**Лекция 6.** Вербальные алгоритмы. Нормальные алгорифмы Маркова.

**ЦЕЛЬ**: Ознакомить с понятиями: Вербальные алгоритмы. Нормальные алгорифмы Маркова, принцип нормализации и его обоснование, самоприменимость и переводимость. Ассоциативные исчисления в алфавите.

**ВОПРОСЫ**:

1. Вербальные алгоритмы.
2. Нормальные алгорифмы Маркова, принцип нормализации и его обоснование, самоприменимость и переводимость.
3. Ассоциативные исчисления в алфавите.

**ОСНОВНЫЕ ПОНЯТИЯ:** Вербальные алгоритмы. Нормальные алгорифмы Маркова, принцип нормализации и его обоснование, самоприменимость и переводимость. Ассоциативные исчисления в алфавите.

**Нормальный алгоритм Маркова**— один из стандартизованных вариантов представления об алгорифме (алгоритме). Понятие нормального алгоритма введено А. Марковым в конце 1940-х годов, по отношению к алгоритмам Маркова принято использовать авторский вариант (термин алгорифм). Алгорифмы Маркова составляют теоретическую основу системы программирования, использующую язык РЕФАЛ. Нормальные алгорифмы оказались удобным средством для построения многих разделов конструктивной математики.

Информация*,* обрабатываемая алгорифмом Маркова, представляется словом в некотором фиксированном алфавите *A*. ***Алгорифм* (*программа)*** представляется последовательностью пар слов α *i* → β*i* в алфавите *A*. Пары, составляющие алгорифм, называются также подстановками и записываются в виде α → β , где α, β – слова в алфавите *A*, причем β может быть пустым (обозначаем λ). Некоторые подстановки помечаются восклицательным знаком и называются заключительными. В паре слов подстановки левое (первое) слово непустое, а правое (второе) слово может быть пустым символом. Например,

1. ab\to bd
2. db\to ba
3. bba\to abb
4. c\to \lambda

Слова будем обозначать латинскими буквами P, Q, R. Одно слово может быть составной частью другого слова. Тогда первое слово называется подсловом второго или вхождением во второе.

Определение 1. Будем говорить, что имеется вхождение слова P в слово R, если существуют слова V1 и V2 (возможно, пустые), такие, что R = V1 P V2 (1).

Если слово V1 имеет наименьшую дину из всех слов вида (1), то говорят о первом вхождении P в слово R.

***Функционирвание.*** Во входном слове ищется фрагмент, совпадающий с левой частью первой подстановки. Если такой фрагмент находится, то он во входном слове заменяется на ее правую часть, в противном случае рассматривается вторая подстановка из алгорифма и так далее. Вычисления заканчиваются когда ни одна из левых частей подстановок не является фрагментом обработанного к данному моменту слова или когда выполнена заключительная подстановка. Процесс может оказаться и бесконечным.

Пример 1: в алфавите А = {a, b} рассмотрим следующие слова:

1. P1 = aa входит в слово R = abaabab, так как R = V1P1V2, где V1 = ab, V2 = bab. В этом случае V1 и V2 определены однозначно.

2. P2 = aba входит в слово R = abaabab, так как R = V1 P V2, где V1 = λ, V2 = abab; R = W1 P2W2, где W1 = aba, W2 = b. В этом случае V1 и V2 определены неоднозначно.

Замечание: в дальнейшем будем разыскивать первые вхождения данных слов в некоторые слова и заменять их на другие слова, возможно, пустые.

Пример 2: в результате замены первого вхождения слова P = ba в слове R = aababab на слово: 1) Q = a; 2) Q = b; 3) Q = baa; 4) на пустое слово, получим следующие слова (табл. 1).

Таблица 1.

|  |  |  |
| --- | --- | --- |
|  | P = ba | результат |
| Q = a | aababab | aaabab |
| Q = b | aababab | aabbab |
| Q = baa | aababab | aabaabab |
| λ | aababab | aabab |

Если же вхождение пустого слова в любое слово R заменить на некоторое слово Q, получится слово QR.

