АЛГОРИТМЫ МАРКОВА

Ассоциативные исчисления.

Пусть имеется алфавит (конечный набор различных символов). Составляющие его символы будем называть буквами. Любая конечная последовательность букв алфавита (линейный их ряд) называется **словом** в этом алфавите.

Рассмотрим два слова N и M в некотором алфавите A. Если N является частью M, то говорят, что N входит в M.

Зададим в некотором алфавите конечную систему подстановок
N - M, S - T,..., где N, M, S, T,... - слова в этом алфавите.

Любую подстановку N-M можно применить к некоторому слову K следующим способом: если в K имеется одно или несколько вхождений слова N, то любое из них может быть заменено словом M, и, наоборот, если имеется вхождение M, то его можно заменить словом N.

Пример. Пусть в алфавите A = {a, b, c} имеются слова N = ab, M = bcb, K = abcbcbab, Заменив в слове K слово N на M, получим bcbcbcbab или abcbcbbcb, и, наоборот, заменив M на N, получим aabcbab или abcabab.

Подстановка ab - bcb недопустима к слову bacb, так как ни ab, ни bcb не входит в это слово. К полученным с помощью допустимых подстановок словам можно снова применить допустимые подстановки и т.д.

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 P_1 и P_2 в некотором ассоциативном исчислении называются **смежными**, если одно из них может быть преобразовано в другое **однократным** применением допустимой подстановки.

Последовательность слов $P, P_1, P_2, ..., M$ называется дедуктивной цепочкой, ведущей от слова P *к* слову M, если каждое из двух рядом стоящих слов этой цепочки - смежное.

Если можно построить цепочку, ведущую от слова R к слову S, то в силу симметричности подстановок можно построить и цепочку, ведущую от S к R. Слова R и S будут эквивалентными в данном ассоциативном исчислении (R~S).

Пример.

Алфавит {*a, b, c, d, e*}.

Подстановки:

ac – ca; abac - abace

ad - da; eca - ae

bc - cb; eda - be

bd - db; edb - be

Слова *abcde* и *acbde* - смежные (подстановка *bc* - *cb*). Слова *abcde* и *cadbe* - эквивалентны.

Лемма. Пусть P~Q; тогда, если в каком-либо слове R имеется вхождение P, то в результате подстановки вместо него Q получается слово, эквивалентное R.

Если для ассоциативного исчисления можно указать способ, с помощью которого для любой пары слов S_1 и S_2 можно определить, являются ли эти слова эквивалентными или нет, то говорят, что в этом ассоциативном исчислении разрешима проблема эквивалентности слов.

Пример 1.1. Дано исчисление: алфавит A={a, b, c} и система подстановок:

ba ↔ ab

ca ↔ ac

cb ↔ bc

Доказать, что в подобном исчислении разрешима проблема эквивалентности слов.

Легко заметить, что приведенная система подстановок обладает одним свойством: каждая подстановка меняет порядок букв в слове, не изменяя количества этих букв. Алгоритм определения, являются ли два заданных слова S_1 и S_2 эквивалентными, состоит в подсчете количества букв a, b, c в данных словах: если число букв a, b, c в слове S_1 равно числу тех же букв в слове S_2, то вне зависимости от расположения этих букв в S_1 и S_2 слова будут эквивалентными.

В 1946 и 1947 годах русский математик А.А.Марков и американский математик Э.Пост независимо один от другого построили конкретные примеры ассоциативных исчислений, для каждого из которых проблема эквивалентности слов алгоритмически неразрешима. Тем более не существует алгоритма для распознавания эквивалентности слов в любом исчислении. Первые примеры, построенные для алгоритмической неразрешимости ассоциативных исчислений, оказались весьма громоздкими и насчитывали сотни допустимых подстановок. Была поставлена задача построения по возможности более простых примеров. В этом направлении можно выделить успех математика Г.С.Цейтина, построившего пример ассоциативного исчисления, насчитывающего всего семь допустимых подстановок, в котором проблема эквивалентности слов также алгоритмически неразрешима.

Пример 1.2 (Г.С.Цейтин). Дано ассоциативное исчисление, включающее алфавит A={ a, b, c, d, e} и систему подстановок:

1. ac ↔ ca,
2. ad ↔ da,
3. bc ↔ cb,
4. bd ↔ db,
5. abac ↔ abace,
6. eca ↔ ae,
7. edb ↔ be.

В данном исчислении неразрешима проблема эквивалентности слов.

