Федеральное агентство по образованию
Государственное образовательное учреждение высшего профессионального образования
Амурский государственный университет
(ГОУВПО «АмГУ»)

УТВЕРЖДАЮ
Зав. кафедрой ИиУС
_____ А.В. Бушманов
«___» _____ 2007 г.

Учебно-методический комплекс дисциплины

ЛИНГВИСТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ

для специальности
230201 – Информационные системы и технологии

Составитель: Соловцова Л.А.

2007 г.

Печатается по решению
редакционно-издательского совета
факультета математики
и информатики
Амурского государственного
университета

Лингвистические основы информатики для специальности 230201 «Информационные системы и технологии»: учебно-методический комплекс дисциплины. / Соловцова Л.А. – Благовещенск. Изд-во Амурского гос. ун-та, 2007. 11 с.

ОГЛАВЛЕНИЕ

1. ВЫПИСКА ИЗ ГОСУДАРСТВЕННОГО ОБРАЗОВАТЕЛЬНОГО СТАНДАРТА ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

Направление подготовки дипломированного специалиста

654600 – Информатика и вычислительная техника

Специальность

071900 (230201) – Информационные системы и технологии.

Квалификация – *инженер*

Индекс	*Наименование дисциплин и их основные разделы*	Всего часов
ОПД 17	**Лингвистические основы информатики**	110
	Роль измерений в познании окружающего мира; основные понятия и определения метрологии; виды измерений; погрешности измерений; вероятностные оценки погрешности измерения; средства измерений; метрологические характеристики средств измерения; нормирование метрологических характеристик; электромеханические измерительные приборы; цифровые измерительные приборы; мосты и компенсаторы; осциллографы; средства измерения неэлектрических величин; измерительные информационные системы; подготовка измерительного эксперимента; обработка результатов измерения. .	

2. РАБОЧАЯ ПРОГРАММА

по дисциплине " Лингвистические основы информатики" для специальности 230201 "Информационные системы и технологии"

курс 1 семестр 2

Лекции 36 (час.) Экзамен 5

Практические занятия 18 (час.)

самостоятельная работа _56__ (час.)

Всего часов _110___ час.

Составитель: ст.преподаватель Соловцова Л.А..

Факультет Математики и информатики

Кафедра Информационных и управляющих систем

1. Цели и задачи дисциплины.

1.1. Цели и задачи дисциплины

Целью курса – знакомство с системной моделью формального языка, основами построения компилятора.

1.2. Требования к уровню освоения содержания дисциплины

По завершению обучения по дисциплине студент должен :

- ознакомиться с системной моделью естественного языка;

- знать основные блоки , необходимые для работы компилятора и основные формализмы, которые лежат в основе построения этих блоков.

- уметь не только построить конечный автомат для группы цепочек, но и выполнить его упрощение;

-уметь строить контекстно-свободные грамматики для ограниченного языка и проводить ее нисходящий разбор;

- знать виды грамматик и уметь с ними работать.

2. СОДЕРЖАНИЕ ДИСЦИПЛИНЫ.

2.1. Федеральный компонент.

Дисциплина «Лингвистические основы информатики» является дисциплиной , входящей в блок общеобразовательных дисциплин федерального компонента для специальности 230201 «Информационные системы и технологии». Государственный стандарт – ОПД 17

2.2.Наименование тем их содержание, объем в лекционных часах.

Тематический план лекционных занятий

№	Наименование темы	Кол-во часов
1	Язык как символьная система	2
2	Структура естественного языка	4
3	Структура компилятора	2
4	Конечный автомат	2
5	Получение минимального автомата	2
6	Реализация конечных автоматов	4
7	Автоматы с магазинной памятью	2
8	МП-трансляторы	2
9	Контекстно-свободные грамматики	4

10	Нисходящие методы обработки языков с помощью МП-распознавателей	6
11	Обработка ошибок при нисходящем разборе	4
12	Метод рекурсивного спуска	2
	Итого	36

Тема 1. Язык как знаковая система. (2часа).

Построение системной модели естественного языка.

Тема 2. Структура естественного языка. (4 часа)

Фонетика. Морфология. Синтаксические правила. Слово. Словосочетание. Предложение. Правила построения конструкций языка. Семантика. Анализ и синтез текста, речи.

Тема 3. Структура компилятора.(2 часа)

Обобщенная схема компилятора. Назначение и характеристика основных блоков компилятора.

Тема 4. Конечный автомат. (2 часа)

Определение конечного автомата. Конечный распознаватель. Представление конечного автомата с помощью таблицы переходов. Примеры конечных автоматов.

Тема 5.Получение минимального конечного автомата. (2 часа)

Определение эквивалентного конечного автомата и состояния. Построение таблиц эквивалентности. Алгоритм поиска эквивалентных состояний. Недостижимые состояния и алгоритм их поиска.

Тема 6. Реализация конечных автоматов.(4 часа)

Задачи, решаемые при реализации конечных автоматов. Представление входных символов. Представление состояний. Идентификация слов.

Тема 7. Автоматы с магазинной памятью. (2 часа)

Определение автомата с магазинной памятью. Задачи, решаемые с помощью автоматов с магазинной памятью. Определение состояний автомата, переходов, операций над входом и магазином.

Тема 8. МП-трансляторы.(2 часа)

Определение МП-транслятора. Задачи, решаемые МП-транслятором. Его построение.

Тема 9. Контекстно-свободные грамматики. (4 часа)

Определение и основные понятия контекстно-свободной грамматики. Форма записи КС-грамматики. Вывод. Цепочка вывода. Дерево вывода..

Тема 10. Нисходящие методы обработки языков с помощью МП-распознавателей. (6 часов)

Определение нисходящего метода обработки языков. Определение s-грамматики и построение для нее нисходящего МП-распознавателя.. Определение q-грамматики и построение для нее нисходящего МП-распознавателя.. Определение LL(1)-грамматики и построение для нее нисходящего МП-распознавателя.

Тема 11. Обработка ошибок при нисходящем разборе. (4часа)

Характеристика ошибок, возникающих при нисходящем разборе. Общие подходы при обработке ошибок. Нейтрализация ошибок.

Тема 12. Метод рекурсивного спуска. (2 часа)

Определение рекурсивного спуска при нисходящем разборе. Схема программы для релизации рекурсивного спуска.

2.3. Практические занятия, их содержание и объем в часах

Тематический план практических занятий.

№	Наименование темы	К-во час.
1	Регулярные множества и выражения	2
2	Конечные автоматы	2
3	Построение приведенного автомата	2
4	Реализация конечного автомата	2
5	Автоматы с магазинной памятью	2
6	Нисходящие методы обработки языков с помощью МП-автоматов. S-грамматики	2
7	Нисходящие методы обработки языков с помощью МП-автоматов. Q-грамматики	2
8	Нисходящие методы обработки языков с помощью МП-автоматов.LL(1)-грамматики	2
9	Обработки ошибок при нисходящем разборе	2
	Итого	18

2.5. Самостоятельная работа студентов.

Для самостоятельной работы предлагается следующая тема: «Проектирование лексического блока для языка Turbo Pascal». Остановиться на решении одной из задач:

- правильность построения идентификаторов и ключевых слов;
- раздел описания переменных;
- десятичное число.
- правильность арифметических выражений;

Студентом должен быть представлен отчет о проделанной работе.

2.6. Вопросы к экзамену

1. Структура компилятора.
2. Лексический блок компилятора.
3. Регулярное выражение.
4. Конечные распознаватели.
5. Таблица переходов.
6. Проверка эквивалентности двух состояний.
7. Недостижимые состояния.
8. Получение минимального автомата.
9. Реализация конечных автоматов. Представление входных символов.
10. Представление состояний.
11. выбор переходов.
12. Идентификация слов: метод автомата.
13. Идентификация слов: метод индексов.
14. Идентификация слов: метод линейного списка.
15. Идентификация слов: метод упорядоченного списка.
16. Идентификация слов: метод расстановки
17. Определение автомата с магазинной памятью.
18. Распознавание множества МП-автоматом.
19. перевод с помощью МП-автоматов.
20. Формальные языки и формальные грамматики.
21. Контекстно-свободные грамматики.
22. Выводы.
23. Деревья.
24. Грамматика для констант языка Си++.
25. Грамматика для арифметических выражений.
26. Лишние нетерминалы.
27. Транслирующая грамматика. Нисходящая обработка.
28. Нисходящие методы обработки языков.
29. S-грамматики.
30. Q-грамматики

31. LL(1)- грамматики.
32. Обработка ошибок при нисходящем разборе.
33. Синхронизирующие и начинающие символы.
34. Метод рекурсивного спуска.
35. Генератор кода.
36. Методы оптимизации объектного кода.

2.8.Виды контроля

Текущий контроль за аудиторной и самостоятельной работой обучаемых осуществляется во время проведения аудиторных занятий посредством устного опроса , проведения контрольных работ. Промежуточный контроль осуществляется два раза в семестр на основе анализа аудиторной и самостоятельной работы студента. Итоговый контроль осуществляется после успешного прохождения текущего и промежуточного контроля в виде зачета и устного или письменного экзамена при ответах на два вопроса в билете и дополнительные вопросы экзаменатора.

2.10. Требования к знаниям студентов, предъявляемые на экзамене

Студент, сдающий экзамен по данному предмету, должен показать знания по всем лекционным темам.

Знания студента оцениваются как отличные при полном изложении теоретического материала экзаменационного билета и ответах на дополнительные вопросы со свободной ориентацией в материале и других литературных источниках.

Оценка «хорошо»ставится при твердых знаниях студентом всех разделов курса, но в пределах конспекта лекций и обязательных заданий по самостоятельной работе с литературой.

Оценку «удовлетворительно» студент получает, если дает неполные ответы на теоретические вопросы билета, показывая поверхностное знание

учебного материала, владение основными понятиями и терминологией, при неверном ответе на билет дает ответы на наводящие вопросы.

Оценка «неудовлетворительно» выставляется за незнание студентом разделов курса. Студент не дает полные ответы на теоретические вопросы билета, показывая лишь фрагментарное знание учебного материала, незнание основных понятий и терминологии, наводящие вопросы остаются без ответа.

3. Учебно-методические материалы по дисциплине

3.1 Перечень обязательной (основной) литературы.

1. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. В 2 т. М.:Мир, 1998.

2. Чирков М.К. Основы общей теории конечных автоматов. Л.:Изд-во ЛГУ, 2001.

3. Льюис, Розенкранц. Теория проектирования трансляторов. М.: Мир, 1998.

4. Янгер Д. Н. Распознавание и анализ контекстно-свободных языков за время n^3 // Проблемы математической логики. М.: Мир, 2000.

5. Эрли Дж. Эффективный алгоритм анализа контекстно-свободных языков// Языки и автоматы. М.: Мир, 2000.

6. Т.А. Галаган, Л.А.Соловцова. Практикум по лингвистическим основам информатики. Благовещенск: Амурский гос. у-т, 2005.

3. ГРАФИК САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

Содержание	Объем в часах	Сроки и форма контроля
3.1. Контрольная работа по теме «Конечные автоматы»	2 час.	Собеседование (6 неделя)
3.2. Контрольная работа по теме «Автомат	2 час.	Собеседование

с магазинной памятью»		(14 неделя)
3.3 Контрольная работа теме «Контекстно-свободные грамматики»	2 час.	Собеседование (18 неделя)

4. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПРОВЕДЕНИЮ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

Для самостоятельной работы предлагается следующая тема: «Проектирование лексического блока для языка Turbo Pascal». Остановиться на решении одной из задач:

- правильность построения идентификаторов и ключевых слов;
- раздел описания переменных;

- десятичное число.

- правильность арифметических выражений;

Студентом должен быть представлен отчет о проделанной работе.

5. ПЕРЕЧЕНЬ УЧЕБНИКОВ, УЧЕБНЫХ ПОСОБИЙ

5.1. Перечень основной литературы

1. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. В 2 т. М.:Мир, 1998.

2. Чирков М.К. Основы общей теории конечных автоматов. Л.:Изд-во ЛГУ, 2001.

3. Льюис, Розенкранц. Теория проектирования трансляторов. М.: Мир,

1998.

4. Янгер Д. Н. Распознавание и анализ контекстно-свободных языков за время n^3 // Проблемы математической логики. М.: Мир, 2000.

5. Эрли Дж. Эффективный алгоритм анализа контекстно-свободных языков// Языки и автоматы. М.: Мир, 2000.

6. Т.А. Галаган, Л.А.Соловцова. Практикум по лингвистическим основам информатики. Благовещенск: Амурский гос. у-т, 2005.

6. КОНСПЕКТ ЛЕКЦИЙ

Л Е К Ц И Я № 1

ЯЗЫК КАК ЗНАКОВАЯ СИСТЕМА

Язык –форма существования знания в виде системы знаков плюс правила функционирования этих знаков, служащая средством общения, мышления и выражения. Таким образом , язык имеет две основные функции – он яв-

ляется средством человеческого мышления и средством общения друг с другом.

Общение представляет собой двусторонний процесс передачи информации с определенными целями и по определенным правилам, выраженной на языке, понятном участникам общения, при этом под понятностью понимаются синтаксическая, семантическая и прагматическая однозначность информации, требующая общности языка и знаний участников о предметной области общения.

Кроме этого, язык обладает и моделирующей функцией, то есть обеспечивает представление некоторых характеристик физической или абстрактной системы средствами конкретного языка. О сновой моделирования является экспликация – строгая формулировка содержательного или интуитивного понятия в предметной области моделирования.

В автоматизированных информационных системах средством моделирования служит ограниченный естественный язык – искусственный язык, разработанный для общения человека с ЭВМ.

Появление развитых диалоговых языков общения с ЭВМ еще более усилило интерес исследователей к функциям естественного языка. Для человека идеальным было бы общение с ЭВМ на обычном языке. Кроме того, естественный язык – это единственная известная на сегодня моделирующая система, средствами которой можно описать все многообразие окружающего мира. Основными задачами в области исследования языка как средства для описания действительности и средства общения между человеком и ЭВМ являются следующие:

- автоматизированный анализ текстов на естественном языке;
- переход от языковых представлений к языку описания знаний;
- понимание вопросов и формирование ответов в автоматизированных системах;
- извлечение знаний из текстов на естественном языке.

В языковых исследованиях важное значение придается понятию знак–

способу обозначения определенного понятия, предмета, свойства, используемому для приобретения, хранения, обработки и передачи информации.Лингвисты рассматривают язык как знаковую систему –знаки ее не функционируют независимо друг от друга, а образуют систему , правила которой определяют закономерномти их построения, осмысления и употребления(грамматика, правила смысла).

Этот подход развился в отдельную научную дисциплину, семиотику – науку о том, с помощью каких средств в человеческом обществе происходит общение (передача информации), в том числе :

- как устроены эти средства сами по себе;
- как они применяются;
- каким изменениям они подвержены.

Во всяком общении можно выделить смысл и средства его передачи. Если расчленить этот смысл на элементы и найти средства, которыми выражается каждый из них, – это и будут знаки, соединение определенного способа его выражения (означаемого и означающего). Носителями смысла могут быть действия, понятия, предметы (идеальные, материальные). Например выделяют следующие типы знаков:

знак-копия – воспроизведение, более или менее сходные с обозначаемыми;

знак-признак – связан с обозначаемым предметом как действие со своими причинами (симптомы, приметы);

знак-символ – наглядные образы, используемые для выражения некоторого, часто весьма значительного и отвлеченного, содержания.

В исследованиях знаковых систем применяется ряд специфических понятий; так под синтагмой понимают сложный языковой знак (обычно двучленный), составленный из слов или морфем, соединенных определенным типом связи. Методы изучения синтагм составляют предмет синтагматики.

Синтагматика занимается изучением структурных аспектов сочетания знаков данной системы, правила их образования и преобразования безотно-

сительно к их значениями функциям, в то время как прагматика –изучение отношения, воспринимающего знаковую систему (интерпретатор или адресат), к самой знаковой системе.

Предмет, обозначаемый знаком, называется денотат (референт), а концепт –это информация, которую знак несет о возможных денотатах, о их положении в системе реалий, об их месте в универсуме.

Экстенсионал знака – определяет класс всех допустимых для этого знака денотатов, а интенсионал знака – это характеристика концепта, выраженная через общие свойства денотата.

При анализе тестов используют понятие дискурс (связный текст) – два или более предложений, находящиеся друг с другом в смысловой связи, и лингвистическая совместимость –способность воспринимать и интерпретировать языковую форму представления информации.

Существует еще два важных, основополагающих терминов лингвистики – это:

абстракция – процесс формирования образов реальности (представлений, понятий, суждений) посредством отвлечения и пополнения;

понятие – мысль, отражающая в обобщенной форме предметы и явления действительности и связи между ними посредством фиксации общих и специфических признаков, в качестве которых выступают свойства предметов и явлений и отношения между ними.

Л Е К Ц И Я № 2

СТРУКТУРА ЕСТЕСТВЕННЫХ ЯЗЫКОВ

Человеческая речь –это, прежде всего, звуковая речь. Письменная форма речи представляет собой лишь ее обедненное графическое отображение. Тем не менее письменная речь является эффективным средством человеческого общения.

Существуют различные виды письменности. Чаще всего при создании

используется фонетический принцип, когда с помощью графических символов(букв) обозначаются минимальные смыслоразличительные отрезки звуковой речи (фонемы), а связная речь представляется в виде последовательности букв. Некоторые фонемы могут обозначаться различными буквами, а одни и те же буквы в различных позициях могут обозначать различные фонемы. Положение осложняется исторической традицией, сохраняющей графические образы отрезков речи, изменивших свой первоначальный звуковой состав. Будем рассматривать только письменную форму речи.

В письменных текстах есть много условностей и элементов формализации. Например , довольно условно устанавливаются границы между словами, предложениями и другими единицами речи, а для обозначения этих границ широко применяются различные виды разделителей: пробелы – для обозначения границ между словами, прописные буквы и знаки препинания – для обозначения границ между предложениями и составными частями предложений, абзацные отступы – для обозначения границ между связанными по смыслу группами предложений и т.п.

Слово – законченная последовательность знаков определенной длины, воспринимаемая как элемент обработки с определенным семантическим содержанием. Слово – минимальная, формально выделяемая единица связного текста, но оно – не минимальная единица смысла и может состоять из одной или несколько морфем. В составе слов различают корневые морфемы (корни), префиксы (приставки), суффиксы. Основную смысловую нагрузку несет корень, а префиксы и суффиксы выступают в роли модификаторов смысла. Например, в слове выступающий можно выделить пять морфем: вы-ступ-а-ющ-ий. Здесь морфема ступ – корень слова, морфема вы – префикс, морфемы а, ющ, ий – суффиксы.

Членение слов на морфемы, равно как и членение текста на слова, словосочетания, предложения, сверхфразовые единства – дело непростое (морфология – учение о грамматических формах отдельных слов).

Определению границ слов носители языка обучаются с первых шагов

овладения письменностью, а сами слова фиксируются в нормативных словарях . Подразделение на фонетику , морфологию, и синтаксис произошло посредством дробления и механического членения. Язык изучают не в процессе его становления, а в его состоянии. Его рассматривают как нечто данное и завершенное. Живая речь разлагается на предложения, члены предложения, слова, слоги и звуки. Под слогом понимают часть слова, допускающую независимое обращение и обработку.

Этот метод может привести к ценным наблюдениям, но одновременно может стать источником ошибок. Ошибки начинаются тогда, когда убеждают себя, что указанное членение находит основание в самом организме человеческой речи , что оно представляет собой нечто большее, чем абсолютное, произвольное, механическое и насильственное рассечение.

В процессе функционирования в речи слова приобретают различные формы. Это могут быть формы словоизменения и словообразования. Граница между ними условная. Например, можно считать, что формы склонения существительных и прилагательных, формы спряжения глаголов являются формами словоизменения, а все остальные трансформации слов – формами словообразования.

Изменения форм словмогут носить различный характер. Они могут быть связаны как с изменением основы слова, так и с изменениями его окончания. Изменение окончаний является основным способом образования различных словоизменительных форм слова. В русском языке оно используется как самостоятельно, так и в сочетании с изменением основ слов.

По способу изменения грамматических окончаний(флексий) и своей синтаксической функции русские слова могут быть разбиты на ряд классов, которые получили название флективных. Флективные классы изменяемых слов выделяются на основе анализа и синтаксической функции и систем падежных, личных и родовых окончаний. Классы неизменяемых слов – только по синтаксическому принципу.

По своей синтаксической функции изменяемые слова объединены в

следующие группы: существительные, прилагательные, глаголы в личной форме, глаголы прошедшего времени, краткие прилагательные и причастия, количественные числительные.

В автоматизированных информационных системах, основанных на формализованной записи, используются понятия, выраженные именными словосочетаниями. Словосочетание – смысловое и грамматическое объединение нескольких полнозначных слов. Эти понятия могут обозначать различного рода объекты, их признаки, значения признаков, рубрики классификационных схем и т.п. В именных словосочетаниях главным словом является, как правило, первое слева существительное, а остальные слова служат для уточнения значения главного слова.

Предложение – базовая единица языка, обладающая определенной для данного языка синтаксической и смысловой законченностью. Различают предложение простые и сложные. Обычно простое предложение состоит из группы подлежащего и группы сказуемого. Группа подлежащего обозначает предмет высказывания, группа сказуемого – признак предмета. В общем случае группа подлежащего выражается именным словосочетанием, а группа сказуемого глаголом, кратким прилагательным или кратким причастием с определяющим или дополняющим их словами и словосочетаниями.

Исследование синтаксиса естественного языка – системы правил соединения словоформ, словосочетаний и предложений в естественном языке – оказывает серьезное влияние на разработку синтаксиса современных языков программирования (набор правил, которые определяют основные внутренние структуры и последовательности символов, допустимые в языках программирования).

В исследованиях по структурам текстов особое значение имеют следующие понятия:

семантика – значения языковых единиц – слов, грамматических форм слов, словосочетаний, предложений;

семиотика – наука, изучающая знаковые системы, а также естественные и ис-

кусственные языки как знаковые системы;

метаязык – язык, используемый только для описания другого языка;

метасимвол – символ или знак, используемые для выполнения синтаксических функций, указаний и других вспомогательных целей (скобки, разделители).

Л Е К Ц И Я № 3

СТРУКТУРА КОМПИЛЯТОРА

Транслятор – программа, допускающая в качестве входа программу на исходном языке, а в качестве выхода – другую версию, написанную на языке, который называется объектным. Это машинный язык некоторой вычислительной машины, что дает возможность сразу же выполнить программу. Существует довольно условное деление трансляторов на ассемблеры и компиляторы, транслирующие соответственно языки низкого и высокого уровней.

Компиляторы составляют существенную часть программного обеспечения ЭВМ. Построение компилятора как единого целого – нелегкое и трудоемкое занятие. Поэтому лучше рассматривать процесс компиляции как взаимодействие небольших процессов. Выбор таких подпроцессов для каждого конкретного компилятора может зависеть от особенностей обрабатываемого языка, но в любом случае его лучше сделать с учетом соответствующей теории построения компиляторов.

Основные этапы компиляции – лексический, синтаксический и семантический анализ, оптимизация и генератор кода. Эти три блока имеют доступ к общему набору таблиц, куда можно помещать долговременную и глобальную информацию о программе. Например, таблица имен (называемая таблицей идентификаторов, или таблицей символов), в которой накапливается информация о каждой переменной или идентификаторе). Связи между блоками и таблицами показаны на рис.1. Далее работа блоков описывается более детально.

