НОРМАТИВНЫЙ КОНСПЕКТ ПО ДИСЦИПЛИНЕ «ИНТЕЛЛЕКТУАЛЬНЫЕ СИСТЕМЫ В ГУМАНИТАРНОЙ СФЕРЕ»

Часть 5
«АЛГОРИТМИЧЕСКАЯ МОДЕЛЬ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ. РАЗРАБОТКА СХЕМ НОРМАЛЬНЫХ (МАРКОВСКИХ) АЛГОРИТМОВ»

Введение. Рассмотрим примеры, в которых демонстрируются типичные приемы разработки схем нормальных (марковских) алгоритмов. Буквой *P* будем обозначать входное слово, буквой *A* будем обозначать алфавит входного слова, то есть набор тех символов, которые и только которые могут входить во входное слово *P* (но в процессе выполнения алгоритма в обрабатываемых словах могут появляться и другие символы). Кроме того, в примерах будем справа от формул подстановок указывать их имена. Эти имена не входят в формулы, а нужны для ссылок на формулы при показе пошагового выполнения алгоритмов.

Разработка схемы нормального (марковского) алгоритма: пример с удалением и вставкой символов. Пусть входное слово *P* есть слово в алфавите *A = {a, b, c, d}*. Алгоритм *Ξ*, перерабатывая слово *P*, должен в нем удалить все вхождения символа *c*, а затем заменить первое вхождение подслова *bb* на *ddd*. Например, *Ξ(abbcabbca) = adddabba*.

Нормальными (марковскими) алгоритмами легко реализуются вставки и удаления символов. Вставка новых символов в слово есть замена некоторого «старого» подслова на «новое» подслово с числом символов, большим чем у «старого» подслова. Например, с помощью формулы подстановки *bb→ddd* два символа будут заменены на три символа. При этом не надо заботиться о том, чтобы предварительно освободить место для дополнительных символов: нормальный (марковский) алгоритм «раздвигает» слово автоматически.

Удаление символов есть замена некоторого «старого» подслова на «новое» подслово с числом символов, меньшим чем у «старого» подслова. Например, удаление символа *c* реализуется формулой подстановки *c→* (в ее правой части стоит пустое слово). При этом никаких пустых позиций внутри слова не появляется: нормальный (марковский) алгоритм «сжимает» слово автоматически.

С учетом сказанного нашу задачу должен, казалось бы, решать такой нормальный (марковский) алгоритм:

Ξ: $c\rightarrow$ (R_1)
$bb\rightarrow ddd$ (R_2)

Однако это не так. Проверим этот нормальный (марковский) алгоритм на входном слове *abbcabbca*:

$\Xi(abb\underline{c}abbca)\rightarrow R_1\rightarrow abbabb\underline{c}a\rightarrow R_1\rightarrow a\underline{bb}abba\rightarrow R_2\rightarrow adddda\underline{bb}a\rightarrow R_2\rightarrow$
$\rightarrow adddaddda$

Как видно, алгоритм сначала удалил все символы *c* и только затем заменил первое вхождение *bb* на *ddd*. Но алгоритм на этом не остановился и стал заменять

остальные вхождения *bb*. Дело здесь в том, что, пока применима хотя бы одна формула подстановки, нормальный (марковский) алгоритм продолжает свою работу. Но нам этого не надо, поэтому мы должны принудительно остановить алгоритм после того, как он заменил первое вхождение *bb*. Для этого используются заключительные формулы подстановок, после применения которых алгоритм останавливается. Следовательно, в схеме алгоритма Ξ обычную (незаключительную) формулу *bb→ddd* надо заменить на заключительную формулу *bb→.ddd*:

Ξ: *c→* (R_1)
bb→.ddd (R_2)

Теперь нормальный (марковский) алгоритм Ξ работает правильно:

Ξ*(abbcabbca)*$\to R_1 \to$*abbabbca*$\to R_1 \to$*abbabba*$\to R_2 \to$*adddabba*

Слово, которое получилось после применения заключительной формулы подстановки R_2, является выходным словом, то есть результатом применения нормального (марковского) алгоритма Ξ к входному слову *abbcabbca*.

