# ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

Научная статья

УДК 004.415.52, 004.434
DOI: 10.18101/2304-5728-2025-1-65-78

## МЕТОДИКА АВТОМАТИЗИРОВАННОЙ ОБРАБОТКИ ИНФОРМАЦИИ С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА ПРОГРАММИРОВАНИЯ ВЫСОКОГО УРОВНЯ. ЧАСТЬ 2

**© Федорченко Людмила Николаевна**
кандидат технических наук, старший научный сотрудник,
Санкт-Петербургский федеральный исследовательский центр
Российской академии наук (СПб ФИЦ РАН)
Россия, 199178, г. Санкт-Петербург, 14 Линия В. О., 39
lnf@iias.spb.su

**© Гейда Александр Сергеевич**
кандидат технических наук, заведующий лабораторией прикладной информатики
Санкт-Петербургский федеральный исследовательский центр
Российской академии наук (СПб ФИЦ РАН)
Россия, 199178, г. Санкт-Петербург, 14 Линия В. О., 39
Северо-Западный институт управления — филиал Российской академии
народного хозяйства и государственной службы
Россия, 199178, г. Санкт-Петербург, В. О., Средний пр. В. О., 57/43
geida@iias.spb.su

**Аннотация.** Вторая часть статьи посвящена разработке анализатора языка, определяемого трансляционными грамматиками в регулярной форме. Впервые описываемая методика была применена в 1970-х гг. в компиляторе для Алгола 68. Сформулированы ограничения на грамматику, представленную в форме системы ориентированных графов (типа расширенных диаграмм Вирта), гарантирующие существование языкового процессора, подобного детерминированному магазинному преобразователю, который далее рассматривается как управляющий механизм, сканирующий входную цепочку слева направо, фиксируя последовательность состояний для инициирования действий, составляющих процесс трансляции. Специфику методики составляет *алгоритм регуляризации* грамматики, основанный на эквивалентных преобразованиях синтаксического графа грамматики: устранении рекурсий и вставке итераций. Регуляризация исходной грамматики является *частью полного цикла реализации языка,* состоящего из цикла *пользователя*, и полуавтоматического цикла *разработчика*.

**Ключевые слова**: автоматизированная обработка информации, схема процесса компиляции, синтаксическая модель языка, эквивалентные преобразования, языковой процессор.

**Для цитирования**
*Федорченко Л. Н., Гейда А. С.* Методика автоматизированной обработки информации с использованием языка программирования высокого уровня. Часть 2 // Вестник Бурятского государственного университета. Математика, информатика. 2025. № 1. С. 65–78.

## Введение

Исследования по автоматизированному анализу формальных языков, начавшиеся в 1960-х гг., являются естественной частью фундаментальной теории компьютерной лингвистики [1–3]. Были разработаны различные инструменты для обработки контекстно-свободных и контекстно-зависимых формальных языков, которые необходимо формализовать для изучения их свойств с помощью компьютерной техники.

В наши дни технологии обработки данных с использованием языков широко применяются в системах перевода во многих приложениях для различных компьютерных устройств. Например, в известном проекте GNU[1] по разработке свободного программного обеспечения для построения анализаторов (или трансляторов) инструментальные системы Lex/Yacc[2], Flex/Bison[3] используются для автоматизированного получения конкретного транслятора из синтаксического определения языка. Однако эти анализаторы имеют ограничения на тип входной формальной грамматики, а применяемый алгоритм разбора часто требует значительного объема памяти для хранения промежуточных данных.

При построении анализирующих программ существенной проблемой является эквивалентное преобразование исходной грамматики языка программирования в специальную форму для обеспечения построения корректного трансляционного автомата; другая проблема касается ограничений выбранного метода синтаксического разбора (класс грамматики).

Первая проблема вызвана большим разнообразием форм, используемых для представления языка (BNF, extended BNF, двухуровневые и аффиксные грамматики и т. д.), Вторая проблема часто приводит к языковой неоднозначности (или недетерминированности) распознающего автомата. Решения этих двух проблем тесно связаны с формальным представлением реализуемого языка и средой обработки. Подробнее эти проблемы рассмотрены в работах [4–10].

