МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский государственный университет аэрокосмического приборостроения»

Т.М. Максимова

МЕТОДЫ ТРАНСЛЯЦИИ

Учебное пособие

Санкт-Петербург

2015

Рецензент: д.т.н., профессор А.В.Гордеев

В методическое пособие включён материал, изучаемый в рамках дисциплины «Методы трансляции», относящийся к методам синтаксически управляемой трансляции и сопровождённый заданиями на лабораторные работы. Методическое пособие предназначено для студентов специальностей 09.03.04- Программная инженерия (бакалавр), 02.03.03 - Математическое обеспечение и администрирование информационных систем (бакалавр), изучающих дисциплины «Теория формальных языков» и «Методы трансляции».

Методическое пособие подготовлено на кафедре компьютерной математики и программирования и рекомендовано к изданию редакционно-издательским советом Санкт-Петербургского государственного университета аэрокосмического приборостроения.

1. Трансляция. Основные термины и определения

Трансляторами называют программы, выполняющие перевод текстов программ, написанных на одном (исходном) языке в эквивалентные тексты программ на другом (выходном) языке.

Со словом «транслятор» чаще всего ассоциируется преобразование текста программы, написанного на языке высокого уровня, в последовательность команд, которую надлежит исполнить процессору.

Более общее (абстрактное) представление о трансляторе [3] определяет его как устройство, обеспечивающее для данной строки $x \in \Sigma^*$ вычисление выходной строки $y \in \Delta^*$ такой, что пара (x, y) – элемент множества, называемого трансляцией и определяемого как отношение $\tau \subseteq \Sigma^* \times \Delta^*$. Здесь Σ – алфавит исходного языка, Δ – алфавит выходного языка, * – звёздочка Клини (Стивен Коул Клини – американский математик), обозначающая, нестрого говоря, операцию получения бесконечного множества всевозможных строк из элементов множества-операнда и пустой строки.

По-видимому, наиболее сложными устройствами, реализующими трансляции, являются автоматические переводчики естественных языков, поскольку затруднено построение формальных описаний таких языков.

Среди других областей применения методов трансляции языков программирования можно назвать любую обработку текстовых документов, распознавание изображений и т.д.

Если понимать трансляцию как процедуру построения выходного текста для заданного исходного, следует сказать, что в процессе выполнения трансляции должна быть выстроена структура исходного текста, позволяющая оценить его синтаксическую корректность и, одновременно, выявить его содержание (семантику). Так, например, очевидно, что выражение *a+b+c*, записанное на каком-либо языке программирования высокого уровня, в результате трансляции в код какого-либо процессора окажется последовательностью двух двуместных команд сложения (если не принимать во внимание возможные команды перемещения слагаемых в памяти). Которая из этих двух команд окажется первой в последовательности, зависит от того, какая из двух возможных структур этого выражения будет построена в процессе трансляции. Наглядной формой описания структуры является дерево, крону которого образуют символы исходного текста. Две вышеупомянутые структуры можно изобразить следующими двумя деревьями:

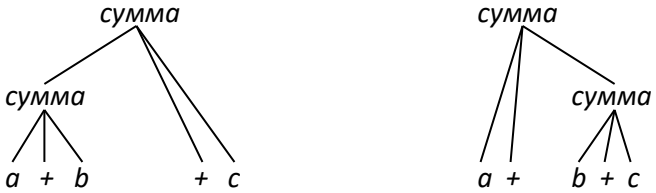


Рис.1. Две структуры арифметического выражения

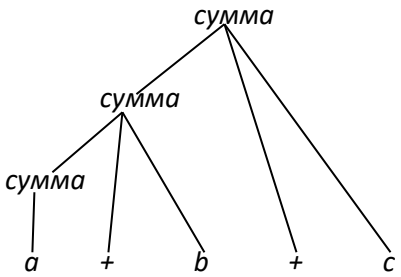
Левая из этих структур предписывает выполнение операций сложения в порядке слева направо, правая из двух структур — выполнение операций сложения в порядке справа налево. Соответствующим образом в каждом из этих случаев и будет сгенерирована последовательность команд сложения в выходном коде. Наверное, не лишним будет заметить здесь, что закон о независимости суммы от перестановки мест слагаемых на компьютерную арифметику, вообще говоря, не распространяется, поскольку точность представления числа в памяти компьютера ограничена.

В решении задачи построения структуры исходного текста транслятор должен опираться на формальное описание исходного языка: его синтаксис (правила написания текстов) и семантику (содержание правильных текстов).

В описание синтаксиса зачастую оказываются вкраплёнными и семантические аспекты языка, что, в частности, наглядно демонстрирует рассмотренный пример с двумя операциями сложения. Описание синтаксиса не должно предоставлять транслятору возможности построения нескольких вариантов структур одного исходного текста. И тогда единственно возможная структура однозначно в нашем примере определит содержание исходного текста: последовательность операций сложения. Таким описанием синтаксиса может быть, скажем, следующее, позволяющее формировать выражение из произвольного количества слагаемых, в том числе – из одного:

сумма — это сумма плюс a
сумма — это сумма плюс b
сумма — это сумма плюс c
сумма — это a
сумма — это b
сумма — это c

По такому описанию возможно построение только одной структуры выражения *a+b+c* :



Операции сложения в соответствии с этой структурой следует выполнять в порядке слева направо.

Задача распознавания исходного текста путём выстраивания его структуры в терминологии формального описания синтаксиса языка называется задачей разбора. Дерево, представляющее структуру распознанного текста, называется деревом разбора.

2. Синтаксически управляемые методы трансляции

Методы, которыми реализуются трансляции, разделяются на прямые и синтаксически управляемые. Прямые методы предполагают создание индивидуальных алгоритмов для трансляции различных конструкций (синтаксических единиц) языка. Синтаксически управляемые методы используют один алгоритм для трансляции всех синтактических единиц языка. Такой алгоритм заключается в исполнении команд некой управляющей таблицы, которая предварительно должна быть построена конструктором транслятора по заданному формальному описанию языка. В дальнейшем сосредоточимся именно на таких методах конструирования трансляторов.

В качестве средства формального описания языков будем использовать форму Бэкуса-Наура (Джон Бэкус – американский учёный в области компьютерных наук, Петер Наур – датский учёный в области компьютерных наук). В её обозначениях приведённое выше описание синтаксиса изображения сумм *a*, *b* и *c* может быть записано в виде 6 правил:

<сумма>::=*< сумма>+a|< сумма>+b|< сумма>+c|a|b|c*

Строго говоря, формальным описанием синтаксиса языка является грамматика, компонентами которой, помимо множества правил, должны быть ещё: множество терминальных символов, множество нетерминальных символов и аксиома. Эти неупомянутые явно компоненты достаточно очевидно просматриваются в правилах:

{*a*, *b*, *c*, *+*} – множество терминальных символов,

{*<сумма>*} – множество нетерминальных символов,

<сумма> – аксиома.

Возможность построения дерева разбора для строки терминальных символов – это ещё и возможность построения вывода этой строки из аксиомы грамматики. Для строки *a+b+c* вывод выглядит так: *<сумма> => <сумма>+c => <сумма>+b+c => a+b+c* . Каждый шаг вывода – это замена в текущей форме одного нетерминального символа на правую часть его правила. Все получаемые в этом процессе формы, включая аксиому и строку терминалов (предложение языка), называются сентенциальными формами.

Если в правилах грамматики слева от символа ::= (он читается как «это есть по определению») находится только один символ (таким символом может быть только нетерминальный), грамматика по классификации Хомского (Ноам Хомский – американский лингвист) относится к классу контекстно-свободных (КС-грамматик). Соответственно, языки, которые описываются КС-грамматиками, являются КС-языками. Синтаксически управляемые трансляторы для таких языков можно строить на базе так называемых магазинных автоматов. Построение магазинного автомата является задачей конструктора транслятора, которая решается по известным алгоритмам, использующим КС-грамматики в качестве исходных данных. Собственно, главной задачей конструктора является построение управляющей таблицы, центральной компоненты автомата. Структура магазинного автомата, как показано на рисунке, предполагает, что процесс его функционирования (трансляции) — последовательное исполнение команд, выбираемых из управляющей таблицы по координатам, указанным содержанием вершины магазина (стека) и очередным читаемым символом исходного (транслируемого) текста.

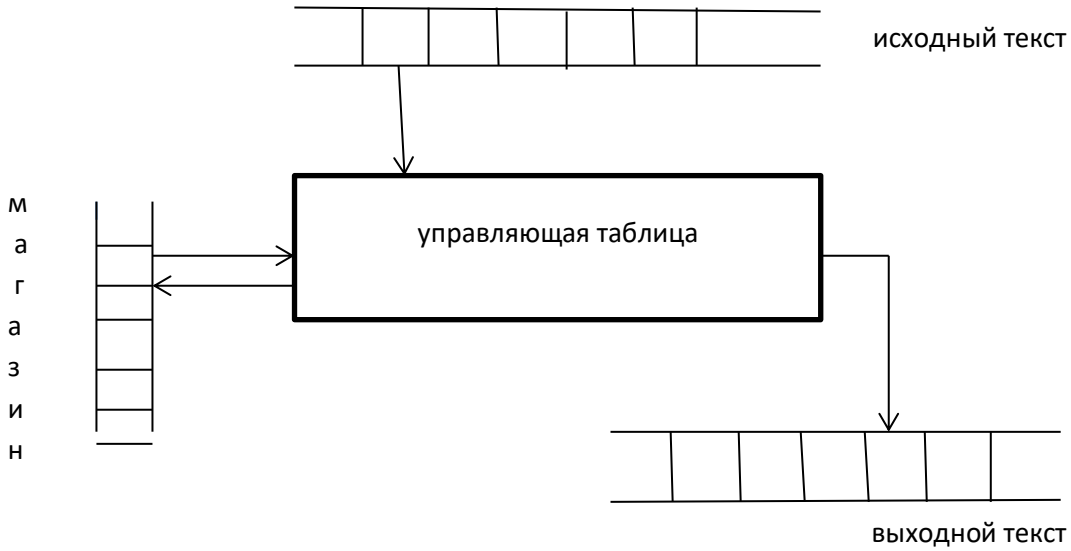


Рис.2. Структура магазинного автомата

Исполняя выбираемые из управляющей таблицы команды, автомат может продвигаться в чтении символов исходного текста в направлении слева направо, изменять содержание стека (удалять из него записи или помещать в него новые записи), генерировать фрагменты выходного текста, последовательно размещая их на выходной ленте.