Проверим наш алгоритм Ξ еще и на входном слове, в которое не входит *bb*:

Ξ*(dcacb)*$\to R_1 \to$*dacb*$\to R_1 \to$*dab*

К последнему слову *dab* не применима ни одна формула подстановки, поэтому, согласно определению нормального (марковского) алгоритма, алгоритм Ξ останавливается и слово *dab* объявляется выходным.

Алгоритм Ξ корректно (в полном соответствии с условием задачи) применим и к пустому слову: $\Xi(\Lambda)=\Lambda$, потому что ни одна из формул подстановок к пустому слову не применима, и алгоритм Ξ сразу останавливается.

Разработка схемы нормального (марковского) алгоритма: пример с перестановкой символов. Входное слово *P* есть слово в алфавите $A = \{a, b\}$. Нормальный (марковский) алгоритм *T* должен преобразовать слово *P* так, чтобы в его начале оказались все символы *a*, а в конце – все символы *b*. Например, *T(babba) = aabbb*.

Казалось бы, для решения этой задачи нужен сложный нормальный (марковский) алгоритм. Но это не так, задача решается с помощью нормального (марковского) алгоритма, содержащего всего лишь одну формулу подстановки:

T: ba→ab (R_1)

Пока в слове *P* справа хотя бы от одного символа *b* есть символ *a*, формула подстановки R_1 будет переносить *a* налево от этого *b*. Формула подстановки R_1 перестает быть применимой, когда справа от *b* нет ни одного *a*. Это и означает, что все *a* оказались слева от *b*. Например:

T($\underline{ba}$*bba)*$\to R_1 \to$*abb*$\underline{ba}$$\to R_1 \to$*ab*$\underline{ba}$*b*$\to R_1 \to$*a*$\underline{ba}$*abb*$\to R_1 \to$*aabbb*

Алгоритм остановился на последнем слове, так как единственная в его схеме формула подстановки к этому слову уже неприменима.

Этот и предыдущий примеры показывают, что нормальными (марковскими) алгоритмами легко реализуются перестановки, вставки и удаления символов. Но для нормальных (марковских) алгоритмов возникает другая проблема: как зафиксировать символ, который должен быть обработан или целое подслово, подлежащее обработке. Рассмотрим эту проблему на следующем примере.

Разработка схемы нормального (марковского) алгоритма: пример с использованием специального знака. Входное слово P есть слово в алфавите $A = \{a, b\}$. Нормальный (марковский) алгоритм Σ должен удалить из слова P, если оно не пустое, его первый символ. Пустое слово P алгоритм Σ не должен менять.

Ясно, что удалив первый символ слова, надо тут же остановиться. Поэтому, казалось бы, задачу решает следующий нормальный (марковский) алгоритм:

Σ: $a\rightarrow.$ (R_1)

$b\rightarrow.$ (R_2)

Но этот алгоритм не решает поставленную задачу, в чем можно убедиться, применив его к слову *bbaba*:

$\Sigma(bb\underline{a}ba)\rightarrow R_1\rightarrow bbba$

Как видно, этот нормальный (марковский) алгоритм удалил не первый символ слова, а первое вхождение символа a, что есть совсем не то же самое. Этот алгоритм будет правильно работать, только если входное слово начинается с символа a. Ясно, что перестановка формул в этом нормальном (марковском) алгоритме не поможет, так как тогда он будет, напротив, неправильно работать на словах, начинающихся с a.

Надо как-то зафиксировать (пометить) первый символ слова, например, поставив перед ним какой-либо знак, скажем *, отличный от символов алфавита слова. После этого уже можно с помощью формул подстановок вида $*\xi\rightarrow.$ заменить этот знак и первый символ ξ входного слова на пустое слово и остановиться:

$bbaba\rightarrow *bbaba\rightarrow baba$

Как же поставить * перед первым символом? Это реализуется формулой подстановки $\rightarrow *$ с пустой левой частью, которая, по определению, приписывает свою правую часть слева к началу слова. Получаем следующий нормальный (марковский) алгоритм:

Σ: $\rightarrow *$ (R_1)

$*a\rightarrow.$ (R_2)

$*b\rightarrow.$ (R_3)

Проверим его на том же входном слове:

$\Sigma(bbaba)\rightarrow R_1\rightarrow *bbaba\rightarrow R_1\rightarrow **bbaba\rightarrow R_1\rightarrow **bbaba\rightarrow ...$

Как видно, этот алгоритм постоянно приписывает слева звездочки и зацикливается. Связано это с тем, что формула подстановки с пустой левой частью применима всегда, поэтому наша формула подстановки R_1 будет работать бесконечно, блокируя доступ к остальным формулам подстановок. Отсюда вытекает очень важное правило: *если в нормальном (марковском) алгоритме есть формула подстановки с пустой левой частью, то ее место – только в самом конце схемы алгоритма* (если, конечно, мы не хотим создать зацикливающийся алгоритм). Учтем это правило и перепишем схему алгоритма:

Σ: $*a\rightarrow.$ (R_1)

$*b\rightarrow.$ (R_2)

$\rightarrow *$ (R_3)

Проверим этот алгоритм:

$\Sigma(bbaba) \rightarrow R_3 \rightarrow \underline{*b}baba \rightarrow R_2 \rightarrow baba$

Ясно: для слов, начинающихся с *a* или *b* (то есть для всех непустых слов, удовлетворяющих условию задачи) этот алгоритм будет работать корректно. Но он зациклится на пустом входном слове, так как постоянно будет применяться формула подстановки R_3. Однако, согласно условию задачи, на таком слове алгоритм должен сразу остановиться. Причина ошибки такова: мы ввели знак * для того, чтобы пометить первый символ слова, а затем уничтожить и *, и этот первый символ. Но в пустом слове нет ни одного символа, поэтому формулы подстановок R_1 и R_2 ни разу не сработают, а постоянно будет выполняться формула подстановки R_3. Следовательно, чтобы учесть частный случай пустого входного слова, надо после формул подстановок R_1 и R_2 записать еще одну формулу подстановки, которая уничтожает «одинокую» звездочку и останавливает алгоритм:

Σ: $*a \rightarrow .$ (R_1)
$*b \rightarrow .$ (R_2)
$* \rightarrow .$ (R_3)
$\rightarrow *$ (R_4)

Теперь алгоритм правильно перерабатывает и пустое слово:

$\Sigma() \rightarrow R_4 \rightarrow \underline{*} \rightarrow R_3 \rightarrow \Lambda$

На этом разработка схемы алгоритма Σ завершена.

Обобщим тот прием, который мы использовали. Пусть в обрабатываемое слово *P* входит несколько раз подслово α:

$P = ... \alpha ... \alpha ... \alpha ...$

Пусть необходимо заменить одно из вхождений α на подслово β. Такая замена реализуется с помощью формулы подстановки $\alpha \rightarrow \beta$. Однако, если мы применим эту формулу подстановки к слову *P*, то будет заменено первое вхождение α. А что делать, если надо заменить какое-то другое вхождение α, скажем второе или последнее? Чтобы на β заменялось не первое вхождение α, а какое-то другое, это другое вхождение надо как-то выделить (пометить), для чего следует рядом с ним (слева или справа) поставить некоторый символ, скажем *, отличный от всех других символов, входящих в *P*:

$P = ... \alpha ... *\alpha ... \alpha ...$

Такой символ будем в дальнейшем называть *специальным знаком*. Его роль – выделить нужное вхождение искомого подслова α среди других его вхождений, сделать требуемое вхождение уникальным. Поскольку только около этого вхождения есть специальный знак, то надо использовать формулу подстановки $*\alpha \rightarrow \beta$, чтобы заменить на β именно это вхождение α, а не какое-то другое.

Этот прием со специальным знаком очень важен, так как при разработке схем нормальных (марковских) алгоритмов он используется часто. Остается еще одна проблема: как специальный знак разместить рядом с нужным вхождением α? Следующие примеры показывают, как это делается.

Разработка схемы нормального (марковского) алгоритма: пример с фиксацией специальным знаком обрабатываемого подслова. Входное слово *P* есть непустое слово в алфавите $A = \{0, 1, 2, 3\}$. Трактуя его как запись неотрицательно-

го целого числа в четверичной системе счисления, нормальный (марковский) алгоритм Δ должен получить запись этого же числа, но в двоичной системе счисления. Для этого надо каждую четверичную цифру (каждую цифру в исходной записи) заменить на пару соответствующих ей двоичных цифр (цифр в записи выходного слова): *0→00*, *1→01*, *2→10*, *3→11*. Например: $\Delta(0123) = 00011011$.

