БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра многопроцессорных систем и сетей

В. В. Рябый

МЕТОДЫ ПРОЕКТИРОВАНИЯ ЛЕКСИЧЕСКИХ И СИНТАКСИЧЕСКИХ АНАЛИЗАТОРОВ

Учебные материалы для студентов факультета прикладной математики и информатики

МИНСК
2015

УДК 004.42(075.8)
ББК 32.973.2-018.2-02-04я73-1
Р98

Утверждено на заседании
кафедры многопроцессорных систем и сетей
26 ноября 2015 г., протокол № 4

Рябый, В. В.
Р98 Методы проектирования лексических и синтаксических анализаторов: учеб. материалы для студентов факультета прикладной математики и информатики / В. В. Рябый. – Минск: БГУ, 2015. – 62 с.

Рассматриваются основные принципы и эффективные методы проектирования лексических и синтаксических анализаторов – главных компонент компиляторов и интерпретаторов фазы анализа.

Предназначено для студентов факультета прикладной математики и информатики.

УДК 004.42(075.8)
ББК 32.973.2-018.2-02-04я73-1

ВВЕДЕНИЕ

В учебных материалах формулируются основные проблемы разработки трансляторов. Рассматриваются принципы и эффективные методы проектирования лексических и синтаксических анализаторов – главных компонент компиляторов и интерпретаторов фазы анализа. Для решения задач лексического и синтаксического анализа процесса трансляции языков программирования приводится необходимый математический аппарат.

Содержащийся материал представляет собой описание технологии разработки алгоритмов поведения автоматов – моделей языковых процессоров, решающих задачи лексического и синтаксического анализа, отражает тематику важнейших разделов дисциплины «Методы трансляции» и адресован студентам специальности «Информатика» и «Прикладная информатика».

1. ТИПЫ ЯЗЫКОВЫХ ПРОЦЕССОРОВ

Языки для общения с вычислительными машинами различаются по сложности. Все они являются искусственными языками, допускающими употреблять строго определенное множество слов, предложений или формул, которые машина может понять. Пользователю для общения с машиной требуется руководство, объясняющее допустимые в языке конструкции и их значения. К степени формализации, описанию языка для разработчика программного обеспечения, с помощью которого машина может воспринимать команды или программы, как предложения искусственного языка, переводить их во внутренние структуры данных для исполнения предъявляются самые строгие требования.

Программу для вычислительной машины, понимающую инструкции и предложения входного языка, используемого прикладным программистом, называют общим термином языковым процессором.

Существует два типа таких программ для обработки языков: интерпретаторы и трансляторы.

Интерпретаторы допускают в качестве входа «инструкция за инструкцией» исходную программу, составленную на языке, называемым исходным языком, и по мере распознавания отдельной инструкции производят вычисления, предписываемые данной инструкцией в контексте этой программы.

Трансляторы допускают в качестве входа программу на исходном языке, а в качестве выхода производят функционально эквивалентную

версию этой программы, написанную на другом языке, который называется объектным (или целевым) языком.

Объектный язык обычно является машинным языком реальной вычислительной машины, причем в этом случае объектную программу можно загрузить и выполнить.

Трансляторы весьма условно делятся на ассемблеры (или автокоды) и компиляторы, которые транслируют соответственно языки низкого уровня (машинно-ориентированные) и высокого уровня, которые не зависят от операционной системы конкретной машины и ее набора команд.

В основе всех процессов обработки языков лежит теория автоматов и формальных языков. Основные проблемы построения компиляторов и технологии их решения, основанные на соответствующих разделах теории, применимы при проектировании любых типов языковых процессоров.

1.1. Процесс компиляции, модель и фазы компилятора

Процесс компиляции логически представим как взаимодействие нескольких процессов, задачи которых легче описать, основываясь на формальных определениях входного и целевого языков.

Компиляция (трансляция или перевод) состоит из двух главных фаз: анализа и синтеза. Под анализом понимается разбиение исходной программы на составные части и создание ее промежуточного представления. Под синтезом понимается конструирование из промежуточного представления целевой программы.

По типу решаемых задач фаза анализа логически представляется как последовательная композиция фаз лексического анализа, синтаксического анализа и семантического анализа. Фаза синтеза логически представляется как композиция нескольких процессов: фазы генерации промежуточного кода (промежуточного представления программы), опциональной фазы оптимизации кода и фазы генерации целевого кода.

Модель компилятора как набор компонент может соответствовать разбиению на фазы. Формально выход текущей фазы является входом очередной. Если очередная фаза не исполняется, пока не сформируется полностью выход текущей, тогда говорят, что две фазы реализуются в двух различных проходах. Концептуально каждой фазе соответствует специальный языковый процессор, который обрабатывает ее вход и переводит в специальную форму программы на выходе. В зависимости от сложности входного языка в реальных компиляторах некоторые фазы допускают объединение в одной компоненте и исполнение в одном проходе. Конструкция однопроходных компиляторов накладывает на язык

требование предшествования объявлений именованных объектов их использованию. Наличие доступных ресурсов (памяти) или простота дизайна может оказаться решающим фактором при выборе числа проходов в процессе компиляции.

В однопроходных компиляторах ведущей фазой является фаза синтаксического анализа. Синтаксис входного языка является определяющим как при формализации схем перевода на целевой язык, так и в дизайне компиляторов. По этой причине методы и технологии конструирования компиляторов называются синтаксически-ориентированными.

Как было выше отмечено, каждой фазе соответствует специальный языковый процессор. Методы и технологии проектирования таких процессоров различаются, основываются на формальных специальных моделях языков и автоматов различной степени общности их применимости.

Конечные автоматы: акцепторы (распознаватели) и преобразователи регулярных языков, их конструирование, лежат в основе методов и технологии разработки лексических анализаторов, называемых иногда сканерами.

Контекстно-свободные грамматики служат для описания контекстно-свободного синтаксиса исходных языков. Магазинные последовательностные автоматы: распознаватели и преобразователи, являются моделями синтаксических анализаторов.

Методы их конструирования лежат в основе технологий разработки синтаксических анализаторов, называемых иногда парсерами.

Предметом содержания учебных материалов являются математические модели и технологии, основанные на таких моделях, необходимые для конструирования эффективных лексических и синтаксических анализаторов.

Основным формализмом для описания трансляции (перевода) исходного языка в целевой язык являются атрибутные трансляционные грамматики. Такое описание трансляции требует определения операционной семантики исходного языка, посредством промежуточного языка виртуальной машины, инструкции которой адекватно выражают операции исходного языка. Контроль правильности при семантическом анализе ограничивается вычислением атрибутов и проверкой контекстных условий, выраженных через соотношения между атрибутами, связанными с грамматически правильными конструкциями.

Проблемы спецификации семантического анализа и генерации целевого кода виртуальной машины, а также основные эквивалентные опти-

мизирующие преобразования, служащие для улучшения качества целевой программы являются предметом описания фазы синтеза.

2. ЛЕКСИЧЕСКИЙ АНАЛИЗ

Вход фазы лексического анализа представляется предложением формального языка L над входным алфавитом Σ.

Пусть $L \subseteq \Sigma^*$ – произвольный непустой язык над алфавитом Σ и $R = \{L_i \mid L_i \subseteq \Sigma^*, i \in [1,r]\}$ – конечное семейство регулярных языков над алфавитом Σ. Семейство языков R называется разделенным, если $L_i \cap L_j = \varnothing$, $1 \le i \ne j \le r$.

Скажем, язык L представим над R, если для любого предложения $x \in L$ найдется последовательность слов $t_1, t_2, \ldots, t_n$ такая, что для каждого слова $t_j, j \in [1,n]$, найдется язык $L_i \in R$, $i \in [1,r]$, что $t_j \in L_i$ и $x = t_1\, t_2 \ldots t_n$.

Регулярный язык $L_i \in R$ называется существенным (по отношению к L), если найдутся предложение x языка L и его представление $x = t_1\, t_2 \ldots t_n$ такие, что слово $t_j \in L_i$ при некотором $j \in [1,n]$. В противном случае язык L_i называется несущественным.

Свойство языка L быть представимым над R сохраняется, если из R исключить несущественный язык.

Семейство языков R называется неизбыточным, если

$$L \subseteq (L_1 \cup L_2 \cup \ldots \cup L_r)^*$$

и все языки семейства R существенны по отношению к L.

Если язык L представим над разделенным семейством R, состоящим только из существенных языков, тогда R называется *регулярным базисом* языка L.

Основная задача лексического анализа заключается в нахождении корректного единственного представления любого предложения входного языка L относительно регулярного базиса R.

Языки регулярного базиса R называют *классами лексем* языка L. Представление $x = t_1\, t_2 \ldots t_r$ предложения x языка L, получаемое в результате лексического анализа, называют разбиением на лексемы. Корректность разбиения на лексемы определяется свойствами регулярного базиса R и синтаксической структурой предложений самого языка L.

Компонента или блок языкового процессора (компилятора или интерпретатора), решающий задачу лексического анализа, называют *лексическим анализатором* или *сканером*. Предложение x языка L обрабатывается сканером последовательно слева направо как входной поток символов алфавита Σ, при этом лексемы последовательно выделяются из входного потока.

Основная проблема, которую должен решать сканер, заключается в корректном определении *правой границы* лексемы. Для ее обнаружения необходима информация об *ограниченном правом контексте* лексемы определенного типа.

Существуют два типа сканеров. Говорят, что сканер действует *прямо*, если находит правый конец лексемы и определяет ее тип, начиная сканирование во входном потоке с первого (левого) символа. Сканер действует *не прямо*, если, начинает сканирование во входном потоке с первого (левого) символа и знает *тип* (класс) лексемы, выделяемой из входного потока. В любом случае для корректного определения правого конца лексемы необходима информация об *ограниченном правом контексте* лексемы определенного типа.

Проиллюстрируем поведение двух типов сканеров при обработке оператора DO Фортран-программы:

DO 10 I = 1,50

Положим, что указатель первого символа выделяемой лексемы установлен на входной символ D.

Сканер, действующий *прямо*, просмотрит все символы до запятой, включая ее самую, чтобы решить, что слово DO является лексемой типа «служебное слово» и сдвинуть указатель на два символа вправо.

Сканер, действующий *не прямо,* выделит DO и сдвинет указатель на два символа вправо, если требуется выделить лексему типа «служебное слово». Если же требуется выделить лексему типа «идентификатор», он выделит слово DO 10 I и сдвинет указатель на семь символа вправо.
Причина такого поведения сканера обусловлена тем, что в Фортране входной символ «пробел» интерпретируется как «пустое слово».

В современных языках программирования, например в C++, символ «пробел» является значащим разделителем. В таких языках правая граница лексемы t во входном потоке определяется однозначно входным символом a таким, что слово ta не является лексемой языка L.

2.1. Регулярные выражения и регулярные множества

Пусть Σ – не пустой конечный алфавит. Определим метаязык регулярных выражений над алфавитом Σ, а также их интерпретацию как функцию L, сопоставляющую каждому регулярному выражению e формальный язык $L(e)$ над алфавитом Σ: $L(e) \subseteq \Sigma^*$. Здесь Σ^* обозначает множество слов конечной длины, составленных из символов алфавита Σ.

Регулярные выражения являются алгебраическим языком – средством определения (спецификации) классов лексем.

Произвольное *регулярное выражение* e и его значение $L(e)$ определяется индуктивно.

Базис

1) Пусть $a \in \Sigma$, тогда a является элементарным регулярным выражением и $L(a)=\{a\}$. Выражение a интерпретируется как обозначение языка, состоящего из единственного слова длины 1, составленного из символа a. Символ a имеет двоякий смысл: как метасимвол он обозначает язык, с другой стороны является символом алфавита языка. Чтобы не вводить излишних обозначений принято разные сущности обозначать одним символом.
2) Метасимвол λ является элементарным регулярным выражением и $L(\lambda)=\{\lambda\}$. Выражение λ интерпретируется как обозначение языка, состоящего из единственного слова длины 0 – пустого слова, не содержащего ни одного символа алфавита Σ.
3) Метасимвол $\varnothing$ является элементарным регулярным выражением и $L(\varnothing)=\varnothing$. Выражение $\varnothing$ интерпретируется как обозначение пустого множества слов – пустого языка.

Индуктивный шаг

Пусть e, e_1, e_2 – регулярные выражения и $L(e)$, $L(e_1)$ $L(e_2)$ – языки им соответствующие, тогда

1) $(e_1|e_2)$ является регулярным выражением и определяет язык $L((e_1 \mid e_2))= L(e_1) \cup L(e_2)$ – объединение языков $L(e_1) \cup L(e_2)$;
2) (e_1e_2) является регулярным выражением и определяет язык $L((e_1e_2))= L(e_1)L(e_2)$ – конкатенацию (сцепление) языков $L(e_1)L(e_2)$;
3) e^* – регулярное выражение определяет язык $L(e^*)= L(e)^*$ – итерацию языка $L(e)$.

Унарный оператор * называется также оператором Клини. Оператор Клини преобразует произвольный язык $L \subseteq \Sigma^*$ в язык L^* по формуле

$$L^* = L^0 \cup L^1 \cup \ldots \cup L^i \cup L^{i+1} \cup \ldots,$$

где $L^0 = \{ \lambda \}$ и $L^{i+1} = L^i L$ для всех $i \geq 0$;

4) (e) – регулярное выражение определяет язык $L((e)) = L(e)$.

Язык $L \subseteq \Sigma^*$ называется *регулярным*, если существует регулярное выражение e над алфавитом Σ такое, что $L = L(e)$.

Унарный оператор $^+$ обозначает позитивную итерацию языка

$$L \subseteq \Sigma^*: \; L^+ = L^1 \cup \ldots \cup L^i \cup L^{i+1} \cup \ldots .$$

Имеет место очевидное тождество: $e^* = (\lambda \mid e^+)$.

Семейство регулярных языков в силу определения замкнуто относительно операции объединения, конкатенации и итерации языков как

множеств слов. Символ ∅ обозначает нейтральный элемент в операции объединения языков. Символ λ обозначает нейтральный элемент в операции конкатенации языков.

Элементарные языки определяют базис алгебры регулярных языков. Любой регулярный язык выражается через базис с помощью конечной последовательности операций объединения, конкатенации и итерации, примененных к элементарным языкам.

Операция объединения является коммутативной «аддитивной» в этой алгебре. Операция конкатенации – не коммутативная «мультипликативная». Относительно каждой из этих операций справедлив ассоциативный закон.

Имеют место два дистрибутивных закона (левый и правый) операции конкатенации по отношению к операции объединения. Справедливы следующие тождества:

$(e_1 (e_2|e_3)) = ((e_1| e_2) (e_1| e_3))$;

$((e_2|e_3) e_1) = ((e_2| e_1) (e_3| e_1))$.

Учитывая приоритет операций, можно опускать круглые скобки: итерация сильнее связывает операнды, чем конкатенация, которая в свою очередь сильнее, чем объединение.

В силу выполнимости ассоциативных законов порядок выполнения однородных бинарных операций не важен, поэтому допускаются следующие формы выражений: $(e_1|e_2|\ldots|e_n)$ или $(e_1e_2\ldots e_n)$, $n \geq 2$, в которых бинарные операции обычно выполняются слева направо.

Постфиксные операторы Клини применяются слева направо. Имеют место также, часто применяемые на практике, следующие тождества:

$(\varnothing \mid e) = e$; $(e|e) = e$;

$(\varnothing e) = (e \varnothing) = \varnothing$; $\varnothing^* = \lambda$;

$(\lambda e) = (e \lambda) = e$;

$(\lambda \mid e)^* = e^*$.

2.2. Диаграммы

Диаграмма является графическим представлением недетерминированного распознавателя регулярного языка. Это ориентированный граф, у которого дуги помечаются символами алфавита регулярного языка и метасимволом λ – символом пустого слова. Допускаются кратные помеченные дуги. В диаграмме выделяются две специальные вершины: *начальная* и *заключительная* (или *финальная*).

Диаграмма *допускает* пустой язык, если не существует ни одного *маршрута*, выходящего из начальной вершины и, заканчивающегося в финальной.

Если существует маршрут из начальной в финальную, то последовательность меток дуг, составляющих маршрут, однозначно определяет слово в алфавите регулярного языка, если выбросить из этой последовательности метки λ. Для некоторого слова может существовать несколько допустимых маршрутов либо ни одного.

Множество допустимых маршрутов определяет множество слов, которое называется множеством слов (или языком), допускаемых диаграммой.

Далее будут рассматриваться только такие диаграммы, у которых полустепень захода начальной вершины и полустепень выхода финальной вершины равны нуль.

Используя структурную индукцию, сопоставим каждому регулярному выражению e диаграмму $D(e)$, которая будет по построению допускать регулярный язык $L(e)$.

Определим диаграмму $D(e) = (X, E)$, где $X=\{x_i \mid 0 \le i \le n\}$ – множество вершин; $E \subseteq X \times (\Sigma \cup \{\lambda\}) \times X$ – множество помеченных дуг; x_0 – начальная вершина, x_n – заключительная вершина.

Базис

Пусть $a \in \Sigma$, тогда диаграмма $D(a)$ имеет единственную дугу (x_0, a, x_1), помеченную символом a. Вершина x_0 – начальная, x_1 – финальная. Диаграмма $D(\lambda)$ для выражения λ имеет единственную дугу (x_0, λ, x_1), помеченную символом λ. Вершина x_0 – начальная, x_1 – финальная.

Диаграмма $D(\varnothing)$ для выражения ∅ не имеет дуг, имеет две вершины x_0 и x_1. Вершина x_0 – начальная, x_1 – финальная.

Индуктивный шаг

Пусть e, e_1, e_2 – произвольные регулярные выражения и $D(e)$, $D(e_1)$, $D(e_2)$ соответствующие им диаграммы с попарно не пересекающимися множествами вершин.

1. Для регулярного выражения $(e_1|e_2)$ диаграмма $D((e_1|e_2))$ конструируется из диаграмм $D(e_1)$ и $D(e_2)$ следующим образом:

а) объединяются множества вершин и дуг диаграмм $D(e_1)$ и $D(e_2)$;

б) начальные вершины образуют класс эквивалентности, который представляет начальную вершину результирующей диаграммы;

в) финальные вершины образуют класс эквивалентности, который представляет финальную вершину результирующей диаграммы;

г) исходные начальные и финальные вершины во всех дугах заменяются соответствующими результирующими классами эквивалентности.

2. Для регулярного выражения (e_1e_2) диаграмма $D((e_1e_2))$ конструируется из диаграмм $D(e_1)$ и $D(e_2)$ следующим образом:

а) объединяются множества вершин и дуг диаграмм $D(e_1)$ и $D(e_2)$;

б) начальная вершина диаграммы $D(e_1)$ представляет начальную вершину результирующей диаграммы;

в) финальная вершина диаграммы $D(e_2)$ представляет финальную вершину результирующей диаграммы;

г) финальная вершина диаграммы $D(e_1)$ и начальная вершина диаграммы $D(e_2)$ образуют класс эквивалентности, который представляет вершину результирующей диаграммы. Исходные финальная вершина диаграммы $D(e_1)$ и начальная вершина диаграммы $D(e_2)$ во всех дугах заменяются соответствующим классом эквивалентности.

3. Для регулярного выражения e^* диаграмма $D(e^*)$ конструируется из диаграммы $D(e)$ следующим образом:

а) финальная вершина диаграммы $D(e)$ и ее начальная вершина образуют класс эквивалентности, который представляет вершину результирующей диаграммы. Исходные финальная вершина и начальная вершины во всех дугах заменяются соответствующим классом эквивалентности;

б) добавляются новые начальная и финальная вершины и две новые дуги помеченные символом пустого слова λ. Одна дуга исходит из новой начальной вершины, а ее конец является классом эквивалентности. Другая дуга имеет началом класс эквивалентности, а конец является новой финальной вершиной.

4. Для выражения (e) диаграмма $D((e)) = D(e)$. Взятие выражения в скобки не изменяет диаграммы самого выражения.

По построению для любого регулярного выражения e соответствующая диаграмма $D(e)$ обладает тем свойством, что полустепень захода начальной вершины и полустепень выхода финальной вершины равны нуль.

