МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра автоматизации обработки информации (АОИ)

Утверждаю:
Зав. кафедрой АОИ
профессор
_____Ю.П. Ехлаков
«___» _____2012 г.

Методические указания по выполнению
лабораторных работ по дисциплине

Теория автоматов и формальных языков

для студентов специальности

231000.62 «Программная инженерия»

Разработчик:
профессор
_____ И.А. Ходашинский

Томск – 2012

Содержание

Введение

В процессе подготовки и выполнения лабораторных работ формируются следующие компетенции:

1) овладение культурой мышления, способность к обобщению, анализу, восприятию информации, постановке цели и выбору путей ее достижения (ОК-1);

2) готовность к кооперации с коллегами, работе в коллективе (ОК-3);

3) понимание основных концепций, принципов, теорий и фактов, связанных с информатикой (ПК-1);

4) способность к формализации в своей предметной области с учетом ограничений используемых методов исследования (ПК-2);

5) навыки моделирования, анализа и использования формальных методов конструирования программного обеспечения (ПК-12).

В качестве технологии интерактивного обучения выбрана «мозговая атака», используемая для четырех приведенных методических указаниях лабораторных работ.

Формы контроля компетенций: защита отчетов по лабораторным работам, оценка работы студентов по результатам проведения «мозговой атаки».

Лабораторная работа №1.
Изучение процесса компиляции.

Цель работы.

Знакомство с общими принципами компиляции; построение компилятора арифметического выражения.

Порядок выполнения работы.

1. **Знакомство с фазами компиляции.**

Исходная программа, написанная на некотором языке программирования, есть не что иное, как строка знаков. Компилятор в конечном итоге превращает эту строку знаков в строку битов – объектный код. В этом процессе часто можно выделить подпроцессы со следующими названиями:

1. Лексический анализ.
2. Работа с таблицами.
3. Синтаксический анализ, или разбор.
4. Генерация кода промежуточного кода.
5. Оптимизация кода.
6. Генерация объектного кода.

1.1. *Лексический анализ.*

Первая фаза – лексический анализ. Входом компилятора, а, следовательно, и лексического анализатора, служит строка символов некоторого алфавита. Основу этого алфавита составляет таблица международной кодировки символов ASCII.

Работа лексического анализатора состоит в том, чтобы сгруппировать определенные терминальные символы в единые синтаксические объекты, называемые *лексемами.* Какие объекты считать лексемами, зависит от определения языка программирования. Лексема – это строка терминальных символов, с которой связывается лексическая структура, состоящая из пары вида (тип лексемы, некоторые данные). Первой компонентой пары является синтаксическая категория, такая, как «константа» или «идентификатор», а вторая – указатель: в ней указывается адрес ячейки, хранящей информацию об этой конкретной лексеме. Для данного языка число типов лексем предполагается конечным.

Таким образом, лексический анализатор – это транслятор, входом которого служит строка символов, представляющая исходную

программу, а выходом — последовательность лексем. Этот выход образует вход синтаксического анализатора.

Рассмотрим следующий оператор присваивания из языка, подобного Паскалю:

$$rub := (d1 + d2) * 26.54$$

На этапе лексического анализа будет обнаружено, что *rub*, *d*1 и *d*2 – лексемы типа идентификатора, а 26.54 – лексема типа константы. Знаки «:=», «+», «(», «)» и «*» сами являются лексемами. Допустим, что все константы и идентификаторы нужно отобразить в лексемы типа <ид>. Тогда выходом лексического анализатора, работающего на нашей входной строке, будет последовательность лексем

$$<ид>_1 := (<ид>_2 + <ид>_3)*<ид>_4.$$

Здесь вторая компонента лексемы (указатель данных) показана в виде нижнего индекса. Символы «:=»,«+», «(», «)» и «*» трактуются как лексемы, тип которых представлен ими самими. Они не имеют связанных с ними данных и, значит, не имеют указателей.

1.2. *Работа с таблицами.*

После того как в результате лексического анализа лексемы распознаны, информация о некоторых из них собирается и сохраняется в одной или нескольких таблицах.

Рассмотрим упрощенный пример таблицы, в которой хранится информация об идентификаторах. В ней перечислены, в частности, все идентификаторы вместе с относящейся к ним информацией.

Допустим, что в тексте встречается оператор

$$rub := (d1 + d2) * 26.54.$$

После просмотра этого оператора таблица может иметь вид табл. 1.1.

Таблица 1.1

Номер элемен-	Идентификатор	Информация
1	*rub*	Переменная с плавающей точкой
2	*d*1	Переменная с плавающей точкой
3	*d*2	Переменная с плавающей точкой
4	26.54	Константа с плавающей точкой

1.3. *Синтаксический анализ*

Выходом лексического анализатора является строка лексем, которая образует вход синтаксического анализатора, исследующего только первые компоненты лексем – их типы. Информация о каждой лексеме (вторая компонента) используется на более позднем этапе процесса компиляции для генерации машинного кода.

Синтаксический анализ, или разбор, как его еще называют,— это процесс, в котором исследуется строка лексем и устанавливается, удовлетворяет ли она структурным условиям, явно сформулированным в определении синтаксиса языка.

Выходом анализатора служит дерево, которое представляет синтаксическую структуру, присущую исходной программе.

Допустим, что выход лексического анализатора – строка лексем

$<\text{ид}>_1 := (<\text{ид}>_2 + <\text{ид}>_3) * <\text{ид}>_4$

Эта строка передает информацию о том, что необходимо выполнить следующие шаги:

1. $<\text{ид}>_2$ прибавить к $<\text{ид}>_3$,
2. результат (1) умножить на $<\text{ид}>_4$,
3. результат (2) поместить в ячейку, зарезервированную для $<\text{ид}>_1$.

Эту последовательность шагов можно представить наглядно с помощью помеченного дерева, показанного на рис. 1.2. Внутренние вершины представляют действия, которые надо выполнить.

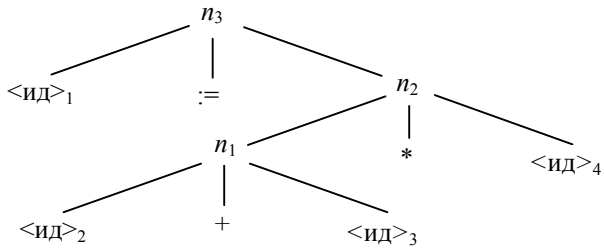

Рис. 1.2. Дерево разбора

Прямые потомки каждой вершины либо представляют аргументы, к которым нужно применить действие (если соответствующая вершина помечена идентификатором или является внутренней), либо помогают определить, каким должно быть это действие (в частности, это делают знаки +, *, :=). Заметим, что скобки, присутствующие в строке,

$$<\text{ид}>_1 = (<\text{ид}>_2 + <\text{ид}>_3) * <\text{ид}>_4$$

в дереве явно не указаны, хотя мы могли бы изобразить их в качестве прямых потомков вершины n_1. Роль скобок только в том, что они влияют на порядок операций. Если бы в упомянутой строке их не было, следовало бы поступить согласно обычному соглашению о том, что умножение «предшествует» сложению, и на первом шаге перемножить $<\text{ид}>_3$ и $<\text{ид}>_4$.

1.4. *Генерация кода*

Дерево, построенное синтаксическим анализатором, используется для того, чтобы получить перевод входной программы. Этот перевод может быть программой в машинном языке, но чаще он бывает программой в промежуточном языке, например, язык ассемблера.

Пусть машина имеет один рабочий регистр (сумматор, или регистр результата) и команды языка ассемблера, вид которых определен в табл. 1.2. Запись «$c(m) \rightarrow$ сумматор» означает, что содержимое ячейки памяти m надо поместить в сумматор. Через $=m$ обозначено собственно численное значение m. Предполагается, что ADD и MPY — операции с плавающей точкой.

Таблица 1.2

Команда	Действие
LOAD m	$c(m) \rightarrow$ сумматор
ADD m	c(сумматор) + $c(m) \rightarrow$ сумматор
MPY m	c(сумматор) * $c(m) \rightarrow$ сумматор
STORE m	c(сумматор) $\rightarrow m$
LOAD $=m$	$m \rightarrow$ сумматор
ADD $=m$	c(сумматор) + $m \rightarrow$ сумматор
MPY $=m$	c(сумматор) * $m \rightarrow$ сумматор

Выходом синтаксического анализатора служит дерево, с помощью этого дерева и информации, хранящейся в таблице имен, можно построить объектный код.