Казалось бы, поставленную задачу решает следующий нормальный (марковский) алгоритм:

Δ: 0→00 (R_1)
1→01 (R_2)
2→10 (R_3)
3→11 (R_4)

Но этот алгоритм некорректен, в чем можно убедиться на входном слове *0123*:
$\Delta(\underline{0}123) \to R_1 \to \underline{0}0123 \to R_1 \to \underline{0}00123 \to R_1 \to 0000123 \to ...$

Алгоритм строит неверный результат да при том еще и зацикливается. Ошибка здесь в том, что после замены четверичной цифры на пару двоичных цифр уже никак нельзя отличить двоичные цифры от четверичных, поэтому приведенный алгоритм начинает заменять и двоичные цифры. Значит, надо как-то отделить ту часть обрабатываемого, в которой уже была произведена замена, от той части, где замены еще не было. Для этого можно пометить слева специальным знаком * ту четверичную цифру, которая сейчас должна быть заменена на пару соответствующих ей двоичных цифр, а после того как такая замена будет выполнена, специальный знак нужно поместить перед следующей четверичной цифрой:

*0123→*0123→00*123→0001*23→000110*3→00011011**

Как видно, слева от специального знака всегда находится та часть числа, которая уже переведена в двоичный вид, а справа – часть, которую еще предстоит заменить. Поэтому никакой путаницы между четверичными и двоичными цифрами уже не будет.

Итак, специальный знак * сначала должен быть размещен слева от первой цифры четверичного числа, а затем он должен «перепрыгивать» через очередную четверичную цифру, оставляя слева от себя соответствующие ей двоичные цифры. В конце же, когда справа от * уже не окажется никакой цифры, специальный знак надо уничтожить и остановиться. Как приписать * слева к входному слову и как уничтожить специальный знак с остановом, мы уже знаем по предыдущему примеру, а вот «перепрыгивание» звездочки реализуется с помощью формул подстановок вида $*\alpha \to \beta\gamma*$, где α есть четверичная цифра, а $\beta\gamma$ есть соответствующая ей пара двоичных цифр.

Получаем следующий нормальный (марковский) алгоритм перевода чисел из четверичной системы в двоичную:

Δ: *0→00* (R_1)
1→01 (R_2)
2→10 (R_3)
3→11 (R_4)
*→. (R_5)

$\rightarrow *$ (R_6)

Проверим этот нормальный (марковский) алгоритм на входном слове *0123*:

$\Delta(0123) \rightarrow R_6 \rightarrow \underline{*}0123 \rightarrow R_1 \rightarrow 00\underline{*}123 \rightarrow R_2 \rightarrow 0001\underline{*}23 \rightarrow R_3 \rightarrow$

$\rightarrow 000110\underline{*}3 \rightarrow R_4 \rightarrow 00011011\underline{*} \rightarrow R_5 \rightarrow 00011011$

Разработка схемы нормального (марковского) алгоритма: пример с перемещением специального знака. Входное слово P есть слово в алфавите $A = \{a, b\}$. Нормальный (марковский) алгоритм Ψ должен приписать символ a к концу слова P. Например: $\Psi(bbab) = bbaba$.

Формула подстановки $\rightarrow a$ приписывает символ a слева к слову P (то есть в начало слова), а не справа (то есть к концу, как того требует условие задачи). Чтобы приписать a справа, надо сначала пометить конец слова. Для этого воспользуемся специальным знаком, который поместим в конец слова P, а затем заменим его (специальный знак) на a:

$P \rightarrow \ldots \rightarrow P* \rightarrow Pa$

Помещение специального знака в конец слова может быть выполнено так: сначала специальный знак * приписывается слева к слову P, а затем «перегоняется» через все буквы слова:

$bbab \rightarrow *bbab \rightarrow b*bab \rightarrow bb*ab \rightarrow bba*b \rightarrow bbab*$

Как же выполнить такой перегон? Нетрудно заметить, что «перепрыгивание» звездочки через какой-то символ ξ – это замена пары $*\xi$ на пару $\xi*$, которая реализуется формулой подстановки $*\xi \rightarrow \xi*$.