Практический интерес, с точки зрения построения реальных трансляторов, представляют детерминированные магазинные автоматы. Каждая ячейка управляющей таблицы такого автомата содержит только одну команду, единственно верную в текущей конфигурации функционирования автомата (при текущем содержании стека и текущем сканируемом символе исходного текста) и позволяющую завершить трансляцию без перебора вариантов решений на отдельных шагах алгоритма и без возвратов к уже прочитанным символам исходного текста. Детерминированные магазинные автоматы могут быть сконструированы только для частных видов КС-грамматик.

Независимо от применяемых методов трансляции, прямых или синтаксически управляемых, из этапа синтаксического анализа обычно выделяют лексический анализ. Задачей лексического анализа является распознавание простейших конструкций исходного текста — лексем. Если говорить о трансляции естественных языков, то лексемами можно считать слова. Анализ корректности предложения, которое собрано из слов, а также содержание этого предложения — это задачи синтаксического и семантического анализов, решение которых существенно упрощается предварительным лексическим анализом.

2.1. Лексический анализ с использованием конечного автомата

Словами в текстах программ обычно считаются идентификаторы, всевозможные константы (числовые, строковые и т.д.), обозначения операций, знаки препинания. Правила грамматик, описывающие такие конструкции, представляют собой частный вид правил КС-грамматик: правая часть правила содержит два символа, один из которых — терминальный, а другой — нетерминальный, или один символ, являющийся терминальным.

Так, например, определение идентификатора как последовательности произвольной длины букв и цифр, начинающейся с буквы, при условии, что буквами считаются *a* и *b*, цифрами — *1* и *2*, формализуется следующими правилами Бэкуса-Наура:

<идентификатор>::=*a*|*b*|< *идентификатор* >*a*|< *идентификатор* >*b*|< *идентификатор*>*1*|< *идентификатор* >*2*

Такой частный вид КС-грамматики по классификации Хомского составляет класс автоматных грамматик, а языки, которые ими описываются, называются автоматными языками. Соответственно, автомат, на базе которого можно построить лексический анализатор, представляет собой частный вид магазинного автомата: у него отсутствует магазин. Эта модель называется конечным автоматом. Автомат всегда пребывает в каком-либо внутреннем состоянии, позволяющем определить координату строки управляющей таблицы, откуда следует выбрать очередную команду для исполнения (координата столбца таблицы, по-прежнему, определяется сканируемым символом исходного текста). Каждая команда управляющей таблицы содержит информацию о следующем внутреннем состоянии автомата.

Формально детерминированный конечный автомат задаётся:

- множеством Σ входных символов,

- множеством V внутренних состояний,

- множеством P переходов между внутренними состояниями под воздействием входных символов,

- F начальным состоянием ($F \in V$),

- множеством $S \subseteq V$последних состояний.

Исходный текст считается принадлежащим языку, принимаемому автоматом, если, начиная с начального состояния, выполнив последовательность переходов по внутренним состояниям под воздействием символов исходного текста, автомат оказывается в одном из последних состоя-

ний. Иными словами, такой исходный текст считается синтаксически (лексически) верным. Результат трансляции слов должен быть удобным, с точки зрения последующих этапов работы транслятора, представлением (кодом) распознанных лексем. Такие коды называют токенами, и они считаются терминальными символами в грамматике, описывающей синтаксис более сложных конструкций языка, то есть синтаксис предложений, состоящих из слов-токенов.

Управляющая таблица конечного автомата, распознающего идентификаторы в соответствии с вышеприведёнными правилами, выглядит так:

	a	*b*	*1*	*2*
начальное состояние	*идентификатор*	*идентификатор*		
идентификатор (последнее состояние)	*идентификатор*	*идентификатор*	*идентификатор*	*идентификатор*

Пустые ячейки таблицы считаются содержащими команду «ошибка».

Нетрудно видеть, что содержание управляющей таблицы определено в соответствии с правилами грамматики. В общем случае при наличии в грамматике правила вида *X* ::= *t*, где *t* — терминальный символ ($t \in \Sigma$), в ячейку управляющей таблицы с координатами (*начальное состояние*,*t*) следует поместить *X*, что будет рассматриваться как команда перехода в состояние *X* из начального состояния под воздействием входного символа *t*.Правило же вида *X* ::= *Y t,* где *Y* — нетерминальный символ ($Y \in V$), предписывает поместить ту же команду *X* в ячейку с координатами (*Y*,*t*), что будет соответствовать переходу автомата в состояние *X* из состояния *Y* под воздействием входного символа *t*. Последнее состояние автомата обычно определяется аксиомой грамматики.

В примере с идентификатором правила используют единственный нетерминальный символ — <*идентификатор*>, а значит, он же в простейшей грамматике является и аксиомой, а при построении конечного автомата соответствующее обозначение (*идентификатор*) приобретает последнее состояние.

Распознавание, например, слова *a21* построенным автоматом будет выполнена за 3 шага:

начальное состояние —*a*→ *идентификатор* —*2*→ *идентификатор* —*1*→ *идентификатор*

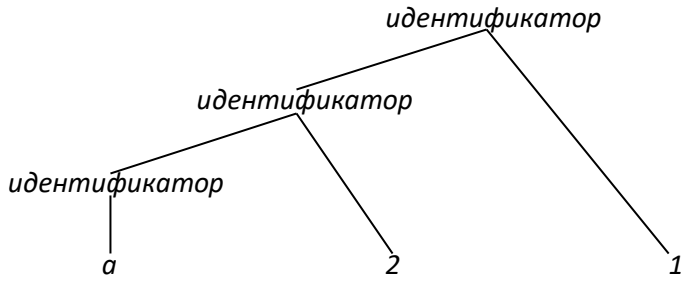
Анализируя идентификатор *a21*, как и любой другой, соответствующий определению идентификатора, автомат по первой букве исходного текста из начального состояния попадает в состояние «*идентификатор*» и остается в нём, пока на его вход продолжают поступать буквы (*a* или *b*) и цифры (*1* или *2*). Поскольку состояние «*идентификатор*» является последним состоянием автомата, автомат вправе остановиться на любом символе из этого потока букв и цифр. Эти остановки могли бы свидетельствовать о том, что любая подстрока, начинающаяся с первой буквы идентификатора, тоже является идентификатором. Если, всё же, речь идет об идентификаторах текстов программ на языках высокого уровня, как правило, требуется в текущей позиции анализируемого текста выявить самое длинное слово, являющееся идентификатором.

В таком случае процесс распознавания идентификатора должен завершиться по обнаружении в исходном тексте символа, не являющегося ни буквой, ни цифрой (ни каким-либо другим символом, разрешённым для использования в идентификаторах).

Аналогичная ситуация может иметь место и при распознавании других лексем, таких, например, как числовые константы.

Лексический анализатор в целом может быть сконструирован как единый конечной автомат с единственным начальным состоянием и несколькими последними состояниями по количеству типов лексем, которые предполагается обнаруживать в исходном тексте. Для распознавания каждой очередной лексемы автомат должен возобновлять свое функционирование с начального состояния. Какой символ исходного текста считать входным воздействием, выводящим автомат из начального состояния, зависит от свойства предыдущей (только что распознанной) лексемы. Если ее длина фиксирована, то входным воздействием для автомата является следующий символ текста, к нему следует переместиться по входной ленте вправо. Если же длина лексем таких типов может быть произвольной, входное воздействие для перехода из начального состояния уже было прочитано из исходного текста, послужив маркером конца лексемы, а теперь ему же предстоит стать первым символом новой лексемы.

В процессе функционирования конечного автомата происходит неявное выстраивание структуры распознаваемого слова в виде дерева разбора. Для идентификатора *a21* дерево разбора выглядит так:



Построение дерева как бы выполняется в направлении снизу вверх. Внутренняя вершина дерева соответствует внутреннему состоянию автомата, ее левый потомок соответствует предыдущему внутреннему состоянию, а правый — входному воздействию, в результате которого был выполнен переход из предыдущего в текущее внутреннее состояние. Переход из начального состояния в состояние «*идентификатор*» изображено в дереве веткой, состоящей из внутренней вершины «*идентификатор*» с единственным потомком *a*.

Направление, в котором выполняется неявное построение дерева разбора конечным автоматом, определяет принадлежность этого способа решения задачи разбора к восходящим алгоритмам.