Формальные основы экспертных систем

Большинство экспертных систем базируется на понятии "формальная продукционная система". Продукционные системы берут свое начало с работ Е.Поста, который в 1943 г. ввел термины продукция и каноническая (продукционная) система. Е.Пост показал, что продукционная система является логической системой, эквивалентной машине Тьюринга.

Другими словами, продукционные системы универсальны, т. е. любая формальная система, оперирующая символами, может быть реализована в виде одной из продукционных систем Е. Поста.

Система продукций Поста задается своим алфавитом C= {c ,...,c}и системой базисных продукций $x_i W \rightarrow W y_i$ (i =1,...,1), где x_i, y_i - слова в алфавите С.

Пусть некоторое слово Y начинается словом x_i . Применить к Y продукцию $x_iW \rightarrow Wy_i$ - это значит вычеркнуть из Y начальный отрезок x_i и затем к оставшемуся слову приписать слово y_i. Например, применив к слову aba продукцию abW →Wc, получим слово ac.

Каждая система продукций понимается как формальная система с правилами вывода p_i (i = 1, ... , n), где p_i (F,Y) считается истинным (применимым), если слово Y получается из F при помощи продукции $x_iW \rightarrow Wy_i$.

Наложив на набор упорядоченных продукций неявную управляющую структуру, перейдем к понятию нормального алгоритма Маркова. В алгоритме Маркова упорядоченные продукции (формулы подстановок) применяются к некоторому заданному слову.

Преобразуемое слово	Марковская подстановка	Результат
138 578 926	(8 578 9 → 00)	130 026
тарарам	(ара → ε)	трам
шрам	(ра → ар)	шарм
функция	(ε → ζ-)	ζ-функция
логика	(ика → ε)	лог
книга	(ε → ε)	книга
поляна	(пор → т)	[неприменима]

Алгоритмы Маркова

Введем понятие алгоритма на основе ассоциативного исчисления: алгоритмом в алфавите А называется понятное точное предписание, определяющее процесс над словами из А и допускающее любое слово в качестве исходного.

Алгоритм в алфавите А задается в виде системы допустимых подстановок, дополненной точным предписанием о том, в каком порядке нужно применять допустимые подстановки и когда наступает остановка.

Пример.

Алфавит: A = {a, b, c}

Система подстановок B:

cb → cc

cca→ ab

ab → bca

Предписание о применении подстановок: в произвольном слове P надо сделать возможные подстановки, заменив **левую часть** подстановок **на правую;** повторить процесс с вновь полученным словом.

Алгоритмы Маркова

Введем понятие алгоритма на основе ассоциативного исчисления: алгоритмом в алфавите А называется понятное точное предписание, определяющее процесс над словами из А и допускающее любое слово в качестве исходного.

Алгоритм в алфавите А задается в виде системы допустимых подстановок, дополненной точным предписанием о том, в каком порядке нужно применять допустимые подстановки и когда наступает остановка.

Пример.

Алфавит: A = {a, b, c}

Система подстановок B:

cb → cc

cca→ ab

ab → bca

Предписание о применении подстановок: в произвольном слове P надо сделать возможные подстановки, заменив **левую часть** подстановок **на правую;** повторить процесс с вновь полученным словом.

Система подстановок B:

cb → cc

cca→ ab

ab → bca

Применяя систему подстановок B из рассмотренного примера к словам babaac и bcacabc получаем:

babaac → bbcaaac → остановка

bcacabc → bcacbcac → bcacccac → bcacabc → бесконечные процесс (остановки нет), так как мы получили исходное слово.

Проанализируем сложение натуральных чисел. На входе алгоритма поступают два натуральных числа x и y. Пусть числа x и y записаны в десятичной системе счисления, например, x = 1253 и y = 2754.

Мы рассматриваем цифры 0, 1, 2, . . . , 9 и знак сложения „+” как буквы. Получаем алфавит

A = {0, 1, 2, . . . , 9, +} из 11 букв.

При операции сложения на вход алгоритма поступает слово 1253+2754 длины 9. На выходе алгоритма имеем слово 4007 в алфавите A . Поэтому сложение натуральных чисел - это процесс переработки слов в алфавите A .

Алгоритмы, которые являются процессами переработки слов, назовем **вербальными алгоритмами**. Поскольку речь идет о произвольных, а не строго заданных правилах действий со словами, то понятие вербального алгоритма является интуитивным понятием.

Среди вербальных алгоритмов выделим некоторые алгоритмы - **нормальные алгоритмы** со строго заданными правилами переработки слов. Поэтому понятие нормального алгоритма является строгим математическим понятием.