В рамках ряда проектов в Санкт-Петербургском ФИЦ РАН и силами студентов СПбГУ разработана инструментальная система SynGT (Syntax Graph Transformations), обеспечивающая решение указанных проблем. В ней правила грамматики заменены на эквивалентные диаграммы (граф-

[1] URL: https://ru.wikipedia.org/wiki/Проект_GNU (дата обращения: 31.03.2025)
[2] Lex – A Lexical Analyzer Generator. URL: http://www.nylxs.com/docs/lexandyacc.pdf (дата обращения: 05.03.2025)
[3] Bison — генератор парсеров GNU. URL: http://www.gnu.org/software/bison/Win flex-bison, http://sourceforge.net/projects/winflexbison/ (дата обращения: 05.03.2025).

схемы), которые используются для представления контекстно-свободной грамматики в регулярной форме (КСР-грамматики), синтеза распознающего автомата и автоматического выполнения эквивалентных преобразований входной грамматики с целью её регуляризации [12–17].

## 1 Проектирование анализирующей части компилятора

При проектировании анализирующей части компилятора работу следует начинать с написания рабочей грамматики входного языка, т. е. грамматики такого класса, на который рассчитан предполагаемый для использования анализатор исходного языка, а следовательно, выполнения преобразований грамматики. Из соображений эффективности фактически используются некоторые подклассы КС-грамматики. Поэтому речь может идти лишь о написании КС-грамматики, аппроксимирующей входной язык с точностью до контекстных свойств (условий), которые должны учитываться дополнительными механизмами, составляющими контекстно зависимую часть компилятора.

Среди наиболее часто используемых подклассов КС-грамматик можно перечислить следующие: LR(1)-грамматики [3–4] или их варианты с дополнительными ограничениями LR(0), SLR(1) [7], LALR(1) [4] и грамматики простого или операторного предшествования [5], рассчитанные на анализаторы типа «снизу вверх»; LL($k$)-грамматики [4], предполагающие использование предсказывающего типа алгоритмы разбора, относящиеся к классу анализаторов типа «сверху вниз», удобные для построения анализаторов рекурсивного спуска [6–9]. Для всех этих грамматик существуют адекватные классы анализаторов, выполняющие синтаксический разбор за линейное время.

Анализаторы типа «снизу вверх» называются так потому, что действуют таким образом, как если бы они строили дерево вывода входной цепочки, начиная снизу от листьев в направлении корня. Фактически они воспроизводят в своем магазине открытые части сентенциальных форм правостороннего вывода в обратном порядке, используя механизм «сдвиг-свертка».

Анализаторы типа «сверху вниз» называются так потому, что действуют таким образом, как если бы они строили дерево вывода входной цепочки, начиная сверху от корня, постепенно наращивая ветви до терминальных листьев. Фактически они воспроизводят в своем магазине открытые части сентенциальных форм левостороннего вывода. По сравнению с анализаторами типа «снизу вверх» эти анализаторы имеют больше информации для характеристики ошибок во входной программе, но ими не покрывается весь класс детерминированных языков [8; 9].

Методы синтаксического разбора, применяемые к реализации языков программирования, и коммерческие инструменты для построения анализаторов, такие как Lex-Yacc, Antlr, SYNTAX [18], Forth Technology [10] и Flex/Bison, также используют преобразования исходной КС-грамматики; некоторые из них выполняются автоматически в процессе построения но-

вого анализатора; другие выполняются вручную, а значит, могут быть подвержены внесению ошибок, что затруднит процесс разработки компилятора в целом.

Процедура регуляризации, подробно описанная в работах [12–17], основана на эквивалентных преобразованиях исходной формальной КС-грамматики, специфицирующей реализуемый язык программирования. Её цель — получить эквивалентную форму, подходящую для оптимизации соответствующего языкового процессора. В инструменте SynGT представлены все необходимые функции преобразований [15], предназначенные для быстрого поиска и разрешения неоднозначностей и устранения недетерминизма при построении анализатора. Разнообразие синтаксических определений языков требует подгонки исходной грамматики под конкретный трансляционный автомат, а это означает, что процесс приведения грамматики к необходимому виду должен быть выполнен (автоматически или вручную) до генерации его синтаксического анализатора.

Одними из требований к технологии построения анализатора являются минимальный размер анализатора в случае его встраивания в микропроцессоры, общая низкая стоимость разработки, минимальное время, необходимое для получения новой рабочей грамматики после преобразований, адаптированной к выбранному методу анализа, и удобный интерфейс с пользователем при разрешении конфликтов.