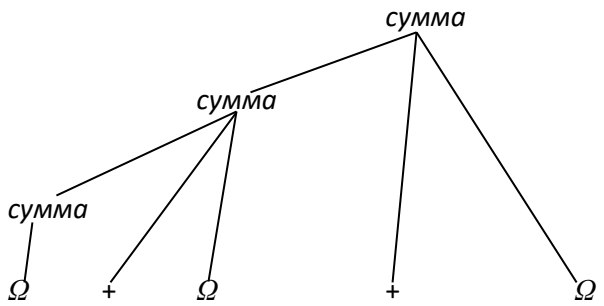
Что же касается семантического анализа, представляется естественным связать его с последними состояниями автомата и подключать семантическую обработку в моменты окончания распознавания лексем. Одна из задач (а в некоторых случаях и единственная задача) семантической обработки обнаруженной лексемы — генерация токена. Обычно для разных лексем генерируются разные токены, для одинаковых — одни и те же. Но возможны и варианты.

Поясним их на примере расширения возможностей изображения сумм, рассмотренном ранее. Пусть это расширение возможностей заключается в том, что слагаемым позволено быть произвольными идентификаторами (в соответствии с только что рассмотренным определением идентификатора), а не только односимвольными *a*, *b* и *c*. Если распознавание этих слагаемых-идентификаторов возложить на лексический анализатор, то его семантическая поддержка должна обеспечить генерацию токенов, которые были бы одинаковыми для всех обнаруженных в исходным тексте идентификаторов. Этот токен, с точки зрения синтаксиса изображения сумм, будет единственным терминальным символом, обозначающим слагаемые. Пусть, например, такой то-

кен будет представлен символом Ω. Тогда грамматика языка изображения сумм идентификаторов будет содержать только 2 правила:

<*сумма*> ::= <*сумма*> + Ω |Ω

С точки зрения синтаксиса языка изображения сумм неважно, какие именно идентификаторы участвуют в сумме, важно только то, что все они являются идентификаторами. Это последнее обстоятельство гарантируется лексическим анализатором, сгенерировавшим токен Ω . Для предложения *a21 + b1 + ab*, например, будет построено дерево:



Однако, с точки зрения семантики изображенной суммы, может быть важным, из каких именно слагаемых-идентификаторов она состоит: возможно, это — имена переменных, которым в выходном (объектном) коде в дальнейшем предполагается назначение адресов в памяти. Тогда в задачу семантической части лексического анализа следует включить и сохранение оригинала обнаруженной лексемы (в данном случае — идентификатора как последовательности символов исходного текста).

На семантическую часть лексического анализатора могут возлагаться и другие задачи, связанные с предметной областью синтаксически управляемой трансляции. Их решение под управлением лексического анализа, выполняемого с помощью конечного автомата, логично было бы назвать лексически управляемой трансляцией.

Программная реализации «лексически управляемой трансляции» может быть выполнена с помощью специально разработанных для этой цели средств, примером которых является утилита Flex. Утилита является генератором C/C++- кода, представляющего собой алгоритм функционирования конечного автомата по встроенной управляющей таблице с чтением символов исходного текста и исполнением действий семантической части лексического анализатора, приписанным последним состояниям автомата. На вход Flex следует подать описание синтаксиса лексем и, при необходимости, описания действий семантической обработки также в виде C/C++- кода.

Не приводя здесь полную формулировку теоремы С. Клини о связи конечных автоматов и языков, являющихся регулярными множествами, отметим только, что следствие из теоремы позволяет воспользоваться для описания синтаксиса лексем не правилами в форме Бэкуса-Наура, а так называемыми регулярными выражениями, поскольку оно утверждает, что каждое множество, принимаемое некоторым конечным автоматом, может быть представлено в виде регулярного выражения.

Это обстоятельство даёт возможность утилите Flex защититься от некорректных входных данных: описания языка, для распознавания которого невозможно построить конечной автомат.

В записи регулярных выражений операндами являются символы алфавита исходного языка, операциями — конкатенация, объединение (дизъюнкция) и итерация. В языке Flex знак конкатенации опускается, дизъюнкция обозначается символом ‘|’, итерация — символом ‘*’. Заметим,

что символ ‘*’ здесь, по-прежнему, имеет смысл звездочки Клини, так как результат итерации — ноль или более вхождений операнда.

Правила образования идентификатора из букв *a* и *b* и цифр *1* и *2*, о котором шла речь в этом разделе, в виде регулярного выражения выглядит так: (*a*|*b*)(*a*|*b*|*1*|*2*)*.

Круглые скобки в записи выражения используются для повышения приоритета операции дизъюнкции. Прочесть эту запись можно следующим образом: «Первый символ идентификатора — *a* или *b*, а за ним может следовать произвольное количество символов *a*,*b*, *1* или *2*».

Входной язык Flex, помимо основных операций над регулярными выражениями, содержит дополнительные средства для более удобной записи регулярных выражений. Подробнее ознакомиться с этими средствами можно в [4].

Что же касается семантической обработки лексемы, соответствующие действия в виде C/C++- кода заключаются в скобки {} и помещаются напротив регулярного выражения. При этом токен, как основной результат трансляции лексемы, чаще всего указывается в качестве операнда return. Например:

(a|b)(a|b|1|2) { return Omega;}*

Оператор *return*, как и любой другой текст, описывающий семантическую обработку лексемы, заносится утилитой Flex в генерируемый ею лексический анализатор таким образом, чтобы исполнение этой семантической обработки в процессе лексического анализа началось по достижении конечным автоматом последнего состояния, соответствующего окончанию распознавания данной лексемы. C/C++- код лексического анализа оформляется утилитой Flex в виде функции *int yylex()*. Оператор *return*, таким образом, предназначен для выхода из этой функции с возвращением целочисленного результата, который снаружи этой функции интерпретируется как токен распознанной лексемы. При использовании символьных имён для обозначения токенов (в примере с лексемой-идентификатором – *Omega*) следует позаботиться о том, чтобы в исполняемом коде эти имена приобрели целочисленные значения. Семантическая обработка распознанной лексемы имеет возможность доступа к этой лексеме как к последовательности символов исходного текста. Лексема помещается функцией *yylex* в локальную переменную *char* yytext*. Кроме того, на этапе лексического анализа можно сформировать некое внутреннее представление обнаруженной лексемы и записать его во внешнюю переменную *yylval*, имея в виду, что оно пригодится на следующем этапе синтаксически управляемой трансляции. По умолчанию эта переменная имеет тип *int*, но при необходимости может быть переопределена. К вопросу о деталях переопределения вернёмся позже.

Если, например, обнаружение идентификатора требуется сопровождать выводом его на экран, а его атрибутом (внутренним представлением) по какой-либо причине должна считаться его длина, то вышеприведённую строку регулярного выражения можно дополнить соответствующими семантическими действиями:

(a|b)(a|b|1|2) { cout<<”обнаружен идентификатор: ”<<yytext<<’\n’;*
yylval=strlen(yytext); return Omega;}

2.2. Синтаксический анализ с использованием автомата с магазинной памятью

Наличие у конечного автомата внутренних состояний позволяет ему запоминать историю поступающих входных воздействий. На один и тот же входной символ автомат может по-разному реагировать, поскольку его реакция определяется не только этим входным символом, но и тем, в каком внутреннем состоянии автомат находится в данный момент. Поэтому конечные автоматы называют ещё последовательностными схемами, так как по внутреннему состоянию автомата

можно судить о том, какова была последовательность символов, поступившая на его вход. Однако, конечной автомат не запоминает историю переходов между внутренними состояниями. И поэтому, как уже было сказано ранее, не любой язык, как множество последовательностей символов (строк), под силу распознавать конечному автомату. Если синтаксис предложений языка невозможно описать регулярными выражениями, то и невозможно сконструировать конечный автомат, способный понимать эти предложения. Одним из простейших примеров такого языка является бесконечное множество предложений типа:

$\underbrace{x...x}_{n}\ \underbrace{y...y}_{n}$, где n=1,2,…

Синтаксис этого языка можно описать правилами КС-грамматики:

$$S ::= x\,y \mid x\,S\,y$$

По этим правилам можно сконструировать автомат магазинного типа в соответствии со схемой на рис.2, принимающий такой язык. Магазин (стек) предназначен для запоминания истории смены внутренних состояний автомата, что и позволяет расширить класс принимаемых этой моделью языков по сравнению с регулярными языками, принимаемыми конечным автоматом.

В общем случае, принимающим для КС-языка является недетерминированый автомат магазинного типа. Ограничимся рассмотрением двух частных случаев КС-языков, для которых возможно построение детерминированных распознавателей: LR(k)-языки и LL(k)-языки. Такие наименования типы этих языков, естественно, получили от типов грамматик, которыми они описываются. Буквой k здесь обозначено целое число, указывающее количество символов исходного текста, которые автомат может считать текущим входным воздействием. Другими словами, k — это длина заголовка каждого столбца управляющей таблицы. Практическую ценность представляют такие грамматики, у которых k = 1. О них в дальнейшем и пойдёт речь.

2.2.1. LR(1)-разбор

Автоматом, построенным на основе LR(1)-грамматики, реализуется алгоритм разбора, относящийся к классу восходящих, так же, как и алгоритм разбора, реализуемый конечным автоматом. В стеке всегда находится некоторое количество имён внутренних состояний автомата. Имя, находящееся в вершине стека, — это имя текущего внутреннего состояния. Координаты ячейки управляющей таблицы, откуда должна быть выбрана очередная команда, определяются так же, как и в случае конечного автомата: внутреннее состояние определяет координату строки, текущий сканируемый символ исходного текста – координату столбца.