Основная задача лексического анализа – разбить входной текст, со-

стоящий из последовательности одиночных символов, на последовательность слов или лексем, то есть выделить эти слова из непрерывной последовательно-

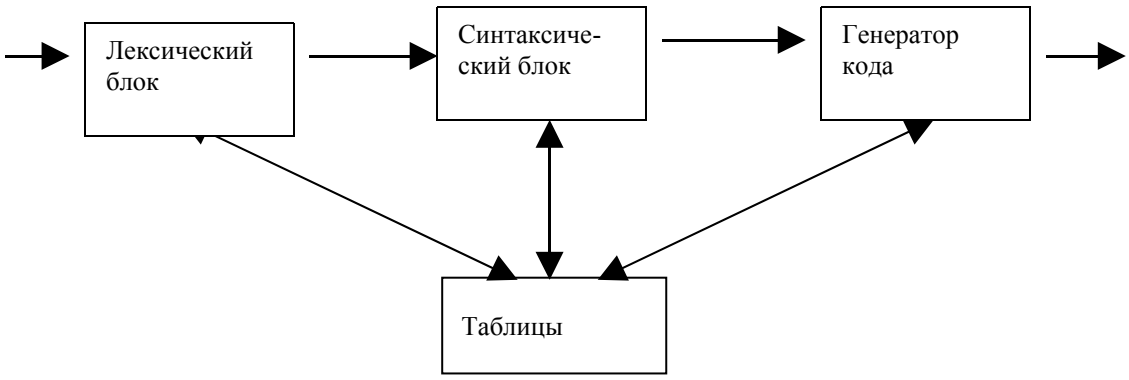


Рис.1.

сти символов. Все символы входной последовательности с этой точки зрения разделяются на принадлежащие каким-либо лексемам и на разделяющие лексемы (разделители).

Обычно все лексемы делятся на классы – числа, идентификаторы, строки. Отдельно выделяют ключевые слова и символы пунктуации (символы-ограничители). Ключевые слова – это некоторое конечное подмножество идентификаторов. В части языков смысл лексемы может зависеть от ее контекста, и провести лексический анализ в отрыве от синтаксического невозможно. С точки зрения дальнейших фаз анализа лексический анализатор выдает информацию двух сортов: для синтаксического анализатора существенна информация о последовательности классов лексем, ограничителей и ключевых слов, а для контекстного – о конкретных значениях отдельных лексем.

Общая схема работы лексического анализатора следующая. Сначала выделяем отдельную лексему, используя символы-разделители. Если выделенная лексема – ограничитель, то он (точнее, его признак) выдается как ре-

зультат лексического анализа.

Ключевые слова – как правило, выделенные идентификаторы – распознаются следующим образом. Возможны два основных способа их выделения либо очередная лексема сначала диагностируется на совпадение с каким-либо ключевым словом и в случае неуспеха делается попытка выделить лексему из какого-либо класса, либо, наоборот, после выборки лексемы идентификатора требуется заглянуть в таблицу ключевых слов для сравнения. Поиск ключевых слов можно вести в основной таблице имен (в которую до начала лексического анализа загружают ключевые слова), либо в отдельной таблице. В первом случае все ключевые слова непосредственно встраивают в конечный автомат лексического анализатора, во втором – конечный автомат содержит только разбор идентификаторов. Если выделенная лексема принадлежит какому-либо другому классу лексем (число, строка и т.д.), выделяется признак класса лексемы, а значение лексемы сохраняется.

Лексический анализатор может работать как самостоятельная фаза трансляции или как подпрограмма, действующая по принципу "дай лексему". В первом случае выходом является файл лексем, во втором лексема выдается при каждом обращении к анализатору (при этом тип лексемы зачастую возвращается как значение функции "лексический анализатор", а значение передается через глобальную переменную). Лексический анализатор может просто выдавать значение каждой лексемы, и тогда построение таблицы переносится на более поздние фазы; он может также самостоятельно строить таблицы объектов (идентификаторов, строк, чисел и т.д.). В этом случае в качестве значения лексемы выдается указатель на вход в соответствующую таблицу.

Основная задача синтаксического анализа – разбор структуры программы. Этот блок переводит последовательность лексем, построенную лексическим блоком, в другую последовательность, более непосредственно отражающую порядок, в котором по замыслу программиста должны выполняться операции в программе. Например, если имеется запись A+B*C,

подразумевается, что числа, представленные идентификатором B и C, будут перемножены и к результату будет прибавлено число, представленное идентификатором A. Указанное выражение можно перевести так: УМНОЖ(B, C, R1) СЛОЖ(A, R1, R2),где УМНОЖ(B, C, R1) интерпретируется как "умножить B на C и заслать результат в R1, а СЛОЖ(A, R1, R2) – как "сложить A и R1 и результат заслать в R2 ".

Таким образом, пять лексем, выданных лексическим блоком, преобразуются в две новые единицы, описывающие то же действие. Эти единицы называются атомами и образуют выход синтаксического блока.

Преимуществом здесь является то, что последовательность атомов отражает порядок, в котором должны выполняться действия. Основной формализм синтаксического анализа – контекстно-свободные грамматики. Результат синтаксического анализа – синтаксическое дерево со ссылками на таблицу имен. На этом этапе обнаруживаются ошибки, связанные со структурой программы.

На этапе контекстного анализа выявляются зависимости между частями программы, которые не могут быть описаны контекстно-свободным синтаксисом. Эту часть работы компилятора называют семантической обработкой. Семантика идентификатора, например, может включать его тип, если это массив- его размерность. Один из видов семантической обработки включает занесение в таблицу имен свойств идентификаторов по мере их выявления. На данном этапе выполняется анализ областей видимости, соответствия параметров, меток и др. Основной формализм – атрибутные грамматики. В процессе контекстного анализа могут быть обнаружены ошибки, связанные с неправильным использованием объектов.

Затем программа с целью оптимизации и удобства генерации может быть переведена во внутреннее представление. Фаз оптимизации может быть несколько.

Генерация кода – последняя фаза трансляции. Результат ее – либо ассемблерный, либо объектный модуль. В процессе генераций кода могут вы-

полняться некоторые локальные оптимизации – такие как распределение регистров, выбор длинных и коротких переходов, учет стоимости команд при выборе их конкретной последовательности. Для генерации кода разработаны различные методы(таблицы решений, сопоставление образцов, включающее динамическое программирование, различные синтаксические методы).

Те или иные фазы транслятора могут либо отсутствовать вовсе, либо объединяться. В простейшем случае однопроходного транслятора нет явной фазы генерации промежуточного представления и оптимизации, остальные фазы объединены в одну, причем нет и явно построенного синтаксического блока.

Л Е К Ц И Я № 4

КОНЕЧНЫЕ АВТОМАТЫ

Теория автоматов лежит в основе теории построения компиляторов. Под автоматом подразумевают не реально существующее устройство, а некоторую математическую модель, свойства и поведение которой можно изучать и которую можно имитировать с помощью программы на реальной, вычислительной машине. Конечные автоматы (КА) обладают рядом свойств, которые позволяют легко использовать их при решении некоторых задач компиляции. Это следующие свойства:

1.Конечный автомат может решать легкие задачи компиляции. В частности, лексический блок зачастую строится на основе конечного автомата.

2.Поскольку при моделировании конечного автомата на ЭВМ обработка одного символа требует небольшого количества операций, программа работает быстро.

3.Моделирование конечного автомата требует фиксированного объема памяти, что упрощает проблемы, связанные с управлением памятью.

4.Существует ряд теорем и алгоритмов, позволяющих конструировать

и упрощать конечные автоматы, предназначенные для тех или иных целей.

Имеются различные определения конечного автомата. Общим в них является то, что они моделируют вычислительные устройства с фиксированным и конечным объемом памяти, которые читают последовательности входных символов, принадлежащих некоторому конечному множеству. Различия в определениях связаны с тем, что автоматы делают на выходе. Наиболее простыми являются автоматы, единственный выход которых – является указание на то, допустима или нет данная входная цепочка (последовательность символов). Допустимая – правильно построенная или синтаксически правильная цепочка. Такие автоматы называют конечными распознавателями.

Примером задачи распознавания может служить проверка нечетности числа единиц в произвольной цепочке, состоящей из нулей и единиц. Соответсвующий конечный автомат, называемый контроллером нечетности, будет допускать все цепочки, содержащие нечетное количество единиц, и отвергать цепочки с четным их числом.

На вход конечного автомата подается цепочка символов из конечного множества, называемого входным алфавитом автомата и представляющего собой совокупность символов, для работы с которыми он предназначен. Как допускаемые, так и отвергаемые цепочки состоят только из символов входного адфавита. Символы, не принадлежащие к входному алфавиту, нельзя подавать на вход автомата. Входной автомат контроллера нечетности состоит из двух символов – 0 и 1.

В каждый момент автомат имеет дело с одним входным символом, а информацию о предыдущих символах входной цепочки сохраняет с помощью конечного множества состояний. Согласно этому представлению автомат помнит о прочитанных раннее символах только то, что при обработке он перешел в некоторое состояние, которое является памятью автомата о прошлом.

Контроллер нечетности должен запоминать, четное или нечетное количество единиц ему встретилось. Поэтому множество состояний автомата со-

держат два состояния, называемые ЧЕТ и НЕЧЕТ.

Одно из состояний должно быть выбрано в качестве начального, в котором автомат начинает работу. Начальным состоянием контроллера нечетности будет ЧЕТ, так как на первом шаге число прочитанных единиц равно нулю, а это число четное.

При чтении очередного входного символа состояние автомата меняется, причем новое зависит от входного символа и текущего состояния. Такое изменения состояния называется переходом. Новое состояние может совпадать со старым.

Работу автомата можно описать с помощью функции δ, называемой функцией перехода. По текущему состоянию $S_{тек}$ и текущему входному символу x она дает новое состояние автомата $S_{нов}$

Определим функцию переходов контроллера нечетности следующим образом:

δ (ЧЕТ, 0)=ЧЕТ

δ (ЧЕТ, 1)=НЕЧЕТ

δ (НЕЧЕТ, 0)=НЕЧЕТ

δ (НЕЧЕТ, 1)=ЧЕТ

Некоторые состояния автомата выбираются в качестве допускающих или заключительных. Если автомат, начав работу в начальном состоянии, при прочтении всей цепочки переходит в одно из допускающих состояний, говорят, что эта входная цепочка допускается автоматом. Если последнее состояние автомата не является допускающим, говорят, что автомат отвергает цепочку. Контроллер нечетности имеет единственное допускающее состояние – НЕЧЕТ.

Конечный автомат задается:

1) конечным множеством входных символов;

2) конечным множеством состояний;

3) функцией перехода F, которая каждой паре, состоящей из входного

символа и текущего состояния, приписывает некоторое новое состояние;

4) состоянием, выделенным в качестве начального;

5) подмножеством состояний, выделенных в качестве допускающих или заключительных.

Один из удобных способов представления конечных автоматов – таблица переходов. Информация размещается в таблице переходов в соответствии со следующими соглашениями.

1. Столбцы помечены входными символами.

2. Строки помечены символами состояний.

3. Элементами таблицы являются символы новых состояний, соответствующих входным символам столбцов и состояниям строк.

4. Первая строка помечена символом начального состояния.

5. Строки, соответствующие допускающим состояниям, помечены справа нулями.

Таблица переходов для контроллера нечетности представлена на рис.2.

	0	1	
ЧЕТ	ЧЕТ	НЕЧЕТ	0
НЕЧЕТ	НЕЧЕТ	ЧЕТ	1

Рис.2.

Она задает онечный автомат у которого:

входное множество={0, 1},

множество состояний = {ЧЕТ, НЕЧЕТ},

переходы δ (ЧЕТ, 0)=ЧЕТ, и т.д.,

начальное состояние = ЧЕТ,

допускающие состояния={НЕЧЕТ}.

На рис.3 представлен еще один пример конечного автомата. Переход автомата из состояния $S_{тек}$ в состояние $S_{нов}$при чтении входного символа а будем обозначать так:

а

$$S_{тек} \rightarrow S_{нов}$$

Входная цепочка допускается автоматом, если он, начав работу в начальном сос

	x	y	z	
1	1	3	4	1
2	2	1	3	0
3	2	4	4	1
4	3	3	3	0

Входное множество –{x, y, z}

Множество состояний ={1, 2, 3, 4}

Функция переходов δ (l, x)=1; δ (1,y)=3 и т.д.

Начальное состояние – 1.

Допускающее состояние {1, 3}.

Рис.3.

состоянии, при прочтении всей цепочки переходит в одно из допускающих состояний. Если последнее состояние автомата не является допускающим, говорят, что автомат отвергает цепочку.

Для описанного автомата цепочка xyzz допустима, так как

$$1 \xrightarrow{x} 1 \xrightarrow{y} 3 \xrightarrow{z} 4 \xrightarrow{z} 3$$

и 3 является допускающим состоянием, тогда как цепочка zyx отвергается, потому что

$$1 \xrightarrow{z} 4 \xrightarrow{y} 3 \xrightarrow{x} 2$$

и 2 – отвергающее состояние.

Для построения конечного автомата необходимо знать множество всех цепочек, распознаваемых автоматом, которое обычно называют регулярным.

Регулярное множество в алфавите T (T – конечный алфавит) определяется рекурсивно следующим образом:

1) {} (пустое множество) – регулярное множество в алфавите T;

2) $\{a\}$ – регулярное множество в алфавите T для каждого *a* из алфавита T;

3) {*e*} –регулярное множество в алфавите T (*e* – пустая цепочка);

4) если P и Q – регулярные множества в алфавите T, то таковы и множества:

а) PUQ (объединение),

б) P*Q (конкатенация) pq p из множества P, q из множества Q,

в) (P)*(итерация) (P)*={e}UPUP*PU...

5) ничто другое не является регулярным множеством в алфавите T. Множество в алфавите T регулярно тогда и только тогда, когда оно либо {}, {e}, {a}, а принадлежит множеству T, либо его можно получить применением конечного числа операций (объединение, конкатенация, итерация).

Примеры регулярных выражений и обозначаемых ими множеств.

Идентификатор=Буква (БукваUцифра)*

Буква={ a,b...z}

Цифра={0,1 ... 9}

Для каждого регулярного множества можно найти по крайней мере одно регулярное выражение, обозначающее это множество. И обратно: для регулярного выражения можно построить регулярное множество, обозначаемое этим выражением. Для каждого регулярного множества существует бесконечное число обозначающих его регулярных выражений.

Л Е К Ц И Я № 5

ПОЛУЧЕНИЕ МИНИМАЛЬНОГО АВТОМАТА

Произвольный конечный автомат можно превратить в эквивалентный ему минимальный, выбрасывая недостижимые состояния и объединяя эквивалентные.

Применительно к конечным распознавателям, назначение которых –

допускать цепочки, эквивалентность состояний можно определить следующим образом. Состояние s конечного распознавателя M эквивалентно состоянию t конечного распознавателя N тогда и только тогда, когда автомат M, начав работу в состоянии s, будет допускать в точности те же, что и автомат N, начавший работу в состоянии t.

Если два состояния s и t одного автомата эквивалентны, то автомат можно упростить, заменив в таблице переходов все вхождения имен этих состояний каким-нибудь новым именем, а затем удалив одну из двух строк, соответствующих s и t. Например, состояния 4 и 5 КА, изображенные на рис.4, явно имеют одинаковые функции, так как оба являются допускающими, оба переходят в состояние 2 при чтении входного символа a и оба переходят в состояние 3 при чтении b.

	a	b	
1	1	4	0
2	3	5	1
3	5	1	0
4	2	3	1
5	2	3	1

Рис.4.

Поэтому можно объединить состояния 4 и 5 в одно состояние и назвать его X. Получается упрощенная таблица состояний, представленная на рис.5.

	a	b	
1	1	4	0
2	3	5	1
3	5	1	0
X	2	3	1

Рис.5.

Обычно эквивалентность состояний менее очевидна. Существует следующее ее определение: два состояния эквивалентны тогда и только тогда,

когда не существует различающей их цепочки. Понятие «эквивалентность состояний» является отношением в математическом смысле; это отношение рефлексивно (каждое состояние эквивалентно себе), симметрично (если s эквивалентно t, то t эквивалентно s) и транзитивно (если s эквивалентно t, а t эквивалентно u, то s эквивалентно u).

Метод проверки эквивалентности состояний основывается на следующем факте. Состояния s и t эквивалентны тогда и только тогда, когда выполняются два условия:

1) условие подобия – состояния s и t должны быть либо оба допускающие, либо оба отвергающие,
2) условие преемственности – для всех входных сисмволов состояния s и t должны переходить в эквивалентные состояния, т.е. их преемники эквивалентны.

Эти условия выполняются тогда и только тогда, когда s и t не имеют различающей цепочки. Если нарушено одно из условий, существует цепочка, различающая эти два состояния. А если не выполняется условие подобия, различающей является пустая цепочка. Если нарушено условие преемственности, то некоторый входной символ x переводит из состояния s и t в неэквивалентные. Поэтому x с приписанной к нему цепочкой, различающей эти новые состояния, образует цепочку, различающую s и t.

Условия 1 и 2 можно использовать в общем методе проверки на эквивалентность произвольной пары состояний. Для этого строятся таблицы эквивалентности состояний. Рассмотрим КА, таблица переходов которого изображена на рис.6, и выявим эквивалентные состояния.

Проверим на эквивалентность состояния 0 и 7. Таблица эквивалентности состояний содержит по одному столбцу для каждого входного символа, – для y и z. Строки будут добавляться в ходе проверки. Первоначально такая таблица имеет вид, представленный на рис.7: имеется одна строка, которая помечена парой состояний, подвергаемых проверке, т.е. парой 0, 7.

	y	z	
0	0	3	0
1	2	5	0
2	2	7	0
3	6	7	0
4	1	6	1
5	6	5	0
6	6	3	1
7	6	3	0

Рис.6.

Сперва необходимо попытаться продемонстрировать неэквивалентность состояний, показав, что нарушается условие подобия. В данном случае это условие выполняется, так как оба состояния являются отвергающие.

	y	z
0, 7		

Рис.7.

Может быть будет нарушено условие преемственности? Чтобы исследовать эту возможность, рассмотрим, как действует на отдельную пару состояний каждый входной символ, и запишем результат в соответствующую ячейку таблицы. Так как состояние 0, 7 под действием входного символа y переходит в состояние 0 и 6 соответственно, то 0, 6 записывается в столбец таблицы, соответствующий символу y. Так как оба состояния 0 и 7 переводятся символом z в состояние 3, то 3 записывается в столбец для z. Таблица эквивалентности примет вид, представленный на рис.8.

Чтобы нарушалось условие преемственности, должны быть неэквивалентны либо состояние 0 и 6, либо состояния 3 и 3. Так как каждое состояние эквивалентно само себе, состояния 3 и 3 эквивалентны автоматически.

y z

0, 7	0, 6	3

Рис. 8.

Чтобы исследовать на эквивалентность состояния 0 и 6, к таблице эквивалентности состояний добавляется новая строка и помечается новой парой 0, 6. Для нее повторяется весь процесс, описанный для пары 0,7. Сперва проверяется условие подобия для пары 0, 6. Оказывается, что 0 и 6 неэквивалентны, так как 6 – допускающее, а 0 – отвергающее состояние. Проверка показала, что состояния 0 и 7 неэквивалентны. Таблицу эквивалентности состояний можно использовать для построения различающей цепочки. Строка 0,6 появилась как результат применения входного символа у к паре 0,7, поэтому у является различающей цепочкой.

Проверим на эквивалентность пару состояний 0, 1. Построение таблицы эквивалентности начинается с пары 0, 1. Эти состояния подобны, поэтому вычисляется результат применения к ним каждого входного символа; полученные состояния помещаются в таблицу. Надежды на эквивалентность 0, 1 оправдаются, если будет установлена эквивалентность пар, помещенных в таблицу, 0, 2 и 3, 5. В таблицу добавляется строка для каждой из этих пар. Результат изображен на рис.9.

Обратившись к строке 0, 2, можно определить, что эти состояния подобны, поэтому можно вычислить следующие пары, чтобы проверить условие преемственности. Результат отображен на рис.10. Из двух новых элементов один дает новую строку, а именно пара 3, 7. Другой элемент уже имеется в таблице, и нет надобности его проверять. Далее по списку проверяется пара 3, 5. Состояния 3, 5 подобны. Вычисляем пары 6, 6 и 5, 7. Так как состояние 6 эквивалентно самому себе, единственной новой строкой в таблице будет 5, 7. Таблица эквивалентности примет вид, представленный на рис.11.

	y	z
0, 1	0, 2	3, 5

	y	z
0, 1	0, 2	3, 5
0, 2		
3, 5		

Рис.9.

	y	z
0, 1	0, 2	3, 5
0, 2	0, 2	3, 7
3, 5		

Рис.10.

	y	z
0, 1	0, 2	3, 5
0, 2	0, 2	3, 7
3, 5	6	5, 7
3, 7		
5, 7		

Рис.11.

Описанные процедуры повторяются для оставшихся пар – 3, 7 и 5, 7. По их окончании не удается получить ни одной новой пары неподобных состояний и ни одной пары, которую надо проверять на подобие. Окончательная таблица представлена на рис.12.

	y	z
0, 1	0, 2	3, 5
0, 2	0, 2	3, 7
3, 5	6	5, 7
3, 7	6	3, 7
5, 7	6	3, 5

Рис.12.

Различающие цепочки для состояний 0, 1 отсутствуют, поэтому эти состояния являются эквивалентными.

Алгоритм проверки эквивалентности двух состояний можно описать следующим образом.

1.Начать построение таблицы эквивалентности состояний с отведения столбца для каждого входного символа. Пометить первую строку парой проверяемых состояний.

2.Выбрать в таблице эквивалентности состояний строку, ячейки которой еще не заполнены, и проверить, подобны ли состояния, которыми она помечена. Если они не подобны, то два исходных состояния неэквивалентны, и процедура оканчивается. Если они подобны, следует вычислить результат применения каждого входного символа к этой паре состояний и записать полученные пары состояний в соответствующие ячейки рассматриваемой строки.

3.Для каждого элемента, полученного на шаге 2, существуют три возможности. Если элементом таблицы является пара одинаковых состояний, для нее не требуется никаких действий. То же самое – если элементом таблицы является пара состояний, которые уже использовались как метки строк. Если же элемент таблицы – пара разных состояний, которая еще не использовалась как метка, для нее нужно добавить новую строку. Порядок состояний в паре не важен, и пары s, t и t, s считаются одинаковыми. После того, как произведены необходимые действия для каждой пары состояний в данной строке, выполняется следующий шаг.

4.Если все строки таблицы эквивалентности заполнены, исходная пара состояний и все пары, порожденные в ходе проверки, эквивалентны, проверка закончена. Если же таблица не заполнена, нужно обработать еще по крайней мере одну ее строку и применить шаг 2.

Так как каждая пара, появившаяся в заполненной таблице, содержит эквивалентные состояния, этот метод проверки дает обычно больше информа-

ции, чем предполагалось сначала. На рис. 12 видно, что, кроме эквивалентности пары (0,1), которая подвергалась проверке, доказана эквивалентность пар (0,2), (3,5), (3,7), (5,7). По свойству транзитивности из эквивалентности пар состояний 0,2 и 0,1 и следует эквивалентность пары 1, 2. Таким образом, состояния 0, 1, 2 эквивалентны друг другу. Аналогично эквивалентны друг другу состояния 3, 5, 7.

Автомат можно упростить , объединив состояния 0, 1, 2 в состояние A, а 3, 5, 7 – в состояние B. Новые имена подставляются в таблицу переходов, лишние строки удаляются и получается более простой – эквивалентный, изображенному на рис.6 . Новый автомат представлен на рис.13.

	y	Z	
A	A	B	0
B	6	B	0
4	A	6	1
6	6	B	1

Рис. 13.