Нормальные алгоритмы Маркова

Теория нормальных алгоритмов (или алгорифмов, как называл их создатель теории) была разработана советским математиком А. А. Марковым (1903–1979) в конце 1940-х - начале 1950-х гг. XX в. Эти алгоритмы представляют собой некоторые правила по переработке слов в каком-либо алфавите, так что исходные данные и искомые результаты для алгоритмов являются словами в некотором алфавите. В этом определении считается, что алгоритмический процесс - это процесс переработки слов некоторого алфавита. Действительно, многие алгоритмические процессы можно рассматривать как процессы переработки слов.

Алфавит и схема нормального алгоритма.

Нормальный алгоритм М задается двумя компонентами:

1)алфавитом алгоритма,

2)схемой алгоритма.

1) Алфавит нормального алгоритма М - это некоторая конечная совокупность символов $A = \{a1, \ldots, an\}$. Элементы из А мы будем называть буквами. Из букв алфавита А составляются слова - конструктивные объекты, которые поступают на вход и перерабатываются в алгоритме М. При этом говорят, что М - алгоритм в алфавите А .

2) Схема нормального алгоритма. Рассмотрим две буквы → и →• , которые не входят в алфавит А. Формулой подстановки назовем выражение

$$u \rightarrow v$$

или выражение

$$u \rightarrow\bullet\ v,$$

где u и v—произвольные слова в алфавите А.

Формула без точки называется *простой формулой*, а формула с точкой называется *заключительной формулой*. В обоих случаях формула имеет левую часть *и* и правую часть *v*, которые должны быть словами в алфавите А .

Схема Z нормального алгоритма А - это конечная упорядоченная последовательность формул подстановок

$$u_1 \rightarrow v_1$$
$$u_2 \rightarrow v_2$$
$$\ldots$$
$$u_k \rightarrow v_k,$$

где вместо некоторых формул $u_i \rightarrow v_i$ могут быть формулы с точкой $u_i \rightarrow\bullet\ v_i$.

Работа нормального алгоритма.

Пусть на вход нормального алгоритма А со схемой *Z* поступило слово *P*. Начинаем процесс, направленный на получение из исходного слова *P* некоторого слова *Q* - результата переработки слова *P*. Совершаем проход по схеме *Z*, двигаясь сверху вниз, выполняя следующие действия 1)–5).

1) Среди левых частей $u_1, u_2, \ldots, u_k$ нужно отыскать первое по порядку слово u_i, которое входит в слово P.

2) Заменить найденное слово u_i в слове P на слово v_i, и получить некоторое слово P1. Возможно несколько вхождений u_i в слово P. Тогда заменяется на v_i первое вхождение u_i в слове P.

3) Если при замене применялась заключительная формула $u_i \rightarrow\bullet\ v_i$, то переработка слова P завершена и алгоритм останавливается. Полученное слово P1 - результат переработки слова P.

4) Если при замене применялась незаключительная формула $u_i \rightarrow v_i$, то слово P1 заново обрабатывается схемой Z и алгоритм продолжает работу.

5) Возможно, что при проходе по схеме Z вообще не обнаружено ни одного вхождения слов $u_1, u_2, \ldots, u_k$ в слово P. Тогда результат переработки - само слово P и алгоритм останавливается.

Таким образом, в результате получаем ровно один из двух случаев I или II.

I. Процесс переработки слов обрывается и получено слово Q = A (P),

т.е. получен результат применения нормального алгоритма A к слову P.

II. Процесс переработки слов бесконечен. Тогда нет результата Q = A(P), и алгоритм A неприменим к слову P.

Очевидно, что описанный выше нормальный алгоритм A удовлетворяет всем требованиям, предъявляемым к понятию алгоритма, и является точным математическим понятием.

Примеры НАМ

1.Алгоритм сложения натуральных чисел.

Алфавит: A = (+, 1).

Система подстановок B:

$1+ \rightarrow +1$

$+1 \rightarrow 1$

$1 \rightarrow\cdot 1$

Слово P: 11+11+111.

Последовательная переработка слова *P* с помощью нормального алгоритма Маркова проходит через следующие этапы:

$P = 11+11+111$ $\quad P_5 = +1+111111$

$P_1 = 1+111+111$ $\quad P_6 = ++1111111$

$P_2 = +1111+111$ $\quad P_7 = +1111111$

$P_3 = +111+1111$ $\quad P_8 = 1111111$

$P_4 = +11+11111$ $\quad P_9 = 1111111$

2. **Вычисление остатка при делении на 4**. Построим нормальный алгоритм, который находит остаток от деления натурального числа x на 4.
Рассмотрим алфавит из двух символов A = {0, |}.

