполных классах в них. Докладчиком получены критерий выразимости произвольных линейных 2-адических функций в системах с сумматором и критерий вхождения сумматора в произвольный замкнутый класс. Также будет описана критериальная система предполных классов в T_0, множестве линейных 2-адических автоматов, сохраняющих 0 в начальный момент времени.

5 марта 2025 года

Лабиринты и автоматные системы

ст. н. с. Н. Ю. Волков

В докладе даётся обзор и критический разбор некоторых моделей взаимодействия систем автоматов в лабиринтах. Предложена формализация модели системы автоматов в лабиринте через автоматную систему. Такая формализация является весьма универсальной и даёт необходимую строгость. Введён класс классических лабиринтов.

Автоматная система S — это множество параметров X с функцией обзора $\beta : X \to A$ ($|A| < \infty$), конечное множество преобразований множества X, параметризованное функцией-преобразователем $\gamma : B \to C(X)$ ($|B| < \infty$, здесь $C(X)$ — частичные отображения X в себя) и конечный автомат $\mathcal{A}$ с входным алфавитом A и выходным алфавитом B. Таким образом, $S = (\mathcal{A}, X, \beta, \gamma)$. Модель автоматной системы является наиболее универсальным вычислителем. Множество автоматных систем, имеющих разные автоматы $\mathcal{A}$, но одни и те же X, A, B, β, γ, называется типом автоматных систем. Тип автоматных систем задаётся вычислительным пространством $\mathcal{X} = (X, A, B, \beta, \gamma)$. Таким образом, $S = (\mathcal{A}, \mathcal{X})$.

Очевидно, что любая система автоматов в лабиринтах может быть смоделирована автоматной системой. Оказалось верно и обратное: любая автоматная система моделируется одним автоматом в классическом лабиринте, или в семействе классических лабиринтов. Таким образом, модель автомата в классическом лабиринте является ведущим частным случаем максимально универсальной модели автоматной системы. При этом, каждый классический лабиринт задаёт вычислительное пространство.

Определяется понятие класса алгоритмов над лабиринтом L (или, что то же самое, над вычислительным пространством X). Каждый алгоритм над лабиринтом L задаётся автоматом, способным перемещаться в этом лабиринте. Два алгоритма эквивалентны, если поведения задающих их автоматов в лабиринте L совпадают при любой точке старта. Этот подход равносилен заданию алгоритма как класса сильно-эквивалентных друг другу автоматных систем над фиксированным вычислительным пространством. Такое определение алгоритма позволяет изучать намного более широкие классы алгоритмов, чем Тьюринговы алгоритмы. В отличие от описания сверхтьюринговых алгоритмов при помощи модели *вычислений с оракулом*, которая есть лишь искусственная надстройка над Тьюринговыми алгоритмами, наш подход позволяет изложить теорию алгоритмов более системно и с общих позиций.

12 марта 2025 года

Проблема полноты в классе линейных дефинитных автоматов

асп. И. В. Молдованов

Проблема проверки полноты конечных подмножеств играет важную роль при исследовании функциональных систем. В классе конечных автоматов с операциями композиции задача проверки полноты конечных подмножеств является алгоритмически неразрешимой, тогда как класс конечных автоматов с операциями суперпозиции не содержит конечных полных систем. Подкласс дефинитных автоматов характеризуется наличием конечных полных систем относительно операций суперпозиции, однако задача проверки полноты конечных подмножеств в данном случае также оказывается алгоритмически неразрешимой.

Ранее был рассмотрен класс линейных автоматов с операциями композиции. Для данного класса был получен алгоритм определения полноты конечных подмножеств. В то же время для линейных автоматов с операциями суперпозиции было установлено отсутствие конечных полных систем. Интерес представляет рассмотрение данной задачи применительно к классу дефинитных линейных автоматов с операциями суперпозиции. В рамках доклада будет освещена критериальная система для класса одноместных линейных дефинитных автоматов, сохраняющих нулевую последовательность, а также представлен алгоритм проверки полноты конечных содержащих константу ноль подмножеств в классе линейных дефинитных автоматов.