В процессе написания рабочей грамматики входного языка естественным образом определяется набор лексических классов, используемых в грамматике взамен терминалов. Это служит исходной информацией для проектирования сканера, задача которого поставлять анализатору очередную лексему — запись, включающую имя (номер) лексического класса и информацию (или ссылку на нее), характеризующую конкретный лексический объект. Поскольку сканер получает микролексемы от транслитератора, то при разрабатывании сканера определяется задание на проектирование транслитератора в виде списка микролексических классов.

Задача бесконтекстного анализатора, как уже отмечалось в [19], состоит в том, чтобы построить конструкционную структуру программы. Результат этого этапа должен быть представлен в виде дерева грамматических конструкций, часто называемого *абстрактным деревом программы.* Каждое его поддерево представляет конкретную конструкцию, входящую в состав программы, а дуга, ведущая к корню поддерева, о котором идет речь, несет информацию о роли данной подконструкции по отношению к конструкции, непосредственной составляющей которой она является.

На рисунке 1 представлено поддерево дерева программы, представляющее конкретный экземпляр присваивания.

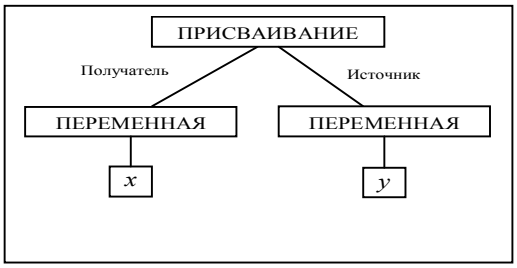



Рис. 1. Поддерево присваивания.

Как видим, переменные *x* и y играют роль получателя и источника данного присваивания. Сами x и y изображают ссылки на элементы таблицы идентификаторов. К моменту генерации машинного кода эти элементы будут содержать смещение относительно начала блока данных области памяти в стеке, отведенной под значение соответствующей переменной и блочные или процедурные уровни, к которым они относятся.

*Таблица идентификаторов* формируется при обработке описаний (констант, переменных, процедур, функций) и изначально содержит информацию о типах идентификаторов, таким образом представляя использующие вхождения идентификаторов. Она применяется на этапе идентификации идентификаторов. Каждое использующее вхождение идентификатора служит ключом доступа к этой таблице. На фазе определения типов конструкций и приведений из этой таблицы извлекаются типы простейших конструкций — использующих вхождений идентификаторов.

Подходящей формой представления программы на этом этапе может служить обратная польская форма, преимущество которой в том, что текстуальный порядок представления действий соответствует порядку их выполнения. Каждая конструкция программы «выстраивается» в таком порядке. Тогда контекстный анализ можно проводить методом «псевдоисполнения» программы подобно тому, как вычисляются выражения, представленные в обратной польской форме (ОПФ). Как известно, управляющий механизм различает в ОПФ операнды и операции: значение операндов помещается в магазин операндов, а операции исполняются, используя столько верхних значений из магазина, какова арность операции. Результат операции замещает использованные операнды. При псевдоисполнении программы вместо значений операндов фигурируют их типы.

Например, если бесконтекстное представление присваивания в обратной польской форме имеет вид: *x y* := , то во время контекстного анализа стековая машина извлечет из таблицы идентификаторов типы операндов *x* и *y* в стек, а пришедшая на вход операция :=, которая «знает», что ее операндов два, и она будет сопоставлять их типы и определять, какие приведения типов над каждым из них планировать.

Априорное значение присваивания — то имя (ссылка), которому присваивалось вещественное значение. В свою очередь, перед использованием этого значения присваивания возможно оно будет приводиться в соответствии с синтаксической позицией присваивания в составе конструкции, составляющей которой оно является.

## 2 Представление данных в объектной программе

Тип данных определяется совокупностью свойств и, в частности, ассортиментом операций, определенных в языке над значениями этого типа. Обычно определяется некоторое небольшое число элементарных или простых типов, таких как целые, вещественные, логические литерные и также небольшое число способов композиции сложных типов, например, массивы и записи (структуры).

Хотя данные простых типов с точки зрения пользователя языка являются неделимыми единицами информации, с точки зрения реализатора языка их представление должно определяться с точностью до бита.

Определить способ представления данных значит ответить на три вопроса:

- какую информацию о значении данного типа следует представлять;
- как кодировать эту информацию;
- какова структура памяти, в которой размещается этот код.