Чтобы упростить автомат необходимо, также удалить из него состояния, которые недостижимы из начального состояния ни для какой входной цепочки.

Л Е К Ц И Я № 6

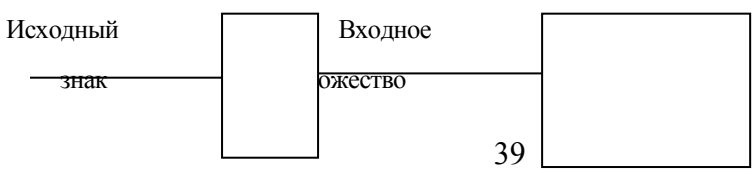
РЕАЛИЗАЦИЯ КОНЕЧНЫХ АВТОМАТОВ

Чтобы смоделировать конечный автомат, необходимо каким-нибудь подходящим способом закодировать его входное множество. Наиболее гибкое решение состоит в представлении входного множества с помощью набора последовательных целых чисел, начиная с 0 или 1. Если входом конечного автомата является выход какого-то другого блока компилятора, то обычно нетрудно по-

строить этот блок так, чтобы он подавал на вход конечного автомата цепочки в наиболее удобном для него виде. Единственное место при построении компилятора, где может возникнуть проблема кодировки, – это процедура чтения символов исходной программы. Если бы множество символов исходной программы непосредственно служило входом для некоторого конечного процессора, содержащего много состояний, то автомат имел бы большую таблицу переходов, так как одни только цифры и латинские буквы образуют множество из 36 символов, обычно же множество символов в несколько раз большее. Удобный способ уменьшения размера таблицы переходов – обработка исходных символов двумя последовательно соединенными автоматами: первый – транслитератор, единственная задача которого — сократить входное множество до приемлемых размеров; второй – выполняет остальную часть работы. Этот вид взаимодействия показан на рис.14, а.

Зависимость между входом и выходом транслитератора можно выразить с помощью таблицы, – например, такой, как на рис.14, б, содержащей перевод каждого символа исходного языка. Выход этого транслитератора называется символьной лексемой. Такая лексема обычно состоит из двух частей – класса и значения. Так, буква *a* будет иметь класс БУКВА и значение *a*, тогда как знак операции + имеет класс ОПЕРАЦИЯ и значение +. Класс лексемы должен служить входом для второго автомата, а ее значение доступно процедурам переходов этого автомата.

Транслитератор — это процессор с одним состоянием, который реализуется как процедура поиска в таблице перевода для каждого входного символа. На многих машинах это можно сделать одной или двумя командами. Множество классов символьных лексем обычно представляется с помощью последовательных натуральных чисел, так как они являются возможными входными символами для второго автомата.



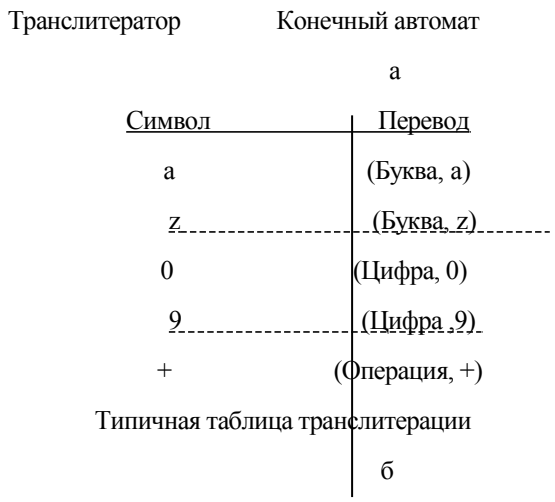


Рис. 14.

Еще один важный вопрос при реализации конечного автомата – представление состояний. Есть два основных способа, с помощью которых программа, моделирующая конечный автомат, может запоминать состояние моделируемого автомата. Первый заключается в запоминании номера, соответствующего текущему состоянию в некотором регистре или ячейке памяти вычислительной машины; этот способ называется явным, поскольку состояние явно задается некоторой переменной. Второй основной метод заключается в том, что для каждого состояния имеется отдельная часть программы. Тот факт, что моделируемый автомат находится в заданном состоянии, «запоминается» тем, что моделирующая программа исполняет часть кода, которая «принадлежит» этому состоянию. Такой метод называется неявным. Тот или иной метод выбирается, исходя из удобства программирования.

При реализации конечного автомата необходимо решить вопрос о выборе переходов. Суть программы, моделирующей конечный процессор, состоит в способе выбора переходов, так как для заданного состояния и очередного входного символа программа должна выполнить переход, указанный в таблице переходов. Поэтому информацию, содержащуюся в таблице переходов, следует где-то хранить, или же включить ее в моделирующую программу.

Предположим, что состояние запоминается неявно. В этом случае косвенно известна нужная строка таблицы переходов и задача сводится к нахожде-

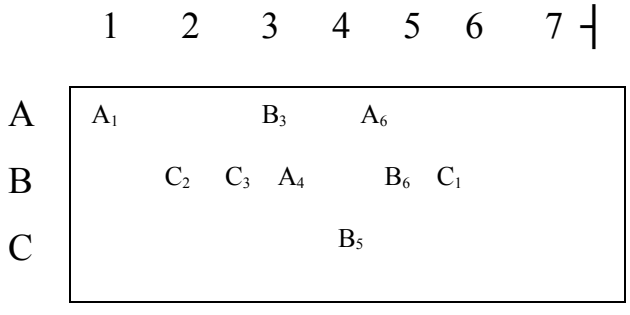
нию перехода только по входному символу. Такая задача должна, разумеется, решаться для каждого состояния в отдельности.

Существуют два метода решения задачи выбора перехода для заданного входного символа: метод «вектор переходов» и метод «список переходов».

Согласно методу вектора переходов адреса или метки тех процедур переходов, на которые должно передаваться управление, хранятся в виде вектора в последовательных ячейках памяти – по одной ячейке для каждого входного символа. Очередной входной символ служит индексом, по которому выбирается элемент вектора, дающий нужный переход. Чтобы этот метод работал, входное множество должно быть представлено каким-нибудь подходящим образом, – например, в виде множества последовательных целых чисел.

Пример такого вектора приведен на рис. 15, б, где изображен вектор переходов для состояния *A* на рис. 15, а. Достоинство метода в том, что нужную процедуру перехода можно выбрать очень быстро. Затраты памяти — по ячейке на каждый элемент строки. В большинстве языков программирования высокого уровня этот метод можно реализовать, используя переключатель или вычисляемый переход.

Согласно методу списка переходов входные символы делятся на два класса: каждому входному символу первого класса приписывается индивидуальный переход, а все символы второго класса имеют общий переход (обычно процедуру, обрабатывающую ошибку). Для первого класса соответствие между входным символом и адресом процедуры перехода задается в виде списка упорядоченных пар. Общий переход для символов из второго класса запоминается отдельно и называется «переходом по неудаче».



а

Адрес A_1	Адрес процедуры обработки ошибки	Адрес процедуры обработки ошибки	Адрес A_1	Адрес процедуры обработки ошибки	Адрес A_1	Адрес процедуры обработки ошибки	Адрес процедуры обработки ошибки

б

Входной Символ	Переход
1	A_1
6	A_6
4	B_3

Переход по неудаче = Процедура обработки ошибки

в

Рис. 15 (а) Построение процессора, (б) вектор переходов,.(в) список переходов.

При поступлении нового входного символа происходит поиск в списке этого символа и соответствующего ему перехода. Если поиск заканчивается неудачей, переходят к процедуре, соответствующую неудаче. Этот метод можно применять, даже если входной символ не является индексом.

На рис.15в показан список переходов для состояния А процессора, изображенного на рис. 15, а. Переход по неудаче вызывают входные индексы 2, 3, 5, 7 и концевой маркер; тем самым пять элементов таблицы объединяются в один переход.

Среднее время, необходимое для выбора перехода методом списка переходов, зависит от длины списка и относительной частоты, с которой входные символы встречаются в исходной программе. Естественно, более выгодно помещать пары, содержащие наиболее часто встречающиеся входные символы, в

начале списка. Тем не менее этот метод работает медленнее, чем метод вектора переходов, а при увеличении длины списка – еще медленнее. Метод списка требует незначительных затрат памяти, когда список короток, однако при длинном списке затраты памяти бывают больше, чем для метода вектора, так как память отводится и для метки процедуры перехода, и для символа, вызывающего этот переход. Таким образом, данным методом лучше всего пользоваться, когда память дорога, а таблица переходов содержит много стандартных переходов, связанных с ошибкой.

Разумеется, при моделировании автомата можно пользоваться смешанным методом. В примере на рис. 15,а при выборе переходов из состояния В был бы предпочтителен метод вектора переходов, тогда как для состояния С метод списка переходов сэкономил бы много места.

Если состояние моделируемого автомата хранится в явном виде, принципы указанных методов векторов и списков по-прежнему применимы, но возможны многочисленные варианты. Одна из возможностей состоит в том, чтобы пользоваться состоянием как индексом для передачи управления одной из уже описанных процедур; другая – в том, чтобы хранить таблицу переходов как единый двумерный массив, индексы которого – это состояния и входные символы. Можно также использовать метод списка для выбора элементов из столбцов, а не из строк таблицы переходов.

В процессе работы компилятор хранит информацию об объектах программы в таблицах. При организации таблиц возникает проблема идентификации слов – распознавание имен идентификаторов и соотнесение их с соответствующими элементами таблиц. Существуют следующие методы идентификации слов: индексов, линейного списка, упорядоченного списка и расстановки. Рассмотрим сущность, недостатки и преимущества каждого из них.

Метод индексов. По входной строке (имя идентификатора) вычисляется индекс, который обеспечивает прямой доступ к элементу таблицы. Таблицу, организованную по этому методу, называют индексированной. Она аналогична одномерному массиву, причем индекс слова служит индексом

компоненты массива.

При использовании этого метода число слов в таблице не должно быть большим; индекс должен легко вычисляться; объем множества слов фиксируется при построении компилятора. Преимущество его в том, что идентификация осуществляется с минимальными затратами времени.

Метод линейного списка –наиболее прямой метод идентификации слов, заключающийся в сравнении входного слова с элементами таблицы до тех пор, пока не произойдет совпадения (если оно возможно).Таблица легко расширяется путем добавления слов на свободные места. Недостаток метода – поиск по длинному списку занимает много времени.

Метод упорядоченного списка. Элементы таблицы упорядочены, – например, лексикографически, т. е. по алфавиту. Предположим, что заданная таблица содержит M элементов. Для предыдущего метода, чтобы найти слово, потребуется (M+1)/2 сравнений, а для слова, которого нет в таблице, – M сравнений. Воспользовавшись упорядоченностью списка, можно уменьшить число сравнений до loqM. Делается это так. Поиск начинают сравнением входного слова со словом, расположенным в середине списка. Если слова совпадают, поиск окончен. В противном случае сравнение показывает, где следует искать входное слово – до или после середины списка.

Дальнейший поиск выполняется в новом списке, который вдвое короче исходного. Поиск продолжается, пока слово не будет найдено или список будет содержать всего один элемент, неравный входному слову, т.е. поиск окончится неудачей.

Недостаток метода – неудобно расширять таблицу, а необходимость вычислить середину списка требует дополнительных затрат времени.

Метод расстановки. По входному слову вычисляется индекс. Он может быть одним и тем же для многих слов. Его можно использовать для нахождения указателей списка. Если указатель равен 0, то входное слово не принадлежит множеству. Иначе элемент таблицы указателей списков указывает на некоторый связанный список слов, индексы которых совпадают с вычис-

ленным. Поиск в этом списке продолжается до тех пор, пока не произойдет совпадения входного слова с элементами списка или не будет достигнут конец списка. В последнем случае множество не содержит входного слова, и это слово можно добавить, просто связав его с последним элементом списка.

Для реализации этого метода необходимо выбрать функцию расстановки – метод вычисления индекса. Обычно используют следующие способы: двоичный код слова представляют как одно число или как ряд чисел, которые каким-нибудь образом комбинируют, чтобы получилось одно число. Затем это число каким-либо способом уменьшают, чтобы получить индекс желаемого размера. Один из способов состоит в использовании остатка от деления числа на некоторую константу – желательно простое число. Другой метод получения индекса – возведение числа в квадрат и выделение средних двоичных разрядов.

Недостаток метода – затраты времени на вычисление индекса, преимущества – множество легко расширяемо; требует умеренных затрат памяти; сравнительно невелико число сравнений для поиска заданного слова.

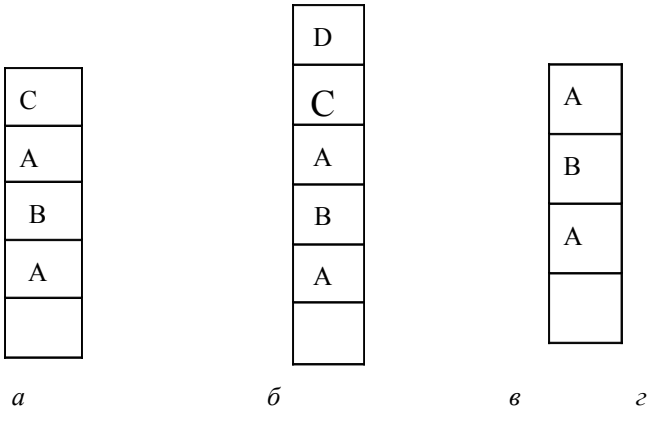
Л Е К Ц И Я № 7

АВТОМАТЫ С МАГАЗИННОЙ ПАМЯТЬЮ

Конечный автомат может решать лишь такие вычислительные задачи, которые требуют фиксированного и конечного объема памяти. В компиляторе, однако, возникает множество задач, которые невозможно решить при таком ограничении, поэтому требуется модель более сложного автомата. Рассмотрим, например, задачу обработки скобок в арифметических выражениях. Арифметическое выражение может начинаться с любого количества левых скобок, и

компилятор должен проверять, имеется ли в выражении такое же число соответствующих правых скобок. Каждый раз самая левая из скобок в выражении будет играть особую роль, так как каждая из таких ролей требует разного числа соответствующих правых скобок для завершения выражения. Иными словами, компилятор должен эффективно подсчитывать левые скобки, чтобы сбалансировать их. Разумеется, конечное множество состояний не годится для запоминания числа необходимых правых скобок, так как множество этих чисел бесконечно. Для систематического решения проблемы, связанной с выражениями, а также для решения многих других проблем компиляции необходимо использовать в компиляторе модели более мощных автоматов.

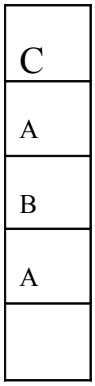
Чтобы получить более мощный автомат, память конечного автомата расширяется за счет дополнительного механизма хранения информации. Один из методов хранения информации – использование магазина или стека. Основная особенность магазинной памяти, с точки зрения работы с нею, состоит в том, что символы можно помещать в магазин и удалять из него по одному, причем удаляемый символ – всегда тот, который был помещен в магазин последним. Когда информация помещается в магазин, говорят, что она в него «вталкивается», когда удаляется из магазина, – что она «выталкивается» из него. Говорят, что информация, только что поступившая в магазин, находится в его верхушке , или наверху.



а *б* *в* *г*

Рис. 16.

На рис. 16 представлены магазины в различных состояниях. Рис.16,а – на дне магазина находится символ ∇ маркер дна магазина – специальный



символ, который помечает начало, а наверху — символ С. Символы расположены в том порядке, в каком они поступали в магазин. Сначала поступил символ нижнее А, затем В, затем верхнее А и, наконец, символ С. Маркер дна магазина является постоянной составляющей магазина. Если втолкнуть в магазин символ D, магазин будет выглядеть так, как показано на рис. 16, б, где D — верхний символ магазина. Если же, наоборот, вытолкнуть из магазина верхний символ С, верхним символом окажется А, и магазин будет выглядеть, как показано на рис. 16, в. В обоих случаях изменениям подвергается только верх магазина, а остальные символы остаются неизменными. Маркер дна никогда не выталкивается из магазина. Так, если верхний символ магазина, как на рис. 16, г, это означает, что других символов в магазине нет. В этом случае говорят, что магазин пуст.

Одна из моделей автомата, где используется магазинный принцип организации памяти, – автомат с магазинной памятью (МП-автомат). В нем очень просто комбинируется память конечного автомата и магазинная память. МП-автомат может находиться в одном из конечного числа состояний и имеет магазин, куда он может помещать и откуда извлекать информацию.

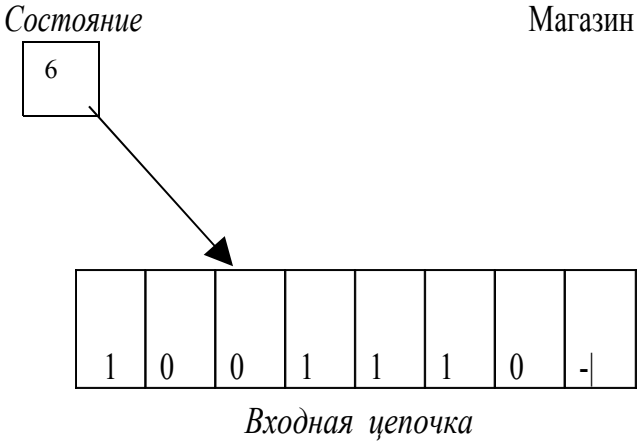


Рис. 17.

Как и в случае конечного автомата, обработка входной цепочки осуществляется рядом мелких шагов. На каждом шаге действия автомата конфигурация его памяти может измениться за счет перехода в новое состояние, а также вталкивания символа в магазин или выталкивания из него. Однако в отличие от конечного автомата МП-автомат может обрабатывать один входной символ в течение нескольких шагов. На каждом шаге управляющее устройство автомата решает, пора ли закончить обработку текущего входного символа и получить, если это так, новый входной символ, либо продолжить обработку текущего символа на следующем шаге. На рис. 17 изображена одна из конфигураций, которая может возникнуть при обработке некоторым гипотетическим МП-автоматом входной цепочки 100110. Для большей наглядности входная цепочка изображена записанной в ячейках файла или ленты с указателем на входной символ, подвергающийся в данный момент обработке.

Каждый шаг процесса обработки задается множеством правил, использующих информацию трех видов: состояние, верхний символ магазина, текущий входной символ.

Это множество правил называется управляющим устройством, или механизмом управления. На рис. 17 информация, поступающая в управляющее устройство, такова: состояние 6, верхний символ магазина С, текущий вход-

ной символ 0.

В зависимости от получаемой информации управляющее устройство выбирает либо выход из процесса (т. е. прекращает обработку), либо переход в новое состояние. Переход состоит из трех операций: над магазином, над состоянием и над входом. Возможные операции могут быть следующими.

Операции над магазином:

1) втолкнуть в магазин определенный магазинный символ.;

2) вытолкнуть верхний символ магазина;

3) оставить магазин без изменений.

Операция над состоянием:

1) перейти в заданное новое состояние;

Операции над входом:

1) перейти к следующему входному символу и сделать его текущим входным символом;

2) оставить данный входной символ текущим, иначе говоря, держать его до следующего шага;

Обработку входной цепочки МП-автомат начинает в некотором выделенном состоянии при определенном содержимом магазина, а текущим входным символом является первый символ входной цепочки. Затем автомат выполняет операции, задаваемые его управляющим устройством. Если происходит выход из процесса, обработка прекращается; если происходит переход, то он дает новый верхний магазинный символ, новый текущий символ – автомат переходит в новое состояние, и управляющее устройство определяет новое действие, которое нужно произвести.

Чтобы управляющие правила имели смысл, автомат не должен требовать следующего входного символа, если текущим символом является концевой маркер, и не должен выталкивать символ из магазина, если это маркер дна. Поскольку маркер дна может находиться исключительно на дне магазина, автомат не должен также вталкивать его в магазин.

На основе всего сказанного МП-автомат определяется следующими пятью объектами:

1) конечным множеством входных символов, включающим входит и концевой маркер;

2) конечным множеством магазинных символов, включающим маркер дна;

3) конечным множеством состояний, включающим начальное состояние;

4) управляющим устройством, которое каждой комбинации входного символа, магазинного символа и состояния ставит в соответствие выход или переход; переход, в отличие от выхода, заключается в выполнении операций над магазином, состоянием и входом, как уже было описано, операции, которые запрашивали бы входной символ после концевого маркера или выталкивали из магазина, а также вталкивали в него маркер дна, исключаются;

5) начальным содержимым магазина, которое представляет собой (при условии, что верхний символ считается расположенным справа) маркер дна, за которым следует (возможно, пустая) цепочка других магазинных символов.

МП-автомат называется МП-распознавателем, если у него два выхода — ДОПУСТИТЬ и ОТВЕРГНУТЬ. Говорят, что цепочка символов входного алфавита (исключая концевой маркер) допускается распознавателем, если под действием этой цепочки с концевым маркером автомат, начавший работу в своем начальном состоянии и с начальным содержимым магазина, делает ряд переходов, приводящих к выходу ДОПУСТИТЬ. В противном случае цепочка отвергается,

При описании переходов МП-автомата будем обозначать действия автомата словами ВЫТОЛКНУТЬ (или, для краткости, ВЫТОЛК), ВТОЛКНУТЬ (или ВТОЛК), СОСТОЯНИЕ, СДВИГ и ДЕРЖАТЬ, причем:

ВЫТОЛКНУТЬ означает вытолкнуть верхний символ магазина;

ВТОЛКНУТЬ (А), где А — магазинный символ, означает втолкнуть символ А в магазин;

СОСТОЯНИЕ(s), где s — состояние, означает, что следующим состояни-

ем становится s;

СДВИГ означает, что текущим входным символом становится следующий входной символ. В некоторых реализациях это может означать сдвиг указателя на входе;

ДЕРЖАТЬ означает, что текущий входной символ надо держать до следующего шага, т. е. оставить его текущим (в некоторых реализациях — оставить указатель на прежнем месте); если автомат содержит в точности одно состояние, слово СОСТОЯНИЕ будем опускать.

Применим МП-распознаватель к проблеме скобок. Каждый раз, когда встречается левая скобка, в магазин будет вталкиваться символ А. Когда будет обнаружена соответствующая правая скобка, символ А будет выталкиваться из магазина. Цепочка отвергается, если на входе остаются правые скобки, а магазин пуст (т. е. во входной цепочке есть лишние правые скобки) или если цепочка прочитана до конца, а в магазине остаются символы А (т. е. входная цепочка содержит лишние левые скобки). Цепочка допускается, если к моменту прочтения входной цепочки до конца магазин опустошается. Полное определение таково.

1. Входное множество {(,), —|}.

2. Множество магазинных символов {А, ▽ }.

3. Множество состояний {s}, где s — начальное состояние.

4. Переходы:

(,A,s = ВТОЛКНУТЬ(А), СОСТОЯНИЕ(s), СДВИГ
(, ,▽= ВТОЛКНУТЬ(А), СОСТОЯНИЕ(s), СДВИГ
),A, s= ВЫТОЛКНУТЬ, СОСТОЯНИЕ(s), СДВИГ
), ,▽= ДЕРЖАТЬ
-|, A, s= ДЕРЖАТЬ
-|, ▽s=ДОПУСТИТЬ

5. Начальное содержимое магазина – ▽

Работа этого МП-автомата изображена на рис. 18, где показан каждый шаг процесса обработки, начиная с начальной конфигурации (а) и кончая допускающей конфигурацией (ж). Такое изображение последовательности конфигураций МП-автомата требует много места, поэтому МП-автомат может быть

представлен в более компактном виде, как на рис.18.