Существует несколько методов построения промежуточного кода по синтаксическому дереву. Один из них, называемый синтаксически управляемым переводом (трансляцией). В нем с каждой вершиной n связывается строка промежуточного кода $C(n)$. Код для вершины n строится сцеплением в фиксированном порядке кодовых строк, связан-

ных с прямыми потомками вершины *n*, и некоторых других фиксированных строк. Процесс перевода идет, таким образом, снизу вверх (т. е. от листьев к корню). Фиксированные строки и фиксированный порядок задаются используемым для перевода алгоритмом.

Здесь возникает важная проблема: для каждой вершины *n* выбрать код *C*(*n*) так, чтобы код, приписываемый корню, оказался искомым кодом всего оператора. Вообще говоря, нужна какая-то интерпретация кода *C*(*n*), которой можно было бы единообразно пользоваться во всех ситуациях, где может встретиться вершина *n*.

Для арифметических операторов присваивания нужная интерпретация получается весьма естественно; мы опишем ее ниже. В общем случае при применении синтаксически управляемой трансляции интерпретация должна задаваться создателем компилятора. Эта задача может оказаться легкой или трудной, и в трудных случаях, возможно, придется учитывать структуру всего дерева.

В качестве характерного примера опишем синтаксически управляемую трансляцию арифметических выражений. Заметим, что на рис. 1.2 есть три типа внутренних вершин, зависящих от того, каким из знаков :=, +, * помечен средний потомок. Эти три типа вершин показаны на рис. 1.3, где треугольниками изображены произвольные поддеревья (возможно, состоящие из единственной вершины). Для любого арифметического оператора присваивания, включающего только арифметические операции + и *, можно построить дерево с одной вершиной (корнем) типа *а* и остальными внутренними вершинами только типов *б* и *в*.

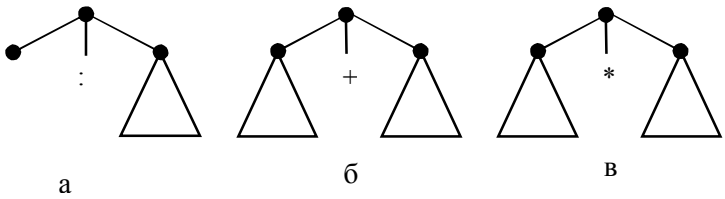

Рис. 1.3. Типы внутренних

Код, соответствующий вершине *n*, будет иметь следующую интерпретацию:

(1) Если *n* — вершина типа *а*, то *C*(*n*) будет кодом, который вычисляет значение выражения, соответствующего правому поддереву, и помещает его в ячейку, зарезервированную для идентификатора, которым помечен левый потомок.

(2) Если *n* — вершина типа *б* или *в,* то строка LOAD *C*(*n*) будет кодом, засылающим в сумматор значение выражения, соответствующего поддереву, над которым доминирует вершина *n.*

Так, для дерева, изображенного на рис. 2.2,
код LOAD $C(n_1)$ засылает в сумматор значение выражения <ид>$_2$ + <ид>$_3$,
код LOAD $C(n_2)$ засылает в сумматор значение выражения (<ид>$_2$ + <ид$_3$>)*<ид>$_4$,
код $C(n_3)$ засылает в сумматор значение последнего выражения и помещает его в ячейку, предназначенную для <ид$_1$>.

Теперь надо показать, как код *C*(*n*) строится из кодов потомков вершины *n.* В дальнейшем мы будем предполагать, что операторы языка ассемблера записываются в виде одной строки и отделяются друг от друга точкой с запятой или началом новой строки. Кроме того, мы будем предполагать, что каждой вершине *n* дерева приписано число *l*(*n*), называемое ее *уровнем,* которое означает максимальную длину пути от этой вершины до листа. Таким образом,

если *n* — лист, то $l(n) = 0$,

если *n* имеет потомков $n_1, ..., n_k$, то $l(n) = \max l(n_i) + 1$, $1 \le i \le k$.

Уровни *l(n)* можно вычислять снизу вверх одновременно с вычислением кодов *C*(*n*). Уровни записываются для того, чтобы контролировать использование временных ячеек памяти. Две нужные нам величины нельзя засылать в одну и ту же ячейку памяти. На рис. 1.4 показаны уровни вершин дерева, изображенного на рис. 1.2.

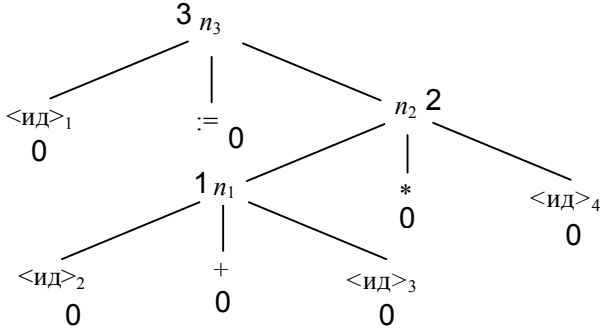

Рис. 1.4. Дерево разбора с уровнями

Теперь определим синтаксически управляемый алгоритм генерации кода, предназначенный для вычисления кодов $C(n)$ всех вершин дерева, состоящего из листьев, корня типа *а* и внутренних вершин типов *б* и *в*.

Алгоритм. Синтаксически управляемая трансляция простых операторов присваивания.

Вход. Помеченное упорядоченное дерево, представляющее оператор присваивания, включающий только арифметические операции + и *. Предполагается, что уровни всех вершин уже вычислены.

Выход. Код в языке ассемблера, вычисляющий этот оператор присваивания.

Алгоритм. Делать шаги (1) и (2) для всех вершин уровня 0, затем для вершин уровня 1 и т. д., пока не будут обработаны все вершины дерева.

(1) Пусть n – лист с меткой <ид>$_j$.

(i) Допустим, что элемент j таблицы идентификаторов является переменной. Тогда $C(n)$ – имя этой переменной.

(ii) Допустим, что элемент j таблицы идентификаторов является константой k. Тогда $C(n)$ –строка $=k$.

(2) Если n – лист с меткой :=, * или +, то $C(n)$ – пустая строка. (В этом алгоритме нам не нужно или мы не хотим выдавать выход для листьев, помеченных :=, * или +.)

(3) Если n – вершина типа *а* и ее прямые потомки – это вершины n_1 n_2 и n_3, то $C(n)$ – строка LOAD $C(n_3)$; STORE $C(n_1)$.

(4) Если n – вершина типа *б* и ее прямые потомки – вершины n_1 n_2 и n_3, то $C(n)$ – строка $C(n_3)$; STORE \$$l(n)$; LOAD $C(n_1)$; ADD \$$l(n)$. Эта последовательность команд занимает временную ячейку, именем которой служит знак \$ вместе со следующим за ним уровнем вершины n. Непосредственно видно, что если перед этой последовательностью поставить LOAD, то значение, которое она в конце концов поместит в сумматор, будет суммой значений выражений поддеревьев, над которыми доминируют вершины n_1 и n_3.

Два замечания относительно выбора временных имен. Во-первых, эти имена должны начинаться знаком \$, так что их нельзя перепутать с именами идентификаторов в Паскале. Во-вторых, в силу способа выбора $l(n)$ можно утверждать, что $C(n)$ не содержит временного имени \$$i$, если i больше $l(n)$. В частности, $C(n_1)$ не содержит \$$l(n)$. Можно, таким образом, гарантировать, что значение, помещенное в ячейку \$$l(n)$, будет еще нахо-

диться там в тот момент, когда его надо прибавить к содержимому сумматора.