2.3. Замкнутые множества и детерминированный конечный автомат (ДКА)

Пусть $D = (X, E)$ – диаграмма, X=[0-n], $0 < n$, 0 – начальное состояние, n – заключительное (финальное) состояние; $E \subseteq X \times \Sigma' \times X$, где $\Sigma' = \Sigma \cup \{\lambda\}$. Здесь и далее вместо имени вершины x_i используется ее уникальный индекс i, множество вершин представлено отрезком [0-n] целых чисел. Термин вершина заменяется термином состояние.

Подмножество состояний $V \subseteq X$ называется *замкнутым*, если для всякой дуги $(i, \lambda, j) \in E$ из того, что $i \in V$, следует, что $j \in V$.

Замыканием произвольного подмножества состояний $V \subseteq X$ называется наименьшее замкнутое подмножество состояний диаграммы D, включающее множество V. Замыкание множества V существует и единственно, обозначается $[V]$. Если V замкнуто, то $V = [V]$.

Очевиден алгоритм построения множества $[V]$, который основан на следующих свойствах:

(а) $V \subseteq [V]$;

(б) из $i \in [V]$ и $(i, \lambda, j) \in E$ следует, что $j \in [V]$;

(в) $[V] \subseteq W$, где W=[W] , $W \subseteq X$.

Определим функцию «перемещения» μ, действующую на подмножестве состояний $V \subseteq X$ при входном символе $a \in \Sigma$:

$$\mu(V, a) = \{j \mid (i, a, j) \in E, i \in V\}.$$

Определим также функцию «*перехода*» g, действующую на произвольном подмножестве состояний V при входном символе $a \in \Sigma$:

$$g(V, a) = [\mu(V, a)].$$

Через q обозначим произвольное замкнутое подмножество состояний, и будем называть *замкнутым состоянием*.

Из определения следует, что при любом входном символе a функция g переводит замкнутое состояние q в замкнутое состояние $g(q, a) = [\mu(q, a)]$.

Пусть q_0 – некоторое фиксированное замкнутое подмножество состояний диаграммы D. Определим и обозначим через Q наименьшее семейство замкнутых подмножеств состояний диаграммы D, удовлетворяющих следующим свойствам:

(г) $q_0 \in Q$;

(д) $g(q, a) \in Q$ для всех $q \in Q$ и $a \in \Sigma$.

Множество Q называется замыканием состояния q_0 относительно функции g (над алфавитом Σ), а состояние q_0 – порождающим.

Алгоритм построения множества Q основан на свойствах (г) и (д) и сводится к порождению всех замкнутых состояний достижимых из q_0.

Вообще, любое множество Q со свойством (д) называется замкнутым относительно функции g (над алфавитом Σ).

Детерминированный конечный автомат (ДКА)

Пусть $L(D) \subseteq \Sigma^*$– язык, определяемый диаграммой D.

Детерминированный конечный автомат $M = (Q, \Sigma, g, q_0, F)$, допускающий в точности язык $L(D)$ над алфавитом Σ, определяется следующим образом.

Начальное состояние $q_0 = [0]$. Здесь [0] для краткости обозначает замыкание одноэлементного множества состояний [{0}].

Множество состояний Q – замыкание начального состояния q_0 относительно функции «перехода» g, которая называется *автоматной* функцией.

Множество *заключительных* (финальных) состояний $F = \{q \mid q \in Q, n \in q\}$, где n – заключительное (финальное) состояние диаграммы D.

Автоматная функция перехода g допускает *расширение* g^*: $Q \times \Sigma^* \to Q$ – функцию, действующую на всех словах $w \in \Sigma^*$:

$g^*(q, \lambda) = q\ , q \in Q;$

$g^*(q, wa) = g(g^*(q, w), a)\ , q \in Q, a \in \Sigma, w \in \Sigma^*\ .$

Множество слов, допускаемых автоматом M, называется *автоматным языком*:

$L(M)=\{w \mid g^*(q_0, w) \in F, w \in \Sigma^*\}.$

Автомату M соответствует изоморфное графическое представление в виде диаграммы $D_M = (Q, E)$, где $Q = \{q_0, \ldots, q_{n-1}\}$, $n = |Q|$ – множество вершин (состояний), $E = \{(q_i, a, q_j) \mid g(q_i, a) = q_j;\ q_i, q_j \in Q\}$ – множество помеченных дуг. Состояние q_0 – начальное и множество заключительных состояний F – выделенные состояния диаграммы D_M.

Пусть P – множество всех маршрутов диаграммы D_M, с началом в q_0 и концами в F, тогда язык, допускаемый диаграммой, – $L(D_M) = w(P)$.

Если $p \in P$, тогда $w(p)$ – слово, составленное из символов алфавита Σ – пометок соответствующих дуг маршрута p. Если $|p|=0$ (p – пустой маршрут), тогда $w(p) = \lambda$ – пустое слово. Очевидно $L(D_M) = L(M)$ – автоматный язык.

2.4. Декартово произведение детерминированных конечных автоматов

Пусть $M_i = (Q_i, \Sigma, g_i, q_i^0, F_i)$, $i \in [1,n]$, $1<n$, – семейство n детерминированных конечных автоматов над общим алфавитом Σ.

При $n=2$ *декартовым произведением* автоматов M_1 и M_2 называется такой ДКА $M_1 \times M_2 = (Q, \Sigma, g, q_0, F)$, что 2-вектор $q_0 = (q_1^0, q_2^0)$ определяет начальное состояние; множество состояний Q суть наименьшее подмножество 2-векторов множества $Q_1 \times Q_2$, удовлетворяющее свойствам:

(а) $q_0 \in Q$;

(б) Q – замкнуто относительно функции g: $Q \times \Sigma \to Q$, причем функция g определяется следующим образом:

$$g((q_1, q_2), a) = (g_1(q_1, a), g_2(q_2, a)) \text{ , } (q_1, q_2) \in Q, a \in \Sigma.$$

Множество заключительных состояний $F = Q \cap (F_1 \times Q_2 \cup Q_1 \times F_2)$ или $F = \{(q_1, q_2) \in Q \mid q_1 \in F_1 \text{ или } q_2 \in F_2\}$.

Обозначим через $M_{1,2} = M_1 \times M_2$ – произведение автоматов M_1 и M_2. Непосредственно из определений вытекают следующие свойства:

(1) $L(M_{1,2}) = L(M_1) \cup L(M_2)$;

(2) автомат $(Q, \Sigma, g, q_0, F \cap F_1 \times F_2)$ допускает язык $L(M_1) \cap L(M_2)$;

(3) $L(M_1) \cap L(M_2) = \varnothing$, если $F \cap F_1 \times F_2 = \varnothing$;

(4) автомат (Q, Σ, g, q_0, T_1) допускает язык $L(M_1)$, где

$$T_1 = \{(q_1, q_2) \in F \mid q_1 \in F_1\};$$

(5) автомат (Q, Σ, g, q_0, T_2) допускает язык $L(M_2)$, где

$$T_2 = \{(q_1, q_2) \in F \mid q_2 \in F_2\};$$

(6) $F = T_1 \cup T_2$;

(7) автомат $(Q, \Sigma, g, q_0, T_1 \cap T_2)$ допускает язык $L(M_1) \cap L(M_2)$;

(8) $L(M_1) \cap L(M_2) = \varnothing$, если $T_1 \cap T_2 = \varnothing$;

(9) $L(M_1) = L(M_2)$, если $T_1 = T_2$;

(10) $T_1 = T_2$, если $T_1 = F$ и $T_2 = F$;

(11) $L(M_1) \subseteq L(M_2)$, если $T_1 \subseteq T_2$.

Свойство (9) позволяет определить эквивалентность автоматов M_1 и M_2.

Декартово произведение ДКА, как определено выше, подчиняется ассоциативному закону: $((M_1 \times M_2) \times M_3) = (M_1 \times (M_2 \times M_3))$ в следующем смысле. Состояния автоматов $((M_1 \times M_2) \times M_3)$, $(M_1 \times (M_2 \times M_3))$ и $(M_1 \times M_2 \times M_3)$ имеют соответственно различные представления $((q_1,q_2),q_3)$, $(q_1,(q_2,q_3))$ и

(q_1,q_2,q_3), между которыми существует очевидная биекция h, устойчивая по отношению автоматных функций. Причем начальные состояния и множества финальных состояний находятся в биективном отношении. Биекция h устанавливает изоморфизм автоматов и, следовательно, их эквивалентность.

При $n\geq 2$ выражение $M_1\times M_2\times\ldots\times M_n = (Q, \Sigma, g, q_0, F)$ означает, что состояния автомата представляются n-векторами и, соответствующим образом определяется на векторах автоматная функция g и множество заключительных состояний F.

Обозначим через $M_{1,2,\ldots,n} = M_1\times M_2\times\ldots\times M_n$ и $pr_i(q) = q_i \in Q_i$, $q = (q_1, \ldots, q_i, \ldots, q_n) \in Q$, тогда имеют место следующие утверждения:

(12) $L(M_{1,2,\ldots,n}) = L(M_1) \cup L(M_2) \cup \ldots \cup L(M_n)$;

(13) автомат (Q, Σ, g, q_0, T_i) допускает язык $L(M_i)$, где

$$T_i = \{q\in F \mid pr_i(q)\in F_i\};$$

(14) $F = T_1 \cup T_2 \cup \ldots \cup T_n$;

(15) $L(M_i) \cap L(M_j) = \varnothing$, если $T_i \cap T_j = \varnothing$;

(16) $L(M_i) = L(M_j)$, если $T_i = T_j$;

(17) $M_{1,2,\ldots,n+1} = M_{1,2,\ldots,n} \times M_{n+1}$, $n\geq 1$;

(18) Если $L(M_{1,2,\ldots,n}) \cap L(M_{n+1}) = \varnothing$ для всех $n \in [1, m]$, $m\geq 2$, тогда

$$L(M_i) \cap L(M_j) = \varnothing,\ 1\leq i\neq j \leq m.$$

Все свойства (1)-(18) легко проверяются и необходимы для проверки корректности проектирования и разработки лексических анализаторов.

2.5. Разбиение алфавита

Пусть $L \subseteq \Sigma^*$ – регулярный язык в алфавите Σ. На множестве всех слов Σ^* язык L определяет отношение эквивалентности R_L: xR_Ly , если для любых слов $u,v\in \Sigma^*$ $uxv \in L$ тогда и только тогда, когда $uyv \in L$. Отношение R_L устойчиво относительно операции конкатенации слов. Поэтому, если $x\ R_Ly$, то говорят, что слово x *конгруэнтно* слову y по модулю L или $x = y(\mathrm{mod}\ L)$. Каждому слову x однозначно соответствует множество всех слов, конгруэнтных по модулю L – класс конгруэнтности $[x]=\{y\mid x = y(\mathrm{mod}\ L)\}$.

Пусть Π_L – разбиение множества всех слов Σ^* на классы конгруэнтности по модулю L. Если L – регулярный язык, то число классов разбиения Π_L конечно.

Обозначим через $\Sigma^1(\mathrm{mod}\ L)$ – подмножество классов конгруэнтности $[a]$, $a\in\Sigma^1$. Тогда *конгруэнтность* по mod L индуцирует разбиение сим-

волов алфавита Σ на классы конгруэнтности по модулю L, которое также будем обозначать Π_Σ или $\Sigma(\mathrm{mod}\ L)$. Очевидно, следующее

Свойство 1. Символы $a, b \in \Sigma$ *конгруэнтны* по модулю L или $a = b(\mathrm{mod}\ L)$, если для любых слов $u,v \in \Sigma^*$ $uav \in L$ тогда и только тогда, когда $ubv \in L$.

Пусть B – новый алфавит, символы которого находятся в биективном отношении β с классами разбиения Π_Σ алфавита Σ. Определим отображение $\gamma: \Sigma \to B$: $\gamma(a) = b$, если $\beta(b)=[a]\cap\Sigma$, $a \in \Sigma$ и $b \in B$.

Отображение γ индуцирует суръективный гомоморфизм свободных моноидов Σ^* и B^* : $\gamma(\Sigma^*) = B^*$.

Свойство 2. Регулярный язык $L \subseteq \Sigma^*$ замкнут относительно $\gamma^{-1}\gamma$, т. е. $\gamma^{-1}\gamma(L) = L$.

Свойство 2 является следствием известного общего утверждения: для того чтобы язык $L \subseteq \Sigma^*$ был замкнут относительно $\gamma^{-1}\gamma$, где γ – гомоморфизм моноида Σ^*, необходимо и достаточно, чтобы для всех слов $x, y \in \Sigma^*$ из соотношения $\gamma(x) = \gamma(y)$ следовало $x = y(\mathrm{mod}\ L)$.

Из вышеизложенных фактов непосредственно вытекает следующая техника определения регулярного языка L над алфавитом Σ и конструирования автомата M_Σ, допускающего язык L. В случае, когда $|B|<<|\Sigma|$, она позволит существенно упростить и сократить трудоемкость проектирования и разработки:

(1) Найти разбиение Π_Σ , используя отношение конгруэнтности по модулю L на алфавите Σ;
(2) Определить фактор-алфавит $B = \Sigma(\mathrm{mod}\ L)$ и $\gamma : \Sigma \to B$;
(3) Найти регулярное выражение e_B в алфавите B такое, что
$$L = \gamma^{-1}(L\ (e_B));$$
(4) Построить диаграмму $D_B = D(e_B)$;
(5) Построить автомат M_B такой, что $L\ (D_B) = L\ (M_B)$;
(6) Преобразовать автомат M_B в автомат M_Σ , что $L = L(M_\Sigma)$. Пусть $M_B = (Q, B, g, q_0 , F)$ и $M_\Sigma = (Q', \Sigma, g', q_0 , F')$, тогда $Q = Q'$, $F = F'$ и $g'(q, a) = q'$, если $g(q, b) = q'$ для всех $a \in \gamma^{-1}(b)$, $b \in B$ и $q, q' \in Q$.

2.6. Приведение к общему алфавиту

Пусть $L_i \subseteq \Sigma^*$ – регулярные языки в алфавите Σ, Π_i – разбиения алфавита Σ, B_i – фактор-алфавиты, индуцированные соответственно языками L_i , $i \in [1, n]$. Тогда разбиение $\Pi = \Pi_1 \cap \ldots \cap \Pi_n$ – наибольшая нижняя гра-

ница разбиений Π_i , $i \in [1, n]$, позволяет определить общий алфавит B и выразить отношения между алфавитами B и B_i следующим образом.

Пусть h_i: $B \to B_i$ – суръективное отображение такое, что $h_i(b) = d$, тогда и только тогда, когда класс C_d разбиения Π_i суть объединение всех классов C_b разбиения Π таких, что $b \in h_i^{-1}(d)$. Тогда h_i: $B^* \to B_i^*$ – гомоморфизм свободного моноида B^* на свободный моноид B_i^* и язык L_i замкнут относительно h_i : $L_i = h_i\, h_i^{-1}(L_i)$.

Из вышеизложенных фактов непосредственно вытекает следующая техника определения регулярных языков L_i над алфавитом Σ и конструирования автоматов M_i, соответственно допускающих языки L_i , $i \in [1, n]$:

(1) Найти разбиения Π_i , используя отношение конгруэнтности по модулю L_i на алфавите Σ;
(2) Определить фактор-алфавит $B_i = \Sigma(\mathrm{mod}\ L_i)$ и $\gamma_i : \Sigma \to B_i$;
(3) Найти регулярное выражение e_i в алфавите B_i такое, что
$$L_i = \gamma_i^{-1}(L\ (e_i));$$
(4) Построить диаграмму $D_i = D(e_i)$;
(5) Построить автомат M_i такой, что $L\ (D_i) = L\ (M_i)$;
(6) Вычислить разбиение $\Pi = \Pi_1 \cap \ldots \cap \Pi_n$ – наибольшую нижнюю границу разбиений Π_i , $i \in [1, n]$;
(7) Определить общий алфавит B и выразить отношения между алфавитами B и B_i с помощью отображения h_i: $B \to B_i$;
(8) Преобразовать автомат M_i в алфавите B_i в автомат M_i' в алфавите B. Пусть $M_i = (Q_i, B_i, g_i, q_{0i}, F_i)$ и $M_i' = (Q_i, B, g_i', q_{0i}, F_i)$, тогда $g_i'(q, a) = q'$, если $g_i\ (q, b) = q'$ для всех $a \in h_i^{-1}\ (b)$, $b \in B$ и $q, q' \in Q$.

Замечание к пункту (7). Утверждение «$h_i(b) = d$ справедливо, тогда и только тогда, когда класс C_d разбиения Π_i является объединением всех классов C_b разбиения Π таких, что $b \in h_i^{-1}(d)$, где $b \in B$, $d \in B_i$ » можно выразить на языке регулярных выражений с помощью равенства

$$d = e_d\ ,$$

где $e_d = b_1 \mid \ldots \mid b_m$ – элементарное аддитивное выражение такое, что $h_i^{-1}(d) = \{b_1, \ldots, b_m\}$ – прообраз символа d.

2.7. Проектирование лексического анализатора

Ниже описываются действия, которые необходимо выполнить при разработке алгоритма лексического анализатора.

Пусть $\{L_i \mid L_i \subseteq \Sigma^*, i \in [1,n]\}$ – семейство $n > 1$ регулярных языков над входным алфавитом Σ, где L_i – класс лексем входного языка $L \subseteq \Sigma^*$, $i \in [1, n]$. Предполагается, что каждое предложение языка L однозначно представимо как сцепление лексем. Также предполагается, что все языки семейства непустые ($L_i \neq \varnothing$), λ-свободны ($\lambda \notin L_i$) и попарно не пересекаются ($L_i \cap L_j \neq \varnothing$, $1 \leq i \neq j \leq n$).

Необходимо:

1) по отношению эквивалентности, индуцируемому классом лексем L_i над входным алфавитом Σ, построить соответствующее разбиение Π_i алфавита Σ для всех $i \in [1, n]$. Определить абстрактный алфавит B_i , соответствующий разбиению Π_i . Символы алфавита B_i суть представители классов разбиения Π_i;

2) найти регулярное выражение e_i над алфавитом B_i такое, что $L_i = L(e_i)$ для всех $i \in [1,n]$. Здесь $L(e_i)$ – стандартная интерпретация регулярных выражений;

3) найти эквивалентные диаграммы D_i для всех e_i , $i \in [1,n]$. Здесь $D_i = D(e_i)$ – нестандартная интерпретация регулярных выражений;

4) построить по каждой диаграмме D_i детерминированный конечный автомат (ДКА) M_i , допускающий язык L_i , $i \in [1,n]$. Проверить, что каждый язык L_i λ-свободен и не пуст, $i \in [1,n]$; определить общий абстрактный алфавит B, соответствующий пересечению разбиений Π_i , $i \in [1,n]$:

$$\Pi = \Pi_1 \cap \ldots \cap \Pi_n.$$

Символы общего абстрактного алфавита B суть представители классов разбиения Π;

5) преобразовать автомат $M_i = (Q_i, B_i, g_i, q_{0i}, F_i)$ в автомат $M_i = (Q_i, B, g_i, q_{0i}, F_i)$, заменяя функцию переходов g_i следующим образом: если $g_i(q, a) = p$ для некоторых состояний q, $p \in Q_i$, $a \in B_i$ и символ a представим в общем алфавите B регулярным аддитивным выражением: $a = b_1 \mid \ldots \mid b_m$, где b_1 , …, $b_m \in B$, $m \geq 1$, – представители классов, объединение которых дает класс символа a, тогда $g_i(q, b_j) = p$ для всех $j \in [1, m]$;$\cup$

6) построить автомат $M = M_1 \times M_2 \times \ldots \times M_n$ – декартово произведение n детерминированных конечных автоматов, допускающий объединение

$L_1 \cup L_2 \cup \ldots \cup L_n \subseteq B^*$ исходных языков над алфавитом B. Пусть $M = (Q, B, g, q_0, F)$. Найти F_i, что $F = F_1 \cup F_2 \cup \ldots \cup F_n$, а соответствующий автомат (Q, B, g, q_0, F_i) допускает язык L_i, $i \in [1, n]$. Показать, что семейство $\{F_1, F_2, \ldots, F_n\}$ состоит из попарно не пересекающихся подмножеств, если исходное семейство языков состоит из n попарно не пересекающихся регулярных языков;

7) определить множество состояний $Error=\{q \mid q \in Q, g^*(q, x) \notin F$ для всех $x \in B^*\}$ и множество активных состояний $Active=Q \backslash Error$. Определить множество активных переходов

$$ActiveTransition = \{(q, a) \mid g(q, a)=p \text{ и } q, p \in Active\};$$

для всех $i \in [1,n]$ определить множество переходов

$$EndL_i = \{(q, a) \mid g(q, a)=p, q \in F_i, a \in B, p \in Error\},$$

которые сигнализируют о завершении выделения во входном потоке лексемы класса L_i ; определить множество переходов

$$ErrorL = \{(q, a) \mid g(q, a)=p, q \in Active \setminus F, a \in B, p \in Error\},$$

которые сигнализируют о лексической ошибке.