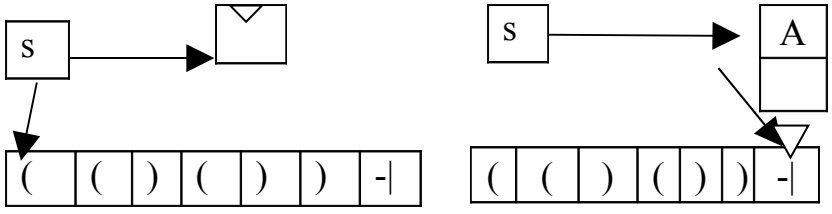
а:∇	[s]	(( ) ( )) -|
б:∇A	[s]	( ) ( )) -|
в :∇ AA	[s]	) ( )) -|
г :∇ A	[s]	( )) -|
д:∇̄ AA	[s]	)) -|
е :∇̄ A	[s]	) -|
ж:∇̄	[s]	-|
ДОПУСТИТЬ		

Рис.18.

Управляющее устройство этого автомата с одним состоянием можно представить в виде управляющей таблицы, как на рис. 20, где показаны действия автомата для каждого сочетания входного символа и верхнего символа магазина. Столбцы таблицы обозначены входными символами, а на пересечении строк и столбцов обозначены соответствующие им действия. Так как этот конкретный автомат имеет лишь одно состояние, информация о состоянии опущена.

Таблица такого вида (т. е. со столбцами для входных символов и строками для символов магазина) является стандартным представлением МП-автоматов с одним состоянием.

Обрабатываемая цепочка (( ) ( ))



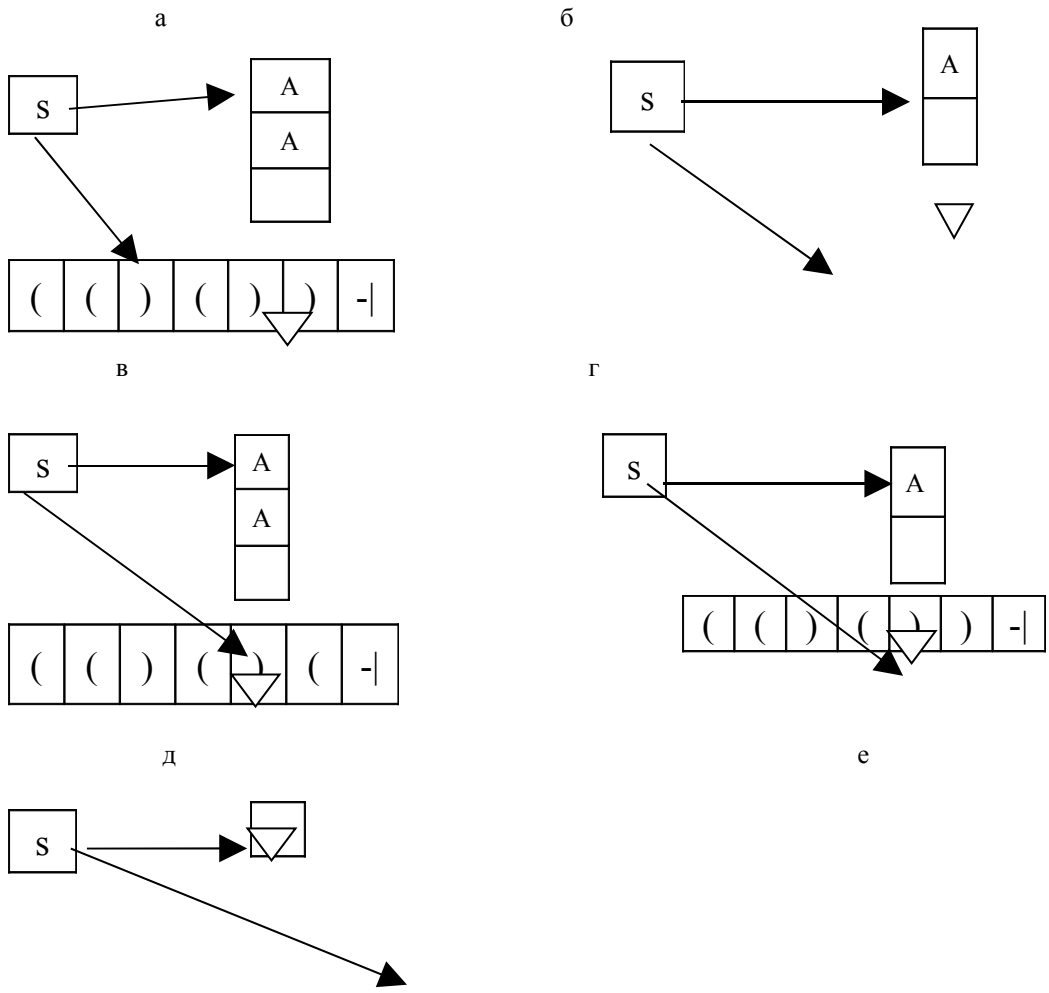


Рис. 19.

(( ) ( )) -|

( ) ⊣

A	ВТОЛКНУТЬ (A) СДВИГ	ВЫТОЛК-НУТЬ СДВИГ	ОТВЕРГНУТЬ
$\bar{V}$	ВТОЛКНУТЬ (A) СДВИГ	ОТВЕРГ-НУТЬ	ДОПУСТИТЬ

Рис. 20.

Л Е К Ц И Я № 8

МП-ТРАНСЛЯТОРЫ

МП-автомат называется МП-транслятором, если при распознавании он порождает выходную цепочку. Чтобы автомат выдавал выходную цепочку, управляющее устройство может наряду с обычными операциями над состоянием, входом и магазином производить операцию на выходе. При отсутствии выходной операции предполагается, что на выход ничего не выдается. Если надо выдать цепочку AB, то в определении соответствующего МП-перехода мы пишем

ВЫДАТЬ (A B).

Чтобы посмотреть, как можно пользоваться операцией ВЫДАТЬ, предлагается решить задачу перевода произвольной цепочки из нулей и единиц в цепочку вида $(1)^n(0)^m$ где n и m – соответственно число единиц и нулей в данной цепочке. Например, цепочка 0 1 1 0 1 1 будет переведена в 1 1 1 1 0 0, так как в этой цепочке четыре единицы и два нуля.

Один из способов такого перевода заключается в том, чтобы выдавать единицы сразу при их появлении на входе, а при появлении на входе нулей помещать их в магазин. Когда встречается концевой маркер, автомат выталкивает из магазина нули и выдает их на выход. Управляющая таблица для автомата с одним состоянием, реализующего этот способ, изображена на рис.21.

	0	1	|-
A	Втолкнуть (A) Сдвиг	Выдать(1) Сдвиг	выдать(0) держать
▽	Втолкнуть (A) Сдвиг	Выдать(1)	Допустить

Начальное содержимое магазина ∇

Рис.21.

Последовательность конфигураций этого автомата при обработке цепочки 0 1 0 1 1 показана на рис. 22. В данном случае магазин служит не для распознавания, а только для перевода. Нули вталкиваются в магазин только для того, чтобы позже автомат выдал их на выходе.

1: ∇ *01011 -|*

2: ∇ *A 1011 -|*

3: ∇ *A ВЫДАТЬ(1) 011 -|*

4: ∇ *AA 1 1 - |*

5: ∇ *AA ВЫДАТЬ(1) 1 - |*

6: ∇ *AA ВЫДАТЬ(1) - |*

7: ∇ *A ВЫДАТЬ(0) -|*

8: ∇ *ВЫДАТЬ(0) -| ДОПУСТИТЬ*

Рис.22

Л Е К Ц И Я № 9

КОНТЕКСТНО-СВОБОДНЫЕ ГРАММАТИКИ

Многие понятия при рассмотрении формальных языков и грамматик аналогичны понятиям, которые используются при изучении естественных языков, – например, английского или русского.

Наиболее фундаментально понятие языка. С теоретической точки зрения

слово «язык» — синоним термина «множество цепочек». Так, язык Си++ можно понимать как множество цепочек, задаваемое некоторым множеством правил. Наиболее интересные языки (такие, как Си++) состоят из бесконечного множества цепочек.

Чтобы отличать употребление слова «язык» в значении точно определенного множества цепочек от употребления этого слова в повседневной речи, множество цепочек называют формальным языком.

Чтобы применить математический подход к проблемам, связанным с языками и их обработкой, необходимо ограничиться множествами цепочек, которые можно определить некоторым точным образом. Есть много способов точного задания таких множеств. Один из них, например, заключается в задании языка как множества, допускаемого каким-нибудь распознавателем цепочек вроде конечного автомата или автомата с магазинной памятью. Другой подход состоит в использовании методов, которые можно считать грамматическими.

Термин «формальная грамматика» применим к любому определению формального языка, основанному на «грамматических правилах», с помощью которых можно порождать и анализировать цепочки аналогично тому, как грамматики используются при изучении естественных языков. Среди формальных грамматик выделяется особый вид, называемый контекстно-свободными грамматиками.

Далее приводится формальная грамматика, которая в какой-то степени напоминает фрагмент грамматики русского языка и задает формальный язык, состоящий из четырех русских предложений. В этой формальной грамматике используются элементы, играющие роль членов предложения или частей речи:

<предложение>

<подлежащее>

<сказуемое>

<дополнение>

<прилагательное>

<существительное>

Чтобы отличать эти элементы от слов из фактического словаря, составляющих предложения языка, их заключают в угловые скобки. В данном примере словарь состоит из пяти слов, или «символов»: ДОМ, ДУБ, ЗАСЛОНЯЕТ, СТАРЫЙ, (точка).

В грамматике есть определенные правила, содержащие информацию о том, как из этих символов можно строить предложения языка. Одно из этих правил таково:

1.<предложение>→ <подлежащее> <сказуемое> <дополнение>.

Это правило интерпретируется следующим образом: «Предложение может состоять из подлежащего, за которым следуют сказуемое, затем дополнение и точка». В грамматике могут быть и иные правила, задающие предложения другой структуры, однако в данной грамматике таких правил нет. Приведем остальные правила данной грамматики:

2.<подлежащее> → <прилагательное> <существительное>

3.<дополнение> → <прилагательное> <существительное>

4.<сказуемое> →ЗАСЛОНЯЕТ

5.<прилагательное> →СТАРЫЙ

6.<существительное >→ ДОМ

7.<существительное >→ДУБ

Такую грамматику можно применить для порождения (или вывода) предложения. По правилу 1 предложение имеет вид

<подлежащее> <сказуемое> <дополнение>.

Так как согласно правилу 2, подлежащим может быть комбинация

<прилагательное> <существительное>

ее можно подставить вместо подлежащего и получить предложение, которое имеет вид

<прилагательное> <существительное> <сказуемое><дополнение>.

Аналогичным образом можно применить правило 3, чтобы заменить <дополнение> и получить

<прилагательное><существительное><сказуемое> <прилагательное > <существительное>.

Теперь можно дважды применить правило 4, чтобы, заменив (прилагательное), получить

СТАРЫЙ <существительное> <сказуемое> СТАРЫЙ <существительное >.

Применяя правила 6 и 7, заменяющие первое и второе (существительное), и правило 4, заменяющее (сказуемое), получаем готовое предложение:

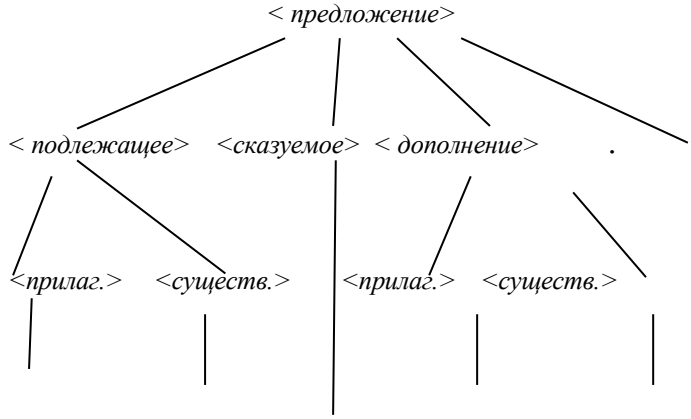
СТАРЫЙ ДОМ ЗАСЛОНЯЕТ СТАРЫЙ ДУБ.

Этот вывод можно наглядно изобразить в виде дерева (рис. 23), которое показывает, какие правила применялись к различным промежуточным элементам, но скрывает порядок их применения.

Таким образом, можно видеть, что результирующая цепочка не зависит от порядка, в котором делались замены промежуточных элементов. Иногда говорят, что дерево представляет собой «синтаксическую структуру» предложения.

Идея вывода подсказывает другие интерпретации правил, подобных правилу

<подлежащее> -> <прилагательное><существительное >



СТАРЫЙ ДОМ ЗАСЛОНЯЕТ СТАРЫЙ ДУБ

Рис. 23.

Вместо того чтобы говорить «<подлежащее> – это <прилагательное>, за которым следует <существительное>», можно сказать, что <подлежащее> «порождает» (или «из него выводится», или «его можно заменить на») <прилагательное> <существительное>.

С помощью этой грамматики можно вывести также три других предложения:

СТАРЫЙ ДУБ ЗАСЛОНЯЕТ СТАРЫЙ ДОМ.

СТАРЫЙ ДОМ ЗАСЛОНЯЕТ СТАРЫЙ ДОМ.

СТАРЫЙ ДУБ ЗАСЛОНЯЕТ СТАРЫЙ ДУБ.

Эти три предложения, а также предложение, выведенное раньше, суть все предложения, порождаемые данной грамматикой. Множество, состоящее из этих четырех предложений, называется языком, который определяется грамматикой (порождается ею или выводится в ней).

Описанная грамматика – это простой пример принадлежащей классу контекстно-свободных грамматик. При их описании используются следующие термины и обозначения.

Такие элементы как <подлежащее> или <существительное>, играющие роль членов предложения или частей речи, называются нетерминальными (вспомогательными) символами или просто нетерминалами. В контекстно-свободной грамматике может быть любое конечное число нетерминалов. При определении языков программирования нетерминалами служат такие элементы как (оператор), (арифметическое выражение) и т.д.

Такие элементы как ДУБ, ЗАСЛОНЯЕТ, играющие роль слов из словаря языка, называются терминальными (основными) символами, или просто терминалами. КС-грамматика может содержать любое конечное число терминалов. В

языках программирования терминалами являются фактически используемые в них слова и символы – такие как DO, + и т. д.

Правила грамматики иногда называются продукциями и в общем виде выглядят так.

Один нетерминал →любая конечная цепочка из терминалов и нетерминалов.

Цепочка справа от стрелки может быть пустой. Пример такого правила:

$$<A> \rightarrow \varepsilon$$

Правило с пустой правой частью называется эпсилон-правилом. КС-грамматика может содержать любое конечное множество продукций. Пример продукции языка программирования:

<оператор>→IF <логическое выражение> THEN <оператор>

Один из нетерминалов выделен как начальный нетерминал, или начальный символ , с которого должны начинаться выводы цепочек языка. Для естественных языков таким нетерминалом может быть <предложение>, для языков программирования – <программа>. Начальный символ часто обозначается через <S>.

Суммируя все сказанное, КС-грамматика задается:

а) конечным множеством нетерминалов;

б) конечным множеством терминалов, которое не пересекается с множеством нетерминалов;

в) конечным множеством правил вида

$$<A> \rightarrow \alpha,$$

где <A> — нетерминал, а α— цепочка терминалов и нетерминалов (возможно, пустая); нетерминал <A>называется левой частью правила, а α – правой его частью;

г) одним нетерминальным символом, выделенным в качестве начального.

Если множество правил приводится без специального указания множества

нетерминалов и терминалов, предполагается, что грамматика содержит в точности те нетерминалы и терминалы, которые встречаются в правилах.

Пусть, например, даны четыре таких правила:

1. <S>→a<A>b<S>

2. <S>→b

3. <A> →<S>ac

4.<A>→ε

Если больше ничего не определено, предполагается, что множество нетерминалов — {S,A}, так как это те нетерминалы, которые встречаются в левых частях правил. Предполагается также, что множество терминалов — {a, b,c}, так как это остальные символы, используемые в правилах. Понятно, что символ ε в правиле 4 представляет пустую цепочку и не является символом грамматики, поэтому правило 4 можно записать без ε.

Можно определять грамматику, задавая ее правила и начальный нетерминал. Нетерминальное и терминальное множества нужно задавать в явном виде лишь, когда они не совпадают с множествами, которые получаются описанным способом.

Для описания грамматик часто используется еще один способ записи, называемый формой Бэкуса – Наура, или БНФ. В этих обозначениях → заменяется символом :: = , за которым может следовать любое число правых частей, разделенных вертикальной чертой |. Здесь нетерминалы также заключаются в угловые скобки:

<S>:=a<A><S>| b

<A>:=<S><A>ε|ε

Форму Бэкуса — Наура используют при описании алгоритмических языков.

Правила грамматики используются для того, чтобы задавать способы подстановки, или замены цепочек. Подстановка осуществляется путем замены

некоторого нетерминала в какой-нибудь заданной цепочке терминалов и нетерминалов правой частью правила, левой частью которого является этот нетерминал. Иногда говорят, что правило применяется к нетерминалу цепочки.

Пусть существует грамматика с начальным нетерминалом (S).

1. <S>→a<A><B>c

2. <S>→ε

3.<A>→c<S><B>

4.<A>→<A>b

5.<B>→b<B>

6. <B>→a

Если дана цепочка

a<A><B>c

5 ↑

и применено правило 5 к нетерминалу <B>, на который указывает стрелка, результат соответствующей подстановки таков:

$$a<A>b<B>c.$$

Эту подстановку необходимо записать так:

a<A><B>c=>a<A>b<B>c

5

Последовательность подстановок называется выводом. Цепочка acabac, например, может иметь следующий вывод, начинающийся с начального нетерминала:

<S> => a <A> <B> c =>a <A> b <B> c=ac <S>

1 4 3

<B>b <B>=>ac<S>ab<B>acab<B>c=>acabac

6 2 6

Каждая цепочка терминалов и нетерминалов, встречающаяся в выводе, называется промежуточной цепочкой этого вывода. Так, в описанном выше выводе семь промежуточных цепочек, включая начальную и заключительную (промежуточную цепочку, выводимую из начального символа, в литературе

иногда называют сентенциальной формой, или выводимой цепочкой).

Мы часто будем употреблять слово «вывод», не указывая начальной цепочки вывода. В таких случаях предполагается, что начальной цепочкой является начальный нетерминал. Если имеется в виду другая начальная цепочка, это указывается явно.

Существование вывода одной цепочки из другой обозначается с помощью символа =>*.

Так, имея в виду, что существует вывод цепочки *acabac* из цепочки (S) или, что эквивалентно, «из цепочки (S) можно вывести цепочку acabac, можно записать <S>=>* acabac.

Язык, задаваемый грамматикой, определяется как множество терминальных цепочек, которые можно вывести из начального символа грамматики. Иногда говорят, что язык «определяется» грамматикой, «порождается» ею или «выводится» в ней. Любой язык, который можно задать контекстно-свободной грамматикой, называется контекстно-свободным языком (кратко – КС-языком).

В приведенном выше примере цепочку acabac можно вывести из начального символа грамматики, поэтому она принадлежит языку, задаваемому грамматикой. С другой стороны, изучение этой грамматики показывает, что цепочка bb, например, не выводится из (S), и, следовательно, bb не принадлежит языку, задаваемому этой грамматикой.

Кроме описанного вывода цепочки, можно построить дерево вывода цепочки КС- языка. Это легко сделать, интерпретируя подстановки как шаги построения дерева. Выше приведена грамматика и вывод в ней терминальной цепочки acabac. Этот вывод можно использовать для построения соответствующего дерева вывода, изображенного на рис. 24, где показан каждый шаг вывода и соответствующее ему дерево. На рис. 24,*а* начальному символу (S) соответствует дерево с одной вершиной (S). Когда к символу (S) применяется правило 1, он заменяется правой частью этого правила, а именно a<A><B>c. Соответствующий шаг построения дерева состоит в том, что до-

бавляются вершины, помеченные символами a, <A>, <B> и c, к которым ведут дуги, исходящие из вершины, помеченной заменяемым символом. Результат показан на рис. 24, *б*. Вывод и соответствующее ему построение дерева продолжается вплоть до рис. 24, *ж*. Окончательный вариант дерева называется деревом вывода терминальной цепочки acabac.

В этом дереве вывода отражено, какие правила были применены во время вывода и к каким вхождениям нетерминалов они применялись. Однако дерево не несет никакой информации о порядке применения правил, кроме очевидного соображения, что правила должны применяться к каждой вершине дерева прежде, чем к нетерминальным вершинам, расположенным ниже ее. Так, рассматривая дерево на рис. 24, *ж*, можно заметить, что правило 4 применялось раньше правила 3, но невозможно сказать, в каком порядке применялись правило 2 и дважды правило 6.

Поскольку порядок подстановок в дереве скрыт, может быть много выводов, соответствующих одному и тому же дереву вывода. Дерево на рис. 34, *ж,* например, соответствует также выводу

<S> => a <A> <B> c => a <A> b <B>c=> ac <S> <B> b <B> c

1 4 3 2

=> ac<B>b <B>c=> acab <B> c => acabac

6 6

Такой вывод называется левым (или левосторонним), так как на каждом шаге заменяется крайний левый нетерминальный символ. Для каждого дерева существует единственный левый вывод, так как благодаря условию выбора крайнего левого нетерминала место каждой подстановки устанавливается единственным образом, а по дереву можно определить единственное правило, которое должно применяться при этой подстановке.

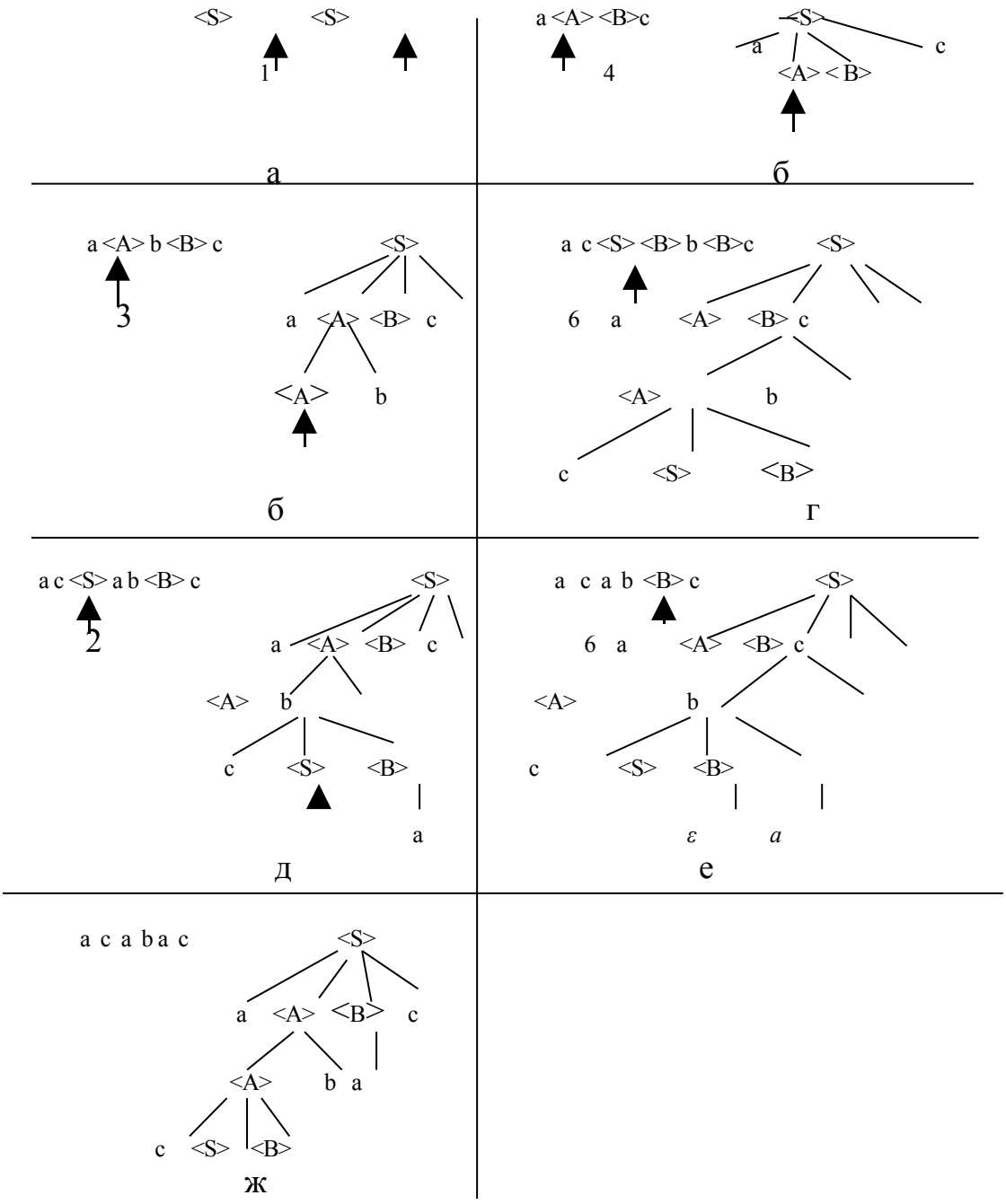


Рис.24.

