Теория автоматов и формальных языков

A1 x A2 x … x An = {(a1, a2, …, an) | a1∊ A1, …, an∊ An }

A x A x … x A = A^n

Функцией, отображающей множество x в множество y

F: X -> Y

F ⊆ X x Y

X — область отправления

Y — область прибытия

{x | (x, y) ∊ F} — область определения

{y | (x, y) ∊ F} — область значения

Если X = A^n, то Y = A

Такая функция — n-местная функция или функция n аргументов

Если область определения равна области отправления, то функция называется полностью определённой, иначе — частично

Если Y={0, 1}, то функция называется предикатом

Алфавит — конечное непустое множество символов

Примеры алфавитов:

V1 = {a, b}

V2 = {0, 1}

V3 = {a, 1, +, =}

Если мы запишем последовательность символов из алфавита, то получим слово. Количество символов в слове — длина слова.

V\* — все слова алфавита

n-местная функция над алфавитом V

(V\*)^n -> V\*

Словарной n-местной функцией можно назвать любую функцию.

Функция называется вычисляемой, если существует алгоритм нахождения её значения по значению аргументов. Для любой вычисляемой функции существует автомат, который вычисляет значение функции по значению аргумента. Этот автомат — машина Тьюринга.

Формальный язык — это множество цепочек. Цепочки всегда конечны. Множество, представляющее язык, может быть как конечным, так и бесконечным. Алфавит всегда конечный.

Цепочка может оканчиваться пустым маркером, который не содержится в алфавите -|. Пустая цепочка называется эпсилон.

Задача распознания заключается в определении, принадлежит ли заданная цепочка к распознаваемому языку.

Для любого множества можно определить характеристическую функцию:

F\_M : V\*-> {0, 1}

F\_M(alpha) = {

1, alpha ∊ M

0, alpha not ∊ M

Для решения характеристической функции используются конечные автоматы

Основное применение теории — разработка трансляторов, проектирование цифровых устройств.

Рекомендуемая литература

Ахо Ольман 2 тома (412 страниц, 685 страниц) 1978 г. Издание в России, написан в 1972

Ахо, Сети, Ульман «Компиляторы. Принципы, технологии и инструменты». Издана в 2003 году. Написана в 1972 году.

Льюис, Разенкранс, Стивенс «Теоретические основы проектирования компиляторов» (отсюда берём обозначения)

Хокрофт, Маквани, Ульман «Введение в теорию автоматов, языков и вычислений».

Бигмулина «Теория формальных грамматик и автоматов».

Если язык представляет собой конечное множество, то его можно задать перечислением.

V = {a, b, c}

L1 = {a, ab, ac}

Цепочки, начинающиеся последовательностью символов a, за которой следует последовательность символов b.

Задание характеристического свойства

L2 = {a^n b^m | n > 0, m > 0}

Задание языка синтаксической диаграммой

<сделать рисунок>

Задание языка формулами Бэкуса-Найера (БНФ)

<Число> ::= <Цифра> <Число> | <Цифра>

<Цифра> ::= 0|1|2|3|4|5|6|7|8|9

Расширенная БНФ отличается от БНФ введением дополнительных обозначений, которые упрощают запись.

<Число>::= {<Цифра>}^+

<Цифра> ::= 0—9

Задание языка автоматом, который распознаёт язык.

Задание языка формальной грамматикой (основной способ, который будем использовать).

Операции над языками

Язык — это множество, поэтому все операции, применимые к множествам, применимы и к формальным языкам

Включение.

Равенство. Истина, если каждая цепочка L1 принадлежит L2 и наоборот

Включение

Строгое включение

Объединение

Пересечение

Разность. Те цепочки L1, которых нет в языке L2

Конкатенация. L1L2={alpha beta alpha ∊ L1 и beta ∊ L2}

Итерация. L\*. Результаты: L^0 = {epsilon}; L^n = LL^(n-1), n>0; L\* = UL^n, n>= 0

U = V\*

Дополнение языка  ~L = U — L

L^+ = UL^n, n>0

Обращение языка. В результате получаем язык, в котором цепочки записаны в обратном порядке.

Формальная грамматика

G = (N, T, P, S)

N — конечное множество нетерминальных символов или нетерминалов

T — конечное множество терминальных символов или терминалов

Терминалы — символы алфавита языка, на которых описывается грамматика

P (правила) — множество alpha -> beta (включают терминалы и нетерминалы)

S — начальный нетерминал, который принадлежит множеству всех нетерминалов

L(G)

Цепочка, выводимая в грамматике по правилу:

1. S — выводимая цепочка

2. Если цепочка alpha beta gamma выводимая и beta -> delta ∊ P alpha delta gamma.

Alpha beta gamma => alpha delta gamma

S => α1 => α2 => … => αn

Вывод — последовательность цепочек, которая удовлетворяет следующим условиям:

Эта последовательность заканчивается терминальной цепочкой

Каждая цепочка, кроме первой, непосредственно выводима из предыдущей, и эти цепочки разделяются знаком =>

S\* => αn

-> означает представляет собой правую часть.

Классы и типы грамматик

Обозначаются числами 0, 1, 2 ,3

0 — грамматики общего вида без ограничений

1 — контекстно зависимые грамматики. Правила имеют вид α A β -> α δ β.

Неукорачивающие грамматики.

2 —

Лекция 3 (от 20.09.25)

Преобразование контекстно-свободных грамматик

Конкретный КС-язык может быть описан различными КС-грамматиками

2 грамматики, которые определяют один и тот же язык, называются эквивалентными. Неоднозначного понимания языка нет, в отличие от естественных языков.