8) определить множество P «действий» (семантических процедур):

$$P = \{pActiveTransition, pEndL_1, \ldots, pEndL_n, pErrorL\},$$

соответствующих переходам пункта 7); определить функцию выхода $f : Q \times B \to P$. Если $f(q,a) = p \in P$, тогда p – действие, выполняемое лексическим анализатором в состоянии $q \in Q$ при входном символе $a \in B$.

Фактически результатом является специальный автомат с выходом – преобразователь простого типа (Q, B, P, g, f, q_0, F).

3. СИНТАКСИЧЕСКИЙ АНАЛИЗ

3.1. Контекстно-свободные грамматики и языки

Контекстно-свободные грамматики служат метаязыком – основным средством для спецификации синтаксиса языков программирования. В классификации по Хомскому они определяют класс 2 – класс контекстно-свободных языков.

Контекстно-свободная (КС) грамматика $G = (N, \Sigma, P, S)$ – это объект из четырех компонентов, где

N – множество вспомогательных символов (нетерминалов),

Σ – множество основных символов (терминалов),

P – множество правил вывода (продукций),

S – начальный символ (аксиома), $S \in N$.

Произвольное правило КС-грамматики имеет форму

$$A \to \gamma, \text{ где } A \in N \text{ и } \gamma \in (N \cup \Sigma)^*.$$

Правила вывода нумеруются от 1 до $| N |$. Форма записи p) $A \to \gamma$ или p: $A \to \gamma$ означает, что правило $A \to \gamma$ имеет номер $p \in [1, | N |]$.

Бинарное отношение $\Rightarrow$ непосредственного вывода на множестве всех слов $(N \cup \Sigma)^*$ объединенного алфавита $(N \cup \Sigma)$ определяется следующим образом: $x \Rightarrow y$, если существует правило вывода р) $A \to \gamma$, $x = x_1 A x_2$ и $y = x_1 \gamma x_2$, где $x, y \in (N \cup \Sigma)^*$.

Отношение вывода в КС-грамматике G определяется как рефлексивное и транзитивное замыкание непосредственного вывода и обозначается символом $\Rightarrow^*$.

КС-языком называется множество слов в основном алфавите выводимых из аксиомы КС-грамматики:

$$L(G) = \{x \mid S \Rightarrow^* x, x \in \Sigma^*\}.$$

По определению $x \in L(G)$, если существует последовательность слов $x_0 , x_1 , \ldots , x_{k-1} , x_k , \ldots , x_{n-1} , x_n \in (N \cup \Sigma)^*$, $n \geq 1$, что $x_{k-1} \Rightarrow x_k$, $0 < k \leq n$, $x_0 = S$ и $x_n = x$. Последнее слово последовательности $x_n = x$ называется предложением КС -языка. Все остальные слова называются сентециальными формами, не завершенными предложениями, содержащими, очевидно, хотя бы один нетерминал.

Если $x_{k-1} \Rightarrow x_k$, $0< k \leq n$, тогда по определению существует правило p_k) $A_k \to \gamma_k$ и представления $x_{k-1} = x_{k-1}' A_k x_{k-1}''$ и $x_k = x_{k-1}' \gamma_k x_{k-1}''$.

Пусть $l_{k-1} = | x_{k-1}'|$ – длина слова x_{k-1}'. При стандартной индексации от 0 значение l_{k-1} определяет позицию нетерминала A_k в слове x_{k-1}, к которому применяется правило вывода p_k.

Тогда последовательность пар чисел

$$\pi = (l_0, p_1), \ldots , (l_{k-1}, p_k), \ldots , (l_{n-1}, p_n)$$

однозначно определяет вывод $S \Rightarrow^* x$, причем $l_0 = 0$. Длина последовательности π определяет длину вывода: $|\pi| = n > 0$.

Задача грамматического разбора (*parsing*) или *синтаксического анализа* заключается в нахождении вывода в грамматике G для слова $y \in \Sigma^*$, если $y \in L(G)$, либо доказательству того, что не существует никакого вывода, если $y \notin L(G)$.

Алгоритм или машина грамматического разбора (*parser*) решает, в частности, проблему распознавания языка. Распознаватель языка в общем случае может и не решать задачу грамматического разбора.

Для предложения языка $x \in L(G)$ может существовать несколько выводов. Поскольку выводы однозначно определяются π-последовательностями, тогда пары выводов различаются, если различаются (не равны) их π-последовательности.

3.2. Синтаксические деревья

Синтаксическое дерево $D(\pi)$ (дерево вывода или дерево грамматического разбора) однозначно строится по заданному выводу π. Дерево $D(\pi)$ в явной форме определяет иерархическую зависимость синтаксических элементов КС-языка (синтагм) в предложении x, которое выводимо с помощью вывода π.

Пусть $G = (N, \Sigma, P, S)$ – КС-грамматика и A – произвольный нетерминал. Грамматика $G_A = (N, \Sigma, P, A)$ определяет КС-язык – язык фраз типа A (синтаксических элементов КС-языка $L(G)$ типа A или элементов КС-языка $L(G)$ синтаксической категории A).

Далее под выводом π понимается произвольный вывод в любой КС - грамматике $G_A = (N, \Sigma, P, A)$.

Вывод π определяется последовательностью частичных выводов π_k длины $k = |\pi_k|$, $k = 0, 1, \ldots, n$; $\pi_n = \pi$: $\pi_0, \pi_1, \ldots, \pi_k, \ldots, \pi_n$; где π_k – начальный сегмент (“префикс”) длины k; $\pi_0 = \lambda$ – “пустой вывод”; $\pi_1 = (l_0, p_1)$;

$$\pi_k = (l_0, p_1), \ldots, (l_{k-1}, p_k).$$

Построение дерева вывода $D(\pi)$ проводится индуктивно по последовательности частичных выводов. Используется изоморфное графическое представление дерева на плоскости. Дерево “перевернуто” так, что корень находится вверху, а листья – внизу.

Узлам дерева соответствуют точки на плоскости. Узлы имеют метки. Корень помечается аксиомой грамматики. Внутренние узлы всегда помечаются нетерминалами. Терминалами могут помечаться только листья. Специальный символ λ, обозначающий пустую строку, может помечать только лист дерева.

Пусть $D(\pi_0), D(\pi_1), \ldots, D(\pi_{k-1}), D(\pi_k), \ldots, D(\pi_n)$ – последовательность деревьев, соответствующая последовательности частичных выводов $\pi_0, \pi_1, \ldots, \pi_{k-1}, \pi_k \ldots, \pi_n$, порождающих последовательность слов:

$$x_0, x_1, \ldots, x_{k-1}, x_k, \ldots, x_n.$$

Каждое дерево $D(\pi_k)$ "упорядочено" в том смысле, что если при его обходе от корня сверху вниз и слева направо выписать метки листьев, игнорируя листья, помеченные символом пустого слова λ, то получим слово x_k, выводимое с помощью π_k при $k = 0, \ldots, n$.

Базис индукции: $k = 0$. $D(\pi_0)$ – содержит один узел, помеченный аксиомой A.

Индуктивный переход: пусть $D(\pi_{k-1})$ – дерево вывода и π_k – вывод, $0 < k \le n$. Тогда $\pi_k = \pi_{k-1}\ (l_{k-1}, p_k)$ и $p_k)\ A_k \to \gamma_k$.

Дерево $D(\pi_k)$ строится следующим образом:

1) копируем $D(\pi_{k-1})$;
2) обходим слева направо и нумеруем от 0 листья дерева $D(\pi_{k-1})$, игнорируя листья, помеченные символом пустого слова λ; находим лист с индексом l_{k-1} – искомый лист, помеченным нетерминалом A_k;
3) искомый лист, помеченный нетерминалом A_k, преобразуем во внутренний родительский узел для символов слова γ_k – правой части правила $p_k)\ A_k \to \gamma_k$:

   а) правило $p_k)\ A_k \to \gamma_k$ имеет форму $p_k)\ A_k \to \lambda$, $\gamma_k = \lambda$; добавляем лист, помечаем его символом λ и связываем его ребром с родительским узлом, помеченный нетерминалом A_k;

   б) правило $p_k)\ A_k \to \gamma_k$ имеет форму $p_k)\ A_k \to B_1, \ldots, B_r$, $\gamma_k = B_1, \ldots, B_r$, $|\gamma_k| = r > 0$, $\gamma_k \in (N \cup \Sigma)^+$; добавляем r листьев, помечаем их слева направо символами $B_1, \ldots, B_r$ и связываем каждый лист ребром с родительским узлом, помеченным нетерминалом A_k .

3.3. Канонические выводы

Пусть π – вывод и $D(\pi)$ – дерево вывода, построенное индуктивно по вышеуказанной процедуре. По построению каждый внутренний узел, помечается нетерминалом, который определяет левую часть правила вывода, а дочерние узлы помечаются слева направо символами слова правой части правила.

Каждому внутреннему узлу можно сопоставить во взаимнооднозначное соответствие правило грамматики, образованное метками родительского и его дочерних узлов. Тогда вывод π определяет (без повторения) обход внутренних узлов дерева вывода D(π). Но деревья допускают различные обходы.

Рассмотрим два обхода:

а) сверху вниз и слева направо (левосторонний) и б) сверху вниз и справа налево (правосторонний).

При первом посещении внутреннего узла выписываем номер правила, ассоциированного с этим узлом. В результате получим две последовательности номеров правил: а) π^L и б) π^R . Эти последовательности однозначно определяют два вывода слова x, выводимого с помощью исходного вывода π.

Последовательность π^L называется *левосторонним* выводом, а π^R – *правосторонним* выводом слова x. Оба вывода называются *каноническими*. Они явно не указывают позиции нетерминалов в сентенциальных формах, к которым применяются правила.

Однако неявно имеется простое правило, которое однозначно определяет позицию нетерминала:

а) при *левостороннем* выводе правило p_k применятся к самому левому нетерминалу слова x_{k-1} для всех $k \in [1, n]$;

б) при *правостороннем* выводе правило p_k применятся к самому правому нетерминалу слова x_{k-1} для всех $k \in [1, n]$. Последовательности π^L и π^R в общем случае представляют различные порядки применения последовательности правил вывода π, но они определяют одно и то же дерево грамматического разбора.

Два вывода называются *эквивалентными*, если они определяют одно и то же дерево вывода. Каждое дерево вывода однозначно определяет канонический вывод. Обратно, каждый канонический вывод однозначно определяет дерево вывода. Следующие три определения зквивалентны.

Грамматика называется *однозначной*, если любое предложение языка имеет единственный левосторонний вывод.

Грамматика называется *однозначной*, если любое предложение языка имеет единственный правосторонний вывод.

Грамматика называется *однозначной*, если любое предложение языка имеет единственное дерево вывода.

Пример неоднозначной грамматики G_1:

1) $E \rightarrow E + E$
2) $E \rightarrow E * E$
3) $E \rightarrow i$

Слово $i + i * i$ имеет два левосторонних вывода и, следовательно, – два дерева грамматического разбора:

a) 1, 3, 2, 3, 3;
b) 2, 1, 3, 3, 3.

Пример однозначной грамматики G_2:

1) $E \rightarrow E + T$
2) $E \rightarrow T$

3) $T \to T * R$

4) $T \to R$

5) $R \to i$

Слово $i + i * i$ имеет единственный левосторонний вывод и, следовательно, единственное дерева грамматического разбора:
1, 2, 4, 5, 3, 4, 5, 5.

КС-язык называется однозначным, если существует однозначная КС-грамматика, его порождающая. Язык $L(G_1)$ однозначный, т. к. $L(G_1) = L(G_2)$.

Пример однозначной грамматики G_3:

1) $E \to E + T$; 2) $E \to E - T$; 3) $E \to T$; 4) $T \to T * R$; 5) $T \to T / R$;

6) $T \to R$; 7) $R \to (E)$; 8) $R \to i$; 9) $R \to c$

Слово $w = i - i * c$

Вывод π (не канонический):

	x_{k-1}	(l_{k-1}, p_k)
x_0	$.E \Rightarrow$	(0, 2)
x_1	$E - .T \Rightarrow$	(2, 4)
x_2	$.E - T * R \Rightarrow$	(0, 3)
x_3	$T - .T * R \Rightarrow$	(2, 6)
x_4	$T - R * .R \Rightarrow$	(4, 9)
x_5	$T - .R * c \Rightarrow$	(2, 8)
x_6	$.T - i * c \Rightarrow$	(0, 6)
x_7	$.R - i * c \Rightarrow$	(0, 8)
x_8	$i - i * c$	

π = (0, 2) (2, 4) (0, 3) (2, 6) (4, 9) (2, 8) (0, 6) (0, 8).

Проекция по второй компоненте определяет порядок применения правил $pr_2(\pi)$ = 2 4 3 6 9 8 6 8.

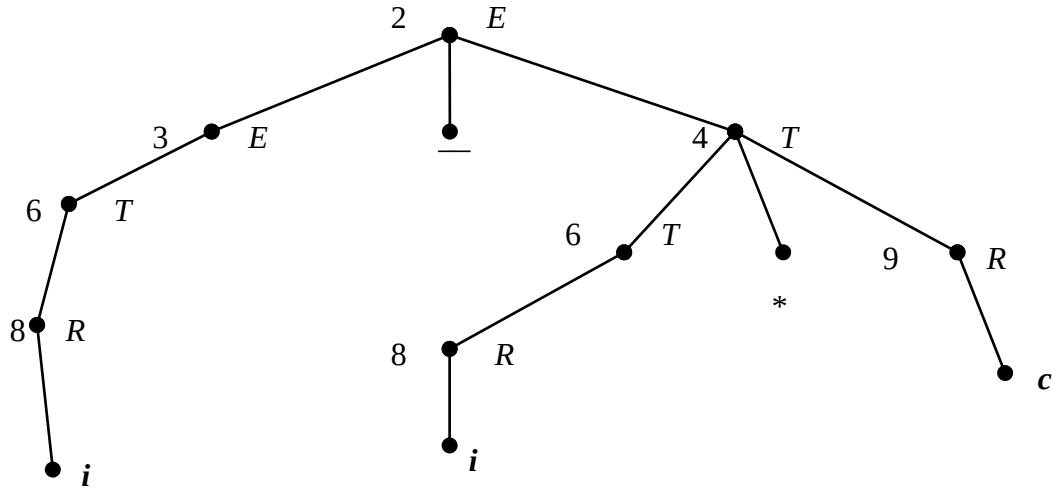
Вывод π^L (канонический левосторонний), π^L = 2 3 6 8 4 6 8 9:

	x_{k-1}	p_k
x_0	$.E \Rightarrow$	2
x_1	$.E - T \Rightarrow$	3
x_2	$.T - T \Rightarrow$	6
x_3	$.R - T \Rightarrow$	8
x_4	$i - .T \Rightarrow$	4
x_5	$i - .T * R \Rightarrow$	6
x_6	$i - .R * R \Rightarrow$	8
x_7	$i - i * .R \Rightarrow$	9
x_8	$i - i * c$	

Вывод π^R (канонический правосторонний), π^R = 2 4 9 6 8 3 6 8:

	x_{k-1}	p_k
x_0	$.E \Rightarrow$	2
x_1	$E - .T \Rightarrow$	4
x_2	$E - T^* .R \Rightarrow$	9
x_3	$E - .T^* c \Rightarrow$	6
x_4	$E - .R^* c \Rightarrow$	8
x_5	$.E - i^* c \Rightarrow$	3
x_6	$.T - i^* c \Rightarrow$	6
x_7	$.R - i^* .R \Rightarrow$	8
x_8	$i - i^* c$	

Порядки применения правил $pr_2(\pi)$, π^L, π^R попарно различаются, но дерево разбора одно и то же для всех трех выводов. Дерево грамматического разбора предложения $w = i - i^* c \in L(G_3)$.



$D(\pi) = D(\pi^L) = D(\pi^R)$

$pr_2(\pi)$ = 2 4 3 6 9 8 6 8

π^L = 2 3 6 8 4 6 8 9

π^R = 2 4 9 6 8 3 6 8

3.4. Функция $FIRST_k(X)$

Пусть $G=(N, \Sigma, P, S)$ – КС-грамматика и $L \subseteq \Sigma^*$, k – фиксированное неотрицательное целое число. Определим функцию $first_k(x)$ над Σ^*:
$first_k(x) = y$, где $y = x$, если $|x| \le k$, если же $|x| > k$, тогда $x = yz$, $|y| = k$. Тогда образ языка L от функции $FIRST_k(X)$ есть

$FIRST_k(L) = \{first_k(x) \mid x \in L\}$.

Для функции $FIRST_k(X)$ имеют место следующие свойства:

1. $FIRST_k(L) = FIRST_k(FIRST_k(L))$, $L \subseteq \Sigma^*$ (*идемпотентность*). В частности $FIRST_k(\varnothing) = \varnothing$ для всех $k \geq 0$;

2. $FIRST_k(L_1 \cup L_2) = FIRST_k(L_1) \cup FIRST_k(L_2)$ для любых $L_1, L_2 \subseteq \Sigma^*$;

3. Для любых $L_1, L_2 \subseteq \Sigma^*$, $k \geq 1$ $FIRST_k(L_1) \cap FIRST_k(L_2) = \varnothing$ влечет $L_1 \cap L_2 = \varnothing$.

Обратное утверждение в общем случае неверно. Контр-пример легко привести.

4. Для любых $L_1, L_2 \subseteq \Sigma^*$ $L_1 \subseteq L_2$ влечет $FIRST_k(L_1) \subseteq FIRST_k(L_2)$;

5. $FIRST_k(L_1L_2) = FIRST_k(FIRST_k(L_1)\ FIRST_k(L_2))$ для любых $L_1, L_2 \subseteq \Sigma^*$.

Выражение в правой части равенства определяет правило вычисления функции от конкатенации двух языков, что позволяет определить специальную операцию над языками – «усеченную» конкатенацию: $L_1 \oplus_k L_2 = FIRST_k(FIRST_k(L_1)\ FIRST_k(L_2))$;

6. Для любых $L_1, L_2, L_3 \subseteq \Sigma^*$ выполнимо

$(L_1 \oplus_k L_2) \oplus_k L_3 = L_1 \oplus_k (L_2 \oplus_k L_3)$ (*ассоциативный закон*);

7. Для любых $L_1, L_2, L_3 \subseteq \Sigma^*$ выполнимы

$(L_1 \cup L_2) \oplus_k L_3 = (L_1 \oplus_k L_3) \cup (L_2 \oplus_k L_3)$

и $L_1 \oplus_k (L_2 \cup L_3) = (L_1 \oplus_k L_2) \cup (L_1 \oplus_k L_3)$ (*дистрибутивные законы*).

Обоснование всех свойств выводится непосредственно путем применения определения $FIRST_k(X)$, законов теории множеств, законов алгебры регулярных множеств, а также очевидного представления любого языка $L \subseteq \Sigma^*$ при $k \geq 0$:

$L = P \cup S$, $P \cap S = \varnothing$,

где $P = \{x \mid x \in L, |x| \leq k\}$ и $S = \{x \mid x \in L, |x| > k\}$.