Поскольку порядок подстановок в дереве скрыт, может быть много выводов, соответствующих одному и тому же дереву вывода. Дерево на рис. 34, *ж,* например, соответствует также выводу

<S> => a <A> <B> c => a <A> b <B>c=> ac <S> <B> b <B> c

1 4 3 2

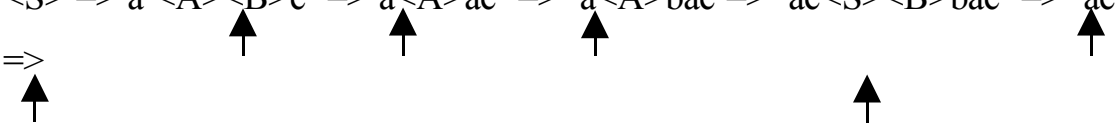
=> ac<B>b <B>c=> acab <B> c => acabac

6 6

Такой вывод называется левым (или левосторонним), так как на каждом шаге заменяется крайний левый нетерминальный символ. Для каждого дерева существует единственный левый вывод, так как благодаря условию выбора крайнего левого нетерминала место каждой подстановки устанавливается единственным образом, а по дереву можно определить единственное правило, которое должно применяться при этой подстановке.

Для каждого дерева существует также единственный правый (или правосторонний} вывод, который получается, если всегда заменять крайний правый нетерминал. Правый вывод, соответствующий дереву на рис. 24, *ж,* таков:

<S> => a <A><B>c => a<A>ac => a<A>bac => ac<S><B>bac => ac<S>abac =>



1 6 4 3 6 2

acabac

Многие методы обработки языков рассчитаны исключительно на левые или правые выводы, так как они очень удобны для систематической обработки. В подобных случаях пишется

$$a \Rightarrow_L B \qquad \text{или} \qquad a \Rightarrow_R B$$

Цепочке языка может соответствовать более чем одно дерево, так как она может иметь разные выводы, порождающие разные деревья. На самом деле это имеет место и для нашей цепочки acabac , Кроме дерева на рис. 24, *ж,* которое еще раз изображено на рис. 24, *а*, ей соответствует также дерево, изображенное на рис. 24, *б*, которое построено по такому выводу:

<S> => a <A><B>c => a<A>b<B>c => ac<S><B>b<B>c => ac<S>ab<B>c =>

1 5 3 6 2

acab<B>c => acabac

6

В этом выводе даже промежуточные цепочки совпадают с промежуточными цепочками первого вывода. Однако в данном случае правило 5 применяется там, где в первом выводе используется правило 4, и соответствующие деревья явно различны.

Когда одна цепочка может иметь несколько деревьев вывода, говорят, что соответствующая грамматика неоднозначна, из всего сказанного можно сделать следующие выводы.

1.Каждой цепочке, выводимой в данной КС-грамматике, соответствуют одно или несколько деревьев вывода.

2.Каждому дереву соответствуют один или более выводов.

3.Каждому дереву соответствуют единственный правый и единственный левый выводы.

4.Если каждой цепочке, выводимой в данной КС-грамматике, соответствует единственное дерево вывода, эта грамматика называется однозначной; в противном случае ее называют неоднозначной.

КС-грамматики могут содержать нетерминалы, которые не участвуют в выводе терминальных цепочек и поэтому может быть исключены из грамматики вместе со всеми правилами, в которые они входили. Нетерминалы, которые не порождают ни одной терминальной цепочки, называются бесплодными, или мертвыми.

Рассмотрим, например, такую грамматику с начальным символом <S>:

1. <S>→a<S>a

2.<S>→b<A>d

3.<S>→c

4.<A>→c<B>d

5.<A>→a<A>d

6.<B>→d<A>f.

Бесплодными здесь являются нетерминалы <A> и <B>. Применив правило 4 к цепочке с символом <A>, можно получить цепочку, содержащую <B>, а применив к этой цепочке правило 6, можно снова получить цепочку, содержащую <A>. Но независимо от того, в каком порядке делаются подстановки, цепочка, содержащая <A > или <B>, всегда переводится в цепочку, которая также содержит <A > или <B >. Поэтому правила, в которые входят <A> или <B>, можно исключить из грамматики, оставив в ней лишь правила 1 и 3. Грамматика, состоящая из правил 1 и 3, порождает те же терминальные цепочки, что и грамматика с правилами 1 – 6.

Аналогична ситуация и в случае, когда в грамматике есть нетерминалы, которые невозможно достичь из начального нетерминала. Рассмотрим, например, такую грамматику с начальным символом <S>:

1. <S>→a<S>b

2.<S>→c

4.<A>→b<S>

5.<A>→a.

Ни одно из двух правил с начальным символом <S > в левой части правила 1 и 2 не содержит в правой части символа <A>. Ни одна цепочка, выводимая из <S>, не может содержать < A >. Поэтому < A > не может участвовать в выводе цепочки из <S>, хотя сам по себе этот нетерминал не является бесплодным. Нетерминал <A > можно удалить из грамматики, оставив лишь правила 1 и 2. Нетерминалы, которые не появляются ни в одной цепочке, выводимой из начального символа, называются недостижимыми нетерминалами.

Нетерминалы, которые бесплодны или недостижимы, называются лишними (бесполезными). Если грамматика получена путем применения какого-нибудь механического способа, в ней часто появляются лишние нетерминалы. Обычно надо проверять, нельзя ли упростить такую грамматику, выбросив лишние нетерминалы. Даже в грамматиках, составленных вручную, лишние нетерминалы могут возникнуть вследствие ошибок разработчика. Поэтому поиск лишних нетерминалов часто может оказаться полезным при отладке граммати-

ки, составленной вручную.

Процедура обнаружения лишних нетерминалов состоит из двух частей: одна – для обнаружения бесплодных, другая – недостижимых нетерминалов. Сперва нужно выполнить первую часть, так как при удалении бесплодных нетерминалов из грамматики другие нетерминалы могут стать недостижимыми.

Терминальный или нетерминальный символ называется продуктивным (живым), если из него выводится какая-либо терминальная цепочка (т. е. если он не является бесплодным нетерминалом). Процедура обнаружения бесплодных нетерминалов основана на следующем свойстве продуктивных символов: если все символы правой части правила продуктивны, продуктивен и символ, стоящий в ее левой части.

Чтобы убедиться в правильности этого утверждения, заметим, что терминальную цепочку можно получить из символа, стоящего в левой части такого правила, применив к нему сначала это правило, а затем заменив каждый нетерминал правой части одной из цепочек, благодаря которым он является продуктивным.

Основная идея процедуры состоит в том, что список начинается с «заведомо» продуктивных нетерминалов, а затем для обнаружения других продуктивных нетерминалов используется описанное свойство, и список пополняется. Шаги процедуры таковы.

1. Нужно составить список нетерминалов, для которых найдется хотя бы одно правило, правая часть которого не содержит нетерминальных символов.

2. Если все нетерминалы, стоящие в его правой части правила, уже занесены в список, добавить в него нетерминал, стоящий в левой части.

3. Если на шаге 2 список больше не пополняется новыми нетерминалами, значит получен список всех продуктивных нетерминалов грамматики, а все нетерминалы, не попавшие в него, являются бесплодными.

То, что список, полученный на шаге 3, содержит только продуктивные нетерминалы, объясняется тем, что в него заносились только нетерминалы,

продуктивные по описанному свойству. Чтобы убедиться, что список содержит все продуктивные нетерминалы, заметим, что если дана последовательность правил, используемых в выводе терминальной цепочки из данного нетерминала, можно рассмотреть эту последовательность в обратном порядке и показать таким образом, что все нетерминалы, участвующие в выводе, продуктивны по свойству продуктивных символов.

Чтобы продемонстрировать эту процедуру в действии, рассмотрим грамматику с начальным символом (S), изображенную на рис. 25,*а*. Благодаря правилу 9 на шаге 1 в список можно поместить нетерминал (C). На шаге 2 можно добавить нетерминал (A), воспользовавшись правилом 5. Наконец, благодаря правилу 2, в список можно добавить (S). Дальнейшие попытки применить шаг 2 оказываются безуспешными, и это говорит о том, что продуктивными нетерминалами являются (C), (A) и (S), а оставшийся нетерминал (B) – бесплодный. Удалив все правила, связанные с (B), получаем грамматику, изображенную на рис. 25, *б.* Символ называется достижимым в грамматике, если он может появиться в какой-нибудь цепочке, выводимой из начального нетерминала.

Процедура обнаружения недостижимых символов грамматики основана на следующем свойстве достижимых символов: если нетерминал в левой части правила достижим, то достижимы и все символы правой его части.

Это свойство выполняется, так как можно сперва вывести цепочку, содержащую символ, который является левой частью правила, и потом применить к ней это правило. Основная идея процедуры в том, что список начинается нетерминалами, которые «заведомо» достижимы, а затем для обнаружения других достижимых нетерминалов используется свойство достижимых символов, и список пополняется. Шаги процедуры таковы.

1. <S>→a<A><B><S>

2. <S>→b <C><A><C>d	2. <S>→b <C><A><C>d
3. <A>→ b<A><B>	
4. <A>→c<S><A>	4. <A>→c<S><A>
5. <A>→c <C><C>	5. <A>→c <C><C>
6. <B>→ b<A><B>	
7. <B>→ c<S><B>	
8. <C>→ c<S>	8. <C>→ c<S>
9. <C>→ c	9. <C>→ c
а	*б*

Рис.25.

1. Нужно образовать одноэлементный список из начального нетерминала.

2. Если найдено правило, левая часть которого уже имеется в списке, включить в список все нетерминалы, содержащиеся в правой его части.

3. Если на шаге 2 новые нетерминалы в список больше не поступают, значит получен список всех достижимых нетерминалов, а нетерминалы, не попавшие в него, являются недостижимыми.

Список, полученный на шаге 3, содержит только достижимые нетерминалы, так как в него вносились только нетерминалы, достижимые по свойству Б. В окончательном списке должны оказаться все достижимые нетерминалы, так как, используя свойство Б и последовательность правил, благодаря которой достижим данный нетерминал, можно показать, что все нетерминалы, участвующие в выводе, попадают в список. Чтобы продемонстрировать процедуру в действии, рассмотрим грамматику с начальным символом <S>, изображенную на рис.26,*а*.

1. <S>→a<A><B>	1. <S>→a<A><B>
2. <S>→<E>	2. <S>→<E>
3. <A>→d<D><A>	3. <A>→d<D><A>
4. <A>→ε	4. <A>→ε
5. <B>→b<E>	5. <B>→b<E>
6. <B>→f	6. <B>→f
7. <C>→a	
8. <C>→ d<S><D>	

9. <C>→ a
10. <D>→ e<A>
11. <E>→f<A>
12. <E>→q

а

10. <D>→ e<A>
11. <E>→f<A>
12. <E>→q

б

Рис.26.

На шаге 1 мы начинаем список, помещая в него начальный символ <S>. Применяя шаг 2 к правилам, содержащим <S> в левой части, а именно к правилам 1 и 2, обнаруживаем, что надо добавить в список нетерминалы <A>, <B> и <E>. Применяя шаг 2 к правилу 3, выясняем, что нужно добавить также нетерминал <D>. При проверке остальных правил новые нетерминалы в список не заносятся, и мы заключаем, что достижимыми являются нетерминалы <S>, <A>, <B>, <D> и <E>, а оставшийся нетерминал <C> – недостижимый. Удалив правила, содержащие (C), получаем грамматику, изображенную на рис. 26,*б*.

1. <S>→ ac
2. <S>→ b<A>
3. <A> →c<B><C>
4. <B> →a<S><A>
5. <C> →b<C>
6. <C>→d

а

1. <S>→ ac
5. <C> →b<C>
6. <C>→d

б

1. <S>→ ac

в

Рис. 27.

Наконец, проиллюстрируем применение обеих процедур на примере грамматики с начальным символом <S>, изображенной на рис. 27, *а.* Применив процедуру для бесплодных символов, мы обнаруживаем, что символы <A> и <B> являются бесплодными. Удалив правила, содержащие эти символы, получаем грамматику на рис. 27, *б*. Теперь, осуществив проверку на недостижимость, выясняем, что символ <C> недостижим. Удалив правила с этим символом, полу-

чаем грамматику, изображенную на рис. 27,*в*. Теперь понятно, что язык, порождаемый грамматикой на рис. 27, *а* содержит одну цепочку – ас. Заметим, что нетерминал <C> стал недостижимым лишь после удаления правила 3. Таким образом, если бы мы устраняли сначала недостижимые нетерминалы, а затем бесплодные, то нам не удалось бы упростить грамматику до конца.

Л Е К Ц И Я № 1 0

НИСХОДЯЩИЕ МЕТОДЫ ОБРАБОТКИ ЯЗЫКОВ С ПОМОЩЬЮ МП-АВТОМАТОВ

Существует нисходящий подход к процессам обработки языков с помощью МП-автоматов, который позволяет распознать не любой КС-язык. Нисходящий подход налагает ограничения на вид входной грамматики.

Рассмотрим такие классы грамматик, для которых можно построить нисходящие МП-автоматов. К ним относятся так называемые s-грамматики, q-грамматики и LL(1)-грамматики.

Контекстно-свободная грамматика называется s-грамматикой (а также разделенной или простой) тогда и только тогда, когда выполняются следующие два условия

1. Правая часть каждого правила начинается терминалом.

2. Если два правила имеют совпадающие левые части, правые части должны начинаться различными терминальными символами.

Например, грамматика

1. <S>→a<T>
2. <S>→<T>b<S>
3. <T>→b<T>
4. <T>→ba

не s-грамматика, так как правая часть правила 2 начинается нетерминальным символом, то есть не удовлетворяет условию 1. Кроме того, правила 3 и 4 начинаются с одинакового терминального символа, что противоречит условию

2.

Рассмотрим еще один пример грамматики

1.<S>→ab<R>

2. <S>→b<R>a<S>

3. <R>→a

4. <R>→b<S>.

Это s-грамматика, так как каждое правило имеет вид, удовлетворяющий условию 1, а правые части двух правил, имеющих в левой части нетерминал <S>, начинаются с различных терминальных символов, что и требуется в условии 2. То же самое можно сказать о правилах, в левой части которых стоит нетерминальный символ <R>.

Для s-грамматики МП-распознаватель с одним состоянием задается следующим образом.

1) Множество входных символов – это множество терминальных символов грамматики, расширенных концевым маркером (┤).

2) Множество магазинных символов состоит из маркера дна ( ), нетерминальных символов грамматики и терминалов, которые входят в правые части правил, за исключением тех, что занимают крайнюю левую позицию.

3) В начале магазин состоит из маркера дна и начального нетерминала.

4) Управление работой МП-автомата с одним состоянием описывается управляющей таблицей, строки которой помечены магазинными символами, столбцы – выходными символами, а элементы описываются ниже.

5) С каждым правилом грамматики сопоставляется элемент таблицы. Правило имеет вид <A>→bα, где <A> – нетерминал, b – терминал, а α – цепочка, состоящая из терминальных и нетерминальных символов. Этому правилу будет соответствовать элемент в строке <A> и столбце b ЗАМЕНИТЬ (α*), СДВИГ, где α*- цепочка α, записанная в обратном порядке. Если правило имеет вид <A>→b, то вместо ЗАМЕНИТЬ используется ВЫТОЛКНУТЬ.

6) Если магазинным символом является терминал b , то элементом таб-

лицы в строке b и столбце b будет ВЫТОЛКНУТЬ, СДВИГ.

7) Элементом таблицы, который находится в строке маркера дна и столбце концевого маркера, является ДОПУСТИТЬ.

8) Элементы таблицы, не описанные ни одним из пунктов 5, 6 и 7, заполняются операцией ОТВЕРГНУТЬ.

Применяя перечисленные правила к s-грамматике, описанной в этом разделе, получим управляющую таблицу (рис.28).

a	b	┤	
ЗАМЕНИТЬ (<R>b) СДВИГ	ЗАМЕНИТЬ (<S>b<R>) СДВИГ	ОТВЕРГНУТЬ	<S>
ВЫТОЛК-НУТЬ сдвиг	ЗАМЕНИТЬ (<R>) СДВИГ	ОТВЕРГНУТЬ	<R>
ОТВЕРГНУТЬ сдвиг	ВЫТОЛКНУТЬ СДВИГ	ОТВЕРГНУТЬ	b
ОТВЕРГНУТЬ	ОТВЕРГНУТЬ	ДОПУСТИТЬ	▽

Рис.28.

Рассмотрим особый класс грамматик – более общий, чем класс s-грамматик. Все грамматики этого класса можно распознать с помощью нисходящих МП-распознавателей. Представители этого класса называются

q-грамматики. Пример:

1. <S>→a<A><S>
2. <S>→b
3. <A>→c<A><S>
4. <A>→ε

Для q-грамматики введем понятия множества терминалов, "следующих за" данным нетерминалом, и множества выбора для данного правила.

Для КС-грамматики с начальным символом <S> и нетерминала <X>

определим СЛЕД(<X>) как множество терминальных символов, которые могут следовать непосредственно за <X> в какой-либо промежуточной цепочке, выводимой из <S>. Это множество называется множеством "следующих за" <X> терминалов.

Анализируя последний пример, видим, что после нетерминала <A> могут идти только терминальные символы *a* и *b*: СЛЕД(<A>)={a,b}.

Множество выбора для данного правила определим следующим образом.

Если правило грамматики имеет вид <A>→bα, где b – терминальный символ, а α – цепочка, состоящая из терминальных и нетерминальных символов, то определим ВЫБОР(<A>→bα)={b}.

Если правило имеет вид <A>→ε, ВЫБОР(<A>→ε)=СЛЕД(<A>).

Если p – номер правила, то ВЫБОР(p) – множество выбора правила p.

Множество выбора правила просто содержит те входные символы, для которых соответствующий МП-автомат должен применить это правило.

Для грамматики, представленной выше, множество выбора будет следующим:

ВЫБОР(1)= ВЫБОР(<S>→a<A><S>)= { a };

ВЫБОР(2)=ВЫБОР(<S>→b)={b};

ВЫБОР(3)=ВЫБОР(<A>→c<A><S>)={c};

ВЫБОР(4)=ВЫБОР(<A>→ε)=СЛЕД(<A>)={a,b}.

Контекстно-свободная грамматика называется q-грамматикой тогда и только тогда, когда выполняются следующие два условия: правая часть каждого правила либо представляет собой пустую строку, либо начинается с терминального символа; множества выбора правила с одной и той же левой частью не пересекаются.

Первое условие просто утверждает, что все правила ограничиваются двумя видами, для которых определено понятие множества выбора. Второе условие указывает, что при построении управляющей таблицы соответствующего МП-автомата конфликтные ситуации не возникают. Наш

пример удовлетворяет обоим условиям, так как правила имеют надлежащий вид и, кроме того, справедливы равенства

ВЫБОР(1) ∩ВЫБОР(2)={a} ∩{b}={}

ВЫБОР(3) ∩ ВЫБОР(4)={c} ∩ {a,b}={}.

Правила построения МП-распознавателя для q-грамматики такие же, как для s-грамматики, кроме 5-го. Его необходимо заменить на два следующих правила.

5.1. Правило грамматики применяется всякий раз, когда магазинный символ является его левой частью, а входной символ принадлежит его множеству выбора. Чтобы применить правило вида <A>→bα, где b – терминальный символ, а α – цепочка терминальных и нетерминаль-ных символов, используется переход ЗАМЕНИТЬ(α*), СДВИГ.

Если правило имеет вид <A>→b, вместо ЗАМЕНИТЪ(ε) можно воспользоваться операцией ВЫТОЛКНУТЬ.

Чтобы применить правило вида <A>→ε, используется переход

ВЫТОЛКНУТЬ, ДЕРЖАТЬ.

5.2.Если имеется ε-правило с нетерминалом <A> в левой части и элемент, соответствующий магазинному символу <A> и входному символу b, не был создан по правилу 5.1, то таким элементом может быть либо применение этого ε-правила, либо операция ОТВЕРГНУТЬ.

Построим управляющую таблицу для q-грамматики, представленной в этом разделе (рис. 29).

LL1-грамматика – третий класс грамматик, для которых легко построить МП-распознаватель. '

КС-грамматика называется LL(1)-грамматикой тогда и только тогда, когда множества выбора правил с одинаковой левой частью не пересекаются.

a b c ┤

ЗАМЕНИТЬ(<S><A>) СДВИГ	ВЫТОЛКНУТЬ СДВИГ	ОТВЕРГНУТЬ	ОТВЕРГНУТЬ	<S>
ВЫТОЛКНУТЬ ДЕРЖАТЬ	ВЫТОЛКНУТЬ ДЕРЖАТЬ	ЗАМЕНИТЬ(<S><A>) СДВИГ	ВЫТОЛКНУТЬ ДЕРЖАТЬ	<A>
ОТВЕРГНУТЬ	ОТВЕРГНУТЬ	ОТВЕРГНУТЬ	ДОПУСТИТЬ	▽

Начальное содержимое магазина ▽ <S>

Рис.29.

Для определения множества выбора введем понятия ПЕРВ(α) и аннулирующей цепочки. По данной контекстно-свободной грамматике и промежуточной цепочке α, состоящей из символов этой грамматики, определим ПЕРВ(α) как множество терминальных символов, которые стоят в начале промежуточных цепочек, выводимых из α, т.е. являются их первыми символами.

Цепочка α, состоящая из символов грамматики, называется аннулирующей тогда и только тогда, когда α=>ε.

Правило грамматики называется аннулирующим тогда и только тогда, когда его правая часть является аннулирующей, то есть его можно использовать для порождения пустой цепочки.

Определим множество выбора для правила произвольного вида следующим образом. Для данного правила <A>→α, где α– цепочка, состоящая из терминалов и нетерминалов, определим ВЫБОР(<A>→α=ПЕРВ(α), если α не аннулирующая, и ВЫБОР(<A>→α)=ПЕРВ(α) U СЛЕД(<A>), если α аннулирующая.