Получение из одной грамматики эквивалентной другой называется преобразованием. Такое преобразование выполняется путём применения правил преобразования.

Множество правил преобразования образуют систему преобразования. Система называется полной, если для любой пары эквивалентных грамматик существует последовательность правил преобразований, которая из одной грамматики получает другую грамматику.

Для КС-свободных грамматик не существует полной системы преобразований. Рассмотри правила преобразований:

1. Устранение лишних символов. Лишние символы:

1. Бесплодные нетерминалы — это нетерминалы, из которых нельзя вывести ни одну терминальную цепочку.

2. Недостижимые символы — это символы (терминалы, нетерминалы), которые не встречаются ни в одной цепочке (терминальной или промежуточной) выводимой из начального нетерминала.

Поиск бесплодных нетерминалов сводится к нахождению продуктивных терминалов.

Для нахождения продуктивных нетерминалов используется следующее правило:

A -> alpha

Если из каждого символа, который есть в альфа, можно вывести терминальную цепочку, то и из A можно вывести терминальную цепочку

Алгоритм

Формируется множество продуктивных терминалов P (пустое)…

Алгоритм поиска недостижимых терминалов аналогичен.

Для нахождения достижимых символов

Если нетерминал достижим, то все его символы из правой части достижимы.

Пример:

1. S->ac

2. S->bA

3. A->cBC

4. B->aSA

5. C->bC

6. C->d

P = {S, A, B, C}

P = {S, C} бесплодные = {A, B}

Убираем правила 2, 3, 4

1. S->ac

2. C->bC

3. C->d

С — недостижим

Сначала необходимо искать бесплодные, и только затем удалять недостижимые

2. Исключение лишних правил. Применяем к грамматике, в которой нет лишних символов

Правило А представляет собой альфа называется лишним, если из нетерминала А можно вывести цепочку альфа, не применяя этого правила. Лишнее правило нужно исключать из грамматики.

1. A->aABa

2. A->bC

3. A->bba

4. B->dBC

5. B->epsilon

6. C->BA

7. C->a

A2=>bC6=>bB5A=>bA2=>bbC7=>bba (3е правило лишнее)

C6=>B5A=>A

1. A->AB

2. A->B

3. A->epsilon

4. B->epsilon

A1=>A3B=>B (1е лишнее)

A2=>B4=>epsilon (3е лишнее)

Но нельзя удалить сразу два, необходимо удалить одно и только затем искать лишние цепочки.

Для разных последовательностей удаления лишних правил будем получать разный результат.

3. Исключение epsilon правил.

Правило вида А представляет собой эпсилон называют эпсилон правилом

Любую грамматику, которая определяет язык, который не содержит пустой цепочки, можно преобразовать в грамматику без эпсилон правил. А если язык содержит пустую цепочку, то грамматику можно преобразовать в грамматику, которая содержит эпсилон для начального терминала

Есть два метода

1. Аннулирующий нетерминал — это нетерминал, из которого можно вывести пустую цепочку.

Найти аннулирующий нетерминал можно следующим способом: исключим все правила, в которых есть хотя бы один терминал

Алгоритм:

1. Находим множество аннулирующих нетерминалов

2. Последовательно просматриваем правила грамматики, и если правила в правой части содержат к вхождений аннулирующих нетерминалов, то это правило заменяется на 2 ^k других правил. Они получаются исключением всеми возможными способами аннулирующих нетерминалов.

3. В результате получения новых правил могут появиться эпсилон правила, правила А представляет собой А. Такие правила бессмысленны и потому удаляются. Могут появиться одинаковые правила. Все эпсилон правила удаляем, и если грамматика порождает пустую цепочку, то добавляем эпсилон-правила для начального нетерминала.

1. S->AaB

2. S->aB

3. S->cC

4. A->AB

5. A->B

6. A->b

7. B->Ba

8. B->epsilon

9. C->AB

10. C->c

Первым действием находим аннулирующие нетерминалы

4, 5, 8, 9

Затем продуктивные

P = {B, A, C}

1

S->AaB

S->aB

S->Aa

S->a

2

S->aB

S->a

3

S->cC

S->c

4

A->AB

A->B

A->A

A->epsilon

5

A->B

A->epsilon

6

A->b

7

B->Ba

B->a

8

B->epsilon

9

C->AB

C->B

C->A

C->epsilon

10

C->c

Оставляем

1

S->AaB

S->aB

S->Aa

S->a

2

3

S->cC

S->c

4

A->AB

A->B

5

A->B

A->epsilon

6

A->b

7

B->Ba

B->a

8

B->epsilon

9

C->AB

C->B

C->A

C->epsilon

10

C->c

Способ 2

В грамматике выбираем одно эпсилон правило, исключаем его, считаем, что в грамматике только один регулирующий нетерминал из левой части этого эпсилон правила и выполняем предыдущий алгоритм.

1

1. S->AaB

2. S->Aa

2

3. S->aB

4. S->a

3

5. S->cC

4

6. A->AB

A->A

5

7. A->B

A->epsilon

6

8. A->B

7

9. B->Ba

10. B->a

8

9

11. C->AB

12. C->A

10

13. C->c

1

1. S->AaB

2. S->aB

2

3. S->Aa

4. S->a

3

S->aB

4

S->a

5

5. S->cC

6

6. A->AB

7. A->B

7

A->B

8

8. A->b

9

9. B->Ba

10

10. B->a

11

11. C->AB

12. C->B

12

13. C->A

C->epsilon

13

14. C->c

1. S->AaB

2. S->aB

3. S->Aa

4. S->a

5. S->cC S->c

6. A->AB

7. A->B

8. A->b

9. B->Ba

10. B->a

11. C->AB

12. C->B

13. C->A

14. C->C