19 марта 2025 года

Решение арифметических задач клеточными автоматами с локаторами

проф. Э. Э. Гасанов

В докладе будет введено понятие клеточного автомата с локаторами. С помощью этих автоматов будут решены следующие задачи: перевод натурального числа из унарного представления в бинарное, перевод натурального числа из бинарного представления в унарное, сложение, умножение и деление натуральных чисел, вычисление суммы большого числа натуральных чисел.

26 марта 2025 года

Универсальные алгоритмы для решения задачи удовлетворения ограничениям

ст. н. с. Д. Н. Жук

Универсальным алгоритмом для задачи удовлетворения ограничениям мы называем алгоритм, который на вход получает экземпляр задачи и за полиномиальное время выдает ответ (иногда правильный, иногда нет) и при этом никак не привязан к языку ограничений и другим параметрам задачи. Обычно такие алгоритмы либо пытаются вывести какое-то противоречие локально, либо сводят задачу к задаче линейного программирования и пытаются получить противоречие там. Если же противоречие получить не удается, алгоритм просто возвращает «Да». Чтобы усилить такие алгоритмы, мы можем сначала зафиксировать какую-то переменную каким-то значением, а потом запустить алгоритм, и если он вернет «Нет», удалить это значение из области значений соответствующей переменной.

Мне удалось придумать универсальную конструкцию, которая позволила для каждого из универсальных алгоритмов описать когда его усиленная версия корректно решает задачу. Кроме этого, удалось показать, что самый сильный из этих алгоритмов решает задачу удовлетворения ограничениям на любом языке ограничений на 7-элементном множестве, но уже на 8-элементном множестве есть язык ограничений, который не решается никакой комбинацией известных универсальных алгоритмов.

2 апреля 2025 года

Использование контрольной суммы для улучшения восстанавливающей способности многоуровневых кодов

доц. Д. В. Алексеев, м. н. с. Д. В. Ронжин

В рамках доклада будет представлено краткое введение в схемы, использующиеся для помехоустойчивого кодирования в СХД (системах хранения данных), и методы оценки надежности этих схем. Будет описан метод для улучшения корректирующей способности многоуровневых кодовых схем, предложенный авторами. Метод основан на использовании дополнительной избыточности данных (CRC) и итерационном декодировании с локализацией ошибок.

О новой системе шифрования с открытым ключом рюкзачного типа

доц. А. А. Ирматов, асп. А. И. Болотников

Задача о чередующемся взвешенном пути возникла при изучении разбиений пространства функций весов уравнениями конфигурации паросочетаний. Для данной задачи была доказана NP-полнота. В докладе будет представлена построенная на её основе новая система шифрования Болотникова—Ирматова с открытым ключом рюкзачного типа.

О сложности задачи существования в нейронной сети предельного цикла заданной длины

доц. Г. В. Боков, асп. А. С. Дробышев

В докладе будет рассказано о модели нейронной сети, представляющей собой ориентированный граф, вершинам которого приписаны пороговые булевы функции. Вершины такого графа называются нейронами, а входящие ребра определяют связь нейрона с соседними нейронами так, что между входящими ребрами и существенными переменными пороговой функции нейрона устанавливается взаимно однозначная связь. Нейронная сеть функционирует в дискретные моменты времени и каждый такт пересчитывает состояние каждого нейрона. Состояние всех нейронов называется конфигурацией сети. Представляет интерес исследовать поведение конфигурации нейронной сети с точки зрения длины предельного цикла, которому она может принадлежать. В докладе будет представлен ряд результатов, связанных с оценкой сложности задачи существования в нейронной сети предельного цикла заданной длины.

9 апреля 2025 года

О модификации диаграммы Мура функцией отрицания

студ. Д. О. Масленников

В докладе вводится понятие результата применения функции $f : B \to B$ к инициальному автомату. После того, как автомат совершает переход из состояния в состояние, происходит изменение его диаграммы: если был выведен символ b, он заменяется на данном переходе на $f(b)$. Полученную словарную функцию называем результатом применения правила f к автомату.

Можно показать, что данная словарная функция является ограниченно-детерминированной с не более чем $n \cdot |B|^{n \cdot |A|}$ остаточных функций, где n — число состояний, поэтому далее рассматриваем её как инициальный автомат приведённого вида.

Для результата применения функции отрицания к сильно связному автомату с входным и выходным алфавитом $\{0, 1\}$ получены верхняя и нижняя оценки числа его состояний: $n \cdot 2^{n+1}$ и 2^{n+1} соответственно.