С учетом всего сказанного получаем следующий нормальный (марковский) алгоритм:

Ψ: $*a \rightarrow a*$ (R_1)

$*b \rightarrow b*$ (R_2)

$* \rightarrow .a$ (R_3)

$\rightarrow *$ (R_4)

Отметим, что при пустом входном слове этот нормальный (марковский) алгоритм сначала введет звездочку, а затем тут же заменит ее на символ a и остановится.

Разработка схемы нормального (марковского) алгоритма: пример со сменой специального знака. Входное слово P есть слово в алфавите $A = \{a, b\}$. Нормальный (марковский) алгоритм Θ должен в слове P заменить на aa последнее вхождение символа a, если оно имеется. Например: $\Theta(bababb) = babaabb$.

Удвоение символа a реализуется формулой подстановки $a \rightarrow aa$. Но чтобы она применялась не к первому вхождению символа a, а к последнему, надо поставить, скажем, справа от последнего символа a специальный знак * и применить формулу подстановки $a* \rightarrow .aa$.

Теперь посмотрим, как поместить * рядом с последним вхождением символа a. Поскольку последнее вхождение – это первое вхождение от конца, то предлагается сначала приписать * слева к слову P, затем перегнать * в конец слова (это мы уже умеем делать), а далее перегнать * справа налево через символы b до ближай-

шего символа a. Еще надо учесть, что в P может и не быть символа a. Поэтому, если звездочка снова достигнет начала слова, ее надо уничтожить и остановиться. Реализуем эту идею в виде следующего нормального (марковского) алгоритма:

Θ: $*a \to a*$ (R_1)
$*b \to b*$ (R_2)
$b* \to *b$ (R_3)
$a* \to .aa$ (R_4)
$* \to .$ (R_5)
$\to *$ (R_6)

Здесь формула подстановки R_6 приписывает * слева к входному слову P, формулы подстановок R_1 и R_2 перегоняют * в конец P, после чего формула R_3 перемещает * справа налево через все b в конце слова до ближайшего (то есть в слове P последнего) символа a, и, наконец, формула R_4 заменяет этот символ на aa и останавливает алгоритм. Формула же R_5 нужна для входных слов, в которые не входит символ a.

Проверим этот алгоритм на входном слове *bababb*:

$\Theta(bababb) \to R6 \to \underline{*b}ababb \to R_2 \to b\underline{*a}abb \to R_1 \to ba\underline{*b}abb \to R_2 \to$
$\to bab\underline{*a}bb \to R_1 \to baba\underline{*b}b \to R_2 \to babab\underline{*b} \to R_2 \to babab\underline{b*} \to R_3 \to$
$\to babab\underline{*b} \to R_2 \to babab\underline{b*} \to R_3 \to babab*b \to ...$

Алгоритм зациклился. Дойдя до конца слова, вместо того, чтобы двигаться справа влево до ближайшего символа a, звездочка начала «прыгать» вокруг последнего символа слова. Дело в том, что формулы подстановок R_1 и R_2, перегоняющие * вправо, мешают формуле R_3, перегоняющей * влево. Отметим, что перестановка этих формул не поможет, так как тогда * начнет «плясать» вокруг первого символа входного слова.

Ошибка произошла из-за того, что мы используем специальный знак * в двух разных целях – как для движения вправо, так и для движения влево. Чтобы этой ошибки не было, надо просто ввести еще один специальный знак, скажем #, распределив между этими специальными знаками обязанности: пусть * движется вправо, а # – влево. Появиться же специальный знак # должен тогда, когда * дойдет до конца слова, то есть когда справа от * не окажется других символов. Такая замена специального знака «исключит из игры» все формулы подстановок со звездочкой в левой части, поэтому они уже не будут мешать формулам со специальным знаком #.