Рассмотрим LL(1)-грамматику и определим для нее множество выбора.

1.<S>→< A>b<B>
2.<S>→d

3.<A>→<C><A>b
4.<A>→<B>
5.<B>→c<S>d
6.<B>→e
7.<C>→a
8.<C>→fd
Начальный нетерминал <S>.

Определим для этой КС-грамматики множество ПЕРВ.

ПЕРВ (<A>b<B>)={a,b,c,f}
ПЕРВ (d)={d}
ПЕРВ(<C><A>b)={a,f}
ПЕРВ (<B>)={c}
ПЕРВ (c<S>d)={c}
ПЕРВ(e)={}
ПЕРВ(a)={a}
ПЕРВ (fd)={f}

В рассматриваемом примере правила 4 и 6 аннулирующие. Для каждого правила определим множество выбора.

ВЫБОР(1)=ПЕРВ(<A>b<B>)={a,b,c,f}

ВЫБОР(2)= ПЕРВ (d)={d}

ВЫБОР(3)=ПЕРВ(<C><A>b)={a,f}

ВЫБОР(4)=ПЕРВ(<B>) U СЛЕД(<A>)={c} U {b}={c,b}

ВЫБОР(5)= ПЕРВ(c<S>d)={c}

ВЫБОР(6)=ПЕРВ(ε) U СЛЕД(<B>)={} U {b,d,-+}

ВЫБОР(7)=ПЕРВ(a)={a}

ВЫБОР (8)= ПЕРВ(fd)={f}.

После нахождения множества выбора можно с уверенностью сказать, что это LL(1)-грамматика, так как множества выбора правил с одинаковой левой частью не пересекаются.

ВЫБОР(1) ∩ ВЫБОР(2)={a,b,c,f} ∩{d}={}

ВЫБОР(3) ∩ ВЫБОР(4)={a,f} ∩{c}={}

ВЫБОР(5) ∩ ВЫБОР(6)={c} ∩ {}={}

ВЫБОР(7) ∩ ВЫБОР(8)={a} ∩ {f}={}

После того как для всех правил найдены множества выбора, можно за-

дать управляющую таблицу нисходящего МП-распознавателя. Правила построения МП-распознавателя для s-грамматики применимы для LL(1)-грамматики, если расширить правило 5.1.

5.1. Любое из правил применяется каждый раз, когда магазинный символ является левой частью правила, а входной символ принадлежит его множеству выбора. Чтобы применить правило вида <A>→bα, где b терминальный символ, а α – цепочка, состоящая из терминальных и нетерминальных символов, используется переход ЗАМЕНИТЬ(α*), СДВИГ.

Чтобы применить правило вида <A>→α, где α – цепочка, которая состоит из терминальных и нетерминальных символов и не начинается с терминала, используется переход ЗАМЕНИТЬ(α*), ДЕРЖАТЬ.

Если цепочка пустая, вместо операции ЗАМЕНИТЬ(ε) можно использовать операцию ВЫТОЛКНУТЬ.

5.2.Если для нетерминала <A> имеется аннулирующее правило и по правилу 5.1 не было создано элемента таблицы, соответствующего магазинному символу <A> и входному символу b, можно применить аннулирующее правило или отвергнуть входную цепочку. Построим управляющую таблицу нисходящего МП-распознавателя (рис.30).

Совокупность правил построения распознавателя устанавливает следующий факт: если язык определяется LL(1) грамматикой, его можно распознать с помощью МП-автомата с одним состоянием, использующего расширенную магазинную операцию ЗАМЕНИТЬ. LL(1)-грамматика – универсальная грамматика по сравнению с s- и q-грамматиками.

a b c d f ┤

<S>

#1	#1	#1	#2	#1	ОТВЕРГ-НУТЬ
#3	#4	#4	ОТВЕРГ-НУТЬ	#3	ОТВЕРГ-НУТЬ
ОТВЕРГ-НУТЬ	#6	#5	#6	ОТВЕРГ-НУТЬ	#6
#7	ОТВЕРГНУТЬ	ОТВЕРГ-НУТЬ	ОТВЕРГ-НУТЬ	#8	ОТВЕРГ-НУТЬ
ОТВЕРГ-НУТЬ	#7	ОТВЕРГ-НУТЬ	ОТВЕРГ-НУТЬ	ОТВЕРГ-НУТЬ	ОТВЕРГ-НУТЬ
ОТВЕРГ-НУТЬ	ОТВЕРГНУТЬ	ОТВЕРГ-НУТЬ	#7	ОТВЕРГ-НУТЬ	ОТВЕРГ-НУТЬ
ОТВЕРГ-НУТЬ	ОТВЕРГНУТЬ	ОТВЕРГ-НУТЬ	ОТВЕРГ-НУТЬ	ОТВЕРГ-НУТЬ	ДОПУ-СТИТЬ

<A>

<B>

<C>

b

d

▽

начальное содержимое магазина ▽ <S>

#1 ЗАМЕНИТЬ(<B>b<A>), ДЕРЖАТЬ
#2 ВЫТОЛКНУТЬ, СДВИГ
#3 ЗАМЕНИТЬ(b<A><C>)ДЕРЖАТЬ
#4 ЗАМЕНИТЬ(<B>), ДЕРЖАТЬ
#5 ЗАМЕНИТЬ(d<S>), СДВИГ
#6 ВЫТОЛКНУТЬ, ДЕРЖАТЬ
#7 ВЫТОЛКНУТЬ, СДВИГ
#8 ЗАМЕНИТЬ(d), СДВИГ

Рис. 30.

Л Е К Ц И Я № 1 1

ОБРАБОТКА ОШИБОК ПРИ НИСХОДЯЩЕМ РАЗБОРЕ

Цепочки, поступающие на вход компилятора, могут не принадлежать распознаваемому языку. Такие цепочки называются неправильными. Когда компилятор встречается с неправильной цепочкой, он должен обнаружить, что в ней основные ошибки, выдать сообщения о них, указывая те, которые мог допустить программист.

Разработчик компилятора решает, сколько и какие сообщения об ошибках выдавать для данной неправильной цепочки. Различные компиляторы для одного и того же языка программирования часто выдают разные сообщения об ошибках для одной и той же неправильной цепочки.

МП-распознаватель обнаруживает, что входная цепочка является неправильной, как только встречает элемент ОТВЕРГНУТЬ в управляющей таблице. Желательно, чтобы в этот момент компилятор сделал сообщение об ошибке, которую, возможно, совершил программист. Далее необходимо, чтобы компилятор изменил конфигурацию МП-автомата таким образом, чтобы процесс обработки входной цепочки можно было продолжить и получить дополнительные сообщения. Процесс изменения конфигурации называется нейтрализацией ошибок. Обычно в управляющей таблице элементы ОТВЕРГНУТЬ – это не просто выходы; они содержат вызовы процедур, которые выдают сообщения об ошибках и осуществляют их нейтрализацию. Нисходящие распознаватели могут давать полезные сообщения об ошибках благодаря «префиксному свойству». Для данных МП-распознавателя и неправильной цепочки символ этой цепочки называется нарушающим, если он является текущим входным символом в тот момент, когда автомат отвергает входную цепочку. Говорят, что МП-автомат обладает префиксным свойством, если для любой недопустимой цепочки цепочка символов, расположенных слева от нарушающего символа, является префиксом какой-нибудь допустимой входной цепочки. Иными словами, распознаватель имеет префиксное свойство, если нарушающий

символ является первым символом, противоречащим гипотезе о том, что входная цепочка является допустимой. Таким образом, нисходящий распознаватель с префиксным свойством обнаруживает самую левую ошибку во входной цепочке.

Если распознаватель обладает префиксным свойством, часто можно получить хорошее сообщение об ошибке, предположив, что она произошла в самом нарушающем символе или где-то поблизости. Чтобы сделать подходящее сообщение об ошибке, компилятор может проанализировать входную цепочку и содержимое магазина в тот момент, когда автомат отверг входную цепочку. Однако часто глубокий анализ не проводится и сообщение основывается только на верхнем магазинном символе и текущем входном символе. Верхний магазинный символ указывает, что компилятор ожидает обнаружить на входе, а текущий входной символ –, что он обнаруживает на самом деле. Таким образом, формат сообщения может быть таким:

«ОЖИДАЛОСЬ (НАЧАЛО) _____, НО ВМЕСТО
ЭТОГО ОБНАРУЖИЛОСЬ_____ .»,

где первый прочерк заменяет верхний магазинный символ (подходящим образом перефразированный, чтобы он было понятен пользователю), а второй – входной символ. Сообщение может быть выражено и другими словами, но в приведенной форме по существу суммирована вся информация, на которой основывается сообщение об ошибке.

В типичном случае сообщение об ошибке содержит еще номер строки, а также позицию нарушающего символа.

В качестве примера рассмотрим проектирование сообщений об ошибках для s-грамматики.

1. <S>→a
2. <S>→(<S><R>)
3. <R>→,<S><R>
4. <R>→)

с начальным символом <S>. Эта грамматика порождает цепочки, подобные выражениям Лисп – таким как

a

(a, (a, a))

((a, (a, a), (a, a), a)

Будем называть их «S-выражениями», предположив, что программист знаком с этим термином.

Управляющая таблица, построенная по данной грамматике, представлена на рис. 31.

	a	,	(	)	┤
<S>	#1	ОТВЕРГНУТЬ a	#2	ОТВЕРГНУТЬ b	ОТВЕРГНУТЬ C
<R>	ОТВЕРГНУТЬ d	#3	ОТВЕРГНУТЬ e	#4	ОТВЕРГНУТЬ f
▽	ОТВЕРГНУТЬ q	ОТВЕРГНУТЬ h	ОТВЕРГНУТЬ i	ОТВЕРГНУТЬ j	ДОПУСТИТЬ

Начальное содержимое магазина : ▽ <S>
#1 ВЫТОЛКНУТЬ, СДВИГ
#2 ЗАМЕНИТЬ (<R><S>), СДВИГ
#3 ЗАМЕНИТЬ (<R><S>), СДВИГ
#4 ВЫТОЛКНУТЬ, СДВИГ

Рис.31.

В этой таблице все элементы ОТВЕРГНУТЬ помечены буквами от *a* до *f*. Для каждого элемента ОТВЕРГНУТЬ необходимо выбрать сообщение об ошибке. В строке <S> содержатся три элемента ОТВЕРГНУТЬ: *a, b, c*. Так как цепочки, порождаемые нетерминалом <S>, называются S-выражениями, сообщение для элементов a и b могут быть такими:

_____ СТОИТ ТАМ, ГДЕ ОЖИДАЕТСЯ S-ВЫРАЖЕНИЕ,

где на месте черты должен стоять входной символ. Такое сообщениене подходит для элемента *c*, так как замена прочерка на концевой маркер или конец выражения мало что говорит среднему пользователю. Посмотрев на

некоторые типичные входные цепочки, где требуется задать сообщение об ошибке для элемента *c*, например,

┤

(┤

(a , ┤,

решаем, что сообщение должно быть таким:

S-ВЫРАЖЕНИЕ НЕ ЗАКОНЧЕНО.

Теперь рассмотрим строку, соответствующую нетерминалу <R>. Так как пользователь, вероятно, ничего не знает о нетерминале <R>, нет смысла упоминать о нем в сообщении. Вместо этого можно перечислить входные, которые могут стоять в начале допустимого продолжения обработанной части входной цепочки. В данном случае допустимые продолжения начинаются с запятой или закрывающей скобки, потому сообщение может быть таким:

_____СТОИТ ТАМ, ГДЕ ОЖИДАЕТСЯ ЗАПЯТАЯ ИЛИ ПРАВАЯ СКОБКА.

Можно дать более подробное сообщение, указывающее возможную причину возникновения ошибки, но это может привести к неверному сообщению. Во всех цепочках, которые заставляют автомат использовать элементы *d* и *e*, содержится цепочка, порожденная нетерминалом <S>, за которой следует a или (. Некоторые типичные цепочки, приводящие к тому, что автомат использует элеты *d* и *e*, таковы (нарушающие символы подчеркнуты)

(a a)

((a) a) ┤

(a (a)) ┤

((a) (a)) ┤

Если в каждую из этих цепочек вставить запятую перед нарушающим символом, это позволит автомату продолжить обработку входной цепочки, а если вставить правую скобку (второй входной символ , соответствующий не-

терминалу <R>), – то нет. Следовательно, для элементов *d* и *e* выбираем сообщение ПРОПУЩЕНА ЗАПЯТАЯ. Это сообщение должно указать пользователю, в чем состоит его ошибка, даже если она записана неточно. Например, если вместо правой скобки ошибочно напечатана левая, могла получиться цепочка ((a (, a) .

Даже при такой цепочке программист, скорее всего правильно поймет сообщение, содержащее номер строки и позицию, занимаемую в ней закрывающей скобкой, а также слова ПРОПУЩЕНА ЗАПЯТАЯ. Для элемента *f* выберем сообщение ПРОПУЩЕНА ПРАВАЯ СКОБКА, так как ясно, что в этой ситуации пропущена правая скобка; это можно проиллюстрировать таким характерным примером:

(a, a ┤ .

Ошибочные элементы *q, h, i, j* в строке, помеченной маркером дна, соответствуют ситуации, в которой распознаватель считает, что S-выражение закончено, но входная цепочка содержит еще какие-то символы. Надлежащее сообщение имеет вид

_____ ВСТРЕЧАЕТСЯ ПОСЛЕ КОНЦА S-ВЫРАЖЕНИЯ.

Для всех элементов ОТВЕРГНУТЬ выбраны сообщения об ошибках.

Теперь рассмотрим проблему нейтрализации ошибок. Один из подходов к этой проблеме состоит в разработке процедур, которые приводят в надлежащее состояние магазин и входную цепочку и затем продолжают процесс обработки, как будто никакой ошибки не было. Если МП-автомат в очередной раз обнаруживает элемент ОТВЕРГНУТЬ, он добавляет к списку сообщений об ошибках сообщение об ошибке, которое дается в этом элементе, и выполняет процедуру нейтрализации ошибок, соответствующую данному элементу. Считаем, что после обнаружения первой ошибки восстановленная конфигурация является «правильной» в том смысле, что она соответствует конфигурации, которая имела бы место, если бы программист не сделал первой ошибки, и поэтому, если после нейтрализации ошибки будут получены еще какие-либо сообщения, они соответствуют уже другим ошибкам, о кото-

рых тоже хотелось бы знать. Риск состоит в том, что конфигурация может быть восстановлена неверно и компилятор в дальнейшем сделает сообщение о фиктивных ошибках. Таким образом, любая попытка нейтрализации ошибок представляет собой компромисс между желанием обнаружить как можно больше ошибок и желанием избежать сообщений о несуществующих ошибках.

Существует два подхода к нейтрализации ошибок – локальный и глобальный. Рассмотрим первый из них. Его идея в том, чтобы каждый элемент

ОТВЕРГНУТЬ в управляющей таблице заполнялся обычными операциями над магазином и входом. Если обнаруживается какая-то конкретная ошибка, автомат вначале выдает сообщение о ней, затем выполняет указанные операции над магазином и входом и, наконец, возобновляет обычный процесс обработки. Разработчик выбирает операции отдельно для каждого элемента таблицы, соответствующего ошибке, на основе интуитивного предположения о том, какая ошибка могла произойти и какие изменения магазина и входной цепочки позволят возобновить процесс обработки наилучшим образом.

На рис.32 показан окончательный вид управляющей таблицы, в которой для каждого ошибочного элемента задана подходящая процедура нейтрализации.

В качестве примера того, как происходит нейтрализация ошибок, на рис.33 показана последовательность конфигураций процесса обработки автоматом, показанным на рис.32., входной цепочки (a , , a.)

В этой последовательности предполагается, что сообщения выдаются с помощью процедуры ПЕЧАТАТЬ с двумя параметрами: позиция нарушающего символа и текст сообщения. Многие компиляторы делят все ошибки на «фатальные» и «нефатальные». Для фатальных ошибок процедура, обрабатывающая ошибку, прекращает компиляцию программы и предотвращает ее выполнение. Для нефатальных ошибок процедура не предпалагает подобных действий, и если далее не обнаруживается фатальных ошибок,

производятся компиляция и полное выполнение программы.

	a	,	(	)	⊣
<S>	#1	a	#2	b	C
<R>	D	#3	E	#4	F
▽	Q	h	I	j	ДОПУСТИТЬ

Начальное содержимое магазина : ▽ <S>

#1 ВЫТОЛКНУТЬ, СДВИГ

#2 ЗАМЕНИТЬ (<R><S>), СДВИГ

#3 ЗАМЕНИТЬ (<R><S>), СДВИГ

#4 ВЫТОЛКНУТЬ, СДВИГ

a, b: ПЕЧАТАТЬ (___ВСТРЕТИЛОСЬ, КОГДА ОЖИДАЛОСЬ S-ВЫРАЖЕНИЕ), ВЫТОЛКНУТЬ,
ДЕРЖАТЬ

c: ПЕЧАТАТЬ (S-ВЫРАЖЕНИЕ НЕ ЗАВЕРШЕНО) ВЫХОД

d, e: ПЕЧАТАТЬ (ПРОПУЩЕНА ЗАПЯТАЯ), ЗАМЕНИТЬ(<R><S>), ДЕРЖАТЬ

f: ПЕЧАТАТЬ (ПРОПУЩЕНА ПРАВАЯ СКОБКА), ВЫХОД

q, i : ПЕЧАТАТЬ(____ВСТРЕТИЛОСЬ ПОСЛЕ КОНЦА S-ВЫРАЖЕНИЯ)
ВТОЛКНУТЬ (<R><S>), ДЕРЖАТЬ

h: ПЕЧАТАТЬ (____ВСТРЕТИЛОСЬ ПОСЛЕ КОНЦА S-ВЫРАЖЕНИЯ)
ВТОЛКНУТЬ(<R>), ДЕРЖАТЬ

j: ПЕЧАТАТЬ(____ВСТРЕТИЛОСЬ ПОСЛЕ КОНЕЦ S-ВЫРАЖЕНИЯ), СДВИГ

Рис.32.

Обычно нефатальные ошибки – те, которые по мнению разработчика могут быть с большой вероятностью устранены процедурой нейтрализации. В тексте сообщений для нефатальных ошибок обычно указывается, что это только предупреждение. В примере такой тип ошибок, по-видимому, соответствует элементам *d* и *e*. Сообщение об ошибке можно заменить на предупреждение

ПРЕДПОЛАГАЕТСЯ, ЧТО ПРОПУЩЕНА ЗАПЯТАЯ.

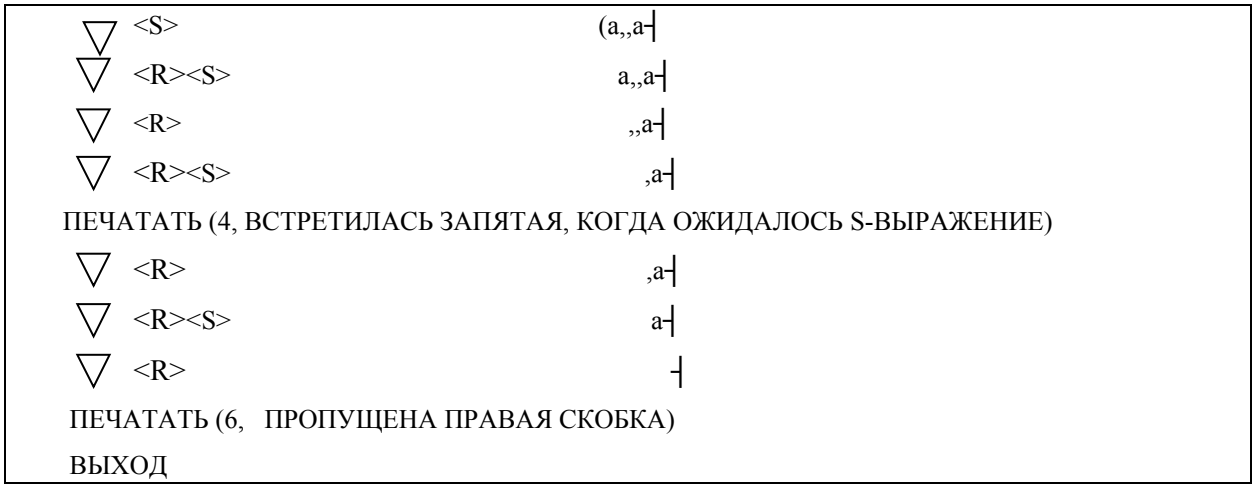


Рис.33.

Второй подход к нейтрализации ошибок можно охарактеризовать как глобальный. Его используют, когда разработчик недостаточно уверен в точности какой-либо из локальных процедур нейтрализации или когда ограничения на память не позволяют иметь для каждого элемента свою нейтрализующую процедуру.

Основная идея заключается в следующем. Входная цепочка считывается до тех пор, пока не будет обнаружен символ из множества «надежных» входных символов. Нейтрализация зависит от этого символа и верхнего символа магазина. Можно выделить два типа надежных символов: синхронизи-

рующие и начинающие.

Синхронизирующий символ – такой, относительно которого разработчик по разумным причинам уверен, что он знает, как возобновить процесс обработки. Множество синхронизирующих символов может включать некоторые символы пунктуации, – такие как запятая, точка с запятой и некоторые зарезервированные слова, – такие как BEGIN, END, THEN.

В ходе процесса нейтрализации ошибок, когда процедура считывания находит один из синхронизирующих символов, из магазина выталкиваются символы до тех пор, пока не будет найден магазинный символ, к которому можно применить данный синхронизирующий символ для правильного продолжения обработки. Тогда процесс обработки продолжается. Простой пример нейтрализации, основанный на использовании синхронизирующего символа, – после обнаружения ошибки игнорируются все символы вплоть до следующей строки или следующего оператора. В этом случае синхронизирующий символ – разделитель операторов или символ «новая строка».

Начинающий символ – такой, который позволяет разработчику предсказать несколько следующих терминальных и нетерминальных символов, даже если он не в состоянии возобновить процесс обработки.

Например, грамматика для S-выражений является частью какой-то большой грамматики, которая не использует скобок ни в каком другом контексте. Следовательно, когда во входной строке встречается левая скобка, разработчик уверен, что началось S-выражение, даже если невозможно полностью согласовать содержимое магазина и входную цепочку.

Если в ходе процесса нейтрализации ошибок считывающая процедура встречает левую скобку, мы можем предсказать, что она начинает цепочку, порожденную <S>. Процедура нейтрализации выполняет операцию ВТОЛКНУТЬ ({ошибка}<S>), а затем продолжает обычный процесс обработки, в котором левая скобка применяется к <S> Символ {ошибка} – это новый символ действия, который действует как мнимый маркер дна магазина. Когда {ошибка} оказывается наверху магазина, мы знаем, что предсказание верно.

Теперь предпринимается действие, выталкивающее символ {ошибка} из магазина и возобновляющее поиск другого синхронизирующего или начинающего символа.

Вообще, обнаружив начинающий символ, процедура нейтрализации вталкивает в магазин символ {ошибка} и символы, предсказываемые данным начинающим символом. Другим примером начинающих символов может быть зарезервированное слово, с которого начинается каждый оператор.

В глобальных процедурах нейтрализации следует рассматривать вместе синхронизирующие и начинающие символы. Например, в грамматике для S-выражений схема нейтрализации ошибок, основанная только на использовании запятой и правой скобки в качестве синхронизирующих символов, работать не будет, так как если считывающая процедура встретит левую скобку в то время, когда она ищет запятую, подсчет скобок будет неверным. Однако если левую скобку рассматривать как начинающий символ, такая комбинация начинающих и синхронизирующих символов будет удовлетворительной.