Вводятся понятие остова — неинициального автомата без функции выхода — и результата применения к нему функции — неинициального аналога результата применения функции к автомату. Рассматриваются две серии примеров применения отрицания к остову. Как следствие, показана неулучшаемость оценок приведённых выше — верхней в общем случае, и нижней для нечётных n.

16 апреля 2025 года

Предикатное задание минимальных клонов трехзначной логики

студ. А. И. Зданович

Минимальные клоны являются нижними элементами решетки замкнутых классов, содержащих тождественную функцию. По аналогии с существованием не конечно порожденного предполного класса, можно задать «двойственный» вопрос о существовании не предикатно описуемого минимального клона. В данном докладе мы обсудим предикатное задание минимальных клонов трехзначной логики, а также приведем не предикатно описуемый минимальный клон для произвольной значности.

Об отношениях на конечном множестве, обладающих внутренней симметрией

студ. А. Е. Жариков

Из соответствия Галуа между клонами и замкнутыми множествами предикатов мы знаем, что любой клон можно задать как класс сохранения какого-то множества предикатов. При этом можно показать, что для этого не требуются все предикаты, а достаточно использовать только существенные предикаты или, ещё точнее, только ключевые предикаты. В данном докладе будет дан ответ на вопрос, какую долю составляют существенные предикаты от числа всех предикатов при различных k. Для $k > 2$ описать все ключевые предикаты достаточно сложно, поэтому будут введены подклассы ключевых предикатов, обладающих дополнительными свойствами, усиливающими их симметрию: абсолютно ключевые и биективно ключевые предикаты. Для случая $k = 3$ будет дано описание биективно ключевых предикатов.

23 апреля 2025 года

Полиномиальная полнота и полнота n-квазигрупп: критерии и методы обеспечения

студ. С. С. Чаплыгина

Доклад посвящен исследованию ряда свойств квазигрупп и n-квазигрупп с точки зрения их применения в задачах криптографии. Рассматриваемые алгебраические структуры представляют интерес для создания различных криптоалгоритмов, таких как шифры, хэш-функции, протоколы аутентификации и другие. Для обеспечения стойкости важно требовать от квазигрупп выполнения особых свойств. В. А. Артамоновым было предложено рассматривать полиномиально полные квазигруппы без собственных подквазигрупп. Следствием результата Й. Хагеманна и К. Херрманна о полиномиальной полноте алгебры, содержащей мальцевскую операцию, а также работы В. А. Артамонова является критерий полиномиальной полноты квазигруппы как алгебры с тремя операциями. В докладе предлагается критерий полиномиальной полноты n-квазигрупп как алгебр с одной операцией (квазигрупповой), а также критерий полноты n-квазигрупп. Вторым результатом, представленным в докладе, являются алгоритмы, которые с помощью изотопных преобразований усиливают криптографические свойства. В случае квазигрупп удается добиться полиномиальной полноты, простоты, неафинности, отсутствия подквазигрупп и тривиальности группы автоморфизмов. В случае n-квазигрупп при $n = 3$ на выходе алгоритма гарантируется либо полиномиальная полнота, либо отсутствие собственных подквазигрупп. При $n > 3$ удается добиться одновременного выполнения обоих свойств. Предложенный алгоритм является обобщением и усилением известного результата Т. Кепки.

14 мая 2025 года

О восстановлении последовательностей машины Тьюринга

асп. В. В. Ушакова

Доклад посвящён изучению машин Тьюринга. Вводятся и рассматриваются последовательности машины Тьюринга: последовательность входных символов, последовательность выходных символов, последовательность состояний, а также последовательность перемещений.

Пусть некоторые из рассматриваемых четырёх последовательностей машины Тьюринга неизвестны. Рассматривается возможность восстановления неизвестных последовательностей по известным. Рассматрива-

ются разные случаи, когда программа и вход машины Тьюринга заданы и не заданы.

Доклады семинара «Теория автоматов»

Во втором полугодии 2025 года на научном семинаре «Теория автоматов» под руководством профессора Эльяра Эльдаровича Гасанова состоялось 10 докладов.