### 2.1 Представление простых типов

Как правило, для этих типов данных удается подобрать машинные форматы таким образом, чтобы операции выполнялись одной или несколькими машинными командами. Обычно для целых выбираются машинные форматы представления с фиксированной запятой, для вещественных — форматы с плавающей запятой, для литер — байты и т. д. По возможности учитываются градации разрядности для целых, длинных или коротких целых, а для вещественных значений — различные градации точности представления, диапазоны значений. Для логических значений предусматривается два различных способа представления — в памяти и на регистре условий.

### 2.2 Представление составных типов данных

Различным способам композиции типов данных, определенным в языке, в реализации должны быть найдены адекватные способы композиции представлений. Например, в Паскале определены следующие способы композиции видов: имена, структуры, массивы, процедуры, объединения, которые могут применяться рекурсивно. Реализатор языка должен определить способы композиции их представлений.

### 2.2.1. Имена

Имена служат для того, чтобы именовать значения. Они являются аналогом переменных, но не обязательно связаны с внешним обозначением в форме идентификатора. Имя начинает именовать некоторое значение в результате присваивания этого значения данному имени. В машинных терминах присваивание можно интерпретировать как замещение одного значения другим в некоторой области памяти. Соответственно имена естественно трактовать как адреса областей памяти, структура и размер которых зависит от вида значений, представление которых в них размещено. Однако следует заметить, что хотя в языке различаются имена разных видов, в реализации все они имеют одинаковое представление в виде адреса. Это возможно благодаря тому, что информация о видах значений полностью учтена в командах, реализующих соответствующие действия над значениями. В тех случаях, когда для выполнения присваиваний в реализации используется универсальная поддерживающая подпрограмма, информация о видах передается в виде машинной константы, представляющей шаблон значений, участвующих в данном присваивании.

Поскольку имена имеют различные области действия, которые при присваиваниях как раз и должны контролироваться (область действия значения источника не должна быть новее области действия значения получателя, иначе имя, которому присваивается некоторое значение, может оказаться доступным в то время, когда именуемое им значение уже не существует), то для точного воспроизведения присваиваний в представление имени помимо адреса области следовало бы включать адрес в стеке, характеризующем область действия представляемого имени.

### 2.2.2. Структуры (записи)

Структуры или записи являются составными значениями, т. е. состоящими из одного или нескольких других значений произвольных видов, называемых ее полями. В языке ссылка на поля производится с помощью идентификаторов — выделителей полей. Характерная операция над структурами — выборка поля с помощью заданного выделителя.

Естественно выбрать представление структуры просто как последовательность представлений ее полей, а выделители полей представлять смещениями областей памяти, занимаемых представлениями полей, относительно начала области, занимаемой всей структурой. Таким образом, выделитель первого поля представляется нулевым смещением, второе поле имеет смещение, равное размеру области занятой первым полем, и т. д. Иногда существенно учитывать выравнивание полей (alignement requirement) из-за того, что каждый тип значений должен размещаться по адресу надлежащей кратности. Чтобы не терять много памяти на выравнивание, имеет смысл упорядочивать поля по убыванию требуемой кратности. Например, сначала разместить все поля, требующие выравнивания 8, затем 4, затем 2 и, наконец, 1.

### 2.2.3. Массивы

Массивы являются составными значениями, состоящими из других значений одного и того же типа, называемыми его элементами. Виды массивов различаются как размерностью (числом измерений), так и видами элементов. Характерная операция над массивами — вырезка. Ее результатом может быть как отдельный элемент, идентифицируемый набором индексов (их число равно размерности массива), так или любая его прямоугольная часть — подмассив той же или меньшей размерности. В любом случае конструкция вырезки состоит из конструкции, результат которой есть массив или имя массива, и индексатора, состоящего из индексов и триммеров, общее число которых равно размерности массива. Индексатор и задает, что именно вырезается.

Очевидно, представление массива состоит из представлений его элементов соответственно их типу. Остается решить вопрос об их размещении в памяти машины и соответствующем способе доступа к ним.

Чтобы доступ был эффективным, разумно элементы расположить простым образом. Например, в одномерном массиве один элемент за другим, в двумерном (матрице) — за элементами первой строки, элементы второй и т. д. В трехмерном массиве — за элементами первого слоя элементы второго и т. д.