(5) Если n – вершина типа *в,* а все остальное, как в (4), то $C(n)$ – строка $C(n_3)$; STORE \$$l(n)$; LOAD $C(n_1)$; MPY \$$l(n)$

Пример. Применим алгоритм к дереву, изображенному на рис. 1.2. То же дерево, на котором явно выписаны коды, сопоставляемые с каждой его вершиной, показано на рис. 1.5. С вершинами, помеченными <ид>$_1$, ..., <ид>$_4$, связаны соответственно коды *rub*, *d*1, *d*2 и =26.54. Теперь мы можем вычислить $C(n_1)$. Так как $l(n_1) = 1$, то формула из правила (4) дает

$$C(n_1) = \quad d2;\ \text{STORE \$1};\ \text{LOAD } d1;\ \text{ADD \$1}$$

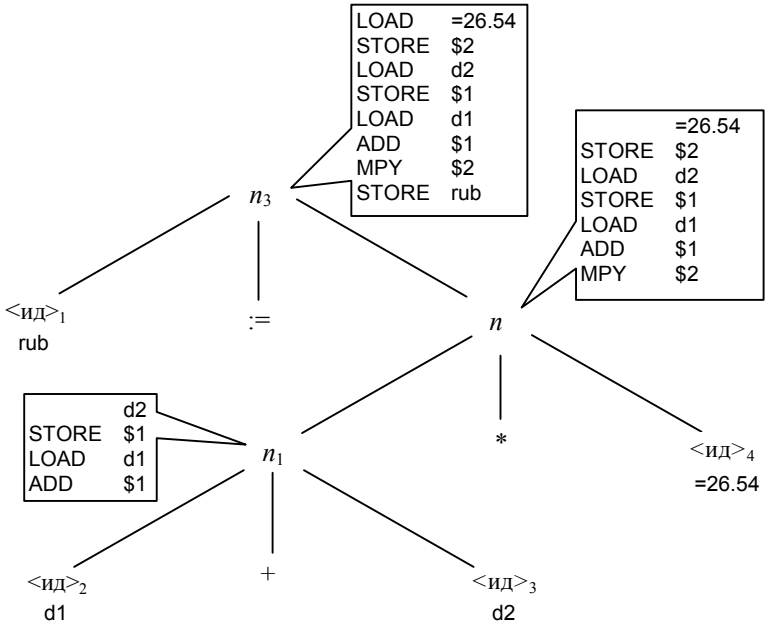

Рис. 1.5. Дерево с кодами

Таким образом, код LOAD $C(n_1)$ вычисляет в сумматоре сумму значений переменных *d1* и *d2*, хотя и делает это неоптимально. Неоп-

тимальность отчасти можно сгладить в процессе оптимизации кода; кроме того, можно разработать правила построения кода, принимающие во внимание особые случаи.

Далее можно вычислить $C(n_2)$ по правилу (5) и получить

$C(n_2)$ равно = 26.54; STORE \$2; LOAD $C(n_1)$; MPY \$2

Здесь $C(n_1)$ – строка, построенная для вершины n_1, а \$2 используется как имя временной ячейки, поскольку $l(n_2) = 2$. Затем вычисляем $C(n_3)$ по правилу (3) и получаем

$C(n_3)$ равно LOAD $C(n_3)$; STORE *rub*

Список команд языка ассемблера (вместо точки с запятой разделителем в нем служит новая строка), который является переводом оператора *rub* := (*d*1 + *d*2) * 26.54 таков:

```
LOAD    =26.54
STORE   $2
LOAD    d2
STORE   $1
LOAD    d1
ADD     $1
MPY     $2
STORE   rub
```

1.5. *Оптимизация кода*

Рассмотрим несколько эвристических преобразования, с помощью которых можно сделать предыдущий код более коротким:

(1) Если «+» – коммутативная операция, можно заменить последовательность команд вида LOAD α; ADD β последовательностью LOAD β; ADD α. Требуется, однако, чтобы в других местах программы не было перехода к оператору ADD β.

(2) Если «*» – коммутативная операция, можно заменить LOAD α; MPY β на LOAD β; MPY α.

(3) Последовательность операторов вида STORE α; LOAD α можно удалить из программы при условии, что либо ячейка α не будет далее использоваться, либо перед использованием α будет заполнена заново. (Чаще можно удалить один лишь оператор LOAD α; для этого только требуется, чтобы к оператору LOAD α не было перехода в других местах программы.)

(4) Последовательность LOAD α; STORE β можно удалить, если за ней следует другой оператор LOAD и нет перехода к

оператору STORE β, а последующие вхождения β будут заменены на α вплоть до того места, где появляется другой оператор STORE β (но исключая его).

Пример. Эти четыре преобразования выбраны из-за их применимости к предыдущей программе. Вообще таких преобразований много и надо пробовать применять их в разных комбинациях. Заметим, что в случае нашей программы правило (1) применимо к последовательности LOAD *d*1; ADD $1, и можно попробовать временно заменить ее на LOAD $1; ADD *d*1, получив при этом код

```
LOAD    =26.54
STORE   $2
LOAD    d2
STORE   $1
LOAD    $1
ADD     d1
MPY     $2
STORE   rub
```

Теперь ясно, что в данной программе можно удалить последовательность STORE $1; LOAD $1 по правилу (3). Таким образом, мы получаем код

```
LOAD    =26.54
STORE   $2
LOAD    d2
ADD     d1
MPY     $2
STORE   rub
```

Теперь к последовательности LOAD = 26.54; STORE $2 можно применить правило (4). Эти две команды удаляются и $2 в команде MPY $2 заменяется на =26.54. Окончательный код таков:

```
LOAD    d2
ADD     d1
MPY     =26.54
STORE   rub
```

Код, приведенный ниже, – самый короткий, какой можно получить с помощью наших четырех и любых других разумных преобразований.

2. **Варианты задания исходного арифметического выражения**.

№ вар.	Выражение
1	$z = (x^2 + 1)(y^2 – 1)+1$
2	$z = 7y^2 –4x^2+7$
3	$z = 2x\, y(y^2 – x +2)$
4	$z = 2y^4 + x – 2$
5	$z = x^2\, y^2 – (x+y+1)$
6	$z = (xy – 1)(x\, y + 1)$
7	$z = 2y^2 + x^3 – 2$
8	$z = (2x^2 + 1)(y^2 – 2)$
9	$z = 2y\,(xy – x^2 –1)$
10	$z = 2x^2 – 2y^2 +5$

3. Задание

3.1. *Безкомпьютерная обработка*

Выполните пять фаз компиляции заданного арифметического выражения на бумаге. В результате необходимо получить следующее: последовательность лексем, таблица имен, дерево разбора, дерево уровней, дерево генерации промежуточного кода, оптимизированная ассемблер-программа.

3.2. *Компьютерная обработка*

- Представьте дерево разбора в виде структуры Вашего любимого языка программирования.
  Внимание. Не нужно писать программу автоматического формирования произвольного дерева разбора.
- Программно реализуйте алгоритм синтаксически управляемой трансляции простых операторов присваивания.
- Программно реализуйте эвристики преобразования для оптимизации кода.

Мозговая атака

Этапы проведения.

1. Предложить два *вопроса для обсуждения*:
   - как изменятся определенные выше алгоритмы и эвристики при добавлении арифметических операций вычитания и деления?
   - предложите свои эвристики для генерации и оптимизации кода.
2. Предложить высказать свои мысли по данным вопросам.

3. Записать все прозвучавшие высказывания без возражений, но, возможно, с уточнениями.
4. По окончанию прочитать все, что записано со слов участников.
5. Обсудить все варианты ответов, выбрать главные и второстепенные.
6. Выяснить, как можно использовать полученные результаты при выполнении данной лабораторной работы.

Отчет

Отчет должен содержать следующие обязательные пункты:

1. Автор (ы) проекта.
2. Последовательность лексем.
3. Таблица имен.
4. Дерево разбора, дерево уровней.
5. Дерево генерации промежуточного кода.
6. Оптимизированная ассемблер-программа.
7. Листинги программ выполнения фаз компиляции арифметического выражения.

Контрольные вопросы

1. Укажите основную функцию компилятора.
2. Назовите основные части компилятора и их назначение.
3. Что такое лексическая структура?
4. Укажите вход и выход лексического анализатора.
5. Для чего необходима таблица имен?
6. Укажите вход и выход синтаксического анализатора.
8. Что такое синтаксическая структура?
9. Какие проблемы возникают при генерации кода?
10. Опишите синтаксически управляемую трансляцию арифметических выражений.
11. Какие проблемы возникают при оптимизации кода?
12. Опишите эвристики для оптимизации кода транслируемых арифметических выражений.

Лабораторная работа №2.
Регулярные языки.

Цель работы.