3 сентября 2025 года

Передача сигнала в модели с фиксированным алгоритмом декодирования на стороне приёмника на основе теории взаимной информации

асп. Д. А. Юдаков

В докладе представлены результаты, составившие содержание кандидатской диссертации. Работа посвящена решению задачи эффективного распределения мощности между пользователями и антеннами беспроводных сетей передачи данных. Предложен алгоритм оптимизации мощности, учитывающий поантенные ограничения базовой станции; рассмотрена функция взаимной информации для передачи сигнала по беспроводной сети с фиксированным алгоритмом декодирования на стороне приемника; доказана возможность представления оптимальных двумерных созвездий в виде декартова произведения одномерных; предложен подход к оптимизации сигнальных созвездий только на стороне передатчика при условии фиксации созвездия на приемнике.

24 сентября 2025 года

О новом исследовательском центре МГУ в сфере искусственного интеллекта

доц. Г. В. Боков

В докладе будет рассказано о научных направлениях деятельности нового Исследовательском центре в сфере искусственного интеллекта Московского университета и ключевых проектах Центра, связанных с разработкой систем по выявлению и устранению токсичности и галлюцинаций в ответах генеративной модели ИИ, систем гибридного ИИ для работы с большими объемами научно-технической информации, моделей вождения беспилотным транспортом с интеграцией данных сенсоров, моделей для генерации и предсказания свойств белков и кристаллических материалов, систем для реконструкции 3D деталей по данным лазерного сканирования и фотографиям, моделей для диагностики заболеваний.

1 октября 2025 года

Методы фотоники в аппаратном ускорении вычислений

ст. н. с. А. А. Грунин, ст. н. с. А. В. Четвертухин

Доклад посвящен вопросу применения методов фотоники для аппаратного ускорения востребованных вычислительных задач, таких как векторно-матричное умножение. Центральной задачей в этой развивающейся области представляется оценка потенциального преимущества фотонных вычислителей (optical processing unit — OPU) перед другими. Преимущество рассматривается по параметрам производительности и энергоэффективности с обязательным учетом физических ограничений. Важным является поиск комбинации из задачи, архитектуры OPU и вычислительных параметров, в которых потенциально возможно достижение преимущества. Оценка преимущества в идеале должна рассматривать не только вычислительное ядро, но и работу всего ускорителя, в т.ч. работу шины и системы памяти. Целями доклада является рассказ о нашем видении исследовательской области и нахождение точек возможного сотрудничества.

8 октября 2025 года

Разработка приложений реального времени на основе отечественной аппаратной и программной платформы

зав. отделом А. И. Грюнталь

В докладе планируется осветить следующие вопросы:

- состав и структура программной платформы;
- состав и структура аппаратной платформы;
- требования к операционной системе реального времени;
- особенности программирования систем реального времени;
- технология кросс-разработки;
- многопроцессорные системы;
- организация вычислений на многопроцессорных комплексах;
- программирование систем сигнальной обработки.

19 октября 2025 года

О многомерном расширении тензорных произведений кодов

н. с. Г. В. Калачев, доц. П. А. Пантелеев

Линейные коды с разреженными проверочными матрицами (LDPC-коды) являются важными компонентами современных систем хранения и передачи данных. В 1970-х годах Майкл Таннер предложил изящный способ построения таких кодов на основе разреженных графов: кодовыми словами служат все возможные пометки ребер элементами конечного поля, при которых для каждой вершины набор пометок инцидентных рёбер образует кодовое слово некоторого фиксированного локального кода, назначенного этой вершине. В 1990-х годах Сипсер и Спилмен показали, что если все локальные коды одинаковы и обладают достаточно большим расстоянием, а исходный граф имеет хорошие свойства расширения, то полученный глобальный код имеет размерность, минимальное расстояние и сложность декодирования, растущие линейно с длиной кода. Эти результаты положили основу современного понимания связи между структурой графа и свойствами кода. Создание квантовых LDPC-кодов и классических локально тестируемых кодов с аналогичными свойствами потребовало построения многомерных аналогов таких конструкций, где вместо графов используются кубические комплексы, а локальные коды задаются в виде тензорных произведений. Однако оказалось, что одного лишь большого расстояния локальных кодов недостаточно: необходима новая характеристика, называемая расширением, которая естественным образом обобщает понятие минимального расстояния. В докладе будет представлен новый результат, показывающий, что над достаточно большим полем почти все тензорные произведения кодов обладают линейным по длине расширением. Этот факт позволяет строить многомерные аналоги кодов Таннера произвольной размерности, открывая широкий круг приложений в теории сложности и отказоустойчивых квантовых вычислениях.