### 2.2.4. Процедуры

Представление процедурных значений будет обсуждаться в следующей части статьи в связи с рассмотрением реализации вызовов и формул — характерных конструкций, в которых используются процедурные значения.

### 2.2.5. Объединения

Строго говоря, значений объединенных типов не бывает. Бывают имена, которым можно присваивать значения не одного фиксированного типа, а нескольких разных типов, определенных описанием этого имени. Под такое имя выделяется область памяти, по объему достаточная для размещения любого из значений, допускаемых этим именем. Эта область также включает тег, показывающий, какой из возможных вариантов типа в настоящий момент располагается в этой области. Фактический способ обработки такого значения выбирается конструкцией, называемой сопоставляющим предложением, которое в зависимости от значения этого тега выбирает соответствующий способ обработки (интерпретации кода, содержащегося в этой области). В Паскале, например, подобный объединенный тип представлен записями с вариантами.

### 3 Основные понятия методики разработки анализаторов

Для описания методики реализации синтаксического анализатора неформально перечислим определения, обозначения и понятия, используемые в теории формальных языков и абстрактных автоматов, часть из них модифицирована. Затем дадим определение КС-грамматики в регулярной форме (КСР-грамматики) и его графического аналога – синтаксической граф-схемы как промежуточной формы при синтезе автомата — анализатора.

В работах [1–4, 12–18] даны строгие определения основных понятий, используемых в доказательствах утверждений при построении «правильного» транслирующего автомата: «Алфавит $V = \{\lambda, a_1, a_2, \ldots, a_n\}$ представляет собой конечный набор букв. $\lambda$ — пустая буква. Слово в алфавите $V$ является либо буквой из $V$, либо конкатенацией $xa$, где $x$ — слово в алфавите $V$ и $a \in V$. Если $x$ — слово в $V$, то $|x|$ обозначает его длину — количество вхождений букв в $x$; слово длины 0, которое не содержит ни одной буквы, называется пустым словом. Если $V$ — алфавит, то $V^*$ — множество всех слов в $V$, включая пустое слово. $V^+$ — множество слов из $V \setminus \{\varepsilon\}$. Язык $L$ над некоторым алфавитом $V$ — это произвольное подмножество $V^*$: $L \subset V^*$. Для бесконечного языка существует проблема его конечного представления, которое само является представлением, то есть словами над некоторым алфавитом с подразумеваемой интерпретацией, связывающей его с конкретным представлением данного языка. Для любых языков (подмножеств $A$ и $B$ в алфавите $V$) $A;B$ будет обозначать их *объединение*, $A,B$ будет обозначать их *произведение* (конкатенацию), а $A\#B$ будет обозначать *обобщенную итерацию* (иногда называемую итерацией с разделителем или обобщённой итерацией по Цейтину). *Регулярным множеством* слов (регулярным языком) над алфавитом $V$ называют следующие множества: $e = \{\varepsilon\}$ — (пустое слово); Ø — (пустое множество слов); $\{a_i\}$ — элементарные множества слов над алфавитом $V$ для любого $a_i \in V$; множества $P \cup Q, PQ, P^*$, где $P$ и $Q$ — регулярные множества в алфавите $V$. Бинарная *обобщённая итерация* (#) может быть определена через традиционную (унарную) операцию Клини как $P\#Q = P,(Q,P)^*$. Таким образом, множество слов в алфавите $V$ является регулярным тогда и только тогда, когда оно равно либо $\{a_i\}$ для некоторого $a_i \in V$ или если оно может быть построено из них с помощью конечного числа операций объединение (;), произведение (,) или обобщенная итерация (#).

В конце прошлого столетия Стивен Коул Клини [2] ввел понятие *регулярное выражение* в алфавите $V$ для представления регулярных множеств. Таким образом, итерации Клини $A^*$ и $A^+$ могут быть идентифицированы с $\#A = A^*$ и $A\# = A^+$ соответственно и обобщённое регулярное выражение определяется следующим образом: «*Обобщённым регу-*

*лярным выражением* множества $A$ называется слово $r(A)$ над расширенным алфавитом $W$, где $W = V \cup \{\#,*,+,;,(,),\varepsilon,\varnothing\}$, причем:

Если $A = \varnothing$, то $r(A) = \varnothing$; если $A = e$, то $r(A) = \varepsilon$; если $A = \{a\}$, где $a \in V$, то $r(A) = a$; если $r(P) = p$ и $r(Q) = q$ — регулярные выражения для множеств $P$ и $Q$ тогда:

$r(A) = (p;q)$ для множества $(P \cup Q)$; $r(A) = (p,q)$ для множества $(PQ)$;

$r(A) = (p^*)$ для множества $P^*$; $r(A) = (p^+)$ для множества $P^+$;

$r(A) = (p\#q)$ для множества $(P\#Q)$.