Знакомство с методами задания регулярных языков; задание языка с помощью праволинейной грамматики; задание языка с помощью конечного автомата.

Порядок выполнения работы.

1. **Знакомство с методами задания регулярных языков**

Регулярные языки задаются с помощью регулярных выражений, с помощью праволинейной грамматики, с помощью детерминированного и недетерминированного конечного автомата.

1.1. *Регулярные множества и регулярные выражения*

Пусть Σ – конечный алфавит. *Регулярное множество в алфавите* Σ определяется рекурсивно следующим образом:

(1) $\varnothing$ (пустое множество) – регулярное множество в алфавите Σ;

(2) $\{e\}$ – регулярное множество в алфавите Σ;

(3) $\{a\}$ – регулярное множество в алфавите Σ для каждого $a \in \Sigma$;

(4) если P и Q – регулярные множества в алфавите Σ, то таковыми же являются и множества

(а) $P \cup Q$,

(б) PQ,

(в) P^*;

(5) ничто другое не является регулярным множеством в алфавите Σ.

Регулярные выражения в алфавите Σ и регулярные множества, которые они обозначают, определяются рекурсивно следующим образом:

(1) $\varnothing$ – регулярное выражение, обозначающее регулярное множество $\varnothing$,

(2) e – регулярное выражение, обозначающее регулярное множество $\{e\}$,

(3) если $a \in \Sigma$, то a – регулярное выражение, обозначающее регулярное множество $\{a\}$,

(4) если p и q – регулярные выражения, обозначающие регулярные множества P и Q соответственно, то

(а) $(p + q)$ – регулярное выражение, обозначающее $P \cup Q$,

(б) (pq) – регулярное выражение, обозначающее PQ;

(в) $(p)^*$– регулярное выражение, обозначающее P^*;

(5) ничто другое не является регулярным выражением.

Несколько примеров регулярных выражений и обозначаемых ими множеств:

(1) 01 обозначает {01}.
(2) 0* обозначает {0}*.
(3) (0+1)* обозначает {0, 1}*.
(4) (0+1)*011 обозначает множество всех цепочек, составленных из нулей и единиц и оканчивающихся цепочкой 011.
(5) $(a+b)$ $(a + b + 0 + 1)^*$ обозначает множество всех цепочек из {0, 1, *a, b*}*,* начинающихся с буквы *a* или *b.*

1.2. *Праволинейные грамматики*

Формально грамматика задается четверкой

$$G = (N, \Sigma, P, S),$$

где N – конечное множество нетерминальных символов;

Σ – конечное множество терминальных символов, непересекающееся с N;

P – конечное множество правил;

S – начальный или исходный символ.

Различают регулярные грамматики с левосторонними продукциями вида

$$A \to Bx \text{ или } A \to Ax \text{ или } A \to x,$$

и регулярные грамматики с правосторонними продукциями вида

$$A \to xB, \text{ или } A \to xA, \text{ или } A \to x, \text{ где } A, B \in N, x \in \Sigma^*.$$

1.3. *Конечный автомат*

Конечным автоматом называется пятерка

$$A = (Q, \Sigma, \delta, q_0, F),$$

где $Q = \{q_0, q_2, \ldots, q_n\}$– конечное множество состояний устройства управления;

$\Sigma = \{a_1, a_2, \ldots, a_k\}$ – конечный входной алфавит;

δ – функция переходов, отображающая $Q \times \Sigma$ во множество подмножеств множества Q, или другими словами, функция переходов по данному текущему состоянию и текущему входному символу указывает все возможные следующие состояния;

q_0 – начальное состояние устройства управления;

F – множество заключительных состояний, причем $F \subseteq Q$.

Конечный автомат называется *детерминированным*, если множество $\delta(q, a)$ содержит не более одного состояния для любых $q \in Q$ и

$a \in \Sigma$. Если множество $\delta(q, a)$ содержит более одного состояния, то автомат называется *недетерминированным*.

1.4. *Эквивалентность языков, определяемых праволинейной грамматикой и конечным автоматом*.

Рассмотрим теперь произвольную автоматную грамматику $G = (N, \Sigma, P, S)$ с правосторонними продукциями:

$$C \to xB, \text{ или } C \to x, \text{ где } C, B \in N, \ x \in \Sigma^*.$$

Построим для нее недетерминированный конечный автомат

$$A = (Q, \Sigma, \delta, q_0, F),$$

где алфавиты Σ в грамматике и автомате совпадают,
$Q = N \cup \{D\}$, причем символ D не должен содержаться в N,
$q_0 = S$,
$F = \{D\}$,
а δ определяется следующим образом:

1) каждой продукции вида $C \to x$ ставится в соответствие команда $\delta(C, x) = D$;
2) каждой продукции вида $C \to xB$ ставится в соответствие команда $\delta(C, x) = B$;
3) других команд нет.

Пример. Пусть язык задан регулярным выражением

$$(a+b)\,(a + b + 0 + 1)^*$$

Грамматика $G = (N, \Sigma, P, S)$, порождающая строки языка задается следующим образом:

$$N = \{S, L\}, \qquad \Sigma = \{a, b, 0, 1\},$$
$$P = \{S \to aL,\ S \to bL,\ S \to a,\ S \to b,\ L \to aL,\ L \to bL,$$
$$L \to 0L,\ L \to 1L,\ L \to a,\ L \to b,\ L \to 0,\ L \to 1\}.$$

Вывод строки $a01b0$ в данной грамматике будет выглядеть следующим образом:

$$\begin{aligned} S &\Rightarrow aL \\ &\Rightarrow a0L \\ &\Rightarrow a01L \\ &\Rightarrow a01bL \\ &\Rightarrow a01b0 \end{aligned}$$

Построим для грамматики G конечный автомат

$$A = (Q, \Sigma, \delta, q_0, F),$$

$\Sigma = \{a, b, 0, 1\}$, $Q = \{S, L\} \cup \{D\}$, $q_0 = S$, $F = \{D\}$,
$\delta(S, a) = L$; $\delta(S, b) = L$; $\delta(S, a) = D$; $\delta(S, b) = D$;

$\delta(L, a) = L$; $\delta(L, b) = L$;
$\delta(L, 0) = L$; $\delta(L, 1) = L$; $\delta(L, a) = D$; $\delta(L, b) = D$;
$\delta(L, 0) = D$; $\delta(L, 1) = D$;

Распознавание строки $a01b0$ данным автоматом будет выполнено следующим образом:

$$(S, a01b0) \vdash (L, 01b0)$$
$$\vdash (L, 1b0)$$
$$\vdash (L, b0)$$
$$\vdash (L, 0)$$
$$\vdash (D, \varepsilon)$$

Теперь рассмотрим произвольный недетерминированный конечный автомат

$$A = (Q, \Sigma, \delta, q_0, F).$$

Такому автомату можно поставить в соответствие следующую автоматную грамматику:

$$G = (N, \Sigma, P, S),$$

где алфавиты Σ в грамматике и автомате совпадают; $N = Q$, $S = q_0$, а множество P строится следующим образом:

1) каждой команде автомата $(q_i, a_j) = q_k$ ставится в соответствие продукция $q_i \to a_j q_k$, если $q_i, q_k \in Q$, $a_j \in \Sigma$;
2) каждой команде $(q_i, a_j) = q_k$ ставится в соответствие еще одна продукция $q_i \to a_j$, если $q_k \in F$;
3) других продукций нет.

Рассмотрим конечный детерминированный автомат $A1$, допускающий все цепочки из символов 0 и 1, которые начинаются и оканчиваются 1. A1=($\{p, q, r\}$, $\{0, 1\}$, δ, p, $\{r\}$), где δ задается следующими командами:

$\delta(p, 1) = q$; $\delta(q, 0) = q$; $\delta(q, 1) = r$; $\delta(r, 0) = q$;
$\delta(r, 1) = r$;

этому автомату поставим в соответствие следующую автоматную грамматику

$$G = (N, \Sigma, P, S),$$

$\Sigma = \{0, 1\}$, $N = \{p, q, r\}$, $S = p$,
$P = \{p \to 1q;$ $q \to 0q;$ $q \to 1r;$ $q \to 1;$ $r \to 0q;$
$r \to 1r;$ $r \to 1\}$

2. **Варианты задания регулярного языка**.