Пусть $y \in (N \cup \Sigma)^*$ и $L_y = \{x \mid y \Rightarrow^* x, x \in \Sigma^*\}$, определим

$FIRST_k(y) = FIRST_k(L_y)$. В силу обозначений, свойств функции $FIRST_k(X)$ и свойств отношения вывода $\Rightarrow^*$ для всех $y_1, y_2 \in (N \cup \Sigma)^*$ выполнимо

$$FIRST_k(y_1y_2) = y_1 \oplus_k y_2.$$

Если $y \in (N \cup \Sigma)^*$ и $|y| = n \geq 2$, тогда $y = Y_1 Y_2 \ldots Y_n$, где $Y_i \in (N \cup \sum)$, $i \in [1, n]$, тогда верно равенство

$$FIRST_k(y) = Y_1 \oplus_k Y_2 \oplus_k \ldots \oplus_k Y_n.$$

Пусть e – произвольное регулярное выражение, построенное над объединенным алфавитом $(N \cup \Sigma)$ КС-грамматики G без использования

операции замыкания Клини. Определим интерпретацию таких ограниченных регулярных выражений посредством функции $FIRST_k(X)$ для $k \geq 1$:

1. $e = \lambda$: $FIRST_k(\lambda) = \{\lambda\}$;

   $e = a, a \in \Sigma$: $FIRST_k(a) = \{a\}$;

   $e = A, A \in N$: $FIRST_k(A) = FIRST_k(L_A)$;

2. Пусть v и w – ограниченные регулярные выражения, $FIRST_k(v)$ и $FIRST_k(w)$ – их значения соответственно, тогда

для $e = (v)$ его значение $FIRST_k(v)$;

для $e = v \mid w$ – $FIRST_k(v \mid w) = FIRST_k(v) \cup FIRST_k(w)$;

для $e = v\,w$ – $FIRST_k(v\,w) = FIRST_k(\,FIRST_k(v)FIRST_k(w))$ или, используя операцию $\oplus_k$, $FIRST_k(v\,w) = v \oplus_k w$.

3.5. Система определяющих уравнений

Определение. Пусть $G = (N, \Sigma, P, S)$ – КС грамматика, $A \in N$. Если множество правил пусто для A, тогда определяющее уравнение для A имеет вид $A=\varnothing$.

Пусть $A \to \gamma_1$, $A \to \gamma_2$, … , $A \to \gamma_m$, $m \geq 1$, – правила грамматики для нетерминала A, тогда определяющее уравнение для A имеет вид

$A = \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_n$.

Совокупность определяющих уравнений всех нетерминалов образуют систему определяющих уравнений грамматики G. Правые части уравнений являются ограниченными регулярными выражениями. Этим выражениям приписываются значения в стандартной интерпретации. Нетерминалы как переменные системы уравнений принимают в качестве значений языки над основным алфавитом Σ.

Решением системы уравнений является набор языков, подстановка которых вместо соответствующих нетерминалов превращает систему уравнений в систему тождеств.

Решение t называется *наименьшим*, если для любого другого решения g выполняется $t(A) \subseteq g(A)$ для всех $A \in N$, где $t(A)$, $g(A) \subseteq \Sigma^*$ -- значения нетерминала A решений t и g соответственно.

Пусть $N = \{A_i \mid 1 \leq i \leq n\}$ и $\boldsymbol{A} = (A_1, A_2, \ldots, A_n)$ – вектор-строка нетерминалов, тогда систему определяющих уравнений можно записать в компактной форме

$$A_i = E_i(\boldsymbol{A}),\ 1 \leq i \leq n,$$

где $E_i(\boldsymbol{A})$ – регулярное ограниченное выражение, определяемое правилами нетерминала A_i.

В векторной форме система принимает вид

$$\boldsymbol{A} = \boldsymbol{E}\ (\boldsymbol{A}),$$

где $\boldsymbol{E}(\boldsymbol{A}) = (E_1(\boldsymbol{A}), E_2(\boldsymbol{A}), \ \dots\ , E_n(\boldsymbol{A}))$– вектор-строка регулярных выражений.

Определим рекуррентную систему соотношений на основе определяющей системы уравнений:

$$A_i^{l+1} = E_i(\boldsymbol{A}^{\,l}\ ) \mid A_i^{\,l}\ ,\ 1 \le i \le \mathrm{n},\ \ l \ge 0,\ \boldsymbol{A}^{\,l} = (A_1^{\,l}, A_2^{\,l}, \dots, A_n^{\,l}\ ),$$

$$\boldsymbol{A}^{\,0} = (A_1^{\,0}, A_2^{\,0}, \dots, A_n^{\,0}\ ) = (\varnothing, \varnothing, \dots, \varnothing).$$

В векторной форме рекуррентная система принимает вид

$$\boldsymbol{A}^{l+1} = \boldsymbol{E}(\boldsymbol{A}^{l}) \mid \boldsymbol{A}^{l},\ l \ge 0,\ \boldsymbol{A}^{0} = (\varnothing, \varnothing, \dots, \varnothing).$$

Легко видеть, что для i и l, $1 \le i \le n$, и $l \ge 0$ язык $A_i^{\,l} \subseteq \Sigma^*$ суть множество терминальных цепочек, выводимых из нетерминала A_i, деревья выводов которых имеют высоту $h \le l$. Очевидно также, что $A_i^{\,l} \subseteq A_i^{\,l+1}$ для всех i и l.

Множество всех терминальных цепочек, выводимых из нетерминала A_i, определяется как $\bigcup\limits_{l=0}^{\infty} A_i^{\,l}$.

Подставляя эти множества вместо нетерминалов, непосредственно проверяем, что они доставляют решение системе определяющих уравнений.

Обозначим полученное решение через g. Предположим, что решение g не минимально. Тогда найдется решение t такое, что $t(A_i) \subseteq g(A_i)$ для всех i, $1 \le i \le n$, и хотя бы для одного i $\ t(A_i) \ne g(A_i)$. В силу допущения найдутся нетерминал A_i, цепочка x, $x \in g(A_i)$, $x \notin t(A_i)$, дерево вывода которой имеет наименьшую высоту $h \ge 1$ среди всех цепочек и нетерминалов с таким свойством.

Высота h не может быть больше 1. Если $h > 1$, тогда для любого внутреннего узла, не корня, помеченного нетерминалом A_j справедливо следующее утверждение: если A_j порождает подцепочку y цепочки x,
то $y \in g(A_j)$ влечет $y \in t(A_j)$.

Если $y \notin t(A_j)$, то это противоречило бы выбору цепочки x. Но тогда, в силу правила построения дерева вывода, для цепочки x как результату конкатенации других цепочек решения t верно $x \in t(A_i)$, что противоречит допущению. Следовательно, высота дерева вывода цепочки x равна 1. Но это означает, что для цепочки $x \in \Sigma^*$ существует правило $A_i \to x$

грамматики и, поэтому необходимо $x \in t(A_i)$, что невозможно. Полученное противоречие доказывает минимальность решения g.

Наименьшее решение системы определяющих уравнений КС-грамматики G называется *наименьшей неподвижной точкой*. Таким образом доказана следующая

Лемма. Наименьшая неподвижная точка системы определяющих уравнений КС-грамматики $G = (N, \Sigma, P, S)$ существует, единственна и имеет вид $t(A_i) = \{x \mid A_i \Rightarrow^* x, x \in \Sigma^*\}$ для всех $A_i \in N = \{A_i \mid 1 \le i \le n\}$.

Непосредственно из доказательства леммы вытекают следующие важные свойства, на которых строится обоснование алгоритма вычисления функции $FIRST_k(X)$.

1. *Множество* $t(A_i)$ всех терминальных цепочек, выводимых из нетерминала A_i, определяется по формуле

$$t(A_i) = \bigcup_{l=0}^{\infty} A_i^l,$$

в которой множества A_i^l вычисляются по рекуррентной системе уравнений

$$A_i^{l+1} = E_i(\boldsymbol{A}^l) \mid A_i^l,\ 1 \le i \le n,\ l \ge 0,\ \boldsymbol{A}^l = (A_1^l, A_2^l, \ldots, A_n^l),$$

$$\boldsymbol{A}^0 = (A_1^0, A_2^0, \ldots, A_n^0) = (\varnothing, \varnothing, \ldots, \varnothing).$$

2. *Множества* A_i^l суть множества терминальных цепочек, выводимых из нетерминала A_i, деревья выводов которых имеют высоту $h \le l$, при чем для всех i и l, $1 \le i \le n$, $l \ge 0$, имеет место включение

$$A_i^l \subseteq A_i^{l+1}.$$

3. Для фиксированных i и l, $1 \le i \le n$, $l \ge 0$, *множество* A_i^l конечно по мощности.

3.6. Алгоритм вычисления функции $FIRST_k(X)$, $k \ge 1$

Пусть $G = (N, \Sigma, P, S)$ – КС-грамматика, $N = \{A_i \mid 1 \le i \le n\}$, $\boldsymbol{A} = (A_1, A_2, \ldots, A_n)$ – вектор-строка нетерминалов и $t(A_i)$, $1 \le i \le n$, – определяют неподвижную точку соответствующей системы определяющих уравнений.

Множество $t(A_i)$ всех терминальных цепочек, выводимых из нетерминала A_i, определяется по формуле

$$t(A_i) = \bigcup_{l=0}^{\infty} A_i^l, \qquad (1)$$

в которой множества A_i^l вычисляются по рекуррентной системе уравнений

$$A_i^{l+1} = E_i(\boldsymbol{A}^l) \mid A_i^l \ , \qquad (2)$$

$$1 \le i \le n,\ l \ge 0,\ \boldsymbol{A}^l = (A_1^l, A_2^l, \ldots, A_n^l),$$

$$\boldsymbol{A}^0 = (A_1^0, A_2^0, \ldots, A_n^0) = (\varnothing, \varnothing, \ldots, \varnothing).$$

Применим к левой и правой частям каждого тождества (2) *функцию* $FIRST_k(X)$. Мы получим, в силу свойств функции $FIRST_k(X)$, подобную (гомоморфную) рекуррентную систему тождеств

$$\underline{A}_i^{l+1} = \underline{E}_i(\underline{\boldsymbol{A}}^l) \mid \underline{A}_i^l \ , \qquad (3)$$

$$1 \le i \le n,\ l \ge 0,\ \underline{\boldsymbol{A}}^l = (\underline{A}_1^l, \underline{A}_2^l, \ldots, \underline{A}_n^l),$$

$$\underline{\boldsymbol{A}}^0 = (\underline{A}_1^0, \underline{A}_2^0, \ldots, \underline{A}_n^0) = (\varnothing, \varnothing, \ldots, \varnothing),$$

где $\underline{A}_i^l = FIRST_k(A_i^l)$ и $FIRST_k(A^l) = \underline{\boldsymbol{A}}^l$.

В векторной форме система (3) принимает вид

$$\underline{\boldsymbol{A}}^{l+1} = \underline{\boldsymbol{E}}(\underline{\boldsymbol{A}}^l) \mid \underline{\boldsymbol{A}}^l \ , \ \underline{\boldsymbol{A}}^0 = \underline{\varnothing}\ , \qquad (4)$$

где $\underline{\varnothing} = (\varnothing, \varnothing, \ldots, \varnothing)$.

Выражение $\underline{E}_i(\underline{\boldsymbol{A}}^l)$ имеет структуру подобную (гомоморфную) структуре $E_i(\boldsymbol{A}^l)$:

а) если для нетерминала A_i множество правил в грамматике пусто, то $E_i(\boldsymbol{A}^l)$ представлено выражением $\varnothing$. Тогда $\underline{E}_i(\underline{A}^l)$ также представлено выражением $\varnothing$, так как $FIRST_k(\varnothing)=\varnothing$;

б) если для нетерминала A_i множество правил в грамматике не пусто и существует правило $A_i \to \lambda$, тогда $E_i(\boldsymbol{A}^l)$ содержит слагаемое λ. Так как $FIRST_k(\lambda) = \lambda$, то $\underline{E}_i(\underline{A}^l)$ содержит слагаемое λ;

в) правило вида $A_i \to a$, $a \in \Sigma$, представлено слагаемым a как в $E_i(\boldsymbol{A}^l)$ так и в $\underline{E}_i(\underline{\boldsymbol{A}}^l)$, поскольку $FIRST_k(a) = a$ при $k \ge 1$.

Правило вида $A_i \to A_j$, $A_j \in N$, в $E_i(\boldsymbol{A}^l)$ представлено слагаемым A_j. Тогда в $\underline{E}_i(\underline{\boldsymbol{A}}^l)$ представлено слагаемым $\underline{A}_j$, так как $\underline{A}_i^l = FIRST_k(A_i^l)$;

г) правило вида $A_i \to \gamma$, $\gamma \in (N \cup \Sigma)^*$, $|\gamma| = r \ge 2$, представлено в $E_i(\boldsymbol{A}^l)$ словом $\gamma = X_1 X_2 \ldots X_r$, $X_j \in (N \cup \Sigma)$, $1 \le j \le r$. В выражении $\underline{E}_i(\underline{\boldsymbol{A}}^l)$ слагаемому γ будет соответствовать слагаемое $\underline{X}_1 \oplus_k \underline{X}_2 \oplus_k \ldots \oplus_k \underline{X}_r$, где $\underline{X}_j = FIRST_k(X_j)$, $1 \le j \le r$.

Фактически вышеописанные правила а) – г) определяют алфавитный гомоморфизм, преобразующий систему (2) в систему (3), причем операция конкатенации языков отображается на «усеченную» конкатенацию $\oplus_k$. Операция объединения языков остается неизменной.

Применим к левой и правой частям равенства (1) *функцию* $FIRST_k(X)$. В силу свойств функции $FIRST_k(X)$ и вышеприведенных обозначений имеем

$$FIRST_k(\boldsymbol{A}_i) = \bigcup_{l=0}^{\infty} \underline{\boldsymbol{A}}_i^{\,l}, \; 1 \le i \le n. \tag{5}$$

В векторной форме эта система равенств имеет вид

$$FIRST_k(\boldsymbol{A}) = \bigcup_{l=0}^{\infty} \underline{\boldsymbol{A}}^{\,l}. \tag{6}$$

Определение. Пусть $\boldsymbol{X} = (X_1, X_2, \ldots, X_n)$ и $\boldsymbol{Y} = (Y_1, Y_2, \ldots, Y_n)$ – векторы, компоненты которых $X_i, Y_i \subseteq \Sigma^*$, $1 \le i \le n$. Определим отношение частичного порядка на множестве векторов:

$\boldsymbol{X} \subseteq \boldsymbol{Y}$, если $X_i \subseteq Y_i$ для всех $i \in [1, n]$.

В силу свойства 4 функции $FIRST_k(X)$ имеем

$$\underline{\boldsymbol{A}}^{\,l} \subseteq \underline{\boldsymbol{A}}^{\,l+1} \text{ для всех } l \ge 0. \tag{7}$$

Последовательность $\{ \underline{\boldsymbol{A}}^{\,l} \mid l \ge 0 \}$ – неубывающая и ограниченная сверху, следовательно найдется наименьшее $l = t \ge 0$, что $\underline{\boldsymbol{A}}^{\,l} = \underline{\boldsymbol{A}}^{\,t}$ для всех $l \ge t$. Отсюда немедленно следует, что

$$FIRST_k(\boldsymbol{A}) = \bigcup_{l=0}^{\infty} \underline{\boldsymbol{A}}^{\,l} = \underline{\boldsymbol{A}}^{\,0} \cup \underline{\boldsymbol{A}}^{\,1} \cup \ldots \cup \underline{\boldsymbol{A}}^{\,t} \tag{8}$$

или ввиду (7)

$$FIRST_k(\boldsymbol{A}) = \underline{\boldsymbol{A}}^{\,t}. \tag{9}$$

Принципиально алгоритм вычисления функции $FIRST_k(X)$ на множестве нетерминалов N КС-грамматики G прост и состоит из двух шагов:

1) по данной грамматике G выписать систему определяющих уравнений, преобразовать её в рекуррентную систему равенств (4)

$$\underline{\boldsymbol{A}}^{\,l+1} = \underline{\boldsymbol{E}}\,(\underline{\boldsymbol{A}}^{\,l}) \mid \underline{\boldsymbol{A}}^{\,l}, \; \underline{\boldsymbol{A}}^{\,0} = \underline{\varnothing};$$

2) вычислять по формуле (4) элементы неубывающей ограниченной сверху последовательности $\{\underline{\boldsymbol{A}}^{\,l} \mid l \ge 0\}$ пока $\underline{\boldsymbol{A}}^{\,l} \ne \underline{\boldsymbol{A}}^{\,l+1}$. Наименьшее $l = t$, при котором достигается условие $\underline{\boldsymbol{A}}^{\,l} = \underline{\boldsymbol{A}}^{\,l+1}$ означает, что функция $FIRST_k(X)$ на множестве нетерминалов N вычислена и $FIRST_k(\boldsymbol{A}) = \underline{\boldsymbol{A}}^{t}$.

Рассмотрим применение данного алгоритма при k=1 к следующей грамматике:

$E \to TR$

$R \to \lambda \mid +TR \mid -TR$

$T \to a \mid i \mid (E)$

Система определяющих уравнений:

$E = TR$

$R = \lambda | +TR | - TR$

$T = a | i | (E)$

Рекуррентная система равенств:

$\underline{E}^{l+1} = \underline{T}^l \oplus_1 \underline{R}^l \mid \underline{E}^l$

$\underline{R}^{l+1} = \lambda | + \oplus_1 \underline{T}^l \ \oplus_1 \underline{R}^l \ | - \oplus_1 \underline{T}^l \oplus_1 \underline{R}^l \mid \underline{R}^l$

$\underline{T}^{l+1} = a \mid i \mid (\oplus_1 \underline{E}^l \ \oplus_1) \mid \underline{T}^l$

$\underline{E}^0 = \varnothing; \ \underline{R}^0 = \varnothing; \ \underline{T}^0 = \varnothing.$

Сходящаяся последовательность векторов ($\underline{E}^l$, $\underline{R}^l$, $\underline{T}^l$).

Компоненты векторов представлены регулярными выражениями.

$(\varnothing, \ \varnothing, \ \varnothing)$

$(\varnothing, \ \lambda, \ a \mid i)$

$(a \mid i , \quad \lambda|+| - \ , \quad a \mid i \quad)$

$(a \mid i , \quad \lambda|+|- \ , \quad a \mid i \mid ( )$

$(a \mid i \mid (, \lambda|+|- \ , \quad a \mid i \mid ( )$

$(a \mid i \mid (, \lambda|+|- \ , \quad a \mid i \mid ( )$

На пятом шаге итерации достигается предел последовательности:
$(FIRST_1 (E), FIRST_1(R), FIRST_1 (T)) = (a \mid i \mid (, \lambda|+|- , a \mid i \mid ( ).$

4. *LL*(*k*)-ГРАММАТИКИ

4.1. Характеристическое свойство класса *LL*(*k*)-грамматик, $k \geq 1$

Определение. Контекстно-свободная грамматика $G = (N, \Sigma, P, S)$ называется $LL(k)$-грамматикой при $k \geq 1$, если из существования двух левых выводов

$$S \Rightarrow^* \omega A \alpha \Rightarrow \omega \gamma_1 \alpha \Rightarrow^* \omega x \in \Sigma^* , A \to \gamma_1 \in P \qquad (1)$$

$$S \Rightarrow^* \omega A \alpha \Rightarrow \omega \gamma_2 \alpha \Rightarrow^* \omega y \in \Sigma^*, A \to \gamma_2 \in P \qquad (2)$$

и выполнения условия $first_k(x) = first_k(y)$ следует, что $\gamma_1 = \gamma_2$.

Ниже формулируемая теорема предоставляет *характеристическое свойство* класса $LL(k)$-грамматик при $k \geq 1$, которое может быть эффективно проверено.