*Ничто другое не является обобщённым регулярным выражением».*

Для каждого регулярного выражения $A$ пусть $L(A)$ будет соответствующим регулярным языком. Каждое регулярное выражение представляет регулярный язык и для каждого регулярного языка существует регулярное выражение, представляющее этот язык (это можно легко доказать с помощью индукции по процессу построения). Однако поскольку регулярный язык можно построить с помощью операций объединения, произведения и итерации многими различными способами, в общем случае различные регулярные выражения могут представлять один и тот же регулярный язык. Здесь возникает известная проблема — как найти минимальное регулярное выражение, представляющее заданный язык.

В работах [14; 18] даны определения КСР-грамматики и синтаксической граф-схемы:

«Контекстно-свободной грамматикой в регулярной форме (КСР-грамматикой) называется пятерка множеств $G_R = (N,T,\Sigma,P,S)$, где $N$ — множество нетерминалов, $T$ — множество терминалов, $\Sigma$ — множество семантик, $P$ — множество КСР-правил, $P = \{A : R_A. \mid A \in N, R_A$ — регулярное выражение над алфавитом $N \cup T \cup \Sigma\}$, $S$ — начальный нетерминал грамматики».

Правила КСР-грамматики (КСР-правила) удобно представлять в виде конечных ориентированных графов с помеченными вершинами и дугами. Такие графы образуют *синтаксическую граф-схему* (СГС), являющуюся графовым аналогом правил КСР-грамматики [11–16], а вывод в грамматике (сентенциальные формы или деревья вывода) заменяется более простой структурой — маршрутом в СГС.

На рисунке 2 представлены три основные операции над элементарными регулярными выражениями

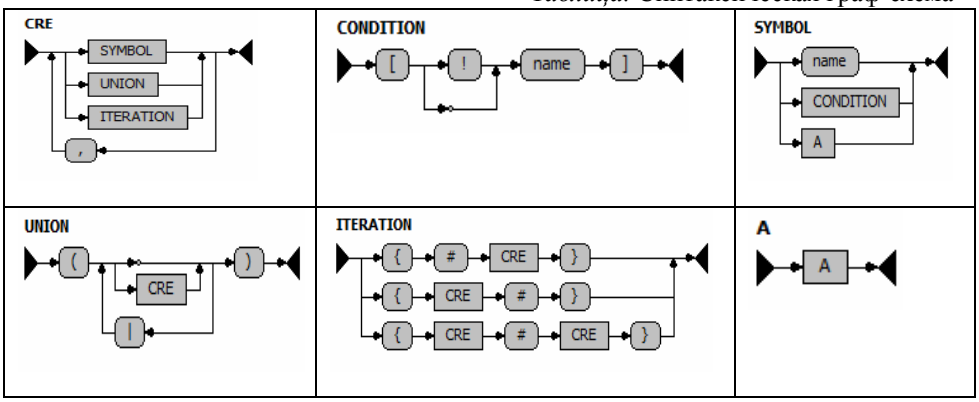

Рис. 2. Представление базовых операций над регулярными выражениями

В качестве примера можно привести грамматику и синтаксическую граф-схему для обусловленных регулярных выражений (ОРВ) — средства формализации требований, используемых при построении семантического графа связей взаимодействующих автоматных объектов.

Правила синтаксиса языка ОРВ записаны в тех же соглашениях, что использованы в статье [20]. Терминальные символы в правилах подчеркнуты. Метасимволы и нетерминалы выделены полужирным начертанием.