Сконструируем автомат магазинного типа, принимающий язык строк символов x и y, упомянутый выше. Первыми символами предложения языка могут быть только символы x. Принимая их в качестве входного воздействия, автомат должен сохранять некое внутреннее состояние, но при этом запоминать, сколько на его вход поступает символов x , заставляющих его переходить из текущего внутреннего состояния в него же. Запоминание необходимо для того, чтобы в дальнейшем можно было проконтролировать равенство количеств символов x и y, а реализуется запоминание стеком, в который помещается имя одного и того же внутреннего состояния с каждым новым поступлением символа x. Назовем это состояние, например, «*учёт* x». Приход символа y должен перевести автомат из состояния «*учёт* x» в состояние «*учёт* y». И теперь каждое новое поступление на вход автомата символа y должно удерживать его в состоянии «*учёт* y», но при этом сопровождаться возвратом по истории смены состояний на 2 шага назад, то есть удалением двух записей из стека. Одна из удаляемых записей это — «*учёт* y», а вторая — «*учёт* x». Если анализируемый текст синтаксически верен, то есть для каждого y найдется парный ему x , и на

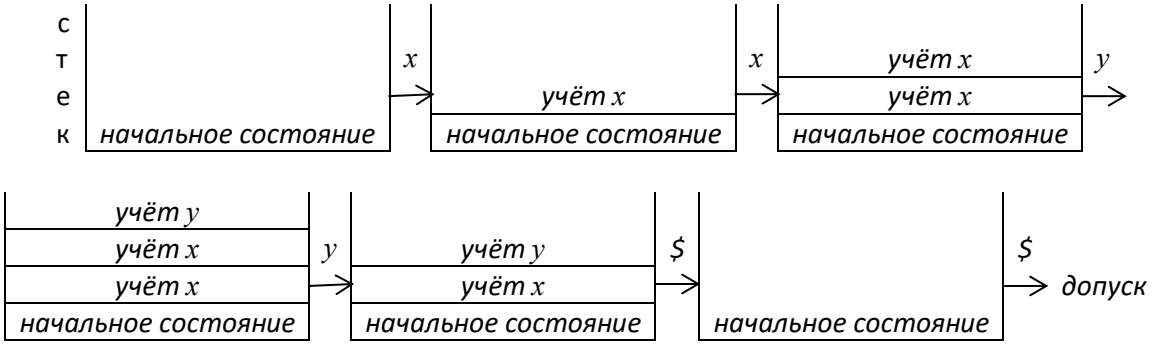
оборот: для каждого x найдется парный ему y, то по окончании сканирования исходного текста в стеке должна остаться одна пара состояний «*учёт* y»-«*учёт* x» и начальное состояние. Имеет смысл удалить из стека и эту пару записей, и, таким образом, заключительной конфигурацией автомата, соответствующей успешному окончанию распознавания исходного текста, можно считать следующую: текст полностью прочитан, в стеке — начальное состояние.

Такая схема функционирования автомата магазинного типа обеспечивается следующей управляющей таблицей (символом $ обозначен маркер правого конца строки анализируемого текста).

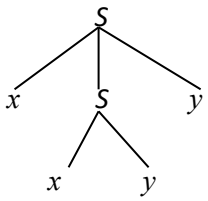
	x	y	$
начальное состояние	*учёт* x		*допуск*
учёт x	*учёт* x	*учёт* y	
учёт y	*удалить 2 записи, учёт* y		*удалить 2 записи*

Словом «*допуск*» в таблице обозначена команда успешного окончания разбора исходного текста, а пустые ячейки таблицы считаются содержащими команду «*ошибка*» — неуспешного окончания разбора.

Распознавание, например, строки $xxyy$ будет выполнено за 5 шагов:

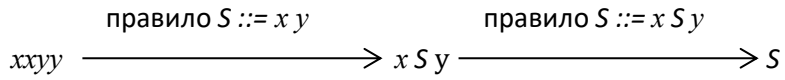


При этом неявно выстраивается в направлении снизу вверх такое дерево:



Каждая инструкция удаления из стека двух записей, можно сказать, предписывает построение очередного уровня дерева. Две записи, удаленные первыми из стека, соответствуют правой части правила $S ::= x\ y$, состоящей из двух символов. Шаг восходящего разбора заменяет правую часть правила на левую, и все последующие удаления из стека — это применения правила $S ::= x\ S\ y$, длина правой части которого равна трём. Но поскольку символ S явно в стек не помещался, на последующих шагах разбора также удаляется по две записи.

Та же самая процедура восходящего разбора строки $xxyy$ может быть изображена последовательностью получаемых сентенциальных форм:



Строка *xxyy* исходного текста, в конечном итоге, свернулась в аксиому *S*, что свидетельствует о синтаксической правильности этой строки.

Процедура построения управляющей таблицы для автомата магазинного типа LR(1) была рассмотрена нами для частного примера грамматики несколько неформально. Однако ее можно облечь в форму строгих правил (алгоритма).

Для того чтобы определить множество внутренних состояний автомата, следует выполнить разметку правил грамматики, на основе которой строится автомат. Разметка предписывает произвольно генерируемые имена внутренних состояний размещать в правых частях правил, в позициях между рядом стоящими символами, перед первым символом и после последнего символа.

Пусть, в отличие от предыдущего варианта автомата, внутренние состояния будут поименованы не длинными идентификаторами (несущими, правда, в себе некий смысл, приписанный внутренним состояниям), а просто числами натурального ряда. Тогда первое правило рассматриваемой грамматики можно разметить так:

$$S ::=_1 x\ {}_2S_3\ y\ {}_4$$

Состояние, имя которого находится в начальной позиции правила для аксиомы, в дальнейшем будет считаться начальным. В нашем случае имя начального состояния — 1. Для детерминированного автомата, естественно, начальные позиции всех правил для аксиомы должны быть отмечены именем начального состояния. Поэтому 1 попадёт и в начальную позицию второго правила рассматриваемой грамматики: $S ::= {}_1\ x\ y$.

Нетрудно догадаться, что по бокам символа в размеченном правиле находятся имена внутренних состояний, между которыми возможен переход. При этом сам символ должен рассматриваться как входное воздействие, из-за которого выполняется переход, слева от символа — имя исходного состояния, справа — имя состояния перехода. Так из разметки первого правила следует, что из состояния 1 под воздействием символа x автомат переходит в состояние 2. И поскольку детерминированный автомат из состояния 1 под воздействием символа x не имеет права переходить ни в какое другое состояние, кроме уже обнаруженного состояния 2, во втором правиле, где уже слева от x появилось 1, справа от x следует поместить 2.

Обобщая эту ситуацию, можно сказать, что если в нескольких правилах слева от одного и того же символа оказалось имя одного и того же состояния, то и справа от этого символа следует поместить одинаковые имена (метки) состояний. Таким образом, второе правило рассматриваемой грамматики теперь будет размечено так: $S ::=_1 x\ {}_2\ y\ {}_5$.

Конфигурации ${}_2S_3$, получившаяся при разметке первого правила, предполагает имитацию чтения символа *S* из исходного текста (в исходном тексте не может быть нетерминальных символов), в результате которой осуществляется переход из состояния 2 в состояние 3. Необходимость такой манипуляции вытекает из необходимости провести автомат по всей правой части правила слева направо (в данном случае — от состояния 1 до состояния 4 в первом правиле, или от состояния 1 до состояния 5 во втором правиле) и прийти к заключению, что правая часть правила в сентенциальной форме может быть заменена на левую, то есть может быть выполнен очередной шаг восходящего разбора. Эта замена сопровождается удалением из стека истории прохождения автомата по данному правилу и последующей имитацией чтения нетерминала левой части правила из исходного текста. Удаление из стека фактически возвращает автомат в состояние, из которого он начинал проход по этому правилу, поскольку имя именно этого состояния после удаления оказывается в вершине стека. Следовательно, при разметке необходимо позаботиться о том, чтобы

имя состояния, стоящего перед нетерминалом, распространилось на начальные позиции всех правил для этого нетерминала (тех правил, где он является левой частью). В рассматриваемых правилах это относится только к имени 2, стоящему перед S.

Окончательно разметка грамматики получается такой:

$$S ::=_{1,2} x\ {}_2S_3\ y\ {}_4\ |_{1,2}\ x\ {}_2\ y_5.$$

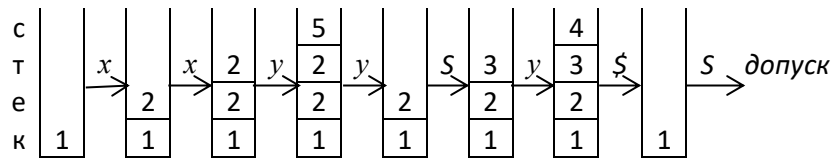
В управляющую таблицу заносятся команды двух типов (не считая команды завершения разбора): перехода (сдвига) и свёртки (приведения). О местах размещения в таблице команд перехода уже было сказано достаточно. Команды свёртки (замены правой части правила на левую) изобразим для начала правилами, по которым выполняется свёртка, хотя для исполнения такой команды нужно знать только длину правой части правила и нетерминал левой части. Длина правой части правила определяет количество записей, которые следуют удалить из стека, нетерминал левой части — имитируемое впоследствии входное воздействие. Место в таблице для команды свертки определяется следующими соображениями. Свёртка должна выполняться тогда, когда автомат оказался в состоянии, соответствующем последней (крайней справа) позиции правила. В рассматриваемом примере это — состояния 4 и 5. При этом допустимыми входными воздействиями являются те символы, которые в какой-либо сентенциальной форме могут оказаться непосредственно справа от нетерминала, к которому сворачивается правая часть правила. (Такие символы называются символами-следователями). Справа от нетерминала *S*, единственного нетерминала в рассматриваемом примере, может находиться символ y (это явно видно в первом правиле грамматики). Кроме того, для аксиомы всегда символом-следователем, в числе возможных прочих, считается маркер правого конца строки. Поскольку маркер присутствует в первоначальной для восходящего разбора сентенциальной форме (на конце строки исходного текста), очевидно, что он сохранится и в заключительной сентенциальной форме (после аксиомы, в которую свернётся строка исходного текста). С учётом этого замечания схему последовательности сентенциальных форм, приведённую выше, можно дополнить маркером *$*:

$$xxyy\$ \xrightarrow{\text{правило } S ::= x\ y} xSy\$ \xrightarrow{\text{правило } S ::= x\ S\ y} S\$$$

Таким образом, выполненной разметке грамматики соответствует следующая управляющая таблица:

	S	*x*	*y*	$
1	*допуск*	2		
2	3	2	5	
3			4	
4			*S ::= x S y*	*S ::= x S y*
5			*S ::=x y*	*S ::=x y*

Её размер оказался несколько большим размера первоначальной таблицы для такой грамматики из-за учёта дополнительных действий имитации чтения нетерминального символа. Распознавание всё той же строки $xxyy\$$ теперь выполняется за 8 шагов:



В этой схеме ещё раз обратим внимание на выполнение команд свёртки. В момент чтения второго символа y автомат находится в состоянии 5. По координатам (5,y) из таблицы выбирается команда $S ::= x\ y$ с длиной правой части правила, равной 2. Поэтому, исполняя эту команду, из стека следует удалить 2 записи и перейти к имитации чтения символа *S*. Аналогично выполняется команда $S ::= x\ S\ y$, выбранная по координатам (4, $). Но в этом случае из стека следует удалить 3 записи, поскольку длина правой части применяемого правила равна 3.