Указанные выше способы не являются универсальными для нейтрализации ошибок; разработчик может предложить другие глобальные процедуры, учитывающие особенности языка.

Л Е К Ц И Я № 1 2

МЕТОД РЕКУРСИВНОГО СПУСКА

Некоторые типы языков можно распознавать и переводить нисходящим методом с помощью МП-автомата. Существует другой метод распознавания и перевода языков, который называется рекурсивным спуском. Этот метод ориентирован на те случаи, когда компилятор программируется на одном из языков высокого уровня, когда допускается использование рекурсивных процедур. Основная идея рекурсивного спуска в том, что каждому нетерминалу грамматики соответствует процедура, которая распознает любую цепочку, порождаемую нетерминалом. Эти процедуры вызывают друг друга, когда это требуется.

Рассмотрим пример. Имеется q-грамматика с начальным символом <S>:

1.<S>→a<A><S>
2.<S>→b
3.<A>→c<A><S>b
4.<A>→e.

На рис. 34 показаны процедуры, распознающие язык, порождаемые данной грамматикой. Процедуры для распознавания <S> и <A> называются соответственно ПРОЦS и ПРОЦА.

Процедуры для распознавания <S> и <A> называются ПРОЦА и ПРОЦS, в них предполагается наличие процедуры, называемой СДВИГ, которая делает сдвиг во входной цепочке и присваивает значение нового входного символа переменной ВХОД.

Процедура ПРОЦS начинает работу, принимая то же решение, что и МП-автомат, когда обнаруживает наверху магазина символ <S>. Процедура рассматривает текущий входной символ и выбирает правило, которое нужно применить, с помощью некоторого переключателя или оператора перехода на вычисляемую метку. Для каждого значения переменной ВХОД указана метка, на которую происходит переход. Считается, что у нас есть процедура обработки ошибки, помеченная ОТВЕРГНУТЬ.

Меткой Р1 помечен код, соответствующий правилу 1. Вначале этот код вызывает процедуру СДВИГ. Текущий входной символ сопоставлен с самым левым символом правой части правила, и нужно сделать сдвиг по входу, чтобы можно было обработать остальные входные символы. Процедура ПРОЦS отыскивает эти цепочки, вызывая процедуру ПРОЦА, чтобы найти цепочку, порожденную <A>,а затем ПРОЦS, чтобы найти цепочку, порожденную <S>. Так как правая часть правила 1 после того, как эти вызовы завершатся, будет исчерпана, далее происходит возврат из процедуры.

Главная программа | Процедура ПРОЦS | Поцедура ПРОЦА

```
"Начало программы"          | Переход на          | Переход на
Установить ВХОД=            | a  P1               | a  P4
Первый входной              | b  P2               | b P4
Символ                      | c  отвергнуть       | c  P3
Вызов ПРОЦ S                | -!   отвергнуть     | -! отвергнуть
Если ВХОД не=-! То          |                     |
ОТВЕРГНУТЬ                  | P1:"Код для         | P3:"Код для правила 3"
ДОПУСТИТЬ                   | Правила1"           | Вызов СДВИГ
                            | Вызов СДВИГ         | Вызов ПРОЦА
                            | Вызов ПРОЦА         | Вызов ПРОЦS
                            | Вызов ПРОЦS         | Если ВХОД не=b
                            | Возврат             | то ОТВЕРГНУТЬ
                            |                     | Вызов СДВИГ
                            |                     | Возврат
                            | P2:"Код для правила 2' | P4:"Код для правила 4"
                            | Вызов СДВИГ         | Возврат
                            | Возврат             |
```

Рис.34.

Код, помеченный P2 и соответствующий правилу 2, очень прост. Известно, что входной символ является желаемым порождением <S>, и потому производится сдвиг по входу и возврат из процедуры.

Соответствующая <A> процедура, ПРОЦА, начинает работу с выбора правила, которое нужно применить. Решение принимается на основе следующих равенств:

ВЫБОР(3)={c}

ВЫБОР(4)=СЛЕД(<A>)= {a,b}.

Код, соответствующий правилу 3, помечен P3. Вначале вызывается СДВИГ, затем ПРОЦА и затем ПРОЦS. После вызова ПРОЦS код достигает точки, в которой текущий входной символ должен совпадать с символом b из правой части третьего правила. Чтобы убедиться, что текущим входным символом на самом деле является *b*, проверяется значение переменной ВХОД. Если это не так, управление передается отвергающей процедуре. Если текущим входным символом является *b*, то программа продолжает работу, сдвигаясь по входной цепочке, поскольку b – это часть цепочки, порождаемой

<A>, которую процедура ПРОЦА обнаружила. Так как теперь правая часть правила полностью обработана, делается возврат из этой процедуры. Соответствующий правилу 4 код с Р4 тривиален. Над входной цепочкой не выполняется никаких операций, так как текущий символ должен обрабатываться дальше. Никакая процедура не вызывается, поскольку правая часть правила пуста. Таким образом, единственное действие – это возврат.

Главная программа – инициирующая вызовы процедур должна гарантировать, что входная цепочка состоит из цепочки, порожденной <S>, за которой следует концевой маркер. Она начинается с того, что входной символ присваивается переменной ВХОД. Затем следуют вызов ПРОЦS, а затем проверка, является ли текущий символ концевым маркером.

П Р А К Т И Ч Е С К О Е З А Н Я Т И Е № 1

РЕГУЛЯРНЫЕ МНОЖЕСТВА И ВЫРАЖЕНИЯ

Цель работы. Ознакомиться с приемами построения регулярных множеств и выражений для представления элементов алгоритмических языков программирования.

Задание

1.Построить регулярное выражение для представления десятичных чисел, описания переменных в алгоритмических языках.

2.Построить регулярное выражение для оператора объявления переменных.

3.Построить регулярное выражение для арифметических выражений.

7. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ

П Р А К Т И Ч Е С К О Е З А Н Я Т И Е № 2

КОНЕЧНЫЕ АВТОМАТЫ

Задание

Цель практического занятия. Изучить методы построения конечных авто-

матов. Освоить приемы работы с конечными автоматами.

1. Построить конечный автомат для одного из представленных ниже множеств цепочек из 0 и 1:

a) между вхождениями единиц четное число нулей,

b) за каждым вхождением пары единиц следует нуль,

c) каждый третий символ – единица,

d) все входные цепочки, начинающиеся 0 и заканчивающиеся 1,

e) все входные цепочки,содержащие в точности три 1,

f) все цепочки, в которых перед и после каждой 1 стоит 0,

g) все цепочки, не содержащие ни одной 1.

2. Записать регулярное выражение для десятичных чисел, оператора объявления переменных в Паскале.

3. Построить конечный автомат, который будет распознавать следующие зарезервированные слова Паскаля: STRING, INTEGER, IN, INLINE.

4. Описать словами множества цепочек, распознаваемых каждым из автоматов, изображенных на рис.

	0	1	
A	B	A	0
B	B	C	0
C	C	C	1

	0	1	
A	B	C	0
B	C	B	1
C	C	C	0

	0	1	
A	B	C	0
B	D	B	1
C	C	D	1
D	D	D	0

	0	1	
A	B	A	0
B	D	C	0
C	C	D	1
D	D	D	0

Рис.35

5. Найти самую короткую цепочку, распознаваемую автоматом, изображенном на рис.36. Найти четыре другие цепочки, распознаваемые этим автоматом. Найти четыре цепочки, которые отвергаются этим автоматом.

	0	1	
A	D	A	0
B	A	C	0
C	A	F	0
D	B	C	0
E	B	C	1
F	E	A	1

Рис.36

6.Для автомата, таблица переходов для которого изображена на рис.37, найти входную цепочку или цепочку с минимальной суммарной длиной – такие, что под их действием:

a) каждое состояние имеет место хотя бы один раз;

b) каждый переход происходит хотя бы один раз.

	0	1	
A	C	B	0
B	D	B	1
C	A	E	0
D	A	A	0
E	F	E	1
F	E	A	1

Рис.37.

7. Построить конечный автомат, который распознает химические формулы, составленные из восьми элементов: H, C, N, O, SI, S, CL и SN. Элементы в формулах разделяются запятыми. Они могут появляться в любом порядке и в любых сочетаниях. Формулы не обязательно представляют реально

существующие соединения. Вот несколько их образцов: H2,O; O,H7; SN,S,O4; CL; N,H7,C7,H5,O2.

8. Построить таблицу переходов конечного автомата, входной алфавит которого состоит из 31 символа: ОДИН, ДВА, ТРИ, … ДЕВЯТЬ, ДЕСЯТЬ, ОДИННАДЦАТЬ … ДЕВЯТНАДЦАТЬ, ДВАДЦАТЬ, ТРИДЦАТЬ … ДЕВЯНОСТО, СТО, ДВЕСТИ … СТА, СОТ; пусть выходной алфавит состоит из 10 символов 0, 1. 2, 3,…9 и пусть он допускает в качестве входа словесную запись любого числа от 1 до 999 и переводит вход в эквивалентную цифровую запись.

П Р А К Т И Ч Е С К О Е З А Н Я Т И Е № 3

ПОСТРОЕНИЕ ПРИВЕДЕННОГО КОНЕЧНОГО АВТОМАТЫ

Цель практического занятия. Ознакомиться с методами построения приведенного конечного автомата. Изучить алгоритмы выявления недостижимых состояний и поиска эквивалентных состояний.

Задание

1.Найти различающую цепочку для пары автоматов, изображенных на рис.37.

	0	1	
A	A	B	1
B	C	D	0
C	D	A	1
D	A	B	0

	0	1	
A	A	D	1
B	A	D	0
C	B	A	1
D	C	B	0

Рис.37.

2.Найти минимальную эквивалентную таблицу для каждого из автоматов, представленных на рис.38.

	0	1	
S1	S1	S3	0
S2	S7	S4	1
S3	S6	S5	0
S4	S1	S4	1
S5	S1	S4	0
S6	S7	S6	1
S7	S7	S3	0

		X	Y	
	1	4	1	1
	2	5	1	1
	3	4	5	0
	4	2	6	0
	5	1	7	0
	6	1	4	1
	7	2	5	1

Рис.38.

3.Для автоматов с рис. 38 найти недостижимые состояния.

4.Привести пример эквивалентных автоматов M и N, чтобы некоторое состояние автомата M было эквивалентно более чем одному состоянию автомата N.

5.Привести пример эквивалентных автоматов M и N, чтобы для некоторого состояния автомата M и N не было эквивалентных ему состояний.

	0	1	2	
A	C	E	Q	0
B	J	E	Q	1
C	J	A	H	0
D	F	A	Q	1
E	E	J	H	0
F	D	I	A	1
Q	H	A	J	0
H	Q	J	B	1
I	D	F	Q	0
J	B	H	Q	1

6.Найти недостижимые состояния автомата, представленного на рис.39.

Рис.39.

7.Рассмотреть множество всех распознавателей с двумя состояниями и

входным алфавитом {0, 1}. Определить, сколько автоматов в этом множестве. Сколько в нем неэквивалентных автоматов? Описать словами множество цепочек, распознаваемое каждым из этих неэквивалентных автоматов.

П Р А К Т И Ч Е С К О Е З А Н Я Т И Е № 4

РЕАЛИЗАЦИЯ КОНЕЧНОГО АВТОМАТА

Цель практического занятия. Выяснить каким образом представляются множества входных символов, множество состояний и таблица переходов для конечных автоматов. Рассмотреть методы решения задачи идентификации слов.

Задание

1.Составить алгоритм для формирования и заполнения таблицы идентификаторов, поступающих в следующем порядке: ABCD, BACD, DAB.C, EL, SON, DELO, DUB, KON. OK.NO, LOO, QROM, SOBA; выбрать и обосновать метод организации таблицы .

2.Составить алгоритм для поиска по таблице идентификаторов, созданной в пункте 1, и дополнить таблицу новым идентификатором в случае неудачного поиска; использовать методы идентификации:

a) метод расстановки со следующими функциями:
   – Rmod7, где R=код ASCII (второго символа)+код ASCII(третьего символа);
   – R*Rmod1 7, где R=сумма кодов ASCII(всех символов);
   – R*R, где R=сумма кодов ASCII(всех символов);
   – 5 первых двоичных разрядов +11 последних двоичных разрядов;
   – R*R,где R=сумма кодов ASCII(всех символов);
   – 5 первых двоичных разрядов (пропустить 5 двоичных разрядов)+5 последующих двоичных разрядов;
   – R*Rmod1 1, где R=сумма кодов ASCII(всех символов);
b) метод упорядоченного списка;
c) метод линейного списка.

3. Для автомата, изображенного на рис.40, построить список переходов

Рис.40.

4.Распознать следующее множество: МАССИВ, МАТРОС, П1,П2, П3, ДОМ, СОН, ДОМАШНИЙ; нарисовать в виде диаграммы структуру распознавателя этого множества, построенного: методом списка переходов, методом списка переходов для расширяющихся множеств, методом упорядоченного списка.

Оценить объем памяти, затрачиваемый при распозновании зарезервированных слов С++ методами: списка переходов, линейного списка, вектора переходов.

П Р А К Т И Ч Е С К О Е З А Н Я Т И Е № 5

АВТОМАТЫ С МАГАЗИННОЙ ПАМЯТЬЮ

Цель практического занятия. Ознакомиться с понятием автомат с магазинной памятью. Освоить методы и приемы работы с автоматом с магазинной памятью.

Задание

	v	C	,	(	)	
1	2	E	E	E	E	0
2	E	E	1	3	E	1
3	4	4	E	E	E	0
4	E	E	3	E	5	0
5	E	E	1	E	E	1
E	E	E	E	E	E	0

Построить МП-распознаватель для каждого из следующих множеств цепочек:

a) $\{1^n\ 0^m \mid n > m > 0\}$;

b) $\{1^n\ 0^m \mid n \geq m > 0\}$;

c) $\{1^n\ 0^n\ 1^m\ 0^m \mid n, m \geq 0\}$;

d) $\{1^n\ 0^m\ 1^n\ 0^m \mid n, m \geq 0\}$;

e) $\{1^n\ 0^m \mid m > n > 0\}$.

2. Для каждого из множеств первой задачи указать цепочку длины, большей 3. Показать последовательность конфигураций соответствующих автоматов, построенных в первом задании при распознавании каждой из цепочек.

3. Написать три цепочки, принадлежащие множеству, распознаваемому МП-автоматом с одним состоянием, изображенным на рис.28. Для каждой из этих цепочек указать соответствующие последовательности конфигураций, допускающие эти цепочки.

	a	b	c	┤
A	ВТОЛКНУТЬ (A) СДВИГ	ОТВЕРГНУТЬ	ВЫТОЛКНУТЬ СДВИГ	ОТВЕРГНУТЬ
B	ВТОЛКНУТЬ (C) СДВИГ	ВЫТОЛКНУТЬ ДЕРЖАТЬ	ВТОЛКНУТЬ (A) СДВИГ	ОТВЕРГНУТЬ
C	ВТОЛКНУТЬ (B) ДЕРЖАТЬ	ВТОЛКНУТЬ (C) СДВИГ	ВЫТОЛКНУТЬ СДВИГ	ОТВЕРГНУТЬ
	ВТОЛКНУТЬ (A) СДВИГ	ОТВЕРГНУТЬ	ОТВЕРГНУТЬ	ДОПУСТИТЬ

▽

Начальное содержимое магазина ▽

Рис. 41.

4. Привести пример такого множество цепочек, которое может распознать МП-автомат с магазинным алфавитом, содержащим маркер дна магазина и еще два символа, но не может распознать никакой МП-автомат, у которого магазинный алфавит состоит из маркера дна магазина и еще одного символа.

5. Составить три допускаемые цепочки и соответствующие последовательности конфигураций для МП-автомата с одним состоянием, представленного на рис.42.

0	1	┤	
ЗАМЕНИТЬ (АА) СДВИГ	ВЫТОЛКНУТЬ СДВИГ	ОТВЕРГНУТЬ	A
ОТВЕРГНУТЬ	ОТВЕРГНУТЬ	ДОПУСТИТЬ	∇

Начальное содержимое магазина ▽

Рис. 42

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 6

НИСХОДЯЩИЕ МЕТОДЫ ОБРАБОТКИ ЯЗЫКОВ С ПОМОЩЬЮ МП-АВТОМАТОВ

Цель: Ознакомиться с методами обработки формальных грамматик с помощью МП-автоматов. Научиться определять тип грамматики и строить множество выбора для них.

Задание

1 . Определить тип грамматики. Построить МП-распознаватель. Варианты грамматик представлены ниже.

Вариант I.

1 .<S>→a<S><A>
2.<S>→b<A>
3.<S>→ε
4.<A>→c<A>
5.<A>→ε

Вариант 2.

1 .<S>→a<A><B><S>
2.<S>→b
3.<A>→b<B>
4.<B>→c<B>
5.<B>→ε

Вариант 3.

1.<S>→a<A>b<S>
2.<S>→b
3.<A>→c<S>
4.<A>→b<A>
5.<A>→a

Вариант 4.

1 .<S>→a<S><A>
2.<S>→ε
3.<A>→a<A>b
4.<A>→b<A>
5.<A>→ε

Вариант 5.

1 .<S>→a<B><S><A>
2.<B>→b
3.<A>→a<A>
4.<A>→b<B>
5.<S>→b

Вариант 6.

1 .<S>→a<S><S><A>
2.<S>→b
3.<A>→b<B>
4.<A>→a<A><A>
5.<B>→ε

Вариант 7.

Вариант 8.

1.<S>→a<S><A>b<B>

2.<S>→b

3.<A>→a<A>

4.<A>→e

5.<B>→b<B>

6.<B>→e

1 .<S>→a<B><S><A

2. <A>→a<B>

3.<B>→b<B>

4.<B>→c<A><B>

5.<B>→ε

2. LL(1)-грамматика порождает дерево вывода, изображенное на рис.43. Выписать последовательность конфигураций МП-распознавателя для этой грамматики, возникающие при обработке следующих входных цепочек:

a) abcccdaa;

b) ε;

c) accc;

d) adab.

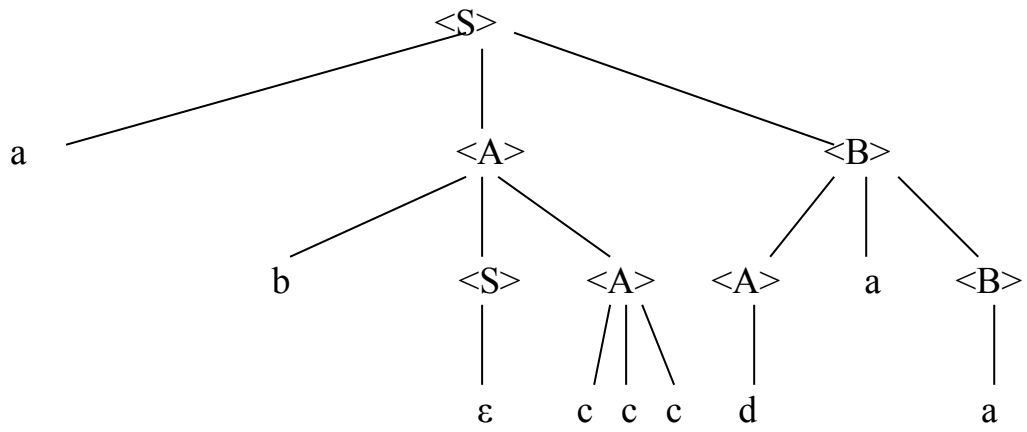


Рис.43.

3. Для следующей грамматики с начальным символом <S>

<S>→ a<A><B>b<C><D> <S>→ε

<A>→ <A><S>d <A>→ε

<B>→ <S><A>c <B>→e<C>

<C>→<C>q <C>→ε

<D>→a<B><D> <D>→ε

a) найти множество СЛЕД для каждого нетерминала;

b) найти ВЫБОР для каждого правила;

c) определить, является ли данная грамматика LL(1)-грамматикой.

4. Построить МП-распознаватель для грамматики арифметических выражений, в которых используются операции + и * .

1. <E>→<T><E->
2. <E->→ +<T><E->
3. <E->→ε
4. <T>→ <P><T->
5. <T->→ *<P><T->
6. <T->→ε
7. <P>→(<E>)
8. <P>→ I

5. Построить q-грамматику для конструкций языка СИ++. Считать терминалами <логическое выражение>, <арифметическое выражение> и <оператор>, т.е. когда доходит до них, вывод прекращается.

a) Описание массивов.

b) Описание условного оператора.

c) Операторы цикла.

6. Разработать алгоритм нахождения множества ВЫБОР для произвольной грамматики.

П Р А К Т И Ч Е С К О Е З А Н Я Т И Е № 7

ОБРАБОТКА ОШИБОК ПРИ НИСХОДЯЩЕМ РАЗБОРЕ

Цель: Ознакомиться с методами обработки ошибок при нисходящем анализе. Научиться разрабатывать сообщения об ошибках. Приобрести навыки разработки процедур нейтрализации ошибок.

Задание

1. Для следующей грамматики с начальным символом <оператор>:
    1. <оператор>→ beqin <описание> , <список операторов> end
    2. <описания>→ d; <описания>
    3. <описания>→ ε
    4. <список операторов>→ s <остальные операторы>
    5. <список операторов>→ε
    6. <остальные операторы>→ ; <список операторов>
    7. <остальные операторы>→ ε

a) показать, что это q-грамматика;

b) построить для нее распознаватель;

c) разработать подходящее множество сообщений об ошибках;

d) разработать локальные процедуры нейтрализации ошибок;

e) разработать глобальные процедуры нейтрализации ошибок.

2. Для грамматики арифметических выражений
    1. <E>→<T><E->
    2. <E->→ +<T><E->
    3. <E->→ε
    4. <T>→ <P><T->
    5. <T->→ *<P><T->
    6. <T->→ε
    7. <P>→(<E>)
    8. <P>→ I

a) построить управляющую таблицу нисходящего МП-распознавателя;

b) разработать сообщения об ошибках для всех отвергающих элемен-

тов таблицы;

с) разработать локальные процедуры нейтрализации ошибок.

8. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ОРГАНИЗАЦИИ МЕЖСЕССИОННОГО КОНТРОЛЯ ЗНАНИЙ СТУДЕНТОВ

1. Межсессионная аттестация студентов проводится дважды в семестр на 7 и 13 неделях 5-го семестра.

2. Аттестационная оценка складывается из оценок, полученных за контрольные работы по результатам промежуточного тестирования.

3. Организация аттестации студентов, проводится в соответствии с положением АмГУ о курсовых экзаменах и зачетах.

9. ТЕСТОВЫЕ ЗАДАНИЯ ДЛЯ ОЦЕНКИ КАЧЕСТВА ЗНАНИЙ ПО ДИСЦИПЛИНЕ

1) Конечный автомат является формализмом
   a) синтаксического блока
   b) генератора кода
   c) лексического блока
   d) семантического блока

2) Основная задача синтаксического анализа
   a) разбить входной текст на последовательность лексем
   b) выделение ключевых слов
   c) генерация объектного модуля
   d) разбор структуры программы

3) Описать множество цепочек , распознаваемых конечным автоматом

	0	1
A		
B		
C		

a) Цепочки из 0 и 1, которые начинаются 0 и заканчиваются 1.
b) Цепочки из 0 и 1, которые начинаются 0 или 1, после чего обязательно присутствует сочетание 01.
c) Цепочки, содержащие в точности три 1.
d) Цепочки, в которых каждый третий символ 1.