Если так и сделать, то получим следующий нормальный (марковский) алгоритм:

Θ: $*a \to a*$ (R_1)
$*b \to b*$ (R_2)
$* \to \#$ (R_3)
$b\# \to \#b$ (R_4)
$a\# \to .aa$ (R_5)
$\# \to .$ (R_6)
$\to *$ (R_7)

Проверим этот алгоритм на прежнем входном слове:

$\Theta(bababb) \to R_7 \to \underline{*b}ababb \to R_2$ *и* R_1 *несколько раз* $\to bababb\underline{*} \to R_3 \to$

$\to bababb\underline{b\#} \to R_4 \to baba\underline{b\#}b \to R_4 \to baba\#bb \to R_5 \to babaabb$

Алгоритм работает корректно. Он работает правильно и тогда, когда во входное слово не входит символ a:

$\Theta(bb) \to R_7 \to \underline{*b}b \to R_2 \to b\underline{*b} \to R_2 \to bb\underline{*} \to R_3 \to b\underline{b\#} \to R_4 \to$

$\to \underline{b\#}b \to R_4 \to \underline{\#}bb \to R_6 \to bb$

Верно работает алгоритм и в случае пустого входного слова:

$\Theta() \to R_7 \to \underline{*} \to R_3 \to \underline{\#} \to R_6 \to \Lambda$

Разработка схемы нормального (марковского) алгоритма: пример с переносом символа через слово. Входное слово P есть слово в алфавите $A = \{a, b\}$. Нормальный (марковский) алгоритм Φ должен перенести в конец непустого слова P его первый символ. Пустое же слово P алгоритм Φ не должен менять. Например: $\Phi(bbaba) = babab$.

Используем в этом примере «говорящие» специальные знаки. Припишем к началу слова специальный знак $[?\}$. Он изображает «головку», намеренную узнать, какой символ в слове стоит первым. Приписывание это реализуется формулой подстановки $\to [?\}$. Естественное место этой формулы – в конце схемы алгоритма.

Если первый символ есть a, то головка «схватывает» его с помощью формулы подстановки $[?\}a \to [a\}$. Аналогично головка «схватывает» первый символ, если им является b: $[?\}b \to [b\}$. Если же входное слово является пустым, то формула подстановки $[?\} \to .$ уничтожает специальный знак $[?\}$ и останавливает работу.

Затем головка «несет в себе» схваченный ею символ, «перепрыгивая» через остальные буквы слова. Это реализуется с помощью формул подстановок вида $[\eta\}\xi \to \xi[\eta\}$, здесь η обозначает схваченный символ, а ξ есть буква, через которую перепрыгивает головка.

Когда головка окажется в конце слова, формула подстановки вида $[\eta\} \to .\eta$ «выбросит» принесенный головкой символ, уничтожит головку и завершит работу. Получился следующий нормальный (марковский) алгоритм:

Φ:
$[?\}a \to [a\}$ (R_1)
$[?\}b \to [b\}$ (R_2)
$[?\} \to .$ (R_3)
$[a\}a \to a[a\}$ (R_4)
$[a\}b \to b[a\}$ (R_5)
$[b\}a \to a[b\}$ (R_6)
$[b\}b \to b[b\}$ (R_7)
$[a\} \to .a$ (R_8)
$[b\} \to .b$ (R_9)
$\to [?\}$ (R_{10})

Проверим этот алгоритм на входном слове $bbaba$:

$\Phi(bbaba) \to R_{10} \to \underline{[?\}b}baba \to R_2 \to \underline{[b\}b}aba \to R_7 \to b\underline{[b\}a}ba \to R_6 \to ba\underline{[b\}b}a \to$

$\to R_7 \to bab\underline{[b\}a} \to R_6 \to baba\underline{[b\}} \to R_9 \to babab$

Алгоритм работает правильно. Проверим его еще на однобуквенном слове:

$Ф(a) \to R_{10} \to \underline{[?\}a} \to R_1 \to \underline{[a\}} \to R_8 \to a$

Все корректно. Убедимся в этом и на пустом слове:

$Ф() \to R_{10} \to [?\} \to R_3 \to Λ$

Разработка схемы нормального (марковского) алгоритма: пример с использованием нескольких специальных знаков. Входное слово *P* есть слово в алфавите *A = {a, b}*. Нормальный (марковский) алгоритм *Г* должен удвоить слово *P*, то есть приписать к *P* (слева или справа) его копию. Например: *Г(abb) = abbabb*.