Теорема. Для того, чтобы КС-грамматика $G = (N, \Sigma, P, S)$ при фиксированном $k \geq 1$ была $LL(k)$-грамматикой, необходимо и достаточно чтобы

для любого нетерминала $A \in N$, любых двух различных его альтернатив $A \to \gamma_1$ и $A \to \gamma_2$ и любого левого вывода $S \Rightarrow^* \omega A\alpha$ выполнялось условие

$$FIRST_k(\gamma_1\alpha) \cap FIRST_k(\gamma_2\alpha) = \varnothing. \quad (3)$$

Определим на множестве символов $N \cup \Sigma$ грамматики G функцию $FOLLOW_k(X)$:

$$FOLLOW_k(X) = \bigcup_{\alpha} FIRST_k(\alpha).$$

Объединение берется по всем *правым контекстам* α *символа* X, для которых существует левый вывод $S \Rightarrow^* \omega X\alpha$, $\omega \in \Sigma^*$

Определение. Грамматика при фиксированном $k \geq 1$ называется $SLL(k)$ (строгой – *strong* $LL(k)$) грамматикой, если для любого нетерминала $A \in N$, любых двух его различных альтернатив $A \to \gamma_1$ и $A \to \gamma_2$ истинно равенство

$$FIRST_k(\gamma_1 FOLLOW_k(A)) \cap FIRST_k(\gamma_2 FOLLOW_k(A)) = \varnothing. \quad (4)$$

Очевидно, истинность условия (4) влечет истинность (3). Обратное утверждение для $k \geq 2$ неверно. Например, грамматика

$$S \to aAaa \mid bAba;$$
$$A \to \lambda \mid b$$

является $LL(2)$, но не $SLL(2)$. Нетерминал S имеет один правый контекст λ, A – два: aa и ba.

Однако при $k = 1$ имеет место

Теорема. КС-грамматика является $LL(1)$-грамматикой тогда и только тогда, когда она является $SLL(1)$-грамматикой.

Если грамматика является $LL(1)$-грамматикой, то либо пустая цепочка не выводится из нетерминала, и в этом случае условия (3) и (4) равносильны условию

$$FIRST_1(\gamma_1) \cap FIRST_1(\gamma_2) = \varnothing. \quad (5)$$

В противном случае должна существовать единственная альтернатива, скажем $A \to \gamma_1$, что $\gamma_1 \Rightarrow_* \lambda$. Тогда условия (3) и (4) равносильны условиям (6) и (7):

$$FIRST_1(\gamma_i) \cap FIRST_1(\gamma_j) = \varnothing,\ 1 \leq i \neq j \leq m, \quad (6)$$
$$FOLLOW_1(A) \cap FIRST_1(\gamma_j) = \varnothing,\ 1 < j \leq m, \quad (7)$$

где $A \to \gamma_1 \mid \ldots \mid \gamma_m$ – альтернативы нетерминала A.

4.2. Эквивалентные преобразования контекстно-свободных грамматик

Определение. КС-грамматики G и G' называются эквивалентными, если $L(G) = L(G')$.

Необходимость в эквивалентных преобразованиях часто возникает при разработке синтаксических анализаторов.

Пусть $G = (N, \Sigma, P, S)$ – КС-грамматика.

Удаление бесполезных символов

Определение. Нетерминал A называется *бесполезным*, если не существует цепочки $x \in \Sigma^*$, что $A \Rightarrow^* x$ или другими словами, если язык, выводимый из нетерминала A, является пустым. В противном случае будем называть нетерминал A *полезным*.

Определение. Терминал a называется *полезным*, если существует полезный нетерминал A и цепочка $xay \in \Sigma^*$, что $A \Rightarrow^* xay$.

В противном случае терминал a называется *бесполезным*.

Построим *неубывающую по включению* последовательность подмножеств полезных символов грамматики G:

$$\{V_i \mid V_i \subseteq V_{i+1} \subseteq N \cup \Sigma,\ i \geq 0\}.$$

1. Обозначим через $V(A \to \gamma)$ – множество символов объединенного алфавита грамматики G, из которых составлены левая и правая части правила $A \to \gamma$. Тогда по определению

$$V_0 = \{X \mid X \in V(A \to \gamma),\ A \to \gamma \in P \text{ и } \gamma \in \Sigma^*\}$$

2. Для всех i ≥ 0 $V_{i+1} = V_i \cup \{A \mid A \to \gamma \in P \text{ и } \gamma \in V_i^*\}$.

Пусть t – наименьшее i, что $V_t = V_{t+1}$. *Тогда V_t – искомое множество полезных символов, $(N \cup \Sigma) \setminus V_t$ – множество бесполезных символов.* Правило $A \to \gamma$ является бесполезным, если либо A, либо некоторый символ цепочки γ бесполезен.

Удаление из исходной грамматики бесполезных правил и символов, исключая аксиому S, преобразует ее в эквивалентную.

Удаление недостижимых символов

Определение. Символ $X \in N \cup \Sigma$ называется недостижимым, если не существует вывода $S \Rightarrow^* yXz$ для $y, z \in (N \cup \Sigma)^*$. В противном случае X называется достижимым.

Построим неубывающую по включению последовательность подмножеств достижимых символов грамматики G:

$$\{U_i \mid U_i \subseteq U_{i+1} \subseteq N \cup \Sigma,\ i \geq 0\}.$$

1. $U_0 = \{S\}$.
2. Для всех $i \geq 0$

$$U_{i+1} = U_i \cup \{X \mid A \to \alpha X \beta \in P \text{ и } A \in U_i\}.$$

Пусть t – наименьшее i, что $U_t = U_{t+1}$. Тогда U_t – искомое множество достижимых символов, $(N \cup \Sigma) \setminus U_t$ – множество недостижимым символов. Правило $A \to \gamma$ является бесполезным, если A недостижим.

Удаление из исходной грамматики бесполезных правил и недостижимых символов преобразует ее в эквивалентную.

Алгоритм преобразования грамматики G в грамматику, не содержащей бесполезных и недостижимых символов

1. Удалить из G *бесполезные* символы. Пусть грамматика G_1 – результат преобразования.

2. Удалить из G_1 *недостижимые* символы. Пусть грамматика G_2 – результат преобразования.

Грамматика G_2 не содержит бесполезных и недостижимых символов.

Если поменять местами преобразования 1 и 2, тогда потребуется на третьем шаге применить еще раз *удаление недостижимых символов*, чтобы получить необходимый результат.

Устраненние λ – правил

Правила вида $A \to \lambda$ называются λ-*правилами*.

Обозначим через $V = \{A|$ существует вывод $A \Rightarrow^* \lambda\}$.

Определим неубывающую по включению последовательность $\{V_i| \ i \geq 0\}$, предел которой есть V.

1. $V_0 = \{A| \ A \to \lambda \in P\}$.

2. $V_{i+1} = V_i \cup \{A| \ A \to \gamma \in P$ и $\gamma \in V_i^*\}, \ i \geq 0$.

Шаг 1.

Пополним исходную грамматику λ-правилами $A \to \lambda$, если $A \in V$. Очевидно, пополненная грамматика эквивалентна исходной.

Шаг 2.

Применим к пополненной грамматике следующее преобразование.

Рассмотрим правило вида $A \to x_0 B_1 x_1 \ \ldots \ x_n B_n x_n$, $n \geq 1$, $B_i \in V$; $x_0, x_i \in ((N \cup \Sigma) \setminus V)^*$.

Заменим это правило множеством правил

$A \to x_0 (\lambda| \ B_1) x_1 \ \ldots \ x_n (\lambda| \ B_n) x_n$.

Такая подстановка производит вместо одного не более чем 2^n правил.

Шаг 3.

Удалим из грамматики все λ-правила $A \to \lambda$ такие, что $A \neq S$.

Если правило вида $S \to \lambda$, где S – аксиома, отсутствует после вышеопределенной подстановки, тогда преобразование закончено. Тогда в преоб-

разованной грамматике нет λ-правил и *язык, выводимый из аксиомы, не содержит пустой цепочки*.

Если правило вида $S \to \lambda$ существует, тогда оно также удаляется, но добавляются новый символ аксиомы S' и правила $S' \to \lambda | S$. В этом случае язык включает в себя пустую цепочку, для которой существует *единственный* вывод $S' \Rightarrow \lambda$.

Грамматика, полученная в результате такого преобразования, называется *свободной от λ-правил*.

Устранение цепных правил

Правило вида $A \to B$, $B \in N$, называется *цепным*.

Пусть исходная грамматика свободна от λ-правил. Для нетерминала A определим множество нетерминалов выводимых из A посредством цепных правил:

$N_A = \{X \mid A \Rightarrow^* X\}$.

Сначала для каждого нетерминала $A \in N$ пополним его множество альтернатив: добавим *не цепное* правило $A \to \gamma$, если $B \to \gamma$ – не цепное правило и $B \in N_A$.

Затем удалим из пополненной грамматики *все цепные правила*. В результате получим грамматику *свободную от цепных правил* и эквивалентную исходной.

Определение. Грамматика, в которой нет *бесполезных* и *недостижимых* символов, называется *приведенной*.

Определение. Грамматика, в которой нет выводов вида $A \Rightarrow^+ A$, где $A \in N$, называется грамматикой *без циклов* или *свободной от циклов*.

Иногда требуют дополнительно, чтобы *приведенная* грамматика была *свободной от λ– правил и циклов.*

Преобразование *исключение цепных правил,* очевидно, дает в результате грамматику *свободную от циклов*.

Факторизация

Пусть некоторое подмножество альтернатив, не обязательно всех, нетерминала A имеет следующее представление:

$$A \to \alpha\beta_1 | \alpha\beta_2 | \ldots | \alpha\beta_n \text{ , } n \geq 2,$$

α– не пустая цепочка.

Тогда под *левой факторизацией* понимается преобразование, заменяющее данное подмножество правил множеством

$$A \to \alpha B$$
$$B \to \beta_1 | \beta_2 | \ldots | \beta_n \text{, где } B \text{ – новый нетерминал.}$$

Аналогично определяется *правая факторизация.*

Устранение левой рекурсии

Определение. Нетерминал A называется *рекурсивным*, если существует вывод $A \Rightarrow^+ \alpha A \beta$. Нетерминал A называется *лево-рекурсивным,* если $\alpha = \lambda$. Если $\beta = \lambda$, то A называется *право-рекурсивным.*

Пусть $A \to A\alpha_1 | A\alpha_2 | \ldots | A\alpha_m | \beta_1 | \beta_2 | \ldots | \beta_n$ – все альтернативы нетерминала A и ни одна из цепочек β_i не начинается на A.

Заменим данное подмножество правил правилами

$$A \to \beta_1 A' | \beta_2 A' | \ldots | \beta_n A'$$

$A' \to \lambda | \alpha_1 A' | \alpha_2 A' | \ldots | \alpha_m A'$, где A' – новый нетерминал.

Такое преобразование устраняет *непосредственную левую рекурсию*. Этого преобразования во многих случаях достаточно для устранения левой рекурсии.

В общем случае для устранения левой рекурсии необходимо наложить специальные условия на исходную грамматику и использовать метод, подобный методу Гаусса исключения переменных, применяемый для решения леволинейных систем с регулярными коэффициентами. В некоторых случаях требуется, чтобы исходная грамматика была приведенной, свободной от λ – правил и циклов.

4.3. Предсказывающий алгоритм грамматического разбора

Пусть $G = (N, \Sigma, P, S)$ – КС-грамматика.

Алгоритм грамматического разбора LL(k)-грамматик основан на моделировании процесса поиска левого вывода входной анализируемой цепочки.

Фактически требуется разработать *специальный детерминированный последовательностный магазинный автомат (ДМПА)* – преобразователь, который принимает на входе произвольную цепочку $x \in \Sigma^*$, а на выходе выдает π– последовательность номеров правил, примененных в левом выводе $S \Rightarrow^* x$, если $x \in L(G)$, в противном случае сигнализирует об ошибке и останавливается.

Возможные *конфигурации автомата* следующие:

$C_0 = (\cdot\, x, S, \lambda)$ – *начальная конфигурация* автомата;

$C_i = (\omega \cdot y, A\alpha, \pi)$ или $C_i = (\omega \cdot y, a\alpha, \pi)$ – произвольные *текущие конфигурации*, где $S \Rightarrow^* \omega A \alpha$ или $S \Rightarrow^* \omega a \alpha$ – *соответствующие левые, час-*

тично построенные выводы π, $a \in \Sigma$, $A \in N$. Точка '·' определяет *текущее положение указателя входного символа*,
ω – проанализированная часть входной цепочки x,
y – не проанализированная часть x.

Возможные *переходные конфигурации*:
$C_{i+1} = (\omega \cdot y, \gamma\alpha, \pi p)$,
где p: $A \to \gamma$ – правило под номером $p \geq 1$, либо
$C_{i+1} = (\omega a \cdot z, \alpha, \pi)$, если $C_i = (\omega \cdot y, a\alpha, \pi)$ и $y = az$.

Переход $C_i \mapsto C_{i+1}$ при $i \geq 0$ означает, что если применяется правило p: $A \to \gamma$, то символ A выталкивается из магазина и затем в магазин заталкиваются символы цепочки $\gamma \neq \lambda$. Если $\gamma = \lambda$ в магазин ничего не заталкивается. В конец последовательности π дописывается номер p примененного правила. Во втором случае указатель входного символа (терминал a) продвигается на один символ вправо, а из магазина один символ (терминал a) выталкивается.

Конфигурация допуска:
$C_i = (x \cdot \lambda, \lambda, \pi)$ при $i \geq 0$ достигается, если указатель находится в конце x (*исчерпан вход*) и магазин *пуст*.

Возможные *конфигурации синтаксических ошибок*:
$C_i = (\omega \cdot bz, a\alpha, \pi)$ и $a \neq b$;
$C_i = (\omega \cdot y, \lambda, \pi)$ и $y \neq \lambda$;
$C_i = (x \cdot \lambda, A\alpha, \pi)$;
$C_i = (x \cdot \lambda, a\alpha, \pi)$.

Конфигурации *допуска* и *синтаксических ошибок* называются заключительными. Достигнув заключительной конфигурации, автомат останавливается, *алгоритм грамматического разбора* (синтаксического анализа) прекращает работу.

Определение. *Текущая конфигурация* вида $C_i = (\omega \cdot y, A\alpha, \pi)$ называется *критической*.

В случае критической конфигурации для реализации перехода $C_i \mapsto C_{i+1}$ или определения ее как ошибочной автомат применяет *k-предсказывающее правило*.

Для *LL*(k)-грамматик *k-предсказывающее правило* вытекает из *характеристического свойства LL*(k)-*грамматик*. Для *SLL*(k)-грамматик аналогичное правило определяется *характеристическим свойством SLL* (k)-*грамматик*.

Пусть p_j: $A \to \gamma_j$ – альтернатива нетерминала A с номером p_j, $j \in [1, n]$, где n – число всех его альтернатив.

Тогда *k-предсказывающее правило для LL(k)-грамматик* следующее:
если $first_k(y) \in FIRST_k(\gamma_j\alpha)$, то применить правило p_j: $A \to \gamma_j$;
если для всех $j \in [1, n]$ $first_k(y) \notin FIRST_k(\gamma_j\alpha)$, то конфигурация C_i регистрирует *синтаксическую ошибку*.

K-предсказывающее правило для SLL (k)-грамматик следующее:
если $first_k(y) \in FIRST_k(\gamma_j FOLLOW_k(A))$, то применить правило p_j: $A \to \gamma_j$; если для всех $j \in [1, n]$ $first_k(y) \notin FIRST_k(\gamma_j FOLLOW_k(A))$, то конфигурация C_i регистрирует *синтаксическую ошибку*.

Характеристические свойства LL(k)- и SLL(k)-грамматик гарантируют детерминированность k-предсказывающего правила.

4.4. Управляющая таблица *SLL(k)*-анализатора

Если для грамматики имеет место характеристическое свойство *LL*(*k*)-грамматики, тогда можно проверить характеристическое свойство *SLL*(*k*)-грамматики. Если последнее имеет место, это позволяет *оптимизировать* представление функции управления поведением анализатора. В случае $k = 1$, что на практике реально встречается, классы *LL*(1)- и *SLL*(1)-грамматик совпадают.

Команды *SLL*(*k*)-анализатора имеют совершенно простой вид.

Определение. Пусть p_j: $A \to \gamma_j$, $j \in [1, n_j]$, – альтернатива нетерминала A и $q = FOLLOW_k(A)$. Множество $FIRST_k(\gamma_j\, q)$ называется управляющим множеством альтернативы p_j: $A \to \gamma_j$ при *k*-контексте q.

Для каждой альтернативы p_j: $A \to \gamma_j$, $j \in [1, n_j]$ и множества $q = FOLLOW_k(A)$ всех правых *k*-контекстов нетерминала A команда *SLL*(*k*) - анализатора принимает форму (*A*-команда)

$$p_j: A \to q_j, \gamma_j, \text{ где } q_j = FIRST_k(\gamma_j\, q).$$

Управляющая таблица SLL(k)-анализатора есть представление функции его поведения на конфигурациях (*не модифицированных*):

1. $M(A, u) = p_j$, если $u \in q_j$ и p_j: $A \to q_j, \gamma_j$ – *SLL*(*k*)-команда альтернативы нетерминала *A*;
2. $M(A, u) = Error$, если условие пункта 1 не выполнено при аванцепочке u для всех *SLL*(*k*)-команд p_j: $A \to q_j, \gamma_j$ нетерминала *A*;
3. $M(a, u)$ = сдвиг, если $a = first_1(u)$;
4. $M(a, u) = Error$, если $a \neq first_1(u)$;
5. $M(\lambda, \lambda)$ = «допуск»;
6. $M(\lambda, u)$ $Error$, $u \neq \lambda$.

4.5. Проектирование *LL*(1)-анализатора

Пусть $G=(N, \Sigma, P, S)$ – КС-грамматика. Ниже описываются действия, которые необходимо выполнить при разработке алгоритма грамматического разбора *LL*(1)-грамматики.

Шаг 1. Применить эквивалентные преобразования:

а) исключить бесполезные и недостижимые символы;

б) устранить левую рекурсию (обычно достаточно устранения непосредственной левой рекурсии);

в) применить левую факторизацию.

Замечание. В общем случае необходимы и другие преобразования, в крайнем – потребуется переопределить входную грамматику вообще или фрагментарно.

Шаг 2. Вычислить значения функции $FIRST_k(X)$ для всех нетерминалов, $k=1$

Шаг 3. Построить список команд *SLL*(*k*)-анализатора, предварительно вычислив значения функции $FOLLOW_k(X)$ для всех нетерминалов, $k=1$.

Шаг 4. Проверить выполнимость характеристического свойства *SLL*(*k*)-грамматики, $k=1$

Если тест для некоторой группы A - команд дал отрицательный ответ (A – нетерминал), остановиться и, возможно, переопределить заново входную грамматику.

Шаг 5. Построить управляющую таблицу *SLL*(*k*)-анализатора, $k=1$, если тест на шаге 4 положителен**.**

Пример.

1: $E \rightarrow E+T$

2: E $\rightarrow E-T$

3: $E \rightarrow T$

4: $T \rightarrow a$

5: $T \rightarrow i$

6: $T \rightarrow (E)$

Аксиома – символ E;

Шаг 1. Применить эквивалентные преобразования

Все символы полезны и достижимы, необходимо устранить непосредственную левую рекурсию для E и T.

Получаем эквивалентную не леворекурсивную грамматику:

1: $E \rightarrow TR$

2: $R \rightarrow \lambda$

3: $R \rightarrow +TR$

4: $R \rightarrow -TR$

5: $T \rightarrow a$

6: $T \rightarrow i$

7: $T \rightarrow (E)$

Шаг 2. Вычислить значения функции $FIRST_k(X)$ для всех нетерминалов, k=1

Сначала выпишем систему определяющих уравнений:

$E = TR$

$R = \lambda | +TR | -TR$

$T = a | i | (E)$

Выпишем рекуррентную систему равенств:

$\underline{E}^{i+1} = \underline{T}^i \oplus_k \underline{R}^i | \underline{E}^i$

$\underline{R}^{i+1} = \lambda | + \oplus_k \underline{T}^i \oplus_k \underline{R}^i | - \oplus_k \underline{T}^i \oplus_k \underline{R}^i | \underline{R}^i$

$\underline{T}^{i+1} = a | i | (\oplus_k \underline{E}^i \oplus_k) | \underline{T}^i$

$\underline{E}^0 = \underline{R}^0 = \underline{T}^0 = \varnothing$

Для упрощения записи можно опустить символ $\oplus_k$, по умолчанию полагая его присутствие**.**

Проведя итерации, получим сходящуюся последовательность векторов ($\underline{E}^i$, $\underline{R}^i$, $\underline{T}^i$).