2.2.2. LL(1)-разбор

LL(1)-разбор относится к классу нисходящих алгоритмов разбора. Процесс начинается с аксиомы и в результате последовательных подстановок в сентенциальные формы вместо нетерминальных символов их правых частей правил (определений этих нетерминальных символов) завершается получением предложения языка. Можно сказать, нисходящий разбор — это вывод заданного предложения.

Для рассматриваемого языка строк из x и y предложение $xxyy$ может быть выведено за 2 шага:

$$S => x\ S\ y => xxyy$$

На каждом шаге нисходящего разбора приходится решать 2 вопроса: какой нетерминал в текущей сентенциальной форме подлежит замене на правую часть правила, и правую часть какого правила следует выбрать для такой замены, если для нетерминала в грамматике есть несколько правил. LL(1)-разбор на первый из этих вопросов отвечает так: замене подлежит самый левый нетерминал текущей сентенциальной формы. Ответ на второй вопрос определяется единственным сканируемым символом исходного текста (в случае LL(k)-разбора — k сканируемыми символами исходного текста): из нескольких правых частей правил выбирается та, для которой сканируемый символ является терминальным префиксом всех выводимых из нее строк. Для того, чтобы такой выбор был однозначным, необходимо, чтобы эти альтернативные правые части правил имели различные терминальные префиксы выводимых из них строк.

В рассматриваемом примере это требование, предъявляемое к грамматике, не выполняется: строки, выводимые из $x\ S\ y$, и строки, выводимые из $x\ y$, имеют один и тот же терминальный префикс — символ x. Эта грамматика не может быть использована алгоритмом LL(1)- разбора. Другими словами, она не обладает свойством LL(1).

Однако существует другая грамматика, описывающая тот же язык и обладающая свойством LL(1).Чтобы получить её из исходной грамматики, следует два конфликтующих правила для нетерминала *S* объединить в одно: *S* ::= xZ. При этом новый нетерминал *Z* определяется двумя правилами, полученными из «остатков» правил для *S* после вынесения из них префикса x:
$Z ::= S\ y | y$. Альтернативные правые части правила для *Z* имеют различные терминальные префиксы выводимых из них строк. Префиксом строк, выводимых из $S\ y$, является x. А символ y сам является префиксом и строкой, выводимой из второго правила для *Z*.

Ответы на упомянутые вопросы нисходящего разбора можно собрать в управляющей таблице LL(1)-разбора, столбцы который озаглавливаются терминальными символами грамматики. В стеке, вершина которого, как и ранее, определяет одну из координат ячейки таблицы, в случае LL(1)-разбора выполняется подстановка правой части правила вместо левой. Поэтому строки управляющей таблицы озаглавливаются нетерминальными и терминальными символами грамматики, то есть всеми символами, которые могут находиться в правых частях правила. Для того, чтобы в процессе разбора правая часть правила просматривалась слева направо, и подстановка выполнялась вместо самого левого нетерминала текущей сентенциальной формы, в стек правая

часть правила помещается посимвольно в обратном направлении, то есть таким образом, чтобы ее первый символ оказался в вершине стека.

Управляющая таблица для LL(1)-разбора, построенная на базе LL(1)-грамматики языка строк из x и y, выглядит так:

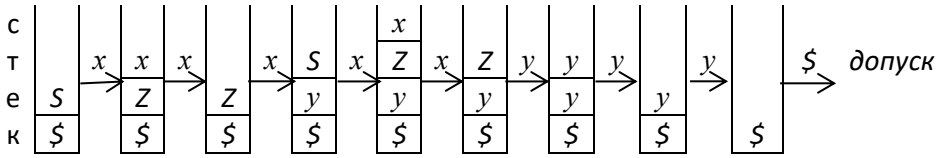
	x	y	$
S	$S ::= xZ$		
Z	$Z ::= S\,y$	$Z ::= y$	
x	*выброс*		
y		*выброс*	
$			*допуск*

Выполнение команд, представленных правилами, заключается в вышеупомянутой подстановке правых частей этих правил вместо левых частей. Команда «*выброс*» выполняется в ситуации совпадения символа в вершине стека с символом, читаемым в исходном тексте. Её выполнение заключается в удалении символа из вершины стека и перемещении по исходному тексту для чтения следующего справа символа.

Команды «*допуск*» и «*ошибка*»(как и ранее, считаем, что пустые ячейки таблицы содержат команду «*ошибка*»), прекращают разбор с, соответственно, успешным или неуспешным результатом.

В начальной конфигурации алгоритма на дне стека находятся 2 записи: маркер дна и аксиома.

Разбор строки $xxyy\$$ выполняется за 9 шагов:



2.3. Синтаксически управляемая трансляция на основе атрибутных грамматик

Синтаксически управляемая трансляция может быть сконструирована на основе атрибутной грамматики [1]. Атрибутной грамматикой называется КС-грамматика, символам которой (терминальным и нетерминальным) приписаны атрибуты. В таких атрибутах, в частности, может формироваться текст, являющийся результатом трансляции (перевода) исходного текста. Генерация выходного текста, таким образом, будет выполняться параллельно с синтаксическим анализом (и под управлением синтаксического анализа) исходного текста.

В зависимости от направления последовательности вычисления атрибутов, приписанных символам правила КС-грамматики, различают синтезируемые и наследуемые атрибуты. Синтезируемым является атрибут, приписанный символу левой части правила и вычисляемый по известным атрибутам символов правой части правила КС-грамматики. Наследуемым является атрибут, приписанный какому-либо символу правой части правила и вычисляемый по известным атрибутам символов, стоящих в правиле левее него, в том числе – символа левой части правила КС-грамматики. Так, например, для правила грамматики

$A ::= B\ C\ D$

символу A можно приписать синтезируемый атрибут A_s , который будет вычисляться как функция атрибутов символов B, C и D: $A_s = f(B_b, C_c, D_d)$. Индекс s будем использовать в обозначениях синтезируемых (synthesized) атрибутов, а в обозначениях наследуемых (inherited) – индекс i. Для атрибутов символов B, C и D выбраны произвольные обозначения, так как при записи функции для

A_s не делалось никаких предположений о способах их вычисления. В соответствии с этим же правилом для *B*, *C* и *D* могут быть определены только наследуемые атрибуты: $B_i = g(A_a)$, $C_i = h(A_a, B_b)$, $D_i = q(A_a, B_b, C_c)$.

Обобщение КС-грамматики, в котором с каждым грамматическим правилом связано множество атрибутов, называется синтаксически управляемым определением.

Очевидно, что вычислением синтезируемых атрибутов удобно сопровождать восходящий алгоритм синтаксического разбора, а вычислением наследуемых – нисходящий. Но возможно и вычисление атрибутов обоих типов при выполнении любого из двух типов алгоритмов синтаксического разбора.

Практическую реализацию вычисления атрибутов рассмотрим на примере языка строк круглых скобок, расставленных в соответствии с правилами их расстановки в арифметических выражениях: количества открывающих и закрывающих скобок должны быть равны, а при просмотре строки слева направо количество закрывающих скобок не должно превышать количества открывающих. Пусть в процессе синтаксического анализа строки скобок требуется решить задачу определения длины префикса этой строки, состоящего только из открывающих скобок. Или, в терминологии обсуждаемой темы, требуется организовать подсчёт длины префикса строки скобок, состоящего только из открывающих скобок, под управлением синтаксического анализа этой строки. Формально эта длина и должна рассматриваться как результат трансляции строки скобок, сколь бы странным это ни казалось. Так, для строки *((( ) ( ))) ( )* результатом трансляции должно быть число 3, а для строки *(( ))) (* трансляция не может быть выполнена, поскольку эта строка синтаксически неверна.

Наиболее удобочитаемым вариантом грамматики для такого языка можно считать следующий: *A* ::= *( )* | *(A)* | *A A* .

Определим функции вычисления синтезируемого атрибута для левой части правила, имея в виду, что значением этого атрибута должно быть натуральное число, равное длине найденного к моменту применения соответствующего правила префикса из открывающих скобок.