№ вар.	Регулярное выражение
1	$(a^*+b^*)c$
2	$a(a+b)^*b$
3	$(aa+bb)^+$
4	$a^n b^m c^k$ где $n,m,k>0$
5	$(a+b)^* aa(a+b)^*$
6	$a^+ b^+ c^* b$
7	$((ba^*)(a^* b))^+$
8	$(10+1)^*(10)^+(1+10)^*$
9	$0(0+1)^*0+1(0+1)^*1$
10	$((1^*0)\ (1^*0)\ (1^*0))^+$

3. Задание

3.1. *Безкомпьютерная обработка*

- Опишите язык, определяемый данным регулярным выражением, приведите примеры строк, принадлежащих и не принадлежащих языку.
- Для выбранного регулярного выражения определите конечный автомат, допускающий строки заданного языка.
- Определите последовательность конфигураций автомата при распознавании строк, принадлежащих и не принадлежащих языку.
- Определенному выше конечному автомату поставьте в соответствие праволинейную грамматику, порождающую строки заданного языка.

3.2. *Компьютерная обработка*

Напишите на Вашем любимом языке программирования программу, моделирующую работу конечного автомата или праволинейной грамматики.

Мозговая атака

Этапы проведения.

1. Предложить *вопрос для обсуждения*:
   - как формализовать процесс построения конечного автомата по заданному регулярному выражению?

2. Предложить высказать свои мысли по данным вопросам.
3. Записать все прозвучавшие высказывания без возражений, но, возможно, с уточнениями.
4. По окончанию прочитать все, что записано со слов участников.
5. Обсудить все варианты ответов, выбрать главные и второстепенные.
6. Выяснить, как можно использовать полученные результаты при выполнении данной лабораторной работы.

Отчет

Отчет должен содержать следующие обязательные пункты:

8. Автор (ы) проекта.
9. Регулярное выражение.
10. Описание языка и примеры строк, принадлежащих и не принадлежащих языку.
11. Описание конечного автомата.
12. Примеры не менее четырех последовательностей конфигураций автомата при распознавании строк, принадлежащих и не принадлежащих языку.
13. Описание праволинейной грамматики.
14. Листинги программ моделирования работы конечного автомата (праволинейной грамматики).

Контрольные вопросы

1. Какими средствами может быть задан регулярный язык?
2. Укажите соответствие между регулярными выражениями и регулярными множествами.
3. Что такое праволинейная грамматика, в чем ее преимущество?
4. Определите структуру конечного автомата и функции его элементов.
5. Какими средствами могут быть описаны функции перехода конечного автомата?
6. Дайте толкования понятиям «конфигурация конечного автомата» и «такт конечного автомата».
7. Укажите отличия между детерминированным и недетерминированным конечным автоматом.
8. Определите язык, допускаемый конечным автоматом.
9. Какие из следующих множеств регулярны? Для тех, которые регулярны, напишите регулярные выражения.

а) Множество цепочек с равным числом нулей и единиц.

б) Множество цепочек из $\{0, 1\}^*$ с четным числом нулей и четным числом единиц.

в) Множество цепочек из $\{0, 1\}^*$, длины которых делятся на 3.

г) Множество цепочек из {0, 1}*, не содержащих подцепочки 101.

Лабораторная работа №3. Контекстно-свободные языки.

Цель работы.

Знакомство с методами задания контекстно-свободных языков; задание языка с помощью КС-грамматики; задание языка с помощью автомата с магазинной памятью.

Порядок выполнения работы.

1. **Знакомство с методами задания КС-языков**

Контекстно-свободные языки задаются с помощью КС-грамматик, с помощью детерминированного и недетерминированного автомата с магазинной памятью.

1.1. *КС-грамматики*

Формально грамматика задается четверкой

$$G = (N, \Sigma, P, S),$$

где N – конечное множество нетерминальных символов;

Σ – конечное множество терминальных символов, непересекающееся с N;

P – конечное множество правил;

S – начальный или исходный символ.

Продукции КС-грамматик имеют следующий вид:

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

1.2. *Автомат с магазинной памятью*

Автомат с магазинной памятью – это распознаватель, имеющий рабочую память, в которой данные хранятся и используются как патроны в магазине автоматического оружия, т.е. в данный момент доступен только верхний элемент магазина.

Автомат с магазинной памятью (МПА)– это семерка

$$M = (Q, \Sigma, T, \delta, q_0, Z_0, F),$$

где $Q = \{q_0, q_2, \ldots, q_n\}$– конечное множество состояний устройства управления;

$\Sigma = \{a_1, a_2, \ldots, a_k\}$ – конечный входной алфавит;

T – конечный алфавит магазинных символов;

δ – функция переходов, отображающая $Q \times \Sigma \times T$ во множество конечных подмножеств множества $Q \times T^*$;

q_0 – начальное состояние устройства управления, $q_0 \in Q$;

Z_0 – начальный символ магазина, $Z_0 \in T$;

F – множество заключительных состояний, $F \subseteq Q$.

Автомат с магазинной памятью имеет две ленты: входную, которую можно только читать и которая содержит распознаваемую входную строку, и рабочую, содержащую магазинные символы, которые можно как читать, так и писать. Управляющее устройство находится в одном из состояний Q , устройство имеет две головки: одну для работы с входной лентой, другую – с рабочей, причем эта головка всегда указывает на верхнюю ячейку магазина. За один такт автомат может выполнить следующие действия над символами магазина:

- стереть символ из верхней ячейки магазина, при этом все символы магазина смещаются вверх на одну ячейку;
- стереть символ из верхней ячейки магазина и записать в магазин строку символов, при этом содержимое магазина сдвигается вниз на длину записываемой строки.

Рассмотрим интерпретацию функции δ для автомата. Эта функция представляется совокупностью команд следующего вида:

$$\delta(q, a, Z) = \{(q_1, \alpha_1), (q_2, \alpha_2), \ldots, (q_n, \alpha_n)\},$$

где $q, q_1, q_2, \ldots, q_n \in Q$, $a \in \Sigma$, $Z \in T$, $\alpha \in T^*$.

При этом считается, что если на входе читающей головки автомата находится символ a, автомат находится в состоянии q, а верхний символ рабочей ленты Z, то автомат может перейти к состоянию q_i, записав при этом на рабочую ленту строку α_i $(1 < i < n)$ вместо символа Z, передвинуть входную головку на один символ вправо. Крайний левый символ α_i должен при этом оказаться в верхней ячейке магазина. Команда

$$\delta(q, \varepsilon, Z) = \{(q_1, \alpha_1), (q_2, \alpha_2), \ldots, (q_n, \alpha_n)\}$$

означает, что независимо от входного символа и, не передвигая входной головки, автомат

перейдет в состояние q_i, заменив символ Z магазина на строку α_i $(1 < i < n)$.

Конфигурацией МПА называется тройка $(q, \omega, \alpha) \in Q \times \Sigma^* \times T^*$,

где $q \in Q$ – текущее состояние устройства управления;

$\omega \in \Sigma^*$ – неиспользованная часть входной строки, первый символ которой находится под входной головкой; если $\omega = \varepsilon$, то считается, что вся входная лента прочитана;

α – содержимое магазина; самый левый символ строки α считается верхним символом магазина, если $\alpha = \varepsilon$, то магазин считается пустым.

Такт работы автомата будем представлять в виде бинарного отношения $\vdash$, определенного на конфигурациях. Например, если $\delta(q_1, a, Z)$ содержит (q_2, γ), то, выполнив такт, автомат перейдет от конфигурации $(q_1, a\omega, Z\alpha)$ к конфигурации $(q_2, \omega, \gamma\alpha)$, здесь $q_1, q_2 \in Q$, $\omega \in \Sigma^*$, $a \in \Sigma$, $Z \in T$, $\alpha, \gamma \in T^*$. Если a не пустой символ $(a \neq \varepsilon)$, то запись

$$(q_1, a\omega, Z\alpha) \vdash (q_2, \omega, \gamma\alpha)$$

говорит о том, что автомат, находясь в состоянии q_1 и имея a в качестве текущего входного символа, расположенного под входной головкой, а Z – в качестве верхнего символа магазина, может перейти в новое состояние q_2, сдвинуть входную головку на одну ячейку вправо и заменить верхний символ магазина строкой γ магазинных символов. Если $\gamma = \varepsilon$, то верхний символ удаляется из магазина, и тем самым магазинный список сокращается.