Искомый нормальный алгоритм имеет схему из одной формулы

$$|||| \to \varepsilon$$

(простой формулы подстановки, правая часть которой - пустое слово).

Пусть на вход алгоритма подается слово

P = 0 ||||…|||| |…|
q r

Оно изображает натуральное число x = 4q+r, где r ∈{0, 1, 2, 3} - остаток от деления числа x на 4. После q проходов по схеме в слове P сотрется 4q палочек и останется r палочек. При q+1 проходе замены нет, алгоритм останавливается. Результат переработки Q равен

0 | . . . |
r

и изображает остаток r.

3. Вычитание единицы (считаем, что $0 - 1 = 0$):

$A = \{|\ ,\ \varepsilon\}$.

$|\ \to .\ \varepsilon$

4. Алгоритм делимости числа на три. Для этого будем стирать по три цифры за один шаг алгоритма, после чего проверим, что осталось. Результатом преобразования будет символ |, если число делится на 3 и пустое слово в противном случае. Схема алгоритма имеет следующий вид:

$||| - \to \varepsilon$

$|| - \to .\ \varepsilon$

$| - \to .\ \varepsilon$

$\varepsilon - \to .\ |$

Примеры: $||||||| \Rightarrow |||| \Rightarrow | \Rightarrow \varepsilon$ (7 не делится на 3),

$||||||||| \Rightarrow ||||| \Rightarrow ||| \Rightarrow \varepsilon \Rightarrow |$ (9 делится на 3).

5. Аннулирующий алгоритм. Зададим нормальный алгоритм М схемой

$a_1 \to \varepsilon$

$a_2 \to \varepsilon$

. . .

$a_k \to \varepsilon$

Пусть $P = ai_1 \ldots ai_m$ - слово на входе.

Вначале алгоритм совершит m проходов по схеме, каждый раз стирая в слове по одной букве с наименьшим индексом. При m+1 проходе на входе пустое слово. Алгоритм останавливается. Получаем результат $M(P) = \varepsilon$.

Отметим, что не все вербальные алгоритмы являются нормальными алгоритмами. Вербальные алгоритмы реализуют произвольные преобразования слов, а нормальные алгоритмы ограничены преобразованиями слов по заданной схеме.

Например, можно задать вербальный алгоритм в алфавите $A = \{a_1, \ldots, a_n\}$, где $n > 1$, с единственным правилом: любое слово $P = a_{i1} \ldots a_{im}$ в алфавите A перерабатывается в перевернутое слово $Q = a_{im} \ldots a_{i1}$ (обращающий алгоритм).

Можно доказать, что **не существует нормального алгоритма** в алфавите A с данным действием.

Поэтому класс нормальных алгоритмов Маркова не совпадает с классом вербальных алгоритмов, а имеет другую формулировку.

Определение 1. Пусть заданы алфавиты A и A1. Если $A \subseteq A1$, то алфавит A1 называется расширением алфавита A .

Определение 2. Нормальный алгоритм в каком-либо расширенииA1 алфавита A называется нормальным алгоритмом над алфавитом A .

Нормальный алгоритм в каком-либо расширении A1 при действии на словах алфавита A реализует более богатый набор преобразований слов по сравнению с нормальным алгоритмом в алфавите A .

Примеры.

1. Алгоритм М левого присоединения слова v. В нем для любого входного слова **u** выходным словом должно быть слово **vu**, которое получено приписыванием к слову **u** слева слова **v**.

Искомый нормальный алгоритм М имеет простейший вид. Его схема состоит из одной формулы подстановки

ε →·v //стоящий слева или справа одиночный символ ε можно опустить.

Пусть имеется входное слово u. Оно имеет вид u = εu. Пустое слово ε заменяется на v. Получается слово vu, что и нужно.

2. Алгоритм правого присоединения. В этом случае нет простейшего алгоритма. Для реализации правого присоединения расширим алфавит $A = \{a_1, \ldots, a_n\}$. Пусть α - некоторая буква, отличная от всех букв алфавита A . Добавив букву α, получим новый алфавит $A1 = A \cup \{\alpha\}$ - расширение алфавита A .