A ::= *( )* $\{ A_s = 1 \}$

A ::= *(A)* $\{ A_s = 1 + A_a \}$

A ::= *A A* $\{ A_s = A_{a1} \}$, где A_{a1} – атрибут первого символа *A* из двух, стоящих в правой части правила. Атрибут левой части правила в этом случае равен атрибуту первого символа *A*, поскольку любая подстановка вместо этого символа приведёт к появлению хотя бы одной закрывающей скобки перед вторым символом *A*, и атрибут второго символа *A*, таким образом, не сможет повлиять на длину префикса из открывающих скобок.

Однако такая грамматика не годится для построения синтаксических анализаторов детерминированного типа, и по этой причине далее будем пользоваться другим описанием синтаксиса того же языка скобок:

A ::= *T Z*

Z ::= *T Z*

Z ::= *Λ*

T ::= *(A)*

T ::= *( )*

Аксиомой здесь является символ *A*, а символ *Λ* является условным обозначением пустой строки.

Вычисления синтезируемых атрибутов в этом варианте грамматики аналогичны предыдущим:

A ::= *T Z* $\{ A_s = T_t \}$

$Z ::= T\ Z$ $\{ Z_s = T_t \}$

$Z ::= \Lambda$ $\{ Z_s = 0 \}$

$T ::= (\ A\ )$ $\{ T_s = 1 + A_a \}$

$T ::= (\ )$ $\{ T_s = 1 \}$

Для вычислений атрибутов в первых двух правилах справедливо то же замечание, что и для вычисления в последнем правиле первого варианта грамматики: синтезируемый атрибут левой части правила не зависит от атрибута символа *Z* и равен атрибуту символа *T*, поскольку любая подстановка вместо символа *T* помещает, как минимум, одну закрывающую скобку перед символом *Z*. Строго говоря, вычисление синтезируемого атрибута для символа *Z* в рассматриваемой задаче не имеет смысла; он не влияет на значение атрибута аксиомы, что нетрудно проследить по приведённым формулам вычислений атрибутов.

2.3.1. Синтаксически управляемая трансляция как вычисление синтезируемых атрибутов

Вышеупомянутая строка *((( ) ( ))) ( )* может быть разобрана восходящим алгоритмом, использующим эту грамматику, с одновременным вычислением префикса из открывающих скобок. Результаты выполнения последовательных шагов такого алгоритма приведены в таблице 1.

Таблица 1

Текущая сентенциальная форма	Применяемое правило грамматики	Атрибуты символов правой части правила	Синтезируемый атрибут левой части правила
((( ) ( ))) ( )	*T ::= ( )*	- -	1
((T ( ))) ( )	*T ::= ( )*	- -	1
((T T)) ( )	*Z ::= Λ*	-	0
((T T Z)) ( )	*Z ::= T Z*	1 0	1
((T Z)) ( )	*A ::= T Z*	1 1	1
((A)) ( )	*T ::= (A)*	- 1 -	2
(T) ( )	*Z ::= Λ*	-	0
(T Z) ( )	*A ::= T Z*	2 0	2
(A) ( )	*T ::= (A)*	- 2 -	3
T ( )	*T ::= ( )*	- -	1
T T	*Z ::= Λ*	-	0
T T Z	*Z ::= T Z*	1 0	1
T Z	*A ::= T Z*	3 1	3
A			

На последнем шаге алгоритма в качестве сентенциальной формы получена аксиома, а приписанный ей атрибут оказался равным 3. Интерпретацию этих результатов можно сформулировать так: строка синтаксически верна, а длина её префикса, состоящего только из открывающих скобок, равна 3.

Для строки *(( )) (,* приведённой в качестве примера синтаксической ошибки, восходящий алгоритм, прежде чем сделать заключение о том, что она неверна, выполнит несколько шагов разбора и, соответственно, вычислений атрибутов. Этот процесс иллюстрируется таблицей 2.

Таблица 2

Текущая сентенциальная форма	Применяемое правило грамматики	Атрибуты символов правой части правила	Синтезируемый атрибут левой части правила
(( )) (	*T ::= ( )*	- -	1

(T)) (	*Z* ::= *Λ*	-	0
(T Z)) (	*A* ::= *T Z*	1 0	1
(A)) (	*T* ::= *(A)*	- 1 -	2
T) (			

К полученной форме, казалось бы, ещё можно применить правило *Z* ::= *Λ*, а затем к получившейся *T Z) (* – правило *A* ::= *T Z*, в результате чего форма приобретёт вид: *A) (* , и к ней уже невозможно применить ни одно из правил грамматики (*Z* ::= *Λ* – не в счёт), поскольку никакая из её подстрок не является правой частью никакого из этих правил. Но о том, что эти два преобразования бесполезны, нетрудно догадаться по присутствию в форме *T) (* закрывающей скобки без предшествующих открывающих. Строго говоря, ни одна из форм, записанных в столбце «Текущая сентенциальная форма» таблицы 2, не является сентенциальной, поскольку в конечном итоге не сворачивается в аксиому. Но алгоритму восходящего разбора об этом становится известно, как минимум, только на 5-м шаге работы, который и становится заключительным с выводом об ошибочности исходной строки.

В рассмотренных примерах разбора строк не приведены основания выбора того или иного правила для свёртки (приведения) той или иной части текущей сентенциальной формы. Хотя для таких простых примеров этот выбор и является достаточно очевидным, он может быть строго обоснован, поскольку, как уже было отмечено, на основе рассматриваемой грамматики может быть построен синтаксический анализатор детерминированного типа. В частности, эта грамматика обладает свойством LR(1). Управляющая таблица для LR(1)-анализатора [2] может быть, например, такой.

	A	T	Z	(	)	$
1	*допуск*	2		6		
2		4	3	6	*Z* ::= *Λ*	*Z* ::= *Λ*
3					*A* ::= *T Z*	*A* ::= *T Z*
4		4	5		*Z* ::= *Λ*	*Z* ::= *Λ*
5					*Z* ::= *T Z*	*Z* ::= *T Z*
6	7	2		6	9	
7					8	
8				*T* ::= *(A)*	*T* ::= *(A)*	*T* ::= *(A)*
9				*T* ::= *( )*	*T* ::= *( )*	*T* ::= *( )*

В таблице используются обозначения внутренних состояний автомата-анализатора числами 1 – 9. Состояние 1 считается начальным. В командах, изображённых целыми числами, указаны состояния соответствующих переходов автомата. Команды, представленные правилами грамматики, предписывают замены в сентенциальных формах правых частей правил на соответствующие левые. Пустые ячейки таблицы считаются содержащими команду «*ошибка*». Следуя указаниям такой таблицы, детерминированный автомат всегда знает, какое единственно правильное действие должно быть выполнено на каждом шаге его работы: это указание выбирается из таблицы по текущему внутреннему состоянию автомата (координата строки таблицы) и текущему сканируемому символу сентенциальной формы (координата столбца таблицы). Так, например, нетрудно видеть, что сканируя префикс исходной строки, состоящий только из открывающих скобок, автомат будет оставаться во внутреннем состоянии 6. Как только такой префикс закончится, т.е. будет обнаружена первая закрывающая скобка, следует применить правило *T* ::= *( )*, т.к. закрывающая скобка инициирует переход автомата из состояния 6 в состояние 9, а 9-я строка таблицы предписывает выполнение команды *T* ::= *( )*. В соответствии с этим правилом обнаруженная пара скобок заменяется в сентенциальной форме нетерминальным символом *T*, как это и показано в таблицах 1 и 2.

Восходящий разбор, сопровождаемый вычислением синтезируемых атрибутов, удобно реализуется с помощью утилиты Bison. Её интерфейс предлагает специальные обозначения для переменных, содержащих значения синтезируемых атрибутов: *$$* – для атрибута нетерминала левой части правила и *$n* – для атрибутов символов правой части правила (*n* – номер позиции символа в правой части правила). При этом атрибуты терминальных символов считаются синтезируемыми на этапе лексического анализа, а наличие доступа к ним через переменные *$n* обеспечивается нахождением их в *yylval*. Если тип значений атрибутов отличен от *int*, его следует указать в директиве *%union* и упомянуть имя этого типа в директиве *%token* перед перечнем символьных обозначений токенов. Например, если атрибутами должны быть строки, то соответствующие директивы могут быть такими: *%union{char *string;}* и *%token<string> Omega*. Подчеркнём, что описание *string* относится только к типу атрибута; токен, представленный символьным обозначением *Omega*, должен быть целочисленным.

Для рассматриваемого примера языка скобок анализатор, подсчитывающий длину префикса открывающих скобок, может быть описан средствами Bison следующим образом (формирования *yylval* здесь не понадобилось, поскольку терминальные символы записаны непосредственно в правилах грамматики в виде символьных констант).

```
%{
#include <iostream>
#include <string>
using namespace std;

void yyerror(char const* msg);

int yylex();
int yyparse();
%}
%%
A:T Z {$$=$1;cout<<$$<<'\n';}  // As = Tt и вывод атрибута As, связанного с аксиомой
;
Z:{$$=0;}                      // Zs = 0
|
T Z {$$=$1;}                   // Zs = Tt
;
T:'('A')' {$$=$2+1;}           // Ts = Aa + 1
|
'('')' {$$=1;}                 // Ts = 1

;
%%
void yyerror(char const* msg)
{
        cerr << msg << endl;
}
int yylex()                    // ввод символов исходного текста
{char c;
cin>>c;
```

```
return c;
}
void main()
{
        if (yyparse() != 0)
// вывод результатов синтаксического анализа
        {
                cout << "Syntax error "  << endl;
        } else
                cout << "Success" << endl;
}
```

2.3.2. Синтаксически управляемая нисходящая трансляция как вычисление наследуемых и синтезируемых атрибутов

При решении задачи синтаксического анализа нисходящим алгоритмом трансляция может быть выполнена в виде схемы вычислений наследуемых и синтезируемых атрибутов. Применение правила грамматики в процессе нисходящего разбора означает замену в текущей сентенциальной форме нетерминального символа левой части этого правила на правую часть этого правила. При этом символы правой части правила могут получить «в наследство» информацию из атрибутов символа левой части правила, если таковая на данный момент имеется. Иными словами, для символов правой части правила могут быть вычислены наследуемые атрибуты. Причём при распространении этого наследства по символам правой части правила в направлении слева направо оно «обогащается» информацией вычисляемых атрибутов. Так символ, занимающий крайнюю правую позицию, получает максимум информации для вычисления наследуемого атрибута: значения атрибутов символа левой части правила и всех символов правой части правила, расположенных левее него. По окончании всех вычислений для символов правой части правила появляется возможность пополнить информацию о нетерминальном символе левой части правила путём вычисления для него синтезируемого атрибута.