Компоненты векторов представлены регулярными выражениями.

$(\varnothing , \varnothing, \varnothing)$

$(\varnothing ,\lambda , a | i)$

$(a | i , \lambda|+| - , a | i)$

$(a | i , \lambda|+|- , a | i | ( )$

$(a | i | (, \lambda|+|- , a | i | ( )$

$(a | i | (, \lambda|+|- , a | i | ( )$

На пятом шаге итерации достигается предел последовательности.

$(FIRST_1 (E), FIRST_1(R), FIRST_1 (T)) = (a | i | (,\lambda|+|- , a | i | ( )$

$\underline{E} = FIRST_1 (E) = a | i | ($

$\underline{R} = FIRST_1 (R) = \lambda | +| -$

$\underline{T} = FIRST_1 (T) = a | i | ($

Шаг 3. Построить список команд $SLL(k)$-анализатора, предварительно вычислив значения функции $FOLLOW_k(X)$ для всех нетерминалов, k=1.

Вычислить $FOLLOW_1(A)$, $A \in N$:

$FOLLOW_1 (E) = \lambda|)$

$FOLLOW_1 (R) = \lambda|)$

$FOLLOW_1 (T) = \lambda| +| - |)$

Вычислить для каждого нетерминала $A \in N$ и его альтернативы $A \to \gamma$ выражение $FIRST_k(\gamma FOLLOW_1(A))$ – управляющее множество аванцепочек альтернативы $A \to \gamma$:

1: $E \to g_1, TR$ $\quad g_1 = a|i|($
2: $R \to g_2, \ \lambda$ $\quad g_2 = \lambda|)$
3: $R \to g_3, \ +TR$ $\quad g_3 = +$
4: $R \to g_4, \ -TR$ $\quad g_4 = -$
5: $T \to g_5, a$ $\quad g_5 = a$
6: $T \to g_6, i$ $\quad g_6 = i$
7: $T \to g_7, (E)$ $\quad g_7 = ($

Очевидно, грамматика является *SLL*(1)-грамматикой и следовательно *LL*(1)-грамматикой.

5. *LR*(*k*)-ГРАММАТИКИ

5.1. Восходящий синтаксический анализ

Существует ряд методов разработки алгоритмов грамматического разбора, реализующих *восходящий синтаксический анализ*, называемый еще анализом типа «*снизу вверх*» (*bottom up*). Свое название он получил из-за принципа построения дерева вывода, которое строится *снизу вверх* – от листьев к корню.

Существуют *универсальные алгоритмы разбора*, применимые ко всему классу КС-языков: *алгоритм Эрли*; *алгоритм Кока-Янгера-Касами*. Временная сложность этих алгоритмов порядка $O(n^3)$ в общем случае – неприемлемо высокая, чтобы применять их на практике. Хотя алгоритм Эрли в классе однозначных грамматик имеет сложность порядка $O(n^2)$, однако для детерминированных языков применяются эффективные алгоритмы с линейной сложностью порядка $O(n)$, где n – длина входа (цепочки символов во входном алфавите).

Этот класс языков определяется *LR*(*k*)-грамматиками, которые образуют наиболее широкий класс грамматик, анализируемых естественным образом *снизу вверх* с помощью *детерминированного магазинного последовательностного автомата (ДМПА)*.

Входом LR(*k*)-анализатора является *анализируемая цепочка* терминалов, которая читается последовательно слева направо (без возвратов), что соответствует букве *L* (*left*).

Выходом является *правый вывод*, что соответствует букве *R* (*right*). На самом деле выходом является *инвертированный правый вывод*, который в точности соответствует *стратегии снизу вверх* построения *синтаксиче-*

ского дерева. Каждая *LL*(*k*)-грамматика является *LR*(*k*)-грамматикой. Класс *LL*(*k*)-грамматик определяет собственное подмножество класса *LR*(*k*)-грамматик. *LL*(*k*)-анализаторы имеют более компактное представление в памяти компьютера, нежели *LR*(*k*)- анализаторы.

Наряду с *LL*(*k*)-грамматиками существуют также другие подклассы *LR*(*k*)-грамматик, в том числе *грамматики предшествования* и *ограниченного правого контекста*, со своими специальными методами анализа. Однако общим для всех методов является стратегия поиска *основы* γ – правой части правила $A \to \alpha$, заменяемой нетерминалом *A*.

Определение. Пусть $G = (N, \Sigma, P, S)$ – КС-грамматика и $S \Rightarrow_r^* \alpha Ax \Rightarrow \alpha\gamma x \Rightarrow_r^* \omega x$ – правый вывод в *G*, $\omega x \in \Sigma^*$. Пусть ω – проанализированная слева направо, а *x* – не проанализированная части входной цепочки ωx; $\alpha\gamma x \Rightarrow^* \omega x$ – частично восстановленный правый вывод. Тогда подцепочка γ называется *основой* цепочки $\alpha\gamma x$. И, говорят, что $\alpha\gamma x$ *левосвертывается* или *редуцируется* с помощью правила $A \to \gamma$ к *правовыводимой* цепочке αAx.

Представим операцию *свертки* с помощью правила $A \to \gamma$ (*свертка* $A \to \gamma$) как переход $C_i \mapsto C_{i+1}$, $i \geq 0$, магазинного автомата, где $C_i = (\alpha\gamma, \omega\bullet x, \rho)$ и $C_{i+1} = (\alpha A, \omega\bullet x, \rho p)$ – конфигурации автомата; *p*: $A \to \gamma$ – правило под номером *p*; символ '•' указывает на текущий входной символ непроанализированной части входа; $\alpha\gamma$ и αA – состояния магазина, причем верхушка магазина находится справа, дно – слева; ρp – инвертированный правый вывод $\alpha Ax \Rightarrow \alpha\gamma x \Rightarrow^* \omega x$.

У начальной конфигурации $C_0 = (\lambda, \bullet\omega x, \lambda)$ магазин пуст и выходная последовательность правил также пуста.

Заключительная конфигурация $C_i = (S, \omega x\bullet, \rho)$, достижимая при некотором $i \geq 0$, сигнализирует, что для входной цепочки ωx построен инвертированный правый вывод ρ; аксиома *S* определяет состояние магазина, а указатель входного символа находится в конце цепочки ωx, причем $x = \lambda$ (исчерпан вход ωx).

Определение. Пусть αx – правовыводимая цепочка, причем либо α– пустая цепочка, либо оканчивающаяся нетерминалом. Тогда α называется *открытой частью* цепочки αx, а подцепочка *x* – *замкнутой частью*.

Состояние магазина начальной конфигурации или конфигурации, полученной после *свертки*, определяют *открытую часть,* а подцепочка правее указателя '•' – *замкнутую часть*.

Ключевая проблема, которая решается алгоритмом разбора типа «снизу вверх» заключается в обнаружении состояния магазина текущей конфигурации, определяющего *основу*.

Чтобы найти *основу*, необходимо найти *правый* и *левый концы основы*. Чтобы обнаружить *правый конец основы* может потребоваться *считать и перенести* в магазин несколько входных символов.

Таким образом, алгоритм разбора принимает следующие решения:

во-первых, *переносить* ли *входной символ* в магазин или делать *свертку*;

во-вторых, если обнаружен *правый конец основы* (принято решение делать *свертку*), то, как *локализовать* в магазине *левый конец основы*;

в третьих, если выделена *основа*, то каким *нетерминалом* ее заменить, чтобы выполнить *свертку*.

Алгоритмы разбора, основанные на принятии решений подобного типа, называют *алгоритмами типа «перенос-свертка»*.

Д. Кнут определил широкий класс грамматик – $LR(k)$-грамматик, для которых всегда можно построить *детерминированные алгоритмы* грамматического разбора типа *«перенос-свертка»*.

5.2. Активные префиксы грамматики

Пусть $G = (N, \Sigma, P, S)$ – КС-грамматика, $G' = (N', \Sigma, P', S')$ – пополненная новой аксиомой S' и правилом $S' \to S$. Правила из множества P нумеруются от 1 до $|P|$, новое правило получает номер 0.

Определение. Пусть $S' \Rightarrow^* \sigma' A x$ – правосторонний вывод в G', $A \to \alpha \in P$, $\alpha = \gamma\beta$. Тогда слова $\sigma' A$ и $\sigma'\gamma$ называются *активными префиксами* грамматики G.

Обозначим через R – множество всех активных префиксов. Очевидно, $R \subseteq V^*$, $V = N \cup \Sigma$.

Д. Кнут показал, что R – регулярное множество и дал конструктивный метод построения ДКА $M = (Q, V, g, q_0 , F)$, допускающего язык $R = L(M)$.

Пусть $\sigma \in R$ – произвольный активный префикс. Тогда найдутся слово $w \in L(G)=L(G')$, правосторонние выводы $S' \Rightarrow^* \sigma x \Rightarrow^* \omega x = w$ и $\sigma \Rightarrow^* \omega$.

Пусть π – последовательность номеров правил правостороннего вывода $S' \Rightarrow^* w$. Тогда $\pi = \pi_1 \pi_2$, где вывод $S' \Rightarrow^* \sigma x$ определяется подпоследовательностью π_1, вывод $\sigma x \Rightarrow^* \omega x = w$ $(\sigma \Rightarrow^* \omega)$ – подпоследовательностью π_2.

Отношению вывода $\Rightarrow*$ соответствует обратное отношение $*\Leftarrow$, называемое отношением *сводимости* или *редукции*.

Запись $w = \omega x *\Leftarrow \sigma x *\Leftarrow S'$ и $\omega *\Leftarrow \sigma$ означает, что сначала ω сводится к активному префиксу σ, затем σx редуцируется к аксиоме S'. На первом этапе применяется подпоследовательностью π_2', на втором – π_1'. Редукция слова $w *\Leftarrow S'$ определяется последовательностью $\pi' = \pi_2'\pi_1'$ – инверсией π.

Последовательность π' называется инвертированным правосторонним выводом слова w.

Обход дерева вывода по маршруту π' соответствует конструированию узлов дерева вывода от листьев к корню. В этом случае говорят, что метод грамматического разбора согласуется с *восходящей стратегией* или *стратегией грамматического разбора снизу-вверх* (*bottom-up*).

Д. Кнут определил класс $LR(k)$-грамматик и эффективный метод разбора *снизу-вверх*, основанный на конструкции ДКА, допускающего язык R активных префиксов грамматики. Входное слово w анализируется слева направо, проанализированный префикс ω входного слова w редуцируется к активному префиксу σ путем применения последовательности правил π_2'. Слово x – не проанализированный суффикс слова w.

Основа и редукция основы

Определение. Пусть $\sigma \in R$ – активный префикс, $A \to \alpha \in P$. Слово α называется *основой* σ, если существует правосторонний вывод $S' \Rightarrow *\sigma' Ax$ и $\sigma = \sigma'\alpha$. Другими словами, если $\sigma = \sigma'\alpha$ и $\sigma' A$ – активные префиксы для правила $A \to \alpha$. В этом случае префикс σ непосредственно редуцируется к префиксу $\sigma' A$ по правилу $A \to \alpha$.

Однако может существовать и другая редукция: $\sigma = \sigma''\beta$ редуцируется к $\sigma'' B$ по правилу $B \to \beta$.

Д. Кнут определил условия, при которых основа и, следовательно, редукция определяются детерминировано (однозначно).

Из определения *основы* и активного префикса σ следует, что существует правосторонний вывод $S' \Rightarrow *\sigma' Ax \Rightarrow \sigma x = \sigma'\alpha x$. Анализатору доступен активный префикс σ и аванцепочка $u = first_k(x)$, где x – не проанализированный суффикс входного слова.

При фиксированном $k \geq 0$ $u \in FOLLOW_k(A)$. Тогда пара $(A \to \alpha\bullet, u)$ называется *допустимой* $LR(k)$-*ситуацией типа свертки* (*редукции*) для активного префикса σ и ограниченного правого контекста u нетерминала A.

5.3. Конструирование ДКА $M = (Q, V, g, q_0, F)$

Для любого $\sigma \in R$ и любого разложения $\sigma = \sigma'\tau$ выполняется $\sigma' \in R$. Другими словами любой префикс активного префикса грамматики является активным префиксом грамматики.

Утверждение следует из определения правостороннего вывода и активного префикса грамматики. В таком случае скажем, что язык R обладает свойством *префиксности*. В частности, пустое слово λ – активный префикс, так как $S' \Rightarrow S \Rightarrow^* w \in L(G)$.

Пусть $M = (Q, V, g, q_0, F)$ – детерминированный конечный автомат, допускающий язык активных префиксов R. Тогда $q_0 \in F$. В силу свойства *префиксности* $Q = F \cup \{q\}$ и $q \notin F$.

5.3.1. $LR(k)$-ситуации, $k \geq 0$

Определение. Пусть $A \to \alpha \in P$, $\alpha = \gamma\beta$, $u \in FOLLOW_k(A)$.
Пара $(A \to \gamma \bullet \beta, u)$ называется $LR(k)$-ситуацией. Через K будем обозначать множество всех $LR(k)$-ситуацией грамматики G'.

Определение. Пусть $\sigma \in R$ – активный префикс. $LR(k)$-ситуация $(A \to \gamma \bullet \beta, u)$ называется допустимой для σ, если $\sigma = \sigma'\gamma$ и существует правосторонний вывод $S' \Rightarrow^* \sigma' A x \Rightarrow \sigma'\gamma\beta x$ и $u = first_k(x)$.

Через $q(\sigma)$ обозначим множество всех допустимых $LR(k)$-ситуаций для активного префикса σ.

Д. Кнут определил множество состояний $F = \{q(\sigma) \mid \sigma \in R\}$, начальное состояние $q_0 = q(\lambda)$. Функция переходов автомата M определяется индукцией по длине активного префикса следующим образом: пусть $\sigma = \sigma' X$, $X \in V$, тогда

$$g(q(\sigma'), X) = q(\sigma).$$

Д. Кнут дал конструктивный метод вычисления начального состояния и индуктивный метод вычисления всех состояний и определения функции переходов $g : Q \times V \to Q$.

5.3.2. Замкнутые множества $LR(k)$-ситуаций

Пусть $K = \{(A \to \gamma \bullet \beta, u) \mid A \to \alpha$ – произвольное правило грамматики, $\alpha = \gamma\beta$, $u \in FOLLOW_k(A)\}$ – множество всех $LR(k)$-ситуаций.

Определение. Пусть $H \subseteq K$. Подмножество H называется *замкнутым*, если из того, что $(A \to \gamma \bullet \beta, u) \in H$ и $\beta = B\beta'$ следует,

что $(B \to \bullet\delta, v) \in H$ для всех альтернатив $B \to \delta$ нетерминала B и всех слов $v \in FIRST_k(\beta' u)$.

Заметим, что $FIRST_k(\beta' u) \subseteq FOLLOW_k(B)$.

Определение. Замыканием подмножества $H \subseteq K$ называется наименьшее по включению замкнутое подмножество $LR(k)$ – ситуацией, включающее H.

Замыкание подмножества H существует, единственно и обозначается через $[H]$, где выражение $[H]$ – результат применения оператора замыкания [] к H.

Выражение $[H]$ вычисляется конструктивно, следуя рекурсивному определению замкнутого множества.

Функция перемещения $\mu(H, X)$, $H \subseteq K$, $X \in V$

Определение. Пусть $H \subseteq K$ и $X \in (N \cup \Sigma)$. Тогда значение функции перемещения $\mu(H, X) = H' \subseteq K$ определяется следующим образом:

$$\mu(H, X) = H' = \{(A \to \gamma X \bullet \beta, u) \mid (A \to \gamma \bullet X\beta, u) \in H\}.$$

Посредством функции *перемещения* $\mu(H, X)$ и оператора *замыкания* $[H]$ множество состояний Q и функция переходов g автомата M, допускающего язык R активных префиксов грамматики, определяется конструктивно по индукции:

a) вычислить начальное состояние $q_0 = [(S' \to \bullet S, \lambda)]$;

b) пусть $q \in Q$ и $q = [q]$ – замкнутое подмножество $LR(k)$-ситуаций, $X \in (N \cup \Sigma)$, тогда $g(q, X) = [\mu(q, X)]$.

Таким образом, все состояния суть замкнутые множества и любое финальное состояние не является пустым множеством $LR(k)$-ситуаций. Пустое множество $LR(k)$-ситуаций – не финальное состояние, следовательно, $Q = F \cup \{\varnothing\}$.

5.4. *LR(k)*-грамматики и функция действия

5.4.1. Выполнимость *LR(k)*-условий

Определение. Пусть $H \subseteq K$ – произвольное замкнутое множество *LR(k)-ситуаций*: $H = [H]$. Множество H называется *противоречивым,* если содержит в себе хотя бы одну пару *различных* элементов одного из двух типов:

а)«свертка, свертка» – $(A \to \gamma \bullet, u)$, $(B \to \eta \bullet, u)$;

б)«свертка, перенос» – $(A \to \gamma \bullet, u)$, $(B \to \beta \bullet a\delta, v)$, причем $u \in FIRST_k(a\delta\, v)$.

В противном случае, множество H называется *непротиворечивым*.

Говорят, что для исходной КС-грамматики G выполняются *LR(k)-условия* или, что грамматика G является *LR(k)-грамматикой,* если для любого активного префикса σ грамматики множество его допустимых *LR(k)-ситуаций* $q(\sigma)$ непротиворечиво либо, что $q(\sigma)$ свободно от коллизий типа «свертка, свертка» и «свертка, перенос».

Тест на выполнимость LR(k)-условий очевиден: построить ДКА $M = (Q, N \cup \Sigma, g, q_0, F)$, что фактически сводится к построению множества его состояний; проверить каждое состояние $q \in Q$ на *непротиворечивость* (отсутствие *коллизий*). Если все состояния непротиворечивы, то грамматика является *LR(k)-грамматикой.*

5.4.2. Определение функции действия

Пусть все состояния множества Q *непротиворечивы*. Тогда на множестве $Q \times \Sigma^{*k}$ можно определить функцию «действия» $f(q, u)$:

$f(q, u)$ = «*свертка*: p», если $(A \to \gamma\bullet, u) \in q$ и p: $A \to \gamma$ − правило грамматики с номером $p > 0$;

$f(q, u)$ = «*допуск*», если $(S' \to S\bullet, \lambda) \in q$, «*допуск*» равносилен значению «*свертка*: 0»;

$f(q, u)$ = «*перенос*», если $(A \to \beta\bullet a\delta, v) \in q$ и $u \in FIRST_k(a\delta\, v)$;

$f(q, u)$ = «*ошибка*» во всех остальных случаях.

Множество $\Sigma^{*k} = FIRST_k(\Sigma^*)$ состоит из всех цепочек $x \in \Sigma^*$ таких, что $|x| \le k$.

Свойства замкнутых множеств

1. $[H_1 \cup H_2] = [H_1] \cup [H_2]$

2. Пусть $h = (A \to \gamma\bullet\beta, u)$ и $H = \{h\}$, тогда $[h]$ обозначает замыкание одноэлементного множества : $[h] = [H] = [\{h\}]$.

3. Пусть $H = \{h_1, \ldots, h_n\}$, тогда $[H] = [h_1] \cup \ldots \cup [h_n]$

4. $[\varnothing] = \varnothing$

5. Пусть $U = \{u_1, u_2, \ldots, u_n\} \subseteq FOLLOW_k(A)$ и $(A \to \gamma\bullet\beta, u_j) \in H$, $j \in [1, n]$. Тогда $(A \to \gamma\bullet\beta, U)$ или $(A \to \gamma\bullet\beta, u_1 \mid u_2 \mid \ldots \mid u_n)$ представляет множество $\{(A \to \gamma\bullet\beta, u_j) \mid j \in [1, n]\}$ $LR(k)$-ситуаций и называется $LR(k)$-*комплексом*. Любое множество H однозначно представимо в виде множества $LR(k)$-*комплексов.*

6. Определения *замкнутых* множеств и *замыкания* имеют силу на языке $LR(k)$-*комплексов* и они равносильны соответствующим определениям на языке $LR(k)$-ситуаций.

7. Определения функции переходов g и функции действия f на языке $LR(k)$-*комплексов* равносильны соответствующим определениям на языке $LR(k)$-ситуаций.