4) Лексический анализатор может работать как самостоятельная фаза трансляции или как подпрограмма, действующая по принципу
a) “дай лексему”
b) “получи лексему”

B	A	0
B	C	0
C	C	1

c) “дай входную последовательность”
d) “допускается ввод”

5) Для поиска эквивалентных состояний строится
a) таблица переходов
b) таблица эквивалентности
c) эквивалентная последовательность
d) вектор переходов

6) Операция вида P*Q для регулярных множеств P и Q называется
a) объединение
b) итерация
c) конкатенация
d) пересечение

7) Если в конечном автомате состояния s и t подобны, то это означает, что
a) их приемники эквивалентны
b) оба состояния либо допускающие, либо отвергающие
c) если на входе 1, то они переходят в одинаковые состояния
d) если при любом входном символе они переходят в идентичные состояния

8) Выход транслитератора, состоящий из двух частей – класса и значения, называется
a) символьная лексема
b) последовательная лексема

c) цепочка символов
d) пересечение

9) Метод переходов, согласно которому адреса процедур переходов хранятся в последовательных ячейках памяти (по одной ячейки для каждого входного символа) называется
a) методом списка переходов
b) метод вектора переходов
c) метод последовательных ячеек
d) метод индексов

10)Магазин в МП-автомате формируется как
a) очередь
b) множество
c) массив
d) стек

11)Операция ВЫТОЛКНУТЬ в МП-автомате означает
a) извлечение одного верхнего магазинного символа
b) извлечение последовательности магазинных символов
c) извлечение заданного количества магазинных символов
d) извлечение одного нижнего магазинного символа

12)Правила контекстно-свободной грамматики называются
a) терминалами
b) нетерминалами
c) операндами
d) продукциями

13)Для упрощения контекстно-свободной грамматики необходимо
a) найти недостижимые состояния
b) найти бесплодные состояния
c) найти вначале недостижимые, а затем бесплодные состояния
d) найти вначале бесплодные, а затем недостижимые состояния

14) Какие правила являются бесплодными для нижеприведённой грамматики?
1.<S>→ac
2.<S>→b<A>
3.<A>→c<B><C>
4.<B>→a<S><A>

5.<C>→b<C>
6.<C>→d

a) <S>,<C>
b) <A>,<B>
c) <A>,<S>
d) <B>,<C>

15)Какие из цепочек распознаются нижеприведённым МП-автоматом?

	0	1	-/
A	ВЫТОЛКНУТЬ СДВИГ	ОТВЕРГНУТЬ	ОТВЕРГНУТЬ
▽	ОТВЕРГНУТЬ	ВТОЛКНУТЬ(A) СДВИГ	ДОПУСТИТЬ

a) 1010101-/
b) 110011-/
c) 1001001-/
d) 101010-/

16) Какая грамматика не может содержать ε-правила?
a) LL(1)-грамматика
b) q-грамматика
c) s-иq-грамматики
d) s-грамматика

17) Какой из терминалов будет принадлежать множеству магазинных символов нижеприведённой грамматики?
1.<S>→a<A>b<S>
2.<S>→b
3.<A>→c<S>b
4.<A>→c<A>
5.<A>d

a) a
b) c
c) b
d) d

18) Для нижеприведённой грамматики с начальным нетерминалом <S> найти множество СЛЕД(<C>)

1.<S>→a<A><B>b<C><D>
2.<S>→ε
3.<A>→<A><S>d
4.<A>→ε
5.<B>→<S><A>c
6.<B>→c<C>
7.<C>→<C>q
8.<C>→q
9.<D>→a<B><D>
10.<D>→ε

a) {a,b,q}
b) {a,c}
c) {b,q}
d) {b,q,c}

19)В методе рекурсивного спуска для ПРОЦА метка вида :

P(№ правила): "Код для правила (№ правила)"
Возврат

применяется для правила вида

a) <A>→α
b) <A>→α*
c) <A>→aα
d) <A>→ε

20) Главная программа в методе рекурсивного спуска для КС-грамматики с начальным нетерминалом <S> начинается с того, что
a) входной символ присваивается переменной ВХОД
b) управление передаётся ПРОЦS
c) проводится проверка, является ли текущий символ концевым маркером
d) проводится проверка, является ли текущий символ маркером дна магазина

10. БИЛЕТЫ ДЛЯ ПРОВЕДЕНИЯ ЭКЗАМЕНА

Экзаменационный билет № 1

1. Структура компилятора. Лексический блок компилятора
2. Построить конечный автомат для распознавания множества цепочек из 0 и 1, удовлетворяющих условию между вхождениями единиц четное число нулей.

3. Определить тип грамматики.

1 .<S>→a<S><A>

2.<S>→b<A>

3.<S>→ε

4.<A>→c<A>

5.<A>→ε

Экзаменационный билет №2

1. Конечный автомат.
2. Определить тип грамматики. Построить МП-распознаватель. .

Вариант 2.

1 .<S>→a<A><B><S>

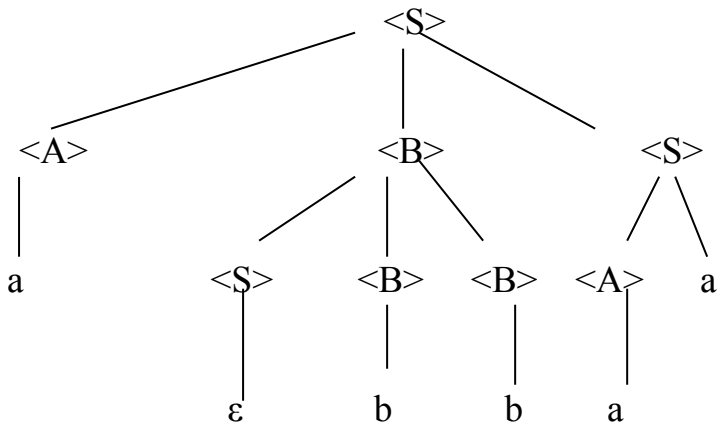
2.<S>→b

3.<A>→b<B>

4.<B>→c<B>

5.<B>→ε

3. КС-грамматика порождает дерево вывода. Построить левый вывод, соответствующий этому дереву.



Экзаменационный билет №3

1. Построение минимального конечного автомата
2. Какие из приведенных цепочек можно вывести в данной грамматике с начальным нетерминалом <S>?

<S>→ a <A> c <B> <A>→ <B> a <B>

<S>→ <B> d <S> <A>→ a <B> c

<B>→ a<S> c <A> <A> → a

<B>→ c <A> <B> <A> → b

a) aacb

b) aababcbadcd

3. Построить МП-распознаватель для грамматики арифметических выражений, в которых используются операции + и * .

1. <E>→<T><E->
2. <E->→ +<T><E->
3. <E->→ε
4. <T>→ <P><T->
5. <T->→ *<P><T->
6. <T->→ε
7. <P>→(<E>)
8. <P>→ I

Экзаменационный билет №4

1. Реализация конечных автоматов. Представление входных символов.
2. Для следующей грамматики с начальным символом <S>

<S>→ a<A><B>b<C><D> <S>→ε

<A>→ <A><S>d <A>→ε

<B>→ <S><A>c <B>→e<C>

<C>→<C>g <C>→ε

<D>→a<B><D> <D>→ε

a) найти множество СЛЕД для каждого нетерминала;

b) найти ВЫБОР для каждого правила;

с) определить является ли данная грамматика LL(1)-грамматикой.

3. Найти грамматику, порождающую множество арифметических выражений.

1. Реализация конечных автоматов. Представление состояний.
2. Описать язык, порождаемый следующей грамматикой, и нарисовать деревья вывода для трех цепочек этого языка.

   <E> → <P><R>

   <P> → (<E>)

   <P>→ I

   <R> → +<P><R>

   <R>→ *<P><R>

   <R>→ ↑<P><R>

   <R>→ ε
3. Построить МП-автомат, который будет выполнять следующий перевод: $1^n\ 0^m$ в $1^n\ 2^{2m}$, где n.>0, m>0;

Экзаменационный билет №6

1. Реализация конечных автоматов. Выбор переходов.
2. Выполнить перевод цепочки $w2w^r$ w принадлежит (0+1)* в цепочку 1^n0^m, где n и m соответственно число 1 и 0 в цепочке w. МП-автомат должен работать в двух фазах.
3. Найти лишние нетерминалы в следующей грамматике с начальным нетерминалом <S>:

<S>→ a<A><B><C> <S>→ b<C><E><S>

<S>→ a<E> <A>→ b<E>

<A>→ <S><C><D> <A>→ d

<B>→d<F><S> <B>→ a<B><C>

<C>→ a<E><S> <C>→ b<E>

<D>→ a<A><C> <D>→ d

<E>→ a<C><E> <E>→ ε

<F>→ <A><B> <F>→ a<F>

Экзаменационный билет №7

1. Идентификация слов. Метод автоматов и метод индексов.
2. Построить МП-автомат, который будет выполнять следующий перевод:

   $1^n\ 0^m\ 1^m\ 0^n$ в $1^m\ 0^{n+m}$, где n.>0, m>0;
3. Какие из приведенных цепочек можно вывести в данной грамматике с начальным нетерминалом <S>? В каждом случае построить левый, правый и дерево вывода.

<S>→ a <A> c <B> <A>→ <B> a <B>

<S>→ <B> d <S> <A>→ a <B> c

<B>→ a<S> c <A> <A> → a

<B>→ c <A> <B> <A> → b

а) aacbccb

в) aacabcbcccaacdca

Экзаменационный билет №8

1. Идентификация слов. Метод линейного и упорядоченного списков.
2. Найти КС-грамматику для языка:

   $\{1^n\ 0^m \mid n > m > 0\}$;
3. Составить три допускаемые цепочки и соответствующие последовательности конфигураций для МП-автомата с одним состоянием, представленного на рис.

0 1 ┤

ЗАМЕНИТЬ (АА) СДВИГ	ВЫТОЛКНУТЬ СДВИГ	ОТВЕРГНУТЬ	A
ОТВЕРГНУТЬ	ОТВЕРГНУТЬ	ДОПУСТИТЬ	∀

На-

чальное содержимое магазина

Экзаменационный билет №9

1. Идентификация слов .Метод расстановки.
2. Написать три цепочки, принадлежащие множеству, распознаваемому МП-автоматом с одним состоянием, изображенным на рис. Для каждой из этих цепочек указать соответствующие последовательности конфигураций, допускающие эти цепочки.

a	b	c	┤
ВТОЛКНУТЬ (A) СДВИГ	ОТВЕРГНУТЬ	ВЫТОЛКНУТЬ СДВИГ	ОТВЕРГНУТЬ
ВТОЛКНУТЬ (C) СДВИГ	ВЫТОЛКНУТЬ ДЕРЖАТЬ	ВТОЛКНУТЬ (A) СДВИГ	ОТВЕРГНУТЬ
ВТОЛКНУТЬ (B) ДЕРЖАТЬ	ВТОЛКНУТЬ (C) СДВИГ	ВЫТОЛКНУТЬ СДВИГ	ОТВЕРГНУТЬ
ВТОЛКНУТЬ (A) СДВИГ	ОТВЕРГНУТЬ	ОТВЕРГНУТЬ	ДОПУСТИТЬ

A

B

C

▽

Начальное содержимое магазина ▽

3. Для следующей грамматики с начальным символом <S>

<S>→ a<A><B>b<C><D> <S>→ε

<A>→ <A><S>d <A>→ε

<B>→ <S><A>c <B>→e<C>

<C>→<C>g <C>→ε

<D>→a<B><D> <D>→ε

определить состав процедур для распознавания грамматики методом рекурсивного спуска;

Экзаменационный билет №10

1. Язык как знаковая система.
2. Определить тип грамматики. Построить МП-распознаватель.

1.<S>→a<A>b<S>

2.<S>→b

3.<A>→c<S>

4.<A>→b<A>

5.<A>→a

3. Для автомата, таблица переходов для которого изображена на рис.6 найти входную цепочку или цепочку с минимальной суммарной длиной, такие, что под их действием:

c) каждое состояние имеет место хотя бы раз;

d) каждый переход происходит хотя бы раз?

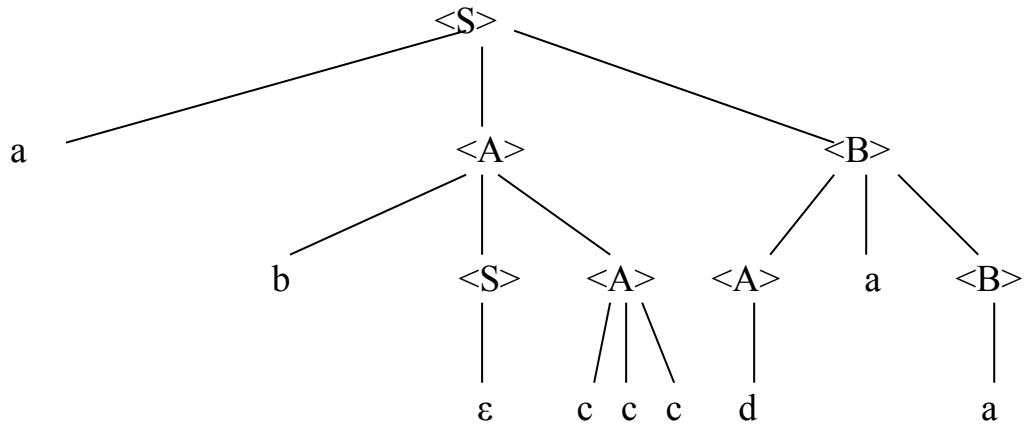
	0	1	
A	C	B	0
B	D	B	1
C	A	E	0
D	A	A	0
E	F	E	1
F	E	A	1

Экзаменационный билет №11

1. Определение автомата с магазинной памятью.
2. . LL(1)-грамматика порождает дерево вывода, изображенное на рис. Выписать последовательность конфигураций МП-распознавателя для этой грамматики, возникающие при обработке следующих входных цепочек:

a) abcccdaa

b) ε



3.Построить регулярное выражение для представления десятичных чисел.

Экзаменационный билет №12

1. Распознавание множества МП-автоматом.

2. Построить конечный автомат для следующего множеств цепочек из 0 и 1 . Каждый третий символ – единица.

3. Какие из приведенных цепочек можно вывести в данной грамматике с начальным нетерминалом <S>? В каждом случае построить левый, правый и дерево вывода.

<S>→ a <A> c <B>

<S>→ <B> d <S>

<B>→ a<S> c <A>

<B>→ c <A> <B>

<A>→ <B> a <B>

<A>→ a <B> c

<A> → a

<A> → b

aacb

aacabcbcccaacbca.

Экзаменационный билет №13

1. Перевод с помощью МП-автоматов.
2. Для следующей грамматики с начальным символом <S>

<S>→ a<A><B>b<C><D>	<S>→ε
<A>→ <A><S>d	<A>→ε
<B>→ <S><A>c	<B>→e<C>
<C>→<C>g	<C>→ε
<D>→a<B><D>	<D>→ε

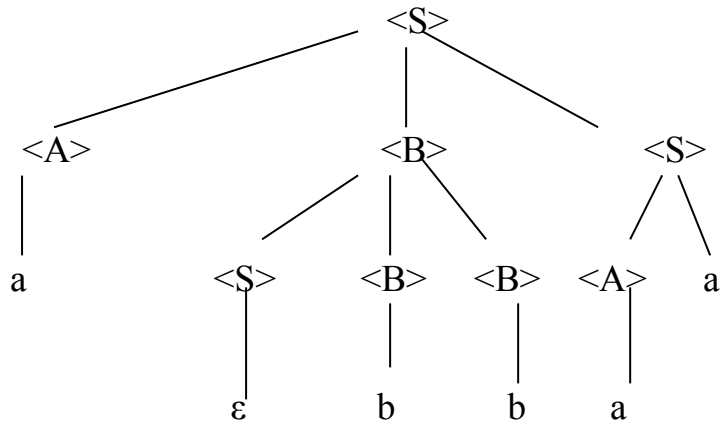
a) найти множество СЛЕД для каждого нетерминала;

b) найти ВЫБОР для каждого правила;

c) определить является ли данная грамматика LL(1)-грамматикой.

3. КС-грамматика порождает дерево вывода, изображенное на рис.

a. Нарисовать дерево вывода терминальной цепочки ab.



Экзаменационный билет №14

1. Формальные языки и формальные грамматики.
2. Составьте алгоритм для формирования и заполнения таблицы иден-

тификаторов, поступающих в следующем порядке: ABCD, BACD, DAB.C, EL, SON, DELO, DUB, KON. OK.NO, LOO, GROM, SOBA. Выбрать и обосновать метод организации таблицы .

3. Для автомата с рис. найти недостижимые состояния

	0	1	
S1	S1	S3	0
S2	S7	S4	1
S3	S6	S5	0
S4	S1	S4	1
S5	S1	S4	0
S6	S7	S6	1
S7	S7	S3	0

Экзаменационный билет №15

1. Контекстно-свободные грамматики. Выводы. Деревья.
2. Построить конечный автомат, который будет распознавать следующие зарезервированные слова Паскаля: STRING, INTEGER, IN, INLINE.
3. Нужно распознать следующее множество: МАССИВ, МАТРОС, П1,П2, П3, ДОМ, СОН, ДОМАШНИЙ. Нарисовать в виде диаграммы структуру распознавателя этого множества, построенного: методом списка переходов, методом списка переходов для расширяющихся множеств, методом упорядоченного списка.

Экзаменационный билет №16

1. Нисходящие методы обработки языков.
2. Найти недостижимые состояния автомата, представленного на рис.

	0	1	2	
A	C	E	G	0
B	J	E	G	1
C	J	A	H	0
D	F	A	G	1
E	E	J	H	0
F	D	I	A	1
G	H	A	J	0
H	G	J	B	1
I	D	F	G	0
J	B	H	G	1

3. Построить МП-автомат, который будет выполнять следующий перевод: $1^m\ 0^n$ в 1^{m-n}, если $m > n$;

0^{n-m}, если $m < n$;

ε, если $m = n$.

Экзаменационный билет №17

1. Транслирующая грамматика. Нисходящая обработка.
2. Найти минимальную эквивалентную таблицу для автомата, представленного на рис.

	0	1	
S1	S1	S3	0
S2	S7	S4	1
S3	S6	S5	0
S4	S1	S4	1
S5	S1	S4	0
S6	S7	S6	1
S7	S7	S3	0

3. Описать словами множества цепочек, распознаваемых автоматом.

0 1

	v	,	(	)	
1	2	E	E	E	0
2	E	E	1	3	E	1
3	4	E	E	E	0
4	E	E	3	E	5	0
5	E	E	1	E	E	1
E	E	E	E	E	E	0

Экзаменационный билет №18

1. Нисходящие методы обработки языков. S-грамматики.
2. Описать словами множество цепочек, распознаваемое МП-автоматом с одним состоянием, изображенным на рис.

0 1 ┤

ВТОЛКНУТЬ (0) СДВИГ	ВЫДАТЬ(1) СДВИГ	ВЫДАТЬ(0) ВЫТОЛКНУТЬ ДЕР- ЖАТЬ	0
ВТОЛКНУТЬ (0) СДВИГ	ВЫДАТЬ(1) СДВИГ	ДОПУСТИТЬ	∀

3. Для автомата построить список переходов

Экзаменационный билет №19

1. Нисходящие методы обработки языков. Q-грамматики.
2. Найти минимальную эквивалентную таблицу для автомата, представленного на рис.

	X	Y	
1	4	1	1
2	5	1	1
3	4	5	0
4	2	6	0
5	1	7	0
6	1	4	1
7	2	5	1

3. Построить конечный автомат для множества цепочек из 0 и 1 .

Все цепочки, в которых после каждой 1 стоит 0.

Экзаменационный билет №20

1. Нисходящие методы обработки языков. LL(1)-грамматики.
2. Для следующей грамматики с начальным символом <оператор>:

    1. <оператор>→ begin <описание> , <список операторов> end
    2. <описания>→ d; <описания>
    3. <описания>→ ε
    4. <список операторов>→ s <остальные операторы>

5. <список операторов>→ε

6. <остальные операторы>→ ; <список операторов>

7. <остальные операторы>→ ε

a) Показать, что это g-грамматика.

b) Построить для нее распознаватель.

c) Разработать подходящее множество сообщений об ошибках.

3. Построить конечный автомат для множества цепочек из 0 и 1 . Все цепочки, не содержащие ни одной 1.

Экзаменационный билет №21

1. Обработка ошибок при нисходящем разборе.
2. Описать словами множества цепочек, распознаваемых автоматом.

	0	1	
A	B	C	0
B	C	B	1
C	C	C	0

3. Построить МП-распознаватель для следующего множества цепочек:

$\{1^n\ 0^m \mid m > n > 0\}$.

Экзаменационный билет №22

1. Метод рекурсивного спуска.
2. Построить МП-распознаватель для грамматики арифметических выражений, в которых используются операции + и * .

1. <E>→<T><E->

2. <E->→ +<T><E->

3. <E->→ε

4. <T>→ <P><T->

5. <T->→ *<P><T->

6. <T->→ε

7. <P>→(<E>)

8. <P>→ I

3. Найти самую короткую цепочку, распознаваемую автоматом. Найти цепочку, распознаваемую этим автоматом. Найти цепочку, которая отвергается этим автоматом.

	0	1	
A	D	A	0
B	A	C	0
C	A	F	0
D	B	C	0
E	B	C	1
F	E	A	1

Экзаменационный билет №23

1. Синхронизирующие и начинающие символы
2. Для грамматики арифметических выражений

1. <E>→<T><E->

2. <E->→ +<T><E->

3. <E->→ε

4. <T>→ <P><T->

5. <T->→ *<P><T->

6. <T->→ε

7. <P>→(<E>)

8. <P>→ I

a) построить управляющую таблицу нисходящего МП-распознавателя;

b) разработать сообщения об ошибках для всех отвергающих элементов таблицы;

c) разработать локальные процедуры нейтрализации ошибок.

3. Построить конечный автомат для множества цепочек из 0 и 1 .

Все входные цепочки,содержащие в точности три 1.

Экзаменационный билет №24

1. Методы оптимизации объектного кода
2. Определить тип грамматики.

1 .<S>→a<S><A>

2.<S>→ε

3.<A>→a<A>b

4.<A>→b<A>

5.<A>→ε

3. Построить конечный автомат, который распознает химические формулы, составленные из восьми элементов: H, C, N, O, SI, S, CL и SN. Элементы в формулах разделяются запятыми. Они могут появляться в любом порядке и в любым сочетаниях. Формулы не обязательно представляют реально существующие соединения. Вот несколько образцов формул: H2,O; O,H7; SN,S,O4; CL; N,H7,C7,H5,O2.

Экзаменационный билет №25

1. Лишние нетерминалы контекстно-свободных грамматик.
2. Определить тип грамматики. Построить МП-распознаватель. .

1.<S>→a<A>b<S>

2.<S>→b

3.<A>→c<S>

4.<A>→b<A>

5.<A>→a

3.Для автомата, таблица переходов для которого изображена на рис. найти входную цепочку или цепочку с минимальной суммарной длиной, такие, что под их действием:

e) каждое состояние имеет место хотя бы раз;

f) каждый переход происходит хотя бы раз?

	0	1	
A	C	B	0
B	D	B	1
C	A	E	0
D	A	A	0
E	F	E	1
F	E	A	1

11. КАРТА КАДРОВОЙ ОБЕСПЕЧЕННОСТИ ДИСЦИПЛИНЫ

Лектор – старший преподаватель Соловцова Любовь Александровна

Руководитель практических работ – старший преподаватель Соловцова Любовь Александровна