Рассмотрим эту схему на примере того же языка скобок, предварительно преобразовав грамматику к виду, удобному нисходящему разбору:

$A ::= T\ Z$ $\quad \{ T_i = A_a ; A_s = T_t \}$

$Z ::= T\ Z$

$Z ::= \Lambda$

$T ::= (\ Y$ $\quad \{ Y_i = T_t + 1 ; T_s = Y_y \}$

$Y ::= A\ )$ $\quad \{ A_i = Y_y ; Y_s = A_a \}$

$Y ::=)$ $\quad \{ Y_s = Y_i \}$

Аксиомой здесь, по-прежнему, является символ A. Обозначения A_a, T_t, Y_y относятся к атрибутам соответствующих нетерминальных символов, значения которых были вычислены по наследуемой и синтезируемой информации и известны на момент применения формулы. Впрочем, в описаниях вычислений синтезируемых атрибутов индексы в этих обозначениях могут быть заменены на индекс s, поскольку к моментам использования значений этих атрибутов полностью завершаются построения выводов из соответствующих нетерминалов и, следовательно, эти атрибуты относятся к синтезируемым. То же можно сказать и о вычислениях атрибутов, наследуемых символами от своих соседей слева в правых частях правил (такой ситуации нет в рассматриваемой

грамматике, поскольку символ *Z* в части правила *T Z* , по-прежнему, не участвует в вычислении итогового результата). В вычислениях же атрибутов, наследуемых от символов левых частей правил, индексы в вышеупомянутых обозначениях заменены индексом i, т.к. у символа левой части правила в этот момент разбора имеется только наследуемая информация, синтезируемая будет вычислена позже, когда будет обработана вся правая часть этого правила.

Следует заметить, что с точки зрения последовательности действий синтаксического разбора и вычислений атрибутов приведённая схема не вполне корректна, поскольку описания всех вычислений, относящихся к правилу грамматики, собраны в единые фигурные скобки. С учётом того, что наследуемые атрибуты вычисляются до синтаксического разбора нетерминального символа, а синтезируемые – после него, схему имеет смысл переписать в следующем виде:

A **::=** $\{ T_i = A_i \}$	***T Z***	$\{ A_s = T_s \}$
Z **::=** $\{ T_i = 0 \}$	***T Z***	
Z **::=**	***Λ***	
T **::= (** $\{ Y_i = T_i + 1 \}$	***Y***	$\{ T_s = Y_s \}$
Y **::=** $\{ A_i = Y_i \}$	***A)***	$\{Y_s = A_s\}$
Y **::=**	***)***	$\{ Y_s = Y_i \}$

Для примера разбора нисходящим алгоритмом возьмём всё ту же строку *((( ) ( ))) ( )*. Её левый (полученный заменами самых левых нетерминалов на каждом шаге) вывод выглядит так:

A *=>* ***T Z*** *=>* ***(Y Z*** *=>* ***(A) Z*** *=>* ***(T Z) Z*** *=>* ***((Y Z) Z*** *=>* ***((A) Z) Z*** *=>* ***((T Z) Z) Z*** *=>* ***(((Y Z) Z) Z*** *=>*
=> ***((( ) Z) Z) Z*** *=>* ***((( ) T Z) Z) Z*** *=>* ***((( ) (Y Z) Z) Z*** *=>* ***((( ) (Y Z) Z) Z*** *=>* ***((( ) ( ) Z) Z) Z*** *=>*
=> ***((( ) ( )) Z) Z*** *=>* ***((( ) ( ))) Z*** *=>* ***((( ) ( ))) T Z*** *=>* ***((( ) ( ))) (Y Z*** *=>* ***((( ) ( ))) ( ) Z*** *=>* ***((( ) ( ))) ( )***

Повторим этот соответствующий нисходящему разбору вывод, сопроводив его вычислением наследуемых и синтезируемых атрибутов (правила грамматики, в отличие от вычислений атрибутов, выделены жирным шрифтом):

A **::= *T Z*** $\{ A_s = 3 \}$
$\{ T_i = 0 \}$ ***T*** **::= (*Y*** $\{ T_s = 3 \}$
$\{ Y_i = 0 + 1 \}$ ***Y*** **::= *A)*** $\{ Y_s = 3 \}$
$\{ A_i = 1 \}$ ***A*** **::= *T Z*** $\{ A_s = 3 \}$
$\{ T_i = 1 \}$ ***T*** **::= (*Y*** $\{ T_s = 3 \}$
$\{ Y_i = 1 + 1 \}$ ***Y*** **::= *A)*** $\{ Y_s = 3 \}$
$\{ A_i = 2 \}$ ***A*** **::= *T Z*** $\{ A_s = 3 \}$
$\{ T_i = 2 \}$ ***T*** **::= (*Y*** $\{ T_s = 3 \}$
$\{ Y_i = 2 + 1 \}$ ***Y*** **::= *)*** $\{ Y_s = 3 \}$
Z **::= *T Z***
$\{ T_i = 0 \}$ ***T*** **::= (*Y*** $\{ T_s = 1 \}$
$\{ Y_i = 0 + 1 \}$ ***Y*** **::= *)*** $\{ Y_s = 1 \}$
Z **::= *Λ***
Z **::= *Λ***
Z **::= *T Z***
$\{ T_i = 0 \}$ ***T*** **::= (*Y*** $\{ T_s = 1 \}$
$\{ Y_i = 0 + 1 \}$ ***Y*** **::= *)*** $\{ Y_s = 1 \}$

Правила грамматики в этой схеме указаны явно и размещены так, что символ левой части правила занимает ту же вертикаль, какую он же занимает в вышестоящей строке, где он должен быть заменён на правую часть этого правила. Так в первых двух строках схемы символы ***T*** разме-

щены по одной вертикали, поскольку именно этот символ в форме ***T Z*** подлежит замене на ***(Y*** , правую часть его правила. Слева от правил в фигурных скобках выписаны значения наследуемых атрибутов. Значения синтезируемых атрибутов выписаны справа от правил и также заключены в фигурные скобки.

Перед началом нисходящего разбора с аксиомой грамматики, символом ***A***, следует связать значение атрибута, которое могло бы быть передано в качестве «наследства» при первой подстановке вместо ***A***. Очевидно, что таким значением должен быть 0, поскольку в момент первой подстановки ничего неизвестно об искомой длине префикса открывающих скобок. Во второй строке схемы этот факт указан в виде $\{ T_i = 0 \}$, что соответствует получению наследуемого атрибута, равного 0, символом ***T***,занимающим первую позицию в применяемом правиле ***A ::= T Z***. В дальнейшем применению каждого правила предшествует вычисление наследуемого атрибута для левой части этого правила в соответствии с указаниями синтаксически управляемого определения. Так, например, перед применением правила ***Y ::= A)*** (третья строка схемы) вычисляется наследуемый атрибут для ***Y***, который этот символ получил «в наследство» от символа ***T*** (вторая строка схемы). Вычисление выполняется в соответствии с указанием синтаксически управляемого определения $\{ Y_i = T_i + 1 \}$ и с учётом того, что в текущий момент $T_i = 0$. По завершении разбора правой части какого-либо правила и, следовательно, по завершении вычисления атрибутов всех символов правой части этого правила появляется возможность вычисления синтезируемого атрибута для символа левой части правила. Простейшим примером такого вычисления является последняя строка схемы с правилом ***Y ::=)***. С единственным символом правой части этого правила не связаны никакие атрибуты, а синтезируемый атрибут символа ***Y***, в соответствии с указанием синтаксически управляемого определения $\{ Y_s = Y_i \}$, совпадает с его наследуемым атрибутом, равным в текущий момент разбора единице. В схеме это указано так: $\{ Y_s = 1 \}$. Впрочем, эти вычисления, приведённые в последних двух строках схемы, никак не влияют на значение синтезируемого атрибута аксиомы, поскольку они относятся к разбору правой части правила для символа ***Z***, атрибуты которого, как это уже неоднократно отмечалось, не могут участвовать в вычислении итогового результата, а потому и не вычисляются вовсе. Итоговый результат представлен в первой строке схемы синтезируемым атрибутом аксиомы: $\{ A_s = 3 \}$. Такое его размещение объясняется тем, что он связан с левой частью первого применяемого правила, символом ***A***. Однако вычисление этого атрибута происходит на последнем шаге алгоритма разбора, после того, как полностью разобрана правая часть этого правила, ***A ::= T Z***. Вычисления синтезируемых атрибутов, можно сказать, происходят в направлении снизу вверх по схеме.