Нормальные алгоритмы являются **вербальными**, то есть предназначенными для применения к словам в различных алфавитах. Определение всякого нормального алгоритма состоит из двух частей: **определения алфавита алгоритма (к словам в котором алгоритм будет применяться) и определения его схемы.** **Схемой нормального алгоритма** называется конечный упорядоченный набор т. н. формул подстановки, каждая из которых может быть простой или заключительной.

Простыми формулами подстановки называются слова вида L→D, где *L* и *D* — два произвольных слова в алфавите алгоритма (называемые, соответственно, левой и правой частями формулы подстановки). Аналогично, заключительными формулами подстановки называются слова вида L→**⋅**D, где *L* и *D* — два произвольных слова в алфавите алгоритма. При этом предполагается, что вспомогательные буквы \toи \to\cdotне принадлежат алфавиту алгоритма (в противном случае на исполняемую ими роль разделителя левой и правой частей следует избрать другие две буквы).

Процесс применения нормального алгоритма к произвольному слову *V* в алфавите этого алгорифма представляет собой дискретную последовательность элементарных шагов, состоящих в следующем. Пусть *V*' — слово, полученное на предыдущем шаге работы алгорифма (или исходное слово *V*, если текущий шаг является первым). Если среди формул подстановки нет такой, левая часть которой входила бы в *V*', то работа алгоритма считается завершённой, и результатом этой работы считается слово *V*'. Иначе среди формул подстановки, левая часть которых входит в *V*', выбирается самая верхняя.

Если эта формула подстановки имеет вид L→D, то из всех возможных представлений слова *V*' в виде *RLS* выбирается такое, при котором *R* — самое короткое, после чего работа алгоритма считается завершённой с результатом *RDS*. Если же эта формула подстановки имеет вид L→D, , то из всех возможных представлений слова *V*' в виде *RLS* выбирается такое, при котором *R* — самое короткое, после чего слово *RDS* считается результатом текущего шага, подлежащим дальнейшей переработке на следующем шаге.

Пример 3. Пусть в алфавите A2={a,b,c} задана система ориентированных подстановок:

1. cb → cc

2. cca → ab

3. ab → bca

4. ca → λ

Этот алгоритм, будучи применен к слову cabc, никогда не обрывается:

– получилось первоначальное слово, т.е. алгоритм закончился.

Если же брать слово baaacca, то после нескольких шагов процесс оборвется на слове bb:

Два нормальных алгоритма отличаются друг от друга алфавитом и системой ориентированных подстановок или даже только порядком подстановок

**Принцип нормализации Маркова**

Принцип Маркова: всякая эффективно вычислимая функция является нормально вычислимой или всякий алгоритм представим в виде нормального алгоритма.

Математически доказать принцип нормализации невозможно, поскольку понятие произвольного алгоритма или эффективно вычислимой функции не являются строго определенными математическими понятиями. Справедливость этого принципа основана на том, что все известные в настоящее время алгоритмы являются нормализуемыми, а способы композиции алгоритмов, позволяющие строить новые алгоритмы из уже известных, не выводят за пределы класса нормализуемых алгоритмов.

Теорема. **Любой нормальный алгорифм эквивалентен некоторой машине Тьюринга**, и наоборот — любая машина Тьюринга эквивалентна некоторому нормальному алгорифму.

Вариант тезиса Чёрча — Тьюринга, сформулированный применительно к нормальным алгорифмам, принято называть «**принципом нормализации**».

**Для формализации понятия алгоритма** российский математик ***А.А.Марков*** предложил использовать ***ассоциативные исчисления***.    (описания основных понятий ассоциативного исчисления см. литературу). Предложенный А.А.Марковым способ уточнения понятия алгоритма основан на понятии нормального алгоритма, который определяется следующим образом. Пусть задан алфавит **A** и система подстановок **B**. Для произвольного слова**P** подстановки из **B** подбираются в том же порядке, в каком они следуют в **B**. Если подходящей подстановки нет, то процесс останавливается. В противном случае берется первая из подходящих подстановок и производится замена ее правой частью первого вхождения ее левой части в **P**. Затем все действия повторяются для получившегося слова **P1**. Если применяется последняя подстановка из системы **B**, процесс останавливается. Совокупность всех слов в данном алфавите вместе с системой допустимых подстановок называют ***ассоциативным исчислением***. Чтобы задать ассоциативное исчисление, достаточно задать алфавит и систему подстановок.