Рассмотрим нормальный алгоритм A1 в новом алфавите A1. В соответствии с определением 2, алгоритм M1 - это алгоритм над старым алфавитом A .
Его схема имеет следующий вид.

$$\alpha a_1 \to a_1 \alpha$$
$$\ldots$$
$$\alpha a_n \to a_n \alpha$$
$$\alpha \to \cdot\ v$$
$$\to\ \alpha$$

Пусть на вход A1 поступило слово **u** в алфавите A . Оно не содержит буквы **α**. Поэтому при первом проходе по схеме сработает последняя формула подстановки **α**. Пустое слово в начале **u** заменится на **α**. В результате к слову **u** слева добавится буква **α**. При следующих проходах по схеме буква **α** перемещается несколько раз вправо, пока не станет в конце слова. При следующем проходе буква **α** заменится на **v** и алгорифм остановится. Слово **u** переработалось в слово **uv**.

3. Алгоритм удаления первой буквы непустого слова.

Алфавит $\Sigma = \{a, b, c\}$.

Вставим в начало слова вспомогательный символ *, а затем удалим его вместе со следующей за ним буквой заключительной подстановкой (таких подстановок в схеме должно быть две по числу букв в алфавите).

Схем подстановок:

$*a \to . \varepsilon$

$*b \to . \varepsilon$

$*c \to . \varepsilon$

$\varepsilon \to *$

Замечание. При разработке схемы следует внимательно относиться к порядку следования подстановок, учитывая, что применяется всегда первая из применимых к слову подстановок.

Вспомогательный символ фиксирует во входном слове начальную позицию. Без его использования отличить первую букву от всех последующих невозможно. Заметим, что на первом шаге нормального алгоритма всегда будет применяться последняя подстановка схемы, а на втором - первая, вторая или третья (заключительная):

$ababc \Rightarrow *ababc \Rightarrow babc$

$caba \Rightarrow *caba \Rightarrow aba$

При записи системы подстановок (т.е. схемы алгоритма) иногда удобно сокращать запись, вводя новые буквы, не принадлежащие алфавиту Σ, в котором определены подстановки. Каждая такая буква означает любую из букв алфавита Σ. Например, в предыдущей задаче, вместо подстановок

$*a \rightarrow . \varepsilon$

$*b \rightarrow . \varepsilon$

$*c \rightarrow . \varepsilon$

можно ввести обобщенную подстановку

$*\mathfrak{z} \rightarrow . \varepsilon$, где $\mathfrak{z} \in \{a, b, c\}$.

Если применить эту схему к пустому слову, то мы получим зацикливание (первые две подстановки останутся после первого шага неприменимыми, а третья не предусматривает остановки): $\varepsilon \Rightarrow * \Rightarrow ** \Rightarrow * * * \Rightarrow \ldots$

Устранить такое поведение можно, добавив третью заключительную подстановку, удаляющую одинокий символ *:

$$*a \to . \varepsilon$$
$$*b \to . \varepsilon$$
$$* \to . \varepsilon$$
$$\varepsilon \to *$$

Теперь поведение на пустом слове следующее: $\varepsilon \Rightarrow * \Rightarrow \varepsilon$.

Применение заключительных подстановок становится возможным только после применения последней подстановки, вставляющей вспомогательный символ * в начало слова.

Таким же образом, добавляя буквы к алфавиту A , можно получить схему для операции перевертывания слов в алфавите A - операции, которую нельзя реализовать схемой в алфавите A . Поэтому в данном случае добавление буквы к алфавиту A и рассмотрение расширенного алфавита A1 позволили схемой в A1 реализовать те действия над словами в исходном алфавите A , которые нельзя было реализовать схемой в A .

Таким образом, добавляя новые буквы к алфавиту A , мы получаем нормальный алгоритм A1 в расширенном алфавите A1, который при ограничении на словах из алфавита A имеет наперед заданное действие.

4. Увеличение на единицу натурального числа в двоичной форме.

Действие алгоритма будет состоять из двух этапов:

1) сначала найдём конец слова (для этого вставленный в начало слова вспомогательный символ будет последовательно перемещаться в его конец);
2) выполним прибавление единицы с переносом единичного разряда (перенос будет реализован движением в обратном направлении), если это окажется необходимым. Схема этого алгоритма имеет следующий вид (для удобства подстановки здесь помечены):

Алфавит A={0,1}. В расширенный алфавит добавим два символа: *a* и *b*.