В результате преобразования грамматики языка строк скобок к виду, удобному для нисходящего разбора, она приобрела свойство LL(1) [2]. Поэтому предписания по выбору правил грамматики в процессе синтаксического разбора какой-либо строки скобок могут быть оформлены в виде управляющей таблицы LL(1)-анализатора:

	(	*)*	*$*
A	*A ::= T Z*		
T	*T ::= (Y*		
Z	*Z ::= T Z*	*Z ::= Λ*	*Z ::= Λ*
Y	*Y ::= A)*	*Y ::=)*	
(	*выброс*		
)		*выброс*	
$			*допуск*

Как и в таблице LR(1)-анализатора, пустые ячейки таблицы считаются содержащими команду «ошибка». В виде правил грамматики представлены команды, предписывающие замену нетерминального символа в текущей сентенциальной форме на соответствующую правую часть

правила, т.е. выполнение шага нисходящего разбора. Команда «выброс» фиксирует совпадение терминального символа в текущей сентенциальной форме с текущим символом анализируемой строки и предписывает «переключение внимания» анализатора на следующий символ этой строки.

Одна из программных реализаций нисходящего разбора, использующего свойство LL(1) грамматики, известна под названием: «Метод рекурсивного спуска» [2]. Структура такой программы фактически повторяет структуру грамматики, описывая каждый нетерминальный символ грамматики программной компонентой – функцией. Содержание такой функции – изображение средствами языка программирования соответствующей правой части правила грамматики, т.е. последовательность вызовов функций нетерминальных символов, находящихся в правой части правила. Терминальные символы правой части правила проверяются на совпадение с текущими сканируемыми символами анализируемой строки (своеобразная реализация исполнения команды «выброс», упомянутой в LL(1)-таблице). Разветвление на несколько правил грамматики внутри функции выполняется по одному терминальному символу, который является текущим сканируемым в анализируемой строке (это и позволяет считать метод реализацией LL(1)-разбора).

Реализация нисходящего разбора методом рекурсивного спуска для рассматриваемого примера LL(1)-грамматики может быть следующей.

```
#include <iostream>
#define AP_left_KW 56
#define AP_right_KW 57
#define AP_end_KW 58
using namespace std;
int number;
int symbol;
int A(int);
int yylex()      // ввод символов исходного текста
{char s;
 cin>>s;
 switch  (s)
 {case '(': return AP_left_KW;
 case ')': return AP_right_KW;
 case '$': return AP_end_KW;
 default : return -1;
 }
}
int Y(int inherited)
{int synthesized;
 switch (symbol)
  {case AP_left_KW:{if(( synthesized=A(inherited))==-1) //Y ::= { Ai = Yi } A )
                              return -1;
                            else if(symbol==AP_right_KW)
                                 return synthesized; //{Ys = As}
                                      else return -1;}
   case AP_right_KW: {symbol=yylex();//Y ::= )
                      return inherited;}//{ Ys = Yi }
  default: return -1;
 }
}
int T(int inherited)
{symbol=yylex();
 return Y(inherited+1);//T ::= (      { Yi = Ti + 1 }  Y     { Ts = Ys }
}
```

```
int Z()
{switch (symbol)
 {case AP_left_KW: T(0); return Z();//Z::={Ti=0}  TZ
  case AP_right_KW: return 0;          //Z::=Λ
  case AP_end_KW: return 0;            //Z::=Λ
  default: return -1;
 }
}
int A(int inherited)
{if(symbol==AP_left_KW)
{int synthesized=T(inherited);         //A ::= { Ti = Ai }     T Z
      Z();
      return synthesized;              //{ As = Ts  }
}
else return -1;
}
void main()
{
symbol=yylex();
      int y=A(0);                      //A - аксиома
// анализ результата трансляции:
if(y<0)cout<<"error\n";
else cout<<"prefix length "<<y<<'\n';
}
```

Терминальные символы грамматики в программе обозначены идентификаторами вида *AP_X_KW* в предположении, что значения этих идентификаторов формируются и возвращаются функцией `int yylex()` лексического анализа. Поле *X* заполнено в идентификаторах терминалов словами `left`, `right` или `end` для, соответственно, открывающей скобки, закрывающей скобки или маркера конца строки. Для работы функций синтаксического анализа текущий терминальный символ помещается в переменную `symbol`.

Наследуемый атрибут нетерминала передаётся внутрь соответствующей функции через её параметр, а синтезируемый атрибут возвращается функцией оператором `return`. При обнаружении синтаксической ошибки функции возвращают -1.

Так, например, функция `int T(int inherited)` сконструирована в соответствии с правилом ***T* ::= (** $\{ Y_i = T_i + 1 \}$ ***Y*** $\{ T_s = Y_s \}$. Терминальный префикс **'('** правой части правила проверяется функциями `A` и `Z` непосредственно перед вызовом функции `T`. Поэтому дополнительную проверку наличия этого символа в анализируемой строке функция `T` может не выполнять, а может сразу же выполнить перемещение по этой строке к следующему справа символу. Такое перемещение выполняется посредством вызова функции `yylex`. Следующий символ ***Y*** правой части правила предписывает выполнение вызова функции `int Y(int inherited)`. При этом правило $\{ Y_i = T_i + 1 \}$ вычисления наследуемого атрибута указывает на то, что аргумент функции `Y` должен быть на 1 больше аргумента функции `T`, а правило $\{ T_s = Y_s \}$ вычисления синтезируемого атрибута – на то, что значение, возвращаемое функцией `T`, должно совпадать со значением, возвращаемым функцией `Y`. Таким образом, этот фрагмент правой части правила атрибутной грамматики в тексте программы может быть описан единственным оператором:

`return Y(inherited+1)`. Явного возвращения -1 в функции `T` не предусмотрено, поскольку, как уже было сказано, внутри нее не анализируется текущий терминальный символ. Тем не менее, функция `T` может возвратить -1 посредством функции `Y`.

3. Задания на лабораторный практикум

Лабораторная работа №1

Восходящий синтаксический анализ

С помощью утилиты Bison разработайте синтаксический LR-анализатор для языка, заданного грамматикой. Грамматику выберите по номеру варианта.

Лабораторная работа №2

Восходящая трансляция на основе вычисления синтезируемых атрибутов

Дополните программу, разработанную в лабораторной работе №1, вычислением синтезируемых атрибутов для программной реализации синтаксически управляемого решения задачи, выбранной по номеру варианта.

Лабораторная работа №3

Нисходящая трансляция на основе вычисления наследуемых и синтезируемых атрибутов

Выполните программную реализацию решения задачи, выбранной по номеру варианта, в виде синтаксически управляемой нисходящей трансляции с использованием наследуемых и синтезируемых атрибутов.

Варианты задания

№ варианта	Грамматика	Постановка задачи на трансляцию
1	S::=L.L S::=L L::=LB L::=B B::=0 B::=1 S – аксиома	Синтаксически управляемое вычисление десятичного значения двоичного числа
2	S::=E E::=E+T E::=T T::=T*F	Синтаксически управляемое вычисление арифметического выражения

	T::=F F::=2 F::=3 F::=4 S – аксиома	
3	S::=E E::=E+T E::=T T::=T*F T::=F F::=a F::=b F::=(E) S – аксиома	Синтаксически управляемое устранение лишних скобок в алгебраическом выражении
4	S::=E E::=E+T E::=T T::=T*F T::=F F::=x F::=5 F::=6 F::=(E) S – аксиома	Синтаксически управляемое дифференцирование алгебраического выражения
5	S::=E E::=E+T E::=T T::=T*F T::=F F::=c F::=d F::=e F::=(E) S – аксиома	Синтаксически управляемый перевод алгебраического выражения из инфиксной формы в постфиксную
6	S::=E E::=EE+ E::=EE* E::=2 E::=3 E::=4 S – аксиома	Синтаксически управляемое вычисление арифметического выражения, представленного в постфиксной форме
7	S::=E E::=E+T E::=T T::=F/F F::=5 F::=6 F::=7 F::=8 S – аксиома	Синтаксически управляемое сложение обыкновенных дробей
8	S::=E E::=E+T E::=T	Синтаксически управляемое преобразование алгебраического выражения вынесением общего множителя за скобки

	T::=T*F T::=F F::=k F::=n F::=m S – аксиома	
9	S::=E E::=E+T E::=T T::=K*x^K T::=K T::= K*x T::= x^K K::=2 K::=3 K::=4 S – аксиома	Синтаксически управляемое преобразование алгебраического выражения приведением подобных членов
10	S::=E E::=T+T T::=L.L L::=B L::=LB B::=0 B::=1 B::=2 B::=3 B::=4 B::=5 B::=6 B::=7 B::=8 B::=9 S – аксиома	Синтаксически управляемое символьное сложение вещественных чисел

График выполнения лабораторных работ

Лабораторная работа	Рейтинг	№ недели отчетности
Восходящий синтаксический анализ	10	5
Восходящая трансляция на основе вычисления синтезируемых атрибутов	10	10
Нисходящая трансляция на основе вычисления наследуемых и синтезируемых атрибутов	30	16

Библиографический список

1. Ахо А., Сети Р., Ульман Дж. Компиляторы: принципы, технологии, инструменты: пер. с англ. М.: Издательский дом «Вильямс», 2003, 768с.
2. Хантер Р. Проектирование и конструирование компиляторов: пер. с англ. М.: Финансы и статистика, 1984, 232с.
3. Мартыненко Б.К. Языки и трансляции: учебное пособие: 2-е изд. испр. и доп. – СПб.: Изд-во С.-Петербургского ун-та, 2013. 268с.
4. Бржезовский А.В., Максимова Т.М., Янкелевич А.А. Теория языков программирования и методы трансляции. Средства автоматизации построения синтаксических анализаторов. Методические указания к выполнению лабораторных работ № 1-2. – СПб: СПбГУАП, 2006, 39с.

Содержание