Если $a = \varepsilon$, будем называть этот такт ε-тактом. В ε-такте текущий входной символ не принимается во внимание и входная головка не сдвигается. Однако состояние управляющего устройства и содержимое памяти могут измениться. Заметим, что ε-такт может происходить и тогда, когда вся входная строка прочитана. Если же магазин пуст, то следующий такт невозможен.

Начальной конфигурацией МПА называется конфигурация вида (q_0, ω, Z_0), где $\omega \in \Sigma^*$, т. е. управляющее устройство находится в начальном состоянии, входная лента содержит строку, которую нужно распознать, а в магазине есть только начальный символ Z_0. Заключительная конфигурация – это конфигурация вида (q, ε, α), где $q_2 \in Q$, $\alpha \in T^*$.

Входная строка ω допускается МПА, если найдется последовательность тактов, переводящая автомат из начальной конфигурации (q_0, ω, Z_0) в заключительную $(q_f, \varepsilon, \alpha)$, здесь q_f – некоторое заключительное состояние $q_f \in F$, $\alpha \in T^*$, ε – пустая строка, или по-другому:

$$(q_0, \omega, Z_0) \vdash^* (q_f, \varepsilon, \alpha).$$

Языком, определяемым (или допускаемым) автоматом P (обозначается $L(P)$), называют множество строк, допускаемых автоматом P.

Теперь мы слегка расширим определение МПА, позволив ему заменять за один такт строку символов ограниченной длины, расположенную в верхней части магазина, другой строкой конечной длины. Напомним, что МПА в первоначальной версии мог на данном такте заменять лишь один верхний символ магазина.

Расширенным МПА назовем семерку

$$P = (Q, \Sigma, T, \delta, q_0, Z_0, F),$$

где δ – отображение конечного подмножества множества $Q \times (\Sigma \cup \{\varepsilon\}) \times T^*$ во множество конечных подмножеств множества $Q \times T^*$, а все другие символы имеют тот же смысл, что и раньше.

Конфигурация определяется так же, как прежде, и мы пишем

$$(q_1, a\omega, \alpha\gamma) \vdash (q_2, \omega, \beta\gamma)$$

если $\delta(q_1, a, \alpha)$ содержит (q_2, β), где $q_1, q_2 \in Q$, $\omega \in \Sigma^*$, $a \in \Sigma \cup \{\varepsilon\}$, $Z \in T$, $\alpha, \beta, \gamma \in T^*$. В этом такте строка α, расположенная в верхней части магазина, заменяется строкой β. Как и прежде, языком $L(P)$, определяемым автоматом P, называется множество

$$\{\omega \mid (q_0, \omega, Z_0) \vdash^* (q_f, \varepsilon, \alpha) \text{ для некоторых } q_f \in F, \alpha \in T^*\}$$

Заметим, что в отличие от обычного МПА расширенный МПА обладает способностью продолжать работу и тогда, когда магазин пуст.

Пусть $P = (Q, \Sigma, T, \delta, q_0, Z_0, F)$ – МПА или расширенный МПА. Будем говорить, что P допускает строку $\omega \in \Sigma^*$ опустошением магазина, если

$$(q_0, \omega, Z_0) \vdash^+ (q, \varepsilon, \varepsilon)$$

для некоторого $q \in Q$. Будем обозначать $L_e(P)$ – множество строк, допускаемых автоматом P опустошением магазина.

1.3. *Нормальная форма Грейбах*

Для каждого контекстно-свободного языка можно найти грамматику, в которой все правые части правил начинаются с терминалов**.** Контекстно-свободная грамматика находится в *нормальной форме Грейбах*, если в ней каждое правило продукций имеет вид:

$A \to a\alpha$, где $A \in N$, $\alpha \in N^*$, $a \in \Sigma$.

Рассмотрим грамматику в нормальной форме Грейбах, порождающую язык $L=\{0^n1^n \mid n>0\}$. Здесь множество $N = \{S, A\}$, множество $\Sigma = \{0, 1\}$, а множество продукций:

$S \to 0SA$

$S \to 0A$

$A \to 1$

Построение грамматики в нормальной форме Грейбах основано на устранении левой рекурсии. Алгоритм устранение левой рекурсии приведен ниже.

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

Выход. Эквивалентная КС-грамматика G^P без левой рекурсии.

Алгоритм.

Шаг 1. Пусть N = $\{A_1, A_2, \ldots, A_n\}$. Преобразуем G так, чтобы в правиле $A_i \to \alpha$ цепочка α начиналась либо с терминала, либо с такого A_j, что $j > i$. С этой целью положим $i = 1$.

Шаг 2. Пусть множество A_i-правил – это

$$A_i \to A_i\alpha_1|\ldots.|A_i\alpha_m|\beta_1|\ldots|\beta_p,$$

где ни одна из цепочек β_j не начинается с A_k, если k<i. (Это всегда можно сделать.) Заменим A_i-правила правилами

$$A_i \to \beta_1|\ldots|\beta_p|\beta_1A_i^1|\ldots|\beta_pA_i^1$$

$$A_i^1 \to \alpha_1|\ldots.|\alpha_m|\alpha_1A_i^1|\ldots.|\alpha_mA_i^1$$

A_i^1 – новый нетерминальный символ. Правые части всех A_i-правил начинаются теперь с терминала или с A_k для некоторого $k > i$.

Шаг 3. Если $i = n$, полученную грамматику G^P считать результатом и остановиться. В противном случае положить $i = i+1$ и $j = 1$.

Шаг 4. Заменить каждое правило вида $A_i \to A_j\alpha$ правилами

$$A_i \to \beta_1\alpha|\ldots|\beta_m\alpha,$$

где $A_i \to \beta_1|\ldots|\beta_m$ – все A_i- правила. Так как правая часть каждого A_i-правила начинается уже с терминала или с A_k для $k > j$, то и правая часть каждого A_i-правила будет теперь обладать этим свойством.

Шаг 5. Если $j = i$ -1, перейти к шагу 2. В противном случае положить $j = j$ +1 и перейти к шагу 4.

Алгоритм преобразования к нормальной форме Грейбах.

Вход. Не леворекурсивная приведенная КС-грамматика $G = (N, \Sigma, P, S)$.

Выход. Эквивалентная грамматика G' в нормальной форме Грейбах.

Алгоритм.

Шаг 1. Построить такой линейный порядок < на N, что каждое A-правило начинается либо с терминала, либо с такого нетерминала B, что $A < B$. Упорядочить $N = \{A_1, \ldots, A_n\}$ так, что $A_1, A_2 < \ldots < A_n$.

Шаг 2. Положить $i = n$—1.

Шаг 3. Если $i = 0$, перейти к шагу 5. В противном случае заменить каждое правило вида $A_i \to A_j\alpha$, где $j > i$, правилами

$$A_i \to \beta_1\alpha|\ldots|\beta_m\alpha,$$

где $A_j \to \beta_1|\ldots|\beta_m$ – все A_i- правила. Позже мы убедимся, что каждая из цепочек $\beta_1, \ldots, \beta_m$ начинается терминалом.

Шаг 4. Положить $i = i$ -1 и вернуться к шагу 3.

Шаг 5. Сейчас правая часть каждого правила (кроме, возможно, $S \to \varepsilon$) начинается терминалом. В каждом правиле $A \to aX_1 \ldots X_k$ заменить $X_j \in \Sigma$ новым нетерминалом X_j`.

Шаг 6. Для новых нетерминалов $X_j`$, введенных на шаге 5, добавить правила $X_j` \rightarrow X_j$.

1.4. *Эквивалентность языков, определяемых КС-грамматикой и автоматом с магазинной памятью*.

Доказано, что если $L(G2)$ – бесконтекстный язык, порождаемый КС-грамматикой $G2 = (N, \Sigma, P, S)$, находящейся в нормальной форме Грейбах, то существует недетерминированный магазинный автомат M такой, что $L_e(M) = L(G2)$. При этом

$$M = (Q, \Sigma, T, \delta, q_0, Z_0, \varnothing),$$

где алфавиты Σ в грамматике и автомате совпадают; $Q = \{q_0\}$, $Z_0 = S$, $T = N$, а δ определяется следующим образом:

(1) если $A \rightarrow a\alpha$ принадлежит множеству правил P грамматики $G2$, то $\delta(q_0, a, A)$ содержит (q_0, α);

(2) если $A \rightarrow a$ принадлежит множеству правил P грамматики $G2$, то $\delta(q_0, a, A) = \{(q_0, \varepsilon)\}$.