22 октября 2025 года

Минимальные универсальные нейронные схемы над кольцом двоично-рациональных чисел

проф. А. А. Часовских, асп. М. В. Агафонова, м. н. с. А. А. Хусаенов, студ. Е. И. Яковенко

Формальный нейрон Мак-Каллока-Питтса строится из линейных функций (умножителей на константы, констант, сумматора) и функции

активации. В качестве последней мы рассматриваем пороговую функцию Хэвисайда. Как показано В. С. Половниковым, замыкая множество, состоящее из перечисленных функций, по операциям суперпозиции, получаем класс кусочно-параллельных функций. Для того, чтобы гарантировать конечную порожденность исследуемого класса, мы рассматриваем класс функций PP, являющийся замыканием по операциям суперпозиции множества линейных функций с коэффициентами из кольца двоично-рациональных чисел и функции Хэвисайда.

Функция из рассматриваемого класса называется минимальной универсальной, если она может быть реализована схемой с двумя входами, содержащей один элемент Хэвисайда. М. А. Агафоновой была найдена такая функция.

Минимальной универсальной схемой в базисе из умножителей на -1 и на $1/2$, сумматора и функции Хэвисайда мы называем схему, содержащую наименьшее количество элементов этого базиса, из которой операциями суперпозиции можно реализовать любую функцию из PP. Мы нашли все минимальные универсальные схемы. В рассматриваемом классе функций найдены замкнутые подклассы, позволившие существенно сократить техническую часть исследования. Исследование замкнутых классов основано на работах Д. В. Ронжина о классах автоматов над кольцом двоично-рациональных чисел. Для минимальных универсальных функций определены некоторые важные свойства.

29 октября 2025 года

Задача определения порядка автоматов, чьи функции переходов и выходов принадлежат замкнутому классу Поста

асп. Н. В. Муравьев

Исследуется задача определения порядка автомата Мили относительно операции суперпозиции. Так как в общем случае она алгоритмически неразрешима, рассматриваются особые классы R-автоматов, чьи алфавиты и множества состояний лежат в многомерных булевых кубах, а функции переходов и выходов принадлежат некоторому замкнутому классу Поста R. Доказано разбиение решетки Поста замкнутых классов относительно разрешимости задачи вычисления порядка для соответствующих R-автоматов.

5 ноября 2025 года

О связи сложности булевых операторов в различных моделях схем

м. н. с. А. А. Ефимов, н. с. Г. В. Калачев

Мы исследуем сложность укладок схем из функциональных элементов (СФЭ) на графах. Через укладки СФЭ на графы с ограничениями можно определить многие известные модели схем: планарные схемы, клеточные схемы, объёмные схемы, многослойные клеточные схемы и некоторые другие. В частности, d-мерные клеточные схемы представляют собой укладки СФЭ на d-мерные решётки. Рассматривается функция сложности перехода между моделями: насколько может возрасти сложность реализации булева оператора при переходе от СФЭ к укладкам на графах.

В докладе будут представлены результаты о связи сложности между СФЭ и укладками на графах. С одной стороны, предложен метод, позволяющий уложить СФЭ сложности N на d-мерную решётку со сложностью порядка $N^{d/(d-1)}$. С другой стороны, для оператора умножения на случайные разреженные двоичные матрицы получена универсальная нижняя оценка через размеры сепараторов графов, на которые укладывается схема. В частности, для укладок на прямоугольные решётки эта нижняя оценка совпадает по порядку с верхней, что даёт точный порядок функции сложности перехода от СФЭ к d-мерным клеточным схемам.

В докладе также будут озвучены направления для дальнейших исследований в этой области.

19 ноября 2025 года

Методы оценки качества и верификации линейных классификаторов для анализа последовательностей микроРНК

н. с. А. П. Жиянов

В докладе представлены результаты, составившие содержание кандидатской диссертации на соискание ученой степени кандидата наук, подготовленной под руководством д.б.н., проф., акад. А. Г. Тоневицкого. Основной целью исследований была разработка методов оценки качества и верификации линейных классификаторов и применение линейных классификаторов к задаче предсказания изоформ микроРНК. Для достижения цели были решены следующие задачи.