**CRE : ( SYMBOL | UNION | ITERATION ) # , .**
**UNION : ( CRE # | ) .**
**ITERATION : { # CRE } | { CRE # } | { CRE # CRE } .**
**SYMBOL :** name **| CONDITION | A .**
**CONDITION : [** !name **|** name **] .**
**A :** A **.**

*Таблица.* Синтаксическая граф-схема




## 4 Алгоритмы регуляризации в разработке анализатора

Основной целью эквивалентных преобразований КСР грамматики является, во-первых, эквивалентное сведение ее к форме, близкой к регулярной, когда в результате подстановок вместо вхождений нетерминалов в правые части правил их порождений, количество правил в грамматике, возможно, уменьшится в идеале до одного правила (КСР-грамматика сводится к одному регулярному выражению) и, во-вторых, выполнение условий, гарантирующих построение детерминированного анализатора.

Термин «регуляризация» перекликается с термином «нормализация». Нормализация — это преобразование в нормальную форму по Грейбах (Greibach S.A.), по Хомскому (Noam Chomsky)), которая является ограниченной формой грамматики, эквивалентной исходной КС-грамматике. КС-грамматика в нормальной форме Грейбах естественным образом преобразуется в МП-автомат, который допускает в точности тот же язык.

Под *регуляризацией контекстно-свободной грамматики (КС-грамматики)* понимается процедура её поэтапного эквивалентного преобразования с целью получения эквивалентной грамматики в форме

грамматики с регулярными выражениями в правых частях правил и не имеющей конфликтов с построением правильной управляющей таблицы автомата.

Как было сказано в [19], построение анализатора языка начинается с преобразования исходной КС-грамматики в эквивалентную, правильно сформированную КСР-грамматику. Базовые операции и алгоритмы преобразования с доказательствами их корректности даны в работах [13–18]. Перечислим их:

- устранение непродуктивных нетерминалов; замена нерекурсивных нетерминалов регулярными выражениями; устранение левой или правой рекурсии; определение общих префиксов; устранение рекурсивных альтернатив для нетерминала; устранение оставшихся рекурсивных нетерминалов; введение дополнительных новых нетерминалов для общих регулярных подвыражений; удаление лишних нетерминалов. Подробное описание процесса эквивалентных преобразований дано в [14].

Набор базовых функций, реализованных в системе SynGT, служит основной цели этой системы — автоматизированному преобразованию КС грамматики с целью ее регуляризации, то есть превращению синтаксической граф-схемы грамматики в минимальный набор конечных автоматов для построения анализатора языка. Если КС грамматика порождает регулярный язык, то синтаксическая граф-схема вырождается в один граф, представляющий эквивалентное регулярное выражение, которое обрабатывается конечным автоматом.

## Заключение

В данной части статьи кратко перечислены этапы разработки анализирующей части при построении транслятора — подготовка входной грамматики, её формирование с целью приведения в форму, пригодную для автоматизированного построения управляющих таблиц анализатора, её регуляризации, которая является частью полного цикла реализации языка. В следующей запланированной статье будут рассмотрены алгоритмы построения состояний преобразователя как абстрактной модели при построении анализатора.

## Литература

1. Ginsburg S. The mathematical theory of context-free languages. New York: Mc Graw-Hill Inc., 1966 / перевод на русский язык и под редакцией А. В. Гладкого. Москва: Мир, 1970. 326 с.

2. Клини С. К. Введение в метаматематику / перевод с английского А. С. Есенина-Вольпина; под редакцией В. А. Успенского. Москва: Изд-во иностр. лит., 1957. 526 с.

3. Гладкий А. В. Формальные грамматики и языки. Москва, 1973. 368 с.

4. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. Москва: Мир, 1978. Т. 1. 612 с.

5. Грис Д. Конструирование компиляторов для цифровых вычислительных машин. Москва: Мир, 1975. 544 с.

6. Касьянов В. Н., Поттосин И. В. Методы построения трансляторов. Новосибирск: Наука, 1986. 343 с.

7. Льюис Ф., Розенкранц Д., Стирнз Р. Теоретические основы проектирования компиляторов. Москва: Мир, 1979. 654 с.

8. Пратт Т. Языки программирования: разработка и реализация / перевод с английского и под редакцией Ю. М. Баяковского. Москва: Мир, 1979. 668 с.

9. Aho A., Sethi R., Ullman J., Compilers: Principles, Techniques and Tools. Addison-Wesley Publishing Company, 1986. 796 p.

10. Baranov S., Lavarenne C. Open C Compiler in Forth. In: EuroForth'95, 27–29 Oct. Schloss Dagstuhl (1995). Москва, 1979. 574 с.