5.4.3. Конфигурации и управляющая таблица $LR(k)$-анализатора

Управляющая таблица $LR(k)$-анализатора — это представление в памяти ЭВМ множества значений функции «действия» f и функции «переходов» g для каждого q в виде одномерной таблицы T_q. Клетки первой части T_q индексируются *аванцепочками* $u \in \Sigma^{*k}$ так, что $T_q[u] = f(q, u)$.

Клетки второй части T_q индексируются символами грамматики $X \in (N \cup \Sigma)$ так, что $T_q[X] = g(q, X)$.

Совокупность всех таблиц T_q можно представить в виде одной *двумерной таблицы* T, причем $T[q, u] = f(q, u)$ и $T[q, X] = g(q, X)$.

Наиболее компактное представление достигается при k=1. Практическое применение нашли только $LR(1)$-*анализаторы*.

Д.Кнут показал, что класс $LR(k)$-языков при фиксированном $k \geq 0$ порождается $LR(1)$-грамматиками. Однако проблема распознавания свойства «быть $LR(k)$-грамматикой при некотором $k \geq 0$» в классе контекстно-свободных грамматик алгоритмически неразрешима.

5.4.4. Конфигурации $LR(k)$-анализатора

$LR(k)$-анализатор – это детерминированный магазинный последовательностный автомат (ДМПА) с выходом (преобразователь). Если $w \in L(G)$, то выходом является ρ – инвертированный правосторонний вывод слова w. Если $w \notin L(G)$, то выходом является ρ_i – частичный инвертированный правосторонний вывод слова $w = \omega \bullet x$, σ_i – активный префикс, ρ_i' – правосторонний вывод $\sigma_i \Rightarrow^* \omega$ проанализированного префикса ω и сигнал о синтаксической ошибке.

Поведение ДМПА определяется переходами $C_i \rightarrow C_{i+1}$, где C_i – текущая конфигурация ДМПА, $i \geq 0$.

Текущая конфигурация $C_i = (M, S, \rho_i)$ соответствует текущему активному префиксу σ_i , к которому *редуцирован* проанализированный префикс ω входного слова $w = \omega x$: $\omega \bullet x \ *\Leftarrow \sigma_i \bullet x$.

$S = \omega \bullet x$; ρ_i' – правосторонний вывод $\sigma_i \Rightarrow^* \omega$, следовательно $\rho_i = \rho_i''$;

$M = q_0\ \sigma_i\ q_t$ – путь ДКА на активном префиксе σ_i от начального состояния q_0 к состоянию $q_t = g^*(q_0, \sigma_i)$, сохраненный в магазине. Состоя-

ние q_0 – на дне магазина, состояние q_t – на вершине магазина, причем $|M| = 2|\sigma_i| + 1$.

Начальная конфигурация имеет вид: $C_0 = (q_0, \bullet w, \lambda)$, где q_0 – начальное состояние, w – входная анализируемая цепочка терминалов.

Пусть $C_i = (q_0X_1q_1 \ldots q_{t-1}X_t\, q_t, \omega \bullet x, \rho_i)$ – текущая конфигурация при $i \geq 0$ и пусть $u = first_k(x)$.

Анализатор сначала применяет к C_i функцию действия f.

Если $T[q_t, u]$ =«*допуск*», то

$$C_i = (q_0X_1q_1 \ldots q_{t-1}X_t\, q_t, \omega \bullet x, \rho_i) = (q_0Sq_1, w \bullet, \rho_i),$$

следовательно, $\omega \bullet x = w \bullet$, $u = \lambda$, $t = 1$ и ρ_i – *инвертированный* правый вывод $S \Rightarrow^* w$ в исходной грамматике G.

Если $T[q_t, u]$ = «*свертка*: p» и p: $A \rightarrow \alpha$ — правило грамматики с номером p, тогда в магазине гарантированно обнаружена *основа* α, из магазина выталкиваются $2|\alpha|$ элементов (при пустой *основе* ничего не выталкивается) : символы *основы* и промежуточные состояния выталкиваются из магазина; затем вталкивается в магазин нетерминал A. Преобразованная промежуточная конфигурация имеет представление

$$C_i' = (q_0X_1q_1 \ldots q_{d-1}A, \omega \bullet x, \rho_i\, p),\ d = t - |\alpha|.$$

К конфигурации C_i' применяется функция переходов g.

Пусть $q_d = g(q_{d-1}, A)$. Если $q_d \neq \varnothing$, то переходная конфигурация имеет вид:

$$C_{i+1} = (q_0X_1q_1 \ldots q_{d-1}A\, q_d, \omega \bullet x, \rho_i\, p).$$

Если $q_d = \varnothing$, то выдается сигнал об *ошибке (синтаксической)*. В простейшем случае *LR(k)-анализатор* завершает работу.

Если $T[q_t, u]$ = «*перенос*», $x = az$, $u = first_k(x)$. Тогда анализатор *переносит* (копирует) входной символ a в магазин и сдвигает на один символ вправо указатель входного символа.

Преобразованная конфигурация имеет представление

$$C_i' = (q_0X_1q_1 \ldots q_da, \omega a \bullet z, \rho_i).$$

К конфигурации C_i' применяется функция переходов g.

Пусть $q_{d+1} = g(q_d, a)$. Если $q_{d+1} \neq \varnothing$, тогда переходная конфигурация имеет вид:

$$C_{i+1} = (q_0X_1q_1 \ldots q_da\, q_{d+1}, \omega a \bullet z, \rho_i).$$

Если $q_{d+1} = \varnothing$, то выдается сигнал об *ошибке (синтаксической)*. В простейшем случае *LR(k)-анализатор* завершает работу.

Если $T[q_t, u]$ = «*ошибка*», то выдается сигнал об *ошибке (синтаксической)*. В простейшем случае *LR(k)-анализатор* завершает работу.

Восстановление после обнаружения ошибки

В реальных компиляторах после обнаружения *синтаксической ошибки* выполняется процедура *восстановления.* Ее смысл заключается в изменении текущей конфигурации с помощью: а) либо изменения состояния магазина; б) либо замены входного символа; в) либо изменения положения указателя входного символа; г) либо применения некоторой комбинации вышеуказанных преобразований для того, чтобы перейти к неошибочной конфигурации. Цель *восстановления* – локализовать ошибку и продолжить разбор, как если бы ошибки не было.

Вполне возможна ситуация, когда преобразования приводят вновь к регистрации ошибки. В этом случае наблюдается эффект *цепной реакции*: генерации серии ошибок, причиной которых явилась только первая, реальная в этой серии.

5.5. Проектирование *LR*(*k*)-анализатора

Пусть $G=(N, \Sigma, P, S)$ – КС-грамматика. Ниже описываются действия, которые необходимо выполнить при разработке алгоритма грамматического разбора *LR*(*k*)-грамматики. Порядок имеет принципиальное значение.

Список операций или преобразований минимально необходим.

Шаг 1. Применить эквивалентные преобразования:

а) исключить бесполезные и недостижимые символы;

Замечание. В крайнем случае, когда грамматика не проходит тест на выполнимость *LR(k) – условий*, потребуется переопределить ее вообще или фрагментарно.

Шаг 2. Пополнить грамматику *G.*

Пусть $G'=(N', \Sigma, P', S')$ – пополнение грамматики *G.*

Шаг 3. Вычислить значения функции $FIRST_k(X)$ для всех нетерминалов.

Шаг 4. Вычислить для пустого активного префикса λ множество допустимых *LR(k) – ситуаций.*

На языке автоматных состояний – это означает, необходимо вычислить начальное состояние q_0 детерминированного конечного автомата ДКА = $(Q, N \cup \Sigma, g, q_0, Q\backslash\{\varnothing\})$, используя формулу

$$q_0 = [(S' \to \bullet S, \lambda)].$$

Шаг 5. Вычислить индуктивно множество состояний Q и определить функцию переходов g: $Q \times (N \cup \Sigma) \to Q$.

Шаг 6. Подвергнуть множество состояний Q тесту на *выполнимость LR(k)-условий.*

Проверить условие противоречивости для всех $q \in Q$. Если все состояния непротиворечивы, заключаем, исходная грамматика G является *LR(k)-грамматикой.*

Тогда переходим к определению функции действия f. В противном случае дальнейшие построения невозможны. Увеличение параметра k на 1 не практично и не эффективно.

Реальное решение – переопределить исходную грамматику.

Шаг 7. Определить функцию действия f: $Q \times \Sigma^{*k} \to A$.

Область значений

A = {«допуск», «перенос», «ошибка»} $\cup$ {«свертка: p»| $p \in [0, r]$},

где r – число правил исходной грамматики G.

Шаг 8. Построить управляющую таблицу *LR(k)-анализатора.*

Пример

1: $S \to SaSb$

2: $S \to \lambda$

Эта грамматика леворекурсивная и, поэтому не является $LL(k)$-грамматикой при любом $k \geq 0$. Но она является $LR(1)$-грамматикой.

Шаг 1. Применить эквивалентные преобразования.

Грамматика не содержит бесполезных и недостижимых символов, поэтому используется, как есть.

Шаг 2. Пополнить грамматику G.

G'= (N', Σ, P', S') – пополнение грамматики G:

0:$S' \to S$

1: $S \to SaSb$

2: $S \to \lambda$

Шаг 3. Вычислить значения функции $FIRST_k(X)$ для всех нетерминалов.

Так как k = 1 и грамматика простая, непосредственно вычисляем

$FIRST_1(S') = \lambda | a$

$FIRST_1(S) = \lambda | a$

Шаг 4. Вычислить для пустого активного префикса λ множество допустимых *LR(k) – ситуаций.*

$$q_0 = [(S' \to \bullet S, \lambda)] =$$

$$= \{(S' \to \bullet S, \lambda); (S \to \bullet SaSb, \lambda); (S \to \bullet, \lambda); (S \to \bullet SaSb, a); (S \to \bullet, a)\} =$$

$$= \{(S' \to \bullet S, \lambda); (S \to \bullet SaSb, \lambda | a); (S \to \bullet, \lambda | a)\}.$$

Шаг 5. Вычислить индуктивно множество состояний Q и определить функцию переходов g: $Q \times (N \cup \Sigma) \to Q$.

0. Множество «*открытых*» состояний $O = \{q_0\}$; множество «*закрытых*» состояний $C = \{\}$—пустое.

1. Исключаем q_0 из O и включаем в ***C***.

Получаем $O = \{\}$, $C = \{q_0\}$.

Вычисляем последовательно

$$g(q_0, S) = [\mu(q_0, S)] =$$
$$= [\{(S' \rightarrow S \bullet, \lambda); (S \rightarrow S \bullet aSb, \lambda|a)\}] =$$
$$= \{(S' \rightarrow S \bullet, \lambda); (S \rightarrow S \bullet aSb, \lambda|a)\} = q_1;$$
$$g(q_0, a) = [\mu(q_0, a)] = \varnothing;$$
$$g(q_0, b) = [\mu(q_0, b)] = \varnothing.$$

Получаем $O = \{q_1, \varnothing\}$, $C = \{q_0\}$, так как q_1, $\varnothing$— новые состояния.

2. Исключаем q_1 из O и включаем в C.

Получаем $O = \{\varnothing\}$, $C = \{q_0, q_1\}$.

Вычисляем последовательно

$$g(q_1, S) = [\mu(q_1, S)] = [\{\}] = \varnothing;$$
$$g(q_1, a) = [\mu(q_1, a)] = [\{(S \rightarrow Sa \bullet Sb, \lambda|a)\}] =$$
$$= [(S \rightarrow Sa \bullet Sb, \lambda|a)] =$$
$$= \{(S \rightarrow Sa \bullet Sb, \lambda|a); (S \rightarrow \bullet SaSb, b); (S \rightarrow \bullet, b); (S \rightarrow \bullet SaSb, a); (S \rightarrow \bullet, a)\} =$$
$$= \{(S \rightarrow Sa \bullet Sb, \lambda|a); (S \rightarrow \bullet SaSb, a|b); (S \rightarrow \bullet, a|b)\} = q_2$$
$$g(q_1, b) = [\mu(q_1, b)] = \varnothing;$$

Получаем $\boldsymbol{O} = \{\varnothing, q_2\}$, $C = \{q_0, q_1\}$,так как q_2— новое состояние.

3. Исключаем $\varnothing$из O и включаем в C.

Получаем $O = \{q_2\}$, $C = \{q_0, q_1, \varnothing\}$.

Вычисляем последовательно

$$g(\varnothing, S) = [\mu(\varnothing, S)] = [\varnothing] = \varnothing$$
$$g(\varnothing, a) = [\mu(\varnothing, a)] = [\varnothing] = \varnothing$$
$$g(\varnothing, b) = [\mu(\varnothing, b)] = [\varnothing] = \varnothing$$

Получаем $O = \{q_2\}$, $C = \{q_0, q_1, \varnothing\}$.

4. Исключаем q_2 из O и включаем в C.

Получаем $O = \{\}$, $C = \{q_0, q_1, \varnothing, q_2\}$.

Вычисляем последовательно

$$g(q_2, S) = [\mu(q_2, S)] =$$
$$= [\{(S \rightarrow SaS \bullet b, \lambda|a); (S \rightarrow S \bullet aSb, a|b)\}] =$$
$$= \{(S \rightarrow SaS \bullet b, \lambda|a); (S \rightarrow S \bullet aSb, a|b)\} = q_3$$
$$g(q_2, a) = [\mu(q_2, a)] = [\{\}] = \varnothing$$
$$g(q_2, b) = [\mu(q_2, b)] = [\{\}] = \varnothing$$

Получаем $O = \{q_3\}$, $C = \{q_0, q_1, \varnothing, q_2\}$.

5. Исключаем q_3 из O и включаем в C.

Получаем O ={}, C ={q_0, q_1, ∅, q_2, q_3}.

Вычисляем последовательно

$$g(q_3, S) = [\mu\ (q_3, S)]= [\{\}]= \varnothing$$

$$g(q_3, a) = [\mu\ (q_3, a)]= [\{(S\rightarrow Sa\bullet Sb,\ a|b)\}]=$$

$$= [\{(S\rightarrow Sa\bullet Sb,\ a|b);\ (S\rightarrow \bullet SaSb,\ b);\ (S\rightarrow \bullet,\ b);\ (S\rightarrow \bullet SaSb,\ a);\ (S\rightarrow \bullet,\ a)\}]=$$

$$= \{(S\rightarrow Sa\bullet Sb,\ a|b);\ (S\rightarrow \bullet SaSb,\ a|b);\ (S\rightarrow \bullet,\ a|b)\}= q_4$$

$$g(q_3, b) = [\mu\ (q_3, b)]=$$

$$= [\{(S\rightarrow SaSb\ \bullet,\ \lambda|a)\}]=$$

$$= \{(S\rightarrow SaSb\ \bullet,\ \lambda|a)\}=q_5$$

Получаем O ={ q_4, q_5 }, C ={q_0, q_1, ∅, q_2, q_3}, q_4, q_5 – новые.

6. Исключаем q_4 из O и включаем в C.

Получаем O ={ q_5}, $\boldsymbol{C}$ ={q_0, q_1, ∅, q_2, q_3, q_4}.

Вычисляем последовательно

$$g(q_4, S) = [\mu\ (q_4, S)]=$$

$$= [\{(S\rightarrow SaS\bullet b,\ a|b);\ (S\rightarrow S\bullet aSb,\ a|b)\}]=$$

$$= \{(S\rightarrow SaS\bullet b,\ a|b);\ (S\rightarrow S\bullet aSb,\ a|b)\}=q_6$$

$$g(q_4, a) = [\mu\ (q_4, a)]=[\{\}]=\varnothing$$

$$g(q_4, b) = [\mu\ (q_4, b)]=[\{\}]=\varnothing$$

Получаем O ={ q_5, q_6}, C ={q_0, q_1, ∅, q_2, q_3, q_4}, q_6 – новое.

7. Исключаем q_5 из O и включаем в C.

Получаем O ={ q_6}, C ={q_0, q_1, ∅, q_2, q_3, q_4, q_5}.

Вычисляем последовательно

$$g(q_5, S) = [\mu\ (q_5, S)]=[\{\}]=\varnothing$$

$$g(q_5, a) = [\mu\ (q_5, a)]=$$

$$= [\{(S\rightarrow Sa\bullet Sb,\ a|b)\}]=$$

$$= [\{(S\rightarrow Sa\bullet Sb,\ a|b)\}]=$$

$$= [\{(S\rightarrow Sa\bullet Sb,\ a|b);\}]= q_4$$

$$g(q_5, b) = [\mu\ (q_5, b)]=$$

$$= [\{(S\rightarrow SaSb\ \bullet,\ a|b)]=$$

$$= \{(S\rightarrow SaSb\ \bullet,\ a|b)\}= q_7$$

Получаем O ={ q_6, q_7}, C ={q_0, q_1, ∅, q_2, q_3, q_4, q_5}, q_7 – новое.

8. Исключаем q_6 из O и включаем в C.

Получаем O ={ q_7}, C ={q_0, q_1, ∅, q_2, q_3, q_4, q_5, q_6}.

Вычисляем последовательно

$$g(q_6, S) = [\mu\ (q_6, S)]=[\{\}]=\varnothing$$

$$g(q_6, a) = [\mu\ (q_6, a)]=$$

$$= [\{(S\rightarrow Sa\bullet Sb,\ a|b)\}]= q_4$$

$$g(q_6, b) = [\mu(q_6, b)]=$$
$$= [\{(S\rightarrow SaSb\bullet, a|b)\}]=$$
$$= \{(S\rightarrow SaSb\bullet, a|b)\}= q_7$$

Получаем $O = \{q_7\}$, $C = \{q_0, q_1, \varnothing, q_2, q_3, q_4, q_5, q_6\}$, новых состояний нет.

9. Исключаем q_7 из O и включаем в C.

Получаем $O = \{\}$, $C = \{q_0, q_1, \varnothing, q_2, q_3, q_4, q_5, q_6, q_7\}$.

Вычисляем последовательно

$$g(q_7, S) = [\mu(q_7, S)]= [\{\}]=\varnothing$$
$$g(q_7, a) = [\mu(q_7, a)]=[\{\}]=\varnothing$$
$$g(q_7, b) = [\mu(q_7, b)]=[\{\}]=\varnothing$$

10. Множество $O = \varnothing$.

Множество состояний $Q = C$.

Функция определена.

Шаг 6. Подвергнуть множество состояний Q тесту на *выполнимость LR(k)-условий*

Все состояния непротиворечивы.

Шаг 7. Определить функцию действия f: $Q \times \Sigma^{*k} \rightarrow A$

Область значений

A = {«допуск», «перенос», «ошибка»} $\cup$ { «свертка:p»| $p \in [0, r]$},

где r – число правил исходной грамматики G.

В нашем случае $r = 2$ и k=1.

Множество аванцепочек $\Sigma^{*1} = \{\lambda, a, b\}$.