    Слова **P1** и **Р2** в некотором ассоциативном исчислении называются ***смежными***, если одно из них может быть преобразовано в другое однократным применением допустимой подстановки. Последовательность, слов **P, P1, ...,M** называется ***дедуктивной цепочкой***, ведущей от слова **P** к слову **М**, если каждое из двух рядом стоящих слов этой цепочки - смежное.    Слова **P** и **М** называют ***эквивалентными***, если существует цепочка от **P** к **М** и обратно.    Может быть рассмотрен специальный вид ассоциативного исчисления, в котором подстановки являются ориентированными: **N->М** (стрелка означает, что подстановку разрешается производить лишь слева направо). Для каждого ассоциативного исчисления существует задача: для любых двух слов определить, являются ли они эквивалентными или нет. Любой процесс вывода формул, математические выкладки и преобразования также являются дедуктивными цепочками в некотором ассоциативном исчислении. Построение ассоциативных исчислений является универсальным методом детерминированной переработки информации и позволяет формализовать понятие алгоритма.

Введем понятие алгоритма на основе ассоциативного исчисления.    ***Алгоритмом в алфавите*** **A** называется понятное точное предписание, определяющее процесс над словами из **A** и допускающее любое слово в качестве исходного. Алгоритм в алфавите **A** задается в виде системы допустимых подстановок, дополненной точным предписанием о том, в каком порядке нужно применять допустимые подстановки и когда наступает остановка.

Такой набор предписаний вместе с алфавитом **A** и набором подстановок **B** определяют нормальный алгоритм. Процесс останавливается только в двух случаях:

* когда подходящая подстановка не найдена;
* когда применена последняя подстановка из их набора.

    Различные нормальные алгоритмы отличаются друг от друга алфавитами и системами подстановок.    Нормальный алгоритм Маркова можно рассматривать как универсальную форму задания любого алгоритма. Универсальность нормальных алгоритмов декларируется принципом нормализации: для любого алгоритма в произвольном конечном алфавите **A** можно построить эквивалентный ему нормальный алгоритм над алфавитом **А**.

В некоторых случаях не удается построит нормальный алгоритм, эквивалентный данному в алфавите **A**, если использовать в подстановках алгоритма только буквы этого алфавита. Однако можно построить требуемый нормальный алгоритм, производя расширение алфавита **A** (добавляя к нему некоторое число новых букв). В этом случае говорят, что построенный алгоритм является алгоритмом над алфавитом **A**, хотя он будет применяться лишь к словам в исходном алфавите **А**.    Если алгоритм **N** задан в некотором расширении алфавита **A**, то говорят, что **N** есть ***нормальный алгоритм над алфавитом* А**.    Условимся называть тот или иной алгоритм ***нормализуемым***, если можно построить эквивалентный ему нормальный алгоритм, и ненормализуемым в противном случае. Принцип нормализации теперь может быть высказан в видоизмененной форме: все алгоритмы нормализуемы.

    Данный принцип не может быть строго доказан, поскольку понятие произвольного алгоритма не является строго определенным и основывается на том, что все известные в настоящее время алгоритмы являются нормализуемыми, а способы композиции алгоритмов, позволяющие строить новые алгоритмы из уже известных, не выводят за пределы класса нормализуемых алгоритмов.

**Способы композиции нормальных алгоритмов:**

1. ***Суперпозиция алгоритмов***. При суперпозиции двух алгоритмов слово первого алгоритма рассматривается как входное слово второго алгоритма **B**, результат суперпозиции **C** может быть представлен в виде **C(p)** и **B(A(p))**.