1) Формализация свойств, которым должны удовлетворять метрики качества классификации. Проверка непротиворечивости этих свойств и наличие их у популярных метрик.

2) Установление верхних оценок вероятности линейной почти разделимости двумерных данных. Разработка статистического критерия однородности выборки, связанного с линейной почти разделимостью, оценка эффективности критерия.

3) Анализ данных экспрессии изоформ микроРНК в различных опухолевых тканях человека. Разработка линейного классификатора, предсказывающего наличие изоформ по нуклеотидной последовательности микроРНК. Оценка качества разработанного классификатора. Выявление нуклеотидных последовательностей, определяющих наличие изоформ.

3 декабря 2025 года

О вычислительных возможностях клеточных автоматов с локаторами

проф. Э. Э. Гасанов, н. с. Г. В. Калачев

Клеточные автоматы с локаторами — новый класс управляющих систем. В первой части доклада будет показано, как схемы из функциональных элементов можно моделировать в клеточных автоматах с локаторами таким образом, чтобы время вычисления было пропорционально глубине схемы. Также будет показано, как в клеточных автоматах моделировать машину Тьюринга с произвольным доступом, чтобы они функционировали потактово эквивалентно.

Во второй части доклада будет рассказано о схемных классах сложности и о скорости решения задач из этих классов с помощью клеточных автоматов с локаторами. В частности, на идейном уровне будет показано, как задачи из класса NC^1 можно решать клеточными автоматами с локаторами за логарифмическое время. Кроме того, будет рассказано о связи класса задач, решаемых клеточными автоматами с локаторами за полиномиальное время, с классом PSPACE.

17 декабря 2025 года

Заседание, посвященное памяти Валерия Борисовича Кудрявцева

К сведению авторов публикаций в журнале «Интеллектуальные системы. Теория и приложения»

В соответствии с требованиями ВАК РФ к изданиям, входящим в перечень ведущих рецензируемых научных журналов и изданий, в которых могут быть опубликованы основные научные результаты диссертаций на соискание ученой степени доктора и кандидата наук, статьи в журнал «Интеллектуальные системы. Теория и приложения» предоставляются авторами в следующей форме:

1. Статьи, набранные в пакете LaTeX, предоставляются к загрузке через WEB-форму http://intsysmagazine.ru/generator_form .
2. К статье прилагаются файлы, содержащие название статьи на русском и английском языках, аннотацию на русском и английском языках (не более 50 слов), список ключевых слов на русском и английском языках (не более 20 слов), информация об авторах: Ф.И.О. полностью, место работы, должность, ученая степень и/или звание (если имеется), для аспирантов ФИО научного руководителя, контактные телефоны (с кодом города и страны), e-mail, почтовый адрес с индексом города (домашний или служебный).
3. Список литературы оформляется в едином формате, установленном системой Российского индекса научного цитирования. Список на русском языке приводится в конце файла с текстом статьи, в то время как список, переведённый на английский язык, прилагается отдельным файлом.
4. За публикацию статей в журнале «Интеллектуальные системы. Теория и приложения» с авторов (в том числе аспирантов высших учебных заведений) статей, рекомендованных к публикации, плата не взимается. Авторам бесплатно предоставляется номер журнала, в котором вышла статья. Журнал распространяется по подписке, экземпляры журнала рассылаются подписчикам наложенным платежом. Условия подписки публикуются в каталоге НТИ «Роспечать», индекс журнала 64559.
5. Доступ к электронной версии последнего вышедшего номера осуществляется через НЭБ «Российский индекс научного цитирования». Номера, вышедшие ранее, размещаются на сайте

http://intsysmagazine.ru,

и доступ к ним бесплатный. Там же будут размещены полные тексты всех публикуемых статей.

Подписано в печать: 25.12.2025
Дата выхода: 31.12.2025
Тираж: 200 экз.
Цена свободная
Свидетельство о регистрации СМИ: ПИ № ФС77–58444 от 25 июня 2014 г., выдано Федеральной службой по надзору в сфере связи, информационных технологий и массовых коммуникаций (Роскомнадзор).