Грамматика в нормальной форме Грейбах, порождающая язык $L=\{0^n1^n | n>0\}$ задается следующим образом: множество $N = \{S, A\}$, множество $\Sigma = \{0, 1\}$, а множество продукций:

$$S \rightarrow 0SA$$
$$S \rightarrow 0A$$
$$A \rightarrow 1$$

Для этой грамматики построим соответствующий ей МПА $M = (Q, \Sigma, T, \delta, q_0, Z_0, \varnothing)$,

где алфавит $\Sigma=\{0, 1\}$, $Q = \{q_0\}$, $T = \{S, A\}$, $Z_0 = S$,

$$\delta(q_0, 0, S) = (q_0, SA)$$
$$\delta(q_0, 0, S) = (q_0, A)$$
$$\delta(q_0, 1, A) = (q_0, \varepsilon)$$

Пусть на ленте автомата записана строка 000111, автомат выполнит следующую последовательность тактов:

$$\begin{aligned}(q_0, 000111, S) &\vdash (q_0, 00111, SA)\\ &\vdash (q_0, 0111, SAA)\\ &\vdash (q_0, 111, AAA)\\ &\vdash (q_0, 11, AA)\\ &\vdash (q_0, 1, A)\\ &\vdash (q_0, \varepsilon, \varepsilon)\end{aligned}$$

2. **Варианты задания КС-языка**.

Дан КС-язык, описать его с помощью грамматики и автомата.

Варианты.

1) Язык $L=\{0^n1^n2 \mid n>0\}$.

2) Язык $L=\{0^n10^n \mid n>0\}$.

3) Язык $L=\{0^n1^m0^k \mid n>0, m>0, k>0\}$.

4) Язык, в котором допустимыми являются все цепочки в алфавите {0,1}, имеющие одинаковое число нулей и единиц.

5) Язык, в котором допустимыми являются все цепочки в алфавите {0,1}, имеющие число нулей в два раза больше единиц.

6) Язык, состоящий из множества всех цепочек нулей и единиц, содержащих четное число нулей и четное число единиц.

3. Задание

3.1. *Безкомпьютерная обработка*

- Приведите примеры строк, принадлежащих и не принадлежащих заданному языку.
- Для данного языка определите КС-грамматику, генерирующую строки заданного языка.
- Постройте выводы строк в заданной грамматике, каждый вывод представьте в виде дерева.
- Определенной выше КС-грамматике поставьте в соответствие автомат с магазинной памятью, допускающий строки заданного языка.
- Определите последовательность конфигураций автомата при распознавании строк, принадлежащих и не принадлежащих языку.

3.2. *Компьютерная обработка*

Напишите на Вашем любимом языке программирования программу, моделирующую работу автомата с магазинной памятью или КС-грамматики.

Мозговая атака

Этапы проведения.

1. Предложить *вопросы для обсуждения*:
   - как формализовать процесс построения КС-грамматики по заданному описанию?
   - как формализовать процесс построения автомата с магазинной памятью по заданному описанию?
2. Предложить высказать свои мысли по данным вопросам.
3. Записать все прозвучавшие высказывания без возражений, но, возможно, с уточнениями.
4. По окончанию прочитать все, что записано со слов участников.
5. Обсудить все варианты ответов, выбрать главные и второстепенные.
6. Выяснить, как можно использовать полученные результаты при выполнении данной лабораторной работы.

Отчет

Отчет должен содержать следующие обязательные пункты:

15. Автор (ы) проекта.
16. Примеры строк, принадлежащих и не принадлежащих языку.
17. Описание КС-грамматики.
18. Примеры деревьев выводов (не менее двух).
19. Описание автомата с магазинной памятью.
20. Примеры не менее четырех последовательностей конфигураций автомата при распознавании строк, принадлежащих и не принадлежащих языку.
21. Листинги программ моделирования работы конечного автомата (праволинейной грамматики).

Контрольные вопросы

1. Какими средствами может быть задан КС-язык?
2. Как строится дерево вывода?
3. Как может быть получена нормальная форма Хомского?
4. Основное назначение нормальной формы Грейбах.
5. Что такое праворекурсивная грамматика, в чем ее преимущество?
6. Определите структуру автомата с магазинной памятью и функции его элементов.
7. Дайте толкования понятиям «конфигурация автомата с магазинной памятью» и «такт автомата с магазинной памятью».
8. Укажите отличия между детерминированным и недетерминированным автоматом с магазинной памятью.
9. Определите язык, допускаемый автоматом с магазинной памятью.

Лабораторная работа №4.
Формализмы перевода.

Цель работы.

Знакомство с методами определения перевода; задание перевода с помощью конечного преобразователя; задание перевода с помощью преобразователя с магазинной памятью.

Порядок выполнения работы.

1. **Знакомство с методами определения перевода.**

Перевод (или трансляция) – это некоторое отношение между цепочками, или, другими словами, перевод есть некоторое множество пар цепочек. Существует два фундаментальных метода определения перевода. Один из них – «схема перевода», которая представляет собой грамматику, снабженную механизмом, обеспечивающим выход для каждой порождаемой цепочки. В другом методе основную роль играет «преобразователь», т. е. распознаватель с выходом, который на каждом такте может выдавать цепочку выходных символов ограниченной длины.

1.1. *Схема синтаксически управляемого перевода*

Одним из формализмов, используемых для определения переводов, является схема синтаксически управляемого перевода (трансляции). Интуитивно такая схема представляет собой просто грамматику, в которой к каждому правилу присоединяется элемент перевода. Всякий раз, когда правило участвует в выводе входной цепочки, с помощью элемента перевода вычисляется часть выходной цепочки, соответствующая части входной цепочки, порожденной этим правилом.

Схемой синтаксически управляемого перевода (или *трансляции)* (сокращенно СУ-схемой) называется пятерка

$$T = (N, \Sigma, \Delta, R, S),$$

где N – конечное множество *нетерминальных символов,*

Σ – конечный *входной алфавит,*

Δ – конечный *выходной алфавит,*

R – конечное множество правил вида

$$A \rightarrow \alpha, \beta,$$

где $\alpha \in (N \cup \Sigma)^*$, $\beta \in (N \cup \Delta)^*$,

S – начальный символ, $S \in N$.

1.2. *Конечные преобразователи*

Конечный преобразователь получится, если конечному автомату позволить выдавать на каждом такте цепочку выходных символов. *Конечным преобразователем* называется шестерка

$$M = (Q, \Sigma, \Delta, \delta, q_0, F)$$

где Q — конечное множество *состояний,*

Σ — конечный *входной алфавит,*

Δ — конечный *выходной алфавит,*

δ — отображение множества $Q \times (\Sigma \cup \{e\})$ в множество конечных подмножеств множества $Q \times \Delta^*$,

$q_0 \in Q$ — *начальное состояние,*

$F \subseteq Q$ — множество *заключительных состояний.*

Определим *конфигурацию* преобразователя M как тройку (q, x, y), где $q \in Q$ —текущее состояние управляющего устройства, $x \in \Sigma^*$ — оставшаяся непрочитанной часть входной цепочки, причем самый левый символ цепочки x расположен под входной головкой, $y \in \Delta^*$ — часть выходной цепочки, выданная вплоть до текущего момента.

Определим бинарное отношение $\vdash_M$ или $\vdash$ на конфигурациях, соответствующее одному такту работы преобразователя *M:* для всех $q \in Q$, $a \in (\Sigma \cup \{e\})$, $x \in \Sigma^*$ и $y \in \Delta^*$, таких, что $\delta(q, a)$ содержит (r, z), будем писать

$$(q, ax, y) \vdash (r, x, yz)$$

Цепочку y назовем *выходом* для цепочки x, если $(q_0, x, e) \vdash^* (q, e, y)$ для некоторого $q \in F$.