Пример 4: рассмотрим алгоритм, позволяющий по любому слову в алфавите А = {a, b} определить, имеет ли оно четную длину.

Условимся считать, что этот алгоритм должен перерабатывать слова в алфавите А = {a, b}, имеющие четную длину (и только их), в пустое слово.

НАМ задается алфавитом В = А и нормальной схемой:



Этот алгоритм может быть задан в виде следующего предписания:

1. Вместо каждой буквы из алфавита А = {a, b} запиши ǀ, переходи к п. 2.

2. Выясни, содержит ли полученное слово вхождение ǁ. Если да, переходи к п. 3, если нет, – к п. 4.

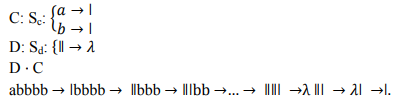
3. Выброси каждое вхождение слова ǁ, переходи к п. 4.

4. Сравни полученное слово с λ. Если оно совпадает, то переходи к п. 5, иначе – к п. 6.

5. Слово имеет четную длину, переходи к п.7.

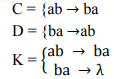
6. Слово имеет нечетную длину, переходи к п.7.

7. Процесс вычисления остановить. Можно выделить два этапа: нахождение длины исходного слова и распознавание четности длины.



1. ***Объединение алгоритмов***. Объединением алгоритмов **А** и **B** в одном и том же алфавите называется алгоритм **С**в том же алфавите, преобразующий любое слово **p** содержащееся в пересечении областей определения алгоритмов **А** и**В**, в записаныe рядом слова **А(р)** и **В(р)**.
2. ***Разветвление алгоритмов***. Разветвление алгоритмов представляет собой композицию **D** трех алгоритмов **A**, **В**и **С**, причем область определения алгоритма **D** является пересечением областей определения всех трех алгоритмов **А**, **В**и **С**, а для любого слова **р** из этого пересечения **D(p) = А(р)**, если **С(р) = е, D(p) = B(p)**, если **C(p) = е**, где **е** - пустая строка.

Пример 5: пусть даны алфавит А = {a, b}, нормальные алгоритмы в том же алфавите, имеющие следующие нормальные схемы:



Рассмотрим работу над словами R = aba и Q = bab

1. C(aba) = baa

D(aba) = aab

K (aba) = aa

H(aba) = aab, так как K(R) ≠ λ

1. C(bab) = bba

D(bab) = abb

K (bab) = λ

H(bab) = bba, так как K(R) = λ

1. ***Итерация алгоритмов***. Итерация (повторение) представляет собой такую суперпозицию **С** двух алгоритмов **А** и**В**, что для любого входного слова **p** соответствующее слово **С(р)** получается в результате последовательного многократного применения алгоритма **А** до тех пор, пока не получится слово, преобразуемое алгоритмом **B**.

Пример 5: А = {a, b}, C = {ab → ba, D = {bbbaa → ab, R = ababb

ababb →baabb →babab →bbaba →bbbaa → ab

Нормальные алгоритмы Маркова являются не только средством теоретических построений, но и основой специализированного языка программирования, применяемого как язык символьных преобразований при разработке систем искусственного интеллекта. Это один из немногих языков, разработанных в России и получивших известность во всем мире. Существует строгое доказательство того, что по возможностям преобразования нормальные алгоритмы Маркова эквивалентны машинам Тьюринга.

**ВЫВОДЫ:**Для формализации понятия алгоритма А.А.Марков предложил использовать ассоциативные исчисления. Нормальные алгоритмы являются вербальными, то есть предназначенными для применения к словам в различных алфавитах.

**Вычисление арифметических функций с помощью нормальных алгоритмов**

Определение 2.

Функция y=F(x1,x2,…,xn) называется арифметической функцией, если аргументы xi и значение y принимают только натуральные значения из N \*={0, 1, 2, 3,…}. Положительные натуральные числа зададим как слова в однобуквенном алфавите А={1}: пусть число n задается в виде слова из n «палочек».