$0b \rightarrow .\ 1$ (а)

$1b \rightarrow b0$ (б)

$b \rightarrow .\ 1$ (в)

$a0 \rightarrow 0a$ (г)

$a1 \rightarrow 1a$ (д)

$0a \rightarrow 0b$ (е)

$1a \rightarrow 1b$ (ё)

$\varepsilon \rightarrow a$ (ж)

Первый этап реализуется применением подстановки (ж) и следующей за ней последовательностью применений подстановок (г) и (д). В результате первого этапа вспомогательный символ *a* оказывается в конце слова.

Второй этап: подстановки (е) или (ё) заменяют *a* на *b* и либо срабатывают заключительные подстановки (а) или (в), либо запускается перенос единичного разряда подстановкой (б). Рассмотрим действие алгоритма на примерах:

ε (ж) $\Rightarrow$ *a* (ж) $\Rightarrow$ *aa* (ж) $\Rightarrow$ *aaa* (ж) $\Rightarrow$ . . .

0 (ж) $\Rightarrow$ *a*0 (г) $\Rightarrow$ 0*a* (е) $\Rightarrow$ 0*b* (а) $\Rightarrow$ 1

1 (ж) $\Rightarrow$ *a*1 (д) $\Rightarrow$ 1*a* (ё) $\Rightarrow$ 1*b* (б) $\Rightarrow$ *b*0 (в) $\Rightarrow$ 10

11 (ж) $\Rightarrow$ *a*11 (д) $\Rightarrow$ 1*a*1 (д) $\Rightarrow$ 11*a* (ё) $\Rightarrow$ 11*b* (б) $\Rightarrow$ 1*b*0 (б) $\Rightarrow$ *b*00 (в) $\Rightarrow$ 100

После достижения первым вспомогательным символом конца слова действуют следующие соображения. Если число заканчивается на цифру 0, то она заменяется на 1 и вычисление завершается. Если число заканчивается на цифру 1, то она заменяется на 0, а затем запускается перенос разряда. Разработанный алгоритм к пустому слову неприменим.

Эту схему нетрудно обобщить на десятичную систему счисления: для этого достаточно добавить подстановки, дублирующие подстановки (а), (г)-(ё) для остальных цифр 2–9 и учесть, что роль 1, как разряда, для которого возникает перенос, теперь будет играть цифра 9.

Таким образом, добавляя новые буквы к алфавиту А , мы получаем нормальный алгоритм М в расширенном алфавите А1, который при ограничении на словах из алфавита А имеет наперед заданное действие.

А.А.Марков предложил следующий тезис.

Принцип нормализации. ***Пусть задан произвольный вербальный алгоритм В в алфавите А . Тогда существует расширение А1 алфавита А и нормальный алгоритм М в алфавите А1 с условием:***
произвольное слово Р в алфавите А перерабатывается нормальным алгорифмом М в тот же самый результат, в который слово Р перерабатывается исходным вербальным алгоритмом В.

Нормальный алгоритм Маркова можно рассматривать как универсальную форму задания любого алгоритма.

Универсальность нормальных алгоритмов декларируется **принципом нормализации**: для любого алгоритма в произвольном конечном алфавите *А* можно построить эквивалентный ему нормальный алгоритм над алфавитом *А,*

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. Ниже (см. следующие слайды) перечислены способы композиции нормальных алгоритмов.

Доказано, что данная формулировка понятия алгоритма (принцип нормализации) эквивалентна другим формулировкам понятия алгоритма: тезису Черча, использующему частично рекурсивные функции, и тезису Тьюринга, использующему понятие вычислительной машины. Поэтому еще раз подкрепляется уверенность в том, что мы нашли и выразили в трех формах фундаментальное понятие математики, логики и информатики - **понятие алгоритма**. При этом частичная рекурсивность, машина Тьюринга и нормальный алгоритм - лишь различные формы выражения этого самостоятельного понятия.

Нормально вычислимые функции

Нормальные алгоритмы в описанной выше форме не производят собственно вычислений, они преобразуют слова, то есть манипулируют символами. Однако результаты таких преобразований вполне можно интерпретировать как вычисления, а значит, нормальными алгоритмами можно воспользоваться для определения вычислимых функций.

Назовём *функцию f*, заданную на некотором множестве слов алфавита Σ, нормально вычислимой, если найдётся такой нормальный алгоритм над этим алфавитом, что каждое слово $P(\in \Sigma^*)$ из области *определения f* преобразуется этим алгоритмом в *слово* $f(P)$.

Заметим, что определение нормально вычислимой функции допускает использование нормальным алгоритмом вспомогательных символов из расширения алфавита Σ. Если нормальный алгоритм на некотором слове не завершается, то считается, что функция на этом слове не определена.