Переводом, определяемым преобразователем M (обозначается $\tau(M)$), назовем множество

$$\{(x, y) \mid (q_0, x, e) \vdash^* (q, e, y)$$

для некоторого $q \in F\}$. Перевод, определяемый конечным преобразователем, будем называть *регулярным переводом* или *конечным преобразованием.*

Заметим, что для того, чтобы выходную цепочку y можно было считать переводом входной цепочки x, цепочка x должна перевести преобразователь M из начального состояния в заключительное.

1.3. *Преобразователь с магазинной памятью*

Преобразователи с магазинной памятью получаются из автоматов с магазинной памятью, если их снабдить выходом и разрешить на каждом такте выдавать выходную цепочку конечной длины.

Преобразователем с магазинной памятью (МП-преобразователем) называется восьмерка

$$P = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, Z_0, F),$$

где $Q = \{q_0, q_2, \ldots, q_n\}$ – конечное множество состояний устройства управления (УУ);

$\Sigma = \{a_1, a_2, \ldots, a_k\}$ – конечный входной алфавит;

Γ – конечный алфавит магазинных символов;

q_0 – начальное состояние устройства управления, $q_0 \in Q$;

Z_0 – начальный символ магазина, $Z_0 \in \Gamma$;

F – множество заключительных состояний, $F \subseteq Q$.

Δ — конечный *выходной алфавит,*

δ — отображение множества $Q\times(\Sigma\cup\{e\})\times\Gamma$ в множество конечных подмножеств множества $Q\times\Gamma^*\times\Delta^*$.

Определим *конфигурацию* преобразователя P как четверку (q, x, α, y),

где $q\in Q$ — текущее состояние устройства управления;

$x\in\Sigma^*$ — неиспользованная часть входной строки, первый символ которой находится под входной головкой; если $x = \varepsilon$, то считается, что вся входная лента прочитана;

α – содержимое магазина; самый левый символ строки α считается верхним символом магазина, если $\alpha = \varepsilon$, то магазин считается пустым.

y — выходная цепочка, выданная вплоть до настоящего момента.

Если $\delta(q, a, Z)$ содержит (r, α, z), то будем писать

$$(q, ax, Z\gamma, y) \vdash (r, x, \alpha\gamma, yz)$$

для любых $x\in\Sigma^*$, $\gamma\in\Gamma^*$ и $y\in\Delta^*$.

Цепочку y назовем *выходом* для x, если $(q_0, x, Z_0, e) \vdash^*(q, e, \alpha, y)$ для некоторых $q\in Q$ и $\alpha\in\Gamma^*$.

Переводом (или *преобразованием*), *определяемым МП-преобразователем* P (обозначается $\tau(P)$), назовем множество

$\{(x, y) \mid (q_0, x, Z_0, e) \vdash^*(q, e, \alpha, y)$ для некоторых $q\in F$ и $\alpha\in\Gamma^*\}$

Аналогично МП-автоматам можно говорить о преобразовании входа x в выход y *опустошением магазина,* если $(q_0, x, Z_0, e) \vdash^*(q, e, e, y)$ для некоторого $q\in Q$.

Переводом, определяемым преобразователем P *опустошением магазина* (обозначается $\tau_e(P)$), назовем множество

$\{(x, y) \mid (q_0, x, Z_0, e) \vdash^*(q, e, e, y)$ для некоторых $q\in Q\}$

Аналогично расширенным МП-автоматам можно определить расширенные МП-преобразователи (у них верх магазина расположен справа).

Рассмотрим МП-преобразователь

$$P = (\{q\},\quad \{a, +, *\},\quad \{+, *, E\},\quad \{a, +, *\},\ \delta,\ q,\ E,\ \{q\})$$

где δ определяется равенствами

$$\delta(q, a, E) = \{(q, e, a)\}$$
$$\delta(q, +, E) = \{(q, EE+, e)\}$$
$$\delta(q, *, E) = \{(q, EE*, e)\}$$
$$\delta(q, e, +) = \{(q, e, +)\}$$
$$\delta(q, e, *) = \{(q, e, *)\}$$

Для входа $+*aaa$ МП-преобразователь P сделает такую последовательность тактов:

$$\begin{aligned}(q, +*aaa, E, e) &\vdash (q, *aaa, EE+, e)\\ &\vdash (q, aaa, EE*E+, e)\\ &\vdash (q, aa, E*E+, a)\\ &\vdash (q, a, *E+, aa)\\ &\vdash (q, a, E+, aa*)\\ &\vdash (q, e, +, aa*a)\\ &\vdash (q, e, e, aa*a+)\end{aligned}$$

Таким образом, P переводит цепочку $+*aaa$ в цепочку $aa*a+$, опустошая магазин. Можно проверить, что

$\tau_e(P) = \{(x, y) |\ x$ — префиксное польское арифметическое выражение в алфавите $\{+, *, a)$ и y — соответствующая постфиксная польская запись$\}$.

2. **Варианты задания перевода**.

Дано вербальное описание перевода, представить его с помощью преобразователя.

Варианты.

1) Отобразить обычную инфиксную запись арифметических выражений в префиксную польскую запись.

2) Отобразить обычную инфиксную запись арифметических выражений в постфиксную польскую запись.

3) Отобразить префиксную польскую запись арифметических выражений в обычную инфиксную запись.

4) Отобразить постфиксную польскую запись арифметических выражений в обычную инфиксную запись.

5) Отобразить арифметическое выражение с избыточными скобками в арифметическое выражение без избыточных скобок.

3. Задание

3.1. *Безкомпьютерная обработка*

- Приведите примеры строк, принадлежащих и не принадлежащих заданному переводу.
- Для данного языка определите СУ-схему, генерирующую строки заданного перевода.
- Постройте выводы строк в заданной схеме.
- Определенной выше СУ-схеме поставьте в соответствие преобразователь, допускающий пары строк заданного перевода.
- Определите последовательность конфигураций преобразователя при переводе, принадлежащих и не принадлежащих переводу.

3.2. *Компьютерная обработка*

Напишите на Вашем любимом языке программирования программу, моделирующую работу преобразователя, решающего задачу перевода.

Мозговая атака

Этапы проведения.

1. Предложить *вопросы для обсуждения*:
   - как формализовать процесс построения схемы синтаксически управляемого перевода по заданному описанию?
   - как формализовать процесс построения преобразователя с магазинной памятью по заданному описанию?
2. Предложить высказать свои мысли по данным вопросам.
3. Записать все прозвучавшие высказывания без возражений, но, возможно, с уточнениями.
4. По окончанию прочитать все, что записано со слов участников.
5. Обсудить все варианты ответов, выбрать главные и второстепенные.
6. Выяснить, как можно использовать полученные результаты при выполнении данной лабораторной работы.

Отчет

Отчет должен содержать следующие обязательные пункты:

22. Автор (ы) проекта.

23. Примеры строк, принадлежащих и не принадлежащих переводу.
24. Описание СУ-схемы.
25. Описание преобразователя.
26. Примеры не менее четырех последовательностей конфигураций преобразователя при распознавании строк.
27. Листинги программ моделирования работы преобразователя.

Контрольные вопросы

1. Какими средствами может быть задан перевод?
2. Как задается схема синтаксически управляемого перевода?
3. Определите структуру конечного преобразователя и функции его элементов.
4. Дайте толкования понятиям «конфигурация конечного преобразователя» и «такт конечного преобразователя».
5. Укажите отличия между детерминированным и недетерминированным конечным преобразователем.
6. Определите перевод, допускаемый конечным преобразователем.
7. Определите структуру преобразователя с магазинной памятью и функции его элементов.
8. Дайте толкования понятиям «конфигурация преобразователя с магазинной памятью» и «такт преобразователя с магазинной памятью».
9. Укажите отличия между детерминированным и недетерминированным преобразователем с магазинной памятью.
9. Определите перевод, допускаемый преобразователем с магазинной памятью.

Список рекомендуемой литературы

1. Мозговой М.В. Классика программирования: алгоритмы, языки, автоматы, компиляторы. Практический подход. - СПб. : Наука и техника, 2006. - 320 с.
2. Молчанов А.Ю. Системное программное обеспечение: Учебник для вузов. – СПб.: Питер, 2006. - 395 с.
3. Ахо А., Ульман Дж. Компиляторы: принципы, технологии, инструменты. – М.: Вильямс, 2001. – 767 с.