Построим алгоритм, который любое слово вида перерабатывает в слово вида , т.е. вычисляет произведение двух чисел.

Пример 6. В алфавите А3={1, \* ,0,v} нормальный алгоритм I3 задается следующей упорядоченной системой ориентированных подстановок:

1. \* 11→0 \* 1

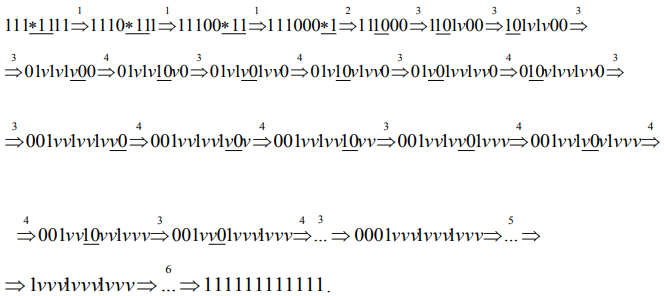
2. \* 1→Λ

3. 10→01v

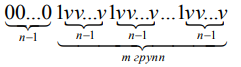
4. v0→0v

5. 0→Λ

6. v→1

Пусть m = 3, n = 4, т.е. зададим I3 слово 1 1 1 2 3 3 3 

Обобщенно можно сказать, что подстановка (1) и (2), примененная к слову , перерабатывает n единиц, расположенных справа от \* , в (n - 1) нулей. При этом m не меняется, а \* исчезает. Далее, каждая перестановка пары 10 с помощью подстановки (3) порождает символ v. При этом 1 не исчезает. Поэтому последовательное применение подстановок (3) и (4), «перетаскивает» все нули на начало слова и каждое такое перемещение нуля через единицы порождает m символов v в итоге подстановки (1), (2), (3),(4) приводят нас к слову вида:



А теперь, применяя подстановку (5), избавимся от нулей и в оставшемся слове, будет содержаться m \* n символов. Применяя подстановку (6) получим (m \* n) расположенных подряд символов и на этом процесс закончится.

Пример 7. Алгоритм I4 для вычисления функции .

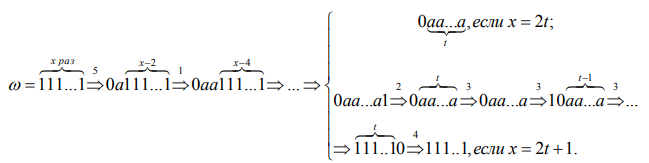
1. a11 → aa

2. a1 → a

3. 0a → 10

4. 0 → Λ . (подстановка с меткой)

5. 11 → 0a



Так как (4) – заключительная подстановка, то процесс завершается, и мы получаем

.

Примеры 6, 7 показывают, как можно приспособить нормальные алгоритмы Маркова для вычисления арифметических функций. Можно показать, что все известные из арифметики и теории чисел функции вычислимы с помощью нормальных алгоритмов. Это обстоятельство позволило А. Маркову предложить в качестве определения понятия алгоритма нормальный алгоритм, и это, в силу теоремы Детловса имеет достаточное основание.

Пример 8. Нормальный алгоритм Маркова в алфавите А={1,0,\*} задан следующей системой ориентированных подстановок:

1) 11\*→\*

2) \*11→\*

3) 1\*1→1

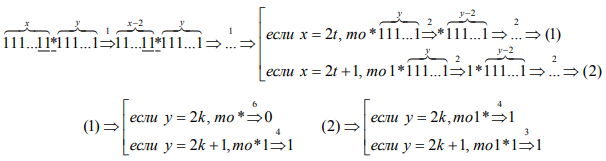
4) \*1→1

5) 1\*→1

6) \*→0

В какое слово перерабатывает этот алгоритм слово вида 111…1\*111…1, то есть, какую арифметическую функцию f(x,y) он вычисляет?

Решение:



алгоритм завершился. Данный алгоритм вычисляет двухместный предикат ⎩ ⎨ ⎧ = в остальных случаях если x и y четные .