11. Fraser C., Hanson D. A retargetable C compiler: Design and implementation. Addison-Wesley Pub. Company, Menlo Park. California, 1995. 564 p.

12. Федорченко Л. Н., Мартыненко Б. К. Эквивалентные преобразования КСР грамматик в регулярной форме в практике построения языковых процессоров. Часть первая. Определение и распознавание КСР-языков посредством синтаксических граф-схем / Ленинградский научно-исследовательский вычислительный центр. Ленинград: Изд-во АН СССР, 1983. 54 с.

13. Федорченко Л. Н. Извлечение крайней рекурсии из КСР грамматики в системе SynGT // Труды СПИИРАН. 2002. Т. 1. С. 350–359.

14. Fedorchenko L. Regularization of Context-Free Grammars. Saarbrucken: LAP LAMBERT Academic Publishing, 2011. 188 p.

15. Федорченко Л. Н. О регуляризации контекстно-свободных грамматик // Известия вузов. Приборостроение. 2006. Т. 49, № 11. С. 50–54.

16. Fedorchenko L. N., Soloviev S. V., Naumov I. N., Mehats L. Syntax Graph Transformations in the System SynGT and their Applications: Rapport IRIT/2003-06-R, UMR 5505 CNRS-INP-UPS. 2003. 15 p.(English).

17. Федорченко Л. Н. Синтаксически управляемая обработка данных для практических задач // Вестник БГУ. 2013. № 9. С. 87–99.

18. Ludmila Fedorchenko, Sergey Baranov. Equivalent Transformations and Regularization in Context–Free Grammars // Bulgarian Academy of Sciences/ Cybernetics and Information Technologies (CIT). Sofia, 2015. Vol. 14, No. 4. P. 11–28.

19. Федорченко Л. Н. Методика автоматизированной обработки информации с использованием языка программирования высокого уровня. (Часть 1) // Вестник Бурятского государственного университета. Математика, информатика. 2024. № 1. С. 46–55.

20. Язык спецификации взаимодействия автоматных объектов / Ф. А. Новиков, И. В. Афанасьева, Л. Н. Федорченко, Т. А. Харисова // Научно-технический вестник информационных технологий, механики и оптики. 2024. Т. 24, № 6. С. 907–915. (на англ. яз.) doi: 10.17586/2226-1494-2023-24-6-907-915.

*Статья поступила в редакцию 28.12.2024; одобрена после рецензирования 31.03.2025; принята к публикации 02.04.2025.*

METHODOLOGY FOR AUTOMATED INFORMATION PROCESSING USING A HIGH LEVEL PROGRAMMING LANGUAGE. PART 2

*Ludmila N. Fedorchenko*
PhD, Senior Researcher
St. Petersburg Federal Research Center of the Russian Academy of Sciences (SPC RAS)
39 14th Line of V.O., St. Petersburg 199178, Russia
lnf@iias.spb.su

*Alexander S. Geida*
PhD, Head of Applied Informatics Laboratory
St. Petersburg Federal Research Center of the Russian Academy of Sciences (SPC RAS)
39 14th Line of V.O., St. Petersburg 199178, Russia
geida@iias.spb.su

*Abstract.* This part of the paper is devoted to the development of a language analyzer defined by translation grammars in a regular form. The described technique was first applied in the 1970s in a compiler for ALGOL 68. Constraints are formulated on the grammar, presented in the form of a system of multicomponent directed graphs (like extended Wirth diagrams), guaranteeing the existence of a language processor similar to a deterministic pushdown transducer, which is then considered as a control mechanism scanning the input chain from left to right, fixing the sequence of states to initiate actions that make up the translation process. The specificity of the technique is the grammar regularization algorithms based on equivalent transformations of the grammar syntactic graph: elimination of recursion and insertion of iterations. Regularization of the original grammar is a part of the full cycle of language implementation, consisting of the user cycle and the semi-automated developer cycle.

*Keywords*: automated information processing, compilation process diagram, syntactic model of language, language processor.

*For citation*
*Fedorchenko L. N., Geida A. S.* Methodology for Automated Information Processing Using a High Level Programming Language. Part 2 // Bulletin of Buryat State University. Mathematics, Informatics. 2025. N. 1. P. 65–78.

*The article was submitted 28.12.2024; approved after reviewing 31.03.2025; accepted for publication 02.04.2025.*