Множество состояний Q =

$\{q_0 = \{(S'\rightarrow \bullet S, \lambda); (S\rightarrow \bullet SaSb, \lambda|a); (S\rightarrow \bullet, \lambda|a)\}$;

$q_1 = \{(S'\rightarrow S\bullet, \lambda); (S\rightarrow S\bullet aSb, \lambda|a)\}$;

$q_2 = \{(S\rightarrow Sa\bullet Sb, \lambda|a); (S\rightarrow \bullet SaSb, a|b); (S\rightarrow \bullet, a|b)\}$;

$q_3 = \{(S\rightarrow SaS\bullet b, \lambda|a); (S\rightarrow S\bullet aSb, a|b)\}$;

$q_4 = \{(S\rightarrow Sa\bullet Sb, a|b); (S\rightarrow \bullet SaSb, a|b); (S\rightarrow \bullet, a|b)\}$;

$q_5 = \{(S\rightarrow SaSb\bullet, \lambda|a)\}$;

$q_6 = \{(S\rightarrow SaS\bullet b, a|b); (S\rightarrow S\bullet aSb, a|b)\}$;

$q_7 = \{(S\rightarrow SaSb\bullet, a|b)\}$;

$\varnothing$

}

$f(q_0, \lambda|\ a|\ b)$ = «свертка:2» | «свертка:2» | «ошибка»

$f(q_1, \lambda|\ a|\ b)$ = «допуск» | «перенос» | «ошибка»

$f(q_2, \lambda|\ a|\ b)$ = «ошибка» | «свертка:2» | «свертка:2»

$f(q_3, \lambda|\ a|\ b)$ = «ошибка» | «перенос» | «перенос»

$f(q_4, \lambda|\ a|\ b)$ = «ошибка» | «свертка:2» | «свертка:2»
$f(q_5, \lambda|\ a|\ b)$ = «свертка:1» | «свертка:1» | «ошибка»
$f(q_6, \lambda|\ a|\ b)$ = «ошибка» | «перенос» | «перенос»
$f(q_7, \lambda|\ a|\ b)$ = «ошибка» | «свертка:1» | «свертка:1»

Шаг 8. Построить управляющую таблицу *LR(k)-анализатора*

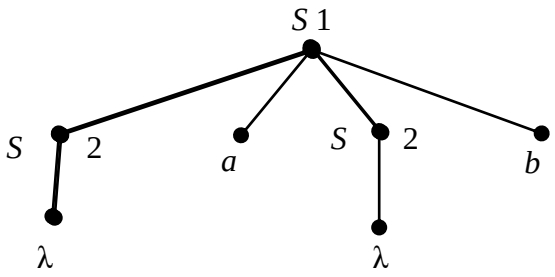
f	λ	a	b	g	S	a	b
q_0	«свертка:2»	«свертка:2»	«ошибка»		q_1	∅	∅
q_1	«допуск»	«перенос»	«ошибка»		∅	q_2	∅
q_2	«ошибка»	«свертка:2»	«свертка:2»		q_3	∅	∅
q_3	«ошибка»	«перенос»	«перенос»		∅	q_4	q_5
q_4	«ошибка»	«свертка:2»	«свертка:2»		q_6	∅	∅
q_5	«свертка:1»	«свертка:1»	«ошибка»		∅	q_4	q_7
q_6	«ошибка»	«перенос»	«перенос»		∅	q_4	q_7
q_7	«ошибка»	«свертка:1»	«свертка:1»		∅	∅	∅

Применим *LR*(1)-анализатор к цепочке $ab \in L(G)$.

$C_0 = (q_0, \bullet\ ab, \lambda)$ $f(q_0, first_1(ab))$ = «свертка:2»
$C_0' = (q_0 S, \bullet\ ab, 2)$ $g(q_0, S) = q_1$
$C_1 = (q_0 S\ q_1, \bullet\ ab, 2)$ $f(q_1, first_1(ab))$ = «перенос»
$C_1' = (q_0 S\ q_1\ a\ , a\bullet\ b, 2)$ $g(q_1, a) = q_2$
$C_2 = (q_0 S\ q_1\ a\ q_2, a\bullet\ b, 2)$ $f(q_2, first_1(b))$ = «свертка:2»
$C_2' = (q_0 S\ q_1\ a\ q_2 S, a\bullet\ b, 22)$ $g(q_2, S) = q_3$
$C_3 = (q_0 S\ q_1\ a\ q_2 S\ q_3, a\bullet\ b, 22)$ $f(q_3, first_1(b))$ = «перенос»
$C_3' = (q_0 S\ q_1\ a\ q_2 S\ q_3\ b, a\ b\bullet, 22)$ $g(q_3, b) = q_5$
$C_4 = (q_0 S\ q_1\ a\ q_2 S\ q_3\ b\ q_5, a\ b\bullet, 22)$ $f(q_5, first_1(\lambda))$ = «свертка:1»
$C_4' = (q_0 S, a\ b\bullet, 221)$ $g(q_0, S) = q_1$
$C_5 = (q_0 S\ q_1, a\ b\bullet, 221)$ $f(q_1, first_1(\lambda))$ = «допуск»

Последовательность $\rho = 221$ – инвертированный правый вывод,
Правый вывод $\pi = \rho' = 122$.

Синтаксическое дерево (дерево грамматического разбора), соответствующее правому выводу $\pi = \rho' = 122$:



5.6. Состояния *LALR*(1)-автомата

Пусть КС-грамматика $G \in LR(k)$. Тогда G допускает построение автомата Кнута $K = (Q, N \cup \Sigma, g, f, q_0, F)$. Детерминированный конечный автомат $M = (Q, N \cup \Sigma, g, q_0, F)$ допускает регулярный язык $R = L(M)$ – множество активных префиксов грамматики G.

По построению каждое состояние $q \in Q$ суть замкнутое подмножество $LR(k)$-ситуаций.

Пусть $s = (A \to \gamma \bullet \beta, u)$ – произвольная $LR(k)$-ситуация и

$$pr_1(s) = A \to \gamma \bullet \beta \quad -$$

проекция $LR(k)$-ситуации s на первую компоненту 2-вектора s, которая называется $LR(0)$-ситуацией.

Каждому состоянию q однозначно соответствует некоторое подмножество $LR(0)$-ситуацией $pr_1(q)$, которое называется *ядром состояния* q. Отображение pr_1 индуцирует на множестве состояний Q отношение эквивалентности: состояния q и p эквивалентны, если $pr_1(q) = pr_1(p)$. Множество состояний Q по отношению эквивалентности разбивается на классы состояний, имеющих одно и тоже ядро.

Пусть $\Pi = \{\Pi_0, \Pi_1, \ldots, \Pi_m\}$, $m \geq 1$, – разбиение множества Q, индуцируемое проекцией pr_1. Каждому классу Π_i , $i \in [0, m]$, сопоставим множество $LR(k)$-ситуаций r_i :

$$r_i = \bigcup_{q \in \Pi_i} q \qquad (1)$$

Очевидно, $pr_1(r_i) \neq pr_1(r_j)$ для всех $0 \leq i \neq j \leq m$, так как ядро множества $LR(k)$-ситуаций r_i определяется ядром всех состояний класса Π_i.

Обозначим через $R = \{r_0, r_1, \ldots, r_m\}$ множество новых состояний. Множество R находится в биективном соответствии с разбиением Π, и все состояния r_i , $i \in [0, m]$, замкнуты в силу (1).

Определим отображение $h : Q \to R$:

$$h(q) = r, \text{ если } pr_1(q) = pr_1(r).$$

Отображение h определено корректно, поскольку каждому состоянию q соответствует однозначно класс Π_i , что $q \in \Pi_i$, и $r = r_i$, $i \in [0, m]$.

Отображение h индуцирует автоматный гомоморфизм $h : M \to M'$, где $M' = (R, N \cup \Sigma, \delta, r_0, F')$, причем $R = h(Q)$, $F' = h(F)$, $r_0 = h(q_0)$ и $h(q') = \delta(h(q), X)$, если $q' = g(q, X)$ для всех $X \in N \cup \Sigma$ и $q, q' \in Q$.

Индукцией по длине входа легко доказать следующее утверждение: ДКА $M' = (R, N \cup \Sigma, \delta, r_0, F')$ допускает множество активных префиксов грамматики G. Следовательно, автоматы M и $M' = h(M)$ эквивалентны.

Как и в методе конструирования $LR(k)$-анализатора вычислим функцию действия f': $R \times \Sigma^{*k} \to 2^A$.

Определение. Функция действия f' ДКА M' однозначна, если для всех $r \in R$ и $u \in \Sigma^{*k}$ справедливо $|f'(r, u)| = 1$.

КС-грамматика G принадлежит классу $LALR(k)$, если функция действия f' ДКА M' однозначна. В противном случае при фиксированном $k \geq 0$ грамматика G не является $LALR(k)$-грамматикой.

Если функция действия f' ДКА M' не однозначна, тогда найдутся состояние r и аванцепочка u такие, что $|f'(r, u)| \geq 2$. Но это равносильно тому, что состояние r при аванцепочке u допускает противоречие (коллизию) хотя бы одного типа:

«перенос»-«свертка:p» или «свертка:p»-«свертка:p'»,

где $p \neq p'$ – номера различных правил грамматики G.

Пример $LALR(1)$-автомата

$LR(1)$-грамматика G с ниже перечисленными правилами

1: $E \to E + T$;

2: $E \to T$;

3: $T \to (E)$;

4: $T \to i$;

где $N = \{E, T\}$, E – аксиома и $\Sigma = \{i, +, (,)\}$, допускает построение $LALR(1)$-автомата. Следовательно, G является $LALR(1)$-грамматикой (*Look Ahead* $LR(1)$-грамматикой: $LR(1)$-грамматикой «с заглядыванием вперед»).

В таблице 1 задано отображение $h : Q \to R$. Первый столбец соответствует множеству состояний R.

Таблица 1

Состояния $LALR(1)$-автомата	Разбиение П множества состояний $LR(1)$-автомата
$r_0 = q_0$	$q_0 = [(\ S' \to \bullet E,\ \lambda\ )]$ $= \{\ (S' \to \bullet E,\ \lambda\ );$ $(E \to \bullet E + T,\ \lambda \mid +);\quad (E \to \bullet T,\ \lambda \mid +);$ $(T \to \bullet i,\ \lambda \mid +);\ (T \to \bullet (,\ \lambda \mid +)\ \}$
$r_1 = q_1$	$q_1 = [\ \{(S' \to E \bullet,\ \lambda\ );\ (E \to E\bullet + T,\ \lambda \mid +)\ \}]$ $= \{(S' \to E \bullet,\ \lambda\ );\ (E \to E\bullet + T,\ \lambda \mid +)\ \}$
$r_2 = q_2 \cup q_7 = (E \to T\bullet,\ \lambda \mid + \mid)\ )$	$q_2 = (E \to T\bullet,\ \lambda \mid +)$ $q_7 = \ (E \to T\bullet,\ \mathbf{)} \mid +\ \ )$
$r_3 = q_3 \cup q_8 = (T \to i\bullet,\ \lambda \mid + \mid)\ )$	$q_3 = \ (T \to i\bullet,\ \lambda \mid +\ \ )$ $q_8 = \ (T \to i\bullet,\ \mathbf{)} \mid +\ \ )$

Продолжение таблицы 1

$r_4 = q_4 \cup q_9$ $= \{(T \to (\bullet E), \lambda \mid +\mid));$ $(E \to \bullet E + T, +\mid)); (E \to \bullet T, +\mid));$ $(T \to \bullet i, +\mid)); (T \to \bullet(E), +\mid))$ }	$q_4 = [(T \to (\bullet E), \lambda \mid + \ )]$ $= \{ (T \to (\bullet E), \lambda \mid + \ );$ $(E \to \bullet E + T, \mathbf{)} \mid +);$ $(E \to \bullet T, \mathbf{)} \mid +);$ $(T \to \bullet i, \mathbf{)} \mid + \ ); (T \to \bullet(E), \mathbf{)} \mid +)$ } $q_9 = [(T \to (\bullet E), \mathbf{)} \mid + \ )]$ $= \{ (T \to (\bullet E), \mathbf{)} \mid + \ );$ $(E \to \bullet E + T, \mathbf{)} \mid + \ );$ $(E \to \bullet T, \mathbf{)} \mid + \ );$ $(T \to \bullet i, \mathbf{)} \mid + \ ); (T \to \bullet(E), \mathbf{)} \mid + \ ) \}$
$r_5 = q_5 \cup q_{11}$ $=\{ (E \to E + \bullet T, \lambda \mid +\mid));$ $(T \to \bullet i, \lambda \mid +\mid)); (T \to \bullet(E), \lambda \mid +\mid)) \}$	$q_5 =[(E \to E + \bullet T, \lambda \mid +)]$ $=\{ (E \to E + \bullet T, \lambda \mid + \ );$ $(T \to \bullet i, \lambda \mid +); (T \to \bullet(E), \lambda \mid +) \}$ $q_{11} = [(E \to E + \bullet T, \mathbf{)} \mid + \ )]$ $= \{(E \to E + \bullet T, \mathbf{)} \mid + \ );$ $(T \to \bullet i, \mathbf{)} \mid + \ ); (T \to \bullet(E), \mathbf{)} \mid +) \mathbf{\}}$
$r_6 = q_6 \cup q_{13}$ $=\{(T \to (E\bullet), \lambda \mid +\mid));$ $(E \to E\bullet + T, +\mid))\}$	$q_6 = [\{(T \to (E\bullet), \lambda \mid +); (E \to E\bullet + T, \mathbf{)} \mid +)\}]$ $=\{(T \to (E\bullet), \lambda \mid +); (E \to E\bullet + T, \mathbf{)} \mid +)\}$ $q_{13} = [\{(T \to (E\bullet), \mathbf{)} \mid +); (E \to E\bullet + T, \mathbf{)} \mid +;\}]$ $= \{ (T \to (E\bullet), \mathbf{)} \mid + \ ); (E \to E\bullet + T, \mathbf{)} \mid + \ ) \}$
$r_7 = q_{10} \cup q_{14} = (E \to E + T\bullet, \lambda \mid +\mid))$	$q_{10} = (E \to E + T\bullet, \lambda \mid +)$ $q_{14} = (E \to E + T\bullet, \mathbf{)} \mid +)$
$r_8 = q_{12} \cup q_{15} = (T \to (E) \bullet, \lambda \mid +\mid))$	$q_{12} = (T \to (E) \bullet, \lambda \mid +)$ $q_{15} = (T \to (E) \bullet, \mathbf{)} \mid + \ )$
$r_9 = q_{16}$	$q_{16} = \varnothing$

Функция переходов δ**:** $Q \times (N \cup \Sigma) \to Q$ *LALR*(1)-автомата представлена в таблице 2.

Таблица 2

δ	E	T	i	(	+	)
$r_0 = q_0$	r_1	r_2	r_3	r_4		
$r_1 = q_1$					r_5	
$r_2 = q_2 \cup q_7$						
$r_3 = q_3 \cup q_8$						
$r_4 = q_4 \cup q_9$	r_6	r_2	r_3	r_4		
$r_5 = q_5 \cup q_{11}$		r_7	r_3	r_4		
$r_6 = q_6 \cup q_{13}$					r_5	r_8
$r_7 = q_{10} \cup q_{14}$						
$r_8 = q_{12} \cup q_{15}$						
$r_9 = q_{16}$						

Пустые ячейки таблицы соответствуют состоянию $r_9 = \varnothing$.

Функция действия f': $Q \times \Sigma^{*k} \rightarrow A$ *LALR*(1)-автомата представлена в таблице 3. Область значений

A = {«допуск», «перенос», «ошибка»} $\cup$ { «свертка: p» | $p \in [0,r]$},

где r – число правил исходной грамматики G.

В нашем случае $r = 4$ и k=1. Множество аванцепочек Σ^{*1} = {λ, i, (, +,)}.

Множество состояний R ={ r_i | $0 \le i \le 9$, $r_9 = \varnothing$}

Таблица 3

f'	λ	i	(	+	)
$r_0 = q_0$		«прнс»	«прнс»		
$r_1 = q_1$	«дпск»			«прнс»	
$r_2 = q_2 \cup q_7$	«сврт:2»			«сврт:2»	«сврт:2»
$r_3 = q_3 \cup q_8$	«сврт:4»			«сврт:4»	«сврт:4»
$r_4 = q_4 \cup q_9$		«прнс»	«прнс»		
$r_5 = q_5 \cup q_{11}$		«прнс»	«прнс»		
$r_6 = q_6 \cup q_{13}$				«прнс»	«прнс»
$r_7 = q_{10} \cup q_{14}$	«сврт:1»			«сврт:1»	«сврт:1»
$r_8 = q_{12} \cup q_{15}$	«сврт:3»			«сврт:3»	«сврт:3»
$r_9 = q_{16}$					

5.7. Вычисление функции $FOLLOW_k(X)$

Метод Кнута позволяет извлечь алгоритм вычисления функции $FOLLOW_k(X)$ на множестве нетерминалов любой контекстно-свободной грамматики G.

Пусть Q – множество состояний автомата $M = (Q, N \cup \Sigma, g, q_0, F)$, допускающего множество активных префиксов грамматики G.

Тогда для каждого нетерминала $A \in N$ грамматики G его множество правых ограниченных терминальных контекстов вычисляется по формуле

$$FOLLOW_k(A) = \bigcup_{q \in Q} \{u \mid (A \rightarrow \alpha \bullet, u) \in q\}$$

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. *Ахо, А*. Компиляторы: принципы, технологии и инструменты / А. Ахо, М. Лам, Р. Сети, Дж. Д. Ульман; пер. с англ. 2-е изд.,– М.: Издат. дом «Вильямс», 2014. – 1184 с.
2. *Ахо, А*. Теория синтаксического анализа, перевода и компиляции : в 2 т. / А. Ахо, Дж. Ульман. – М.: Мир, 1998. Т. 1: Синтаксический анализ. – 612 с.
3. *Ахо, А*. Теория синтаксического анализа, перевода и компиляции : в 2 т. / А. Ахо, Дж. Ульман. – М.: Мир, 1998. Т. 2: Компиляция. – 487 с.
4. *Льюис, Ф.* Теоретические основы проектирования трансляторов / Ф. Льюис, Д. Розенкранц, Р. Стирнз. – М.: Мир, 1995. – 654 с.
5. *Буза, М. К.* Структуры данных и методы трансляции : в 3 ч. учебн. пособие для вузов / М. К. Буза, В. П. Дубков, В. В. Рябый. – Минск.: Выш. шк., 1983. Ч. 2 : Лабораторный практикум по математическому обеспечению ЭВМ. – 173 с.
6. *Керниган, Б.* Язык программирования Си / Б. Керниган, Д. Ритчи / пер. с англ.; под ред. и предисл. В. С. Штаркмана. – 2-е изд., перераб. и доп. – М.: Финансы и статистика, 1992. – 272 с.
7. *Макаров, А. В.* Common Intermediate Language и системное программирование в Microsoft.NET : учебн. пособие / А. В. Макаров, С. Ю. Скоробогатов, А. М. Чеповский. – М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2006. – 328 с.: – ил. – (Серия «Основы информатики и математики»)
8. *Хопкрофт, Дж*. Введение в теорию автоматов, языков и вычислений / Дж. Хопкрофт, Р. Мотвани, Дж. Ульман. – 2-е изд. – М.: Вильямс, 2008. – 528 с.
9. Алгебраическая теория автоматов, языков и полугрупп / под ред. М. Арбиба; пер. с англ. – М.: Статистика, 1975. – 335 с.
10. *Грис, Д.* Конструирование компиляторов для цифровых вычислительных машин / Д. Грис.– М.: Мир, 1975. – 544 с.
11. Семантика языков программирования : сб. ст. – М.: Мир, 1990. – 394 с.
12. Языки программирования / под ред. Ф. Женюи. – М.: Мир, 1972. – 406 с.
13. *Касьянов, В. Н.* Методы построения трансляторов / В. Н. Касьянов, И. В. Поттосин. – Новосибирск : Наука, Сиб. отд. – 1986. – 330 с.
14. *Хантер, Р*. Проектирование и конструирование компиляторов / Р. Хантер / пер. с англ. – М.: Финансы и статистика, 1984. – 232 с.
15. *Рейуорд-Смит, В. Дж.* Теория формальных языков. Вводный курс / В. Дж. Рейуорд-Смит. – М.: Радио и связь, 1988. – 128 с.

ОГЛАВЛЕНИЕ

Учебное издание

Рябый Вячеслав Васильевич

МЕТОДЫ ПРОЕКТИРОВАНИЯ ЛЕКСИЧЕСКИХ И СИНТАКСИЧЕСКИХ АНАЛИЗАТОРОВ

Учебные материалы для студентов факультета прикладной математики и информатики

В авторской редакции

Ответственный за выпуск *В. В. Рябый*

Подписано в печать 30.12.2015. Формат 60×84/16. Бумага офсетная.
Усл. печ. л. 2,79. Уч.-изд. л. 2,6. Тираж 50 экз. Заказ

Белорусский государственный университет
Свидетельство о государственной регистрации издателя, изготовителя, распространителя печатных изданий № 1/270 от 03.04.2014.
Пр. Независимости, 4, 220030, Минск.

Отпечатано с оригинал-макета заказчика
на копировально-множительной технике

факультета прикладной математики и информатики
Белорусского государственного университета.
Пр. Независимости, 4, 220030, Минск.