Предлагается изложенный ниже план решения задачи.

1. К началу обрабатываемого слова приписываем специальный знак *|[?\}*. Первый его символ | изображает «стенку», левее которой находятся уже скопированные символы (в начале работы алгоритма там названных символов нет). Второй символ *[?\}* этого специального знака изображает запоминающую головку, намеренную узнать, какой символ обрабатываемого слова находится справа от нее. Приписывание специального знака *|[?\}* к началу обрабатываемого слова реализуется формулой подстановки *→|[?\}*. Поскольку она есть формула подстановки с пустой левой частью, то другие формулы подстановок будут записываться раньше нее так, чтобы данная формула была последней в схеме разрабатываемого нормального (марковского) алгоритма. Пример результата совершения описанного шага:

abb→|[?\}abb

2. Головка запоминает символ, находящийся справа от нее (при этом *[?\}* превращается в *[a\}* или *[b\}*), срывает его с места и перебрасывает через стенку влево. Эти действия задаются парой формул подстановок:

|[?\}a→a|[a\}

|[?\}b→b|[b\}

Пример результата совершения описанных шагов:

abb→|[?\}abb→a|[a\}bb

3. Головка уходит от стенки вправо и, неся в себе запомненный символ, перепрыгивает через буквы *a*, *b*, *A*, *B* (откуда возьмутся *A* и *B* будет сказано позже), стремясь достичь правого края слова. Эти действия задаются восемью формулами подстановок вида *[η\}ξ→ξ[η\}* (через *η* здесь обозначен запомненный символ *a* или *b*, а *ξ* обозначает перепрыгиваемую букву *a*, *b*, *A* или *B*). Пример результата совершения описанных шагов (при переносе первого копируемого символа «двойники» *A* и *B* еще не встречаются) является таким:

abb→|[?\}abb→a|[a\}bb→a|b[a\}b→a|bb[a\}

4. Достигнув конца слова, головка вписывает здесь символ-двойник того символа, который хранился у нее в памяти и, «честно выполнив свой копирующе-транспортирующий долг», исчезает. Реализующая эти действия пара формул подстановок такова:

[a\}→A

[b\}→B

Почему сейчас печатаются не сами копируемые символы, а их двойники? Дело в том, что нам нужно отличать уже выполненные копии от букв, еще только подлежащих копированию. Скопированные буквы «лежат» слева за стенкой, подлежа-

щие копированию буквы суть *a* и *b*, а уже выполненные копии легко узнаются по тому признаку, что они суть двойники. Пример результата выполнения описанных шагов:

abb→|[?}abb→a|[a}bb→a|b[a}b→a|bb[a}→a|bbA

5. Из стенки появляется новая запоминающая головка, чтобы посмотреть, есть ли еще символы, подлежащие копированию: *|→|[?}*. Эта формула подстановки должна находиться ниже всех других формул подстановок, содержащих изображение головки (то есть подслова *[?}*, *[a}* или *[b}*), чтобы она срабатывала тогда и только тогда, когда «отработанная» головка исчезает. Продолжение начатого ранее примера:

abb→|[?}abb→a|[a}bb→a|b[a}b→a|bb[a}→a|bbA→
→a|[?}bbA→ab|[b}bA→ab|b[b}A→ab|bA[b}→ab|bAB→
→ab|[?}bAB→abb|[b}AB→abb|A[b}B→abb|AB[b}→abb|ABB→
→abb|[?}ABB

6. Итак, в цикле все буквы, подлежавшие копированию, будут скопированы и переброшены через стенку влево. Копия исходного слова, записанная символами-двойниками, полностью готова и стоит справа. Вышедшая из стенки очередная головка *[?}* «наталкивается» на символ-двойник. Тогда стенка | исчезает, а «запоминающе-транспортирующая» головка *[?}* превращается в головку-переводчик *[Xx}*, которая, перепрыгивая через символ-двойник, превращает его в соответствующий символ исходного алфавита. Исчезновение стенки и метаморфоза головок описывается парой формул подстановок:

|[?}A→[Xx}A
|[?}B→[Xx}B

«Трудовая деятельность» головки-переводчика также описывается парой формул подстановок:

[Xx}A→a[Xx}
[Xx}B→b[Xx}

Продолжение начатого ранее примера:

abb→...→abb|[?}ABB→abb[Xx}ABB→abba[Xx}BB→abbab[Xx}B→
→abbabb[Xx}

7. Достигнув конца слова, головка-переводчик исчезает. Соответствующая формула подстановки *[Xx}→.* должна быть завершающей, чтобы остановить работу алгоритма. Завершение начатого выше примера:

abb→...→abbabb[Xx}→abbabb

8. Алгоритм, схема которого будет разработана по описанному выше плану, будет применим к любым непустым словам в алфавите $A = \{a, b\}$ и будет корректно их удваивать в соответствии с условием решаемой нами задачи. А если входное слово будет пустым? Тогда после первого шага сложится следующая ситуация:

Λ→|[?}

Чтобы в этом случае сразу же остановить работу, введем завершающую формулу подстановки *|[?}→.*, поместив ее после всех полезных для обработки непустых входных слов формул подстановок, содержащих в левой части *|[?}*. Теперь и в

отношении пустого входного слова алгоритм будет корректен, ведь удвоенное пустое слово есть слово пустое:

$\Lambda \rightarrow |[?\} \rightarrow \Lambda$

С учетом всего сказанного получаем следующий нормальный (марковский) алгоритм:

Γ: $|[?\}a \rightarrow a|[a\}$ (R_1)
$|[?\}b \rightarrow b|[b\}$ (R_2)
$[a\}a \rightarrow a[a\}$ (R_3)
$[a\}A \rightarrow A[a\}$ (R_4)
$[a\}b \rightarrow b[a\}$ (R_5)
$[a\}B \rightarrow B[a\}$ (R_6)
$[b\}a \rightarrow a[b\}$ (R_7)
$[b\}A \rightarrow A[b\}$ (R_8)
$[b\}b \rightarrow b[b\}$ (R_9)
$[b\}B \rightarrow B[b\}$ (R_{10})
$[a\} \rightarrow A$ (R_{11})
$[b\} \rightarrow B$ (R_{12})
$|[?\}A \rightarrow [Xx\}A$ (R_{13})
$|[?\}B \rightarrow [Xx\}B$ (R_{14})
$|[?\} \rightarrow .$ (R_{15})
$[Xx\}A \rightarrow a[Xx\}$ (R_{16})
$[Xx\}B \rightarrow b[Xx\}$ (R_{17})
$[Xx\} \rightarrow .$ (R_{18})
$| \rightarrow |[?\}$ (R_{19})
$\rightarrow |[?\}$ (R_{20})

Разработка схемы нормального (марковского) алгоритма: пример согласованной работы с различными частями слова. Входное слово P есть слово в алфавите $A = \{a, b\}$. Известно, что слово P имеет четную длину (0, 2, 4, ...). Нормальный (марковский) алгоритм B должен удалить правую половину этого слова. Например: $B(bbab) = bb$.

За основу решения задачи возьмем следующую идею: согласуем просмотр очередной буквы из левой половины с удалением буквы из правой половины. Для достижения этой цели предлагается изложенный ниже план действий.

1. Сначала формула подстановки $\rightarrow|$ приписывает к началу слова символ $|$. Дальше он будет играть роль «стенки», налево от которой перебрасываются буквы левой (остающейся) части слова. Другие формулы подстановок потом добавляются так, чтобы только что названная формула была в схеме алгоритма последней. Пример первого шага переработки непустого входного слова:

$bbab \rightarrow |bbab$

2. Стенка $|$ будет обладать следующим свойством: если она в слове есть, а «вопрошающей головки» $[?\}$ в слове нет, то стенка немедленно «рождает» справа от себя вопрошающую головку. Чтобы обеспечить это, формула подстановки $|\rightarrow|[?\}$ должна быть помещена в схему разрабатываемого алгоритма после всех других

формул подстановки, левые части которых содержат «стенку». А эти другие формулы должны «блокировать» данную формулу, пока в обрабатываемом слове есть вопрошающая головка. Для одновременного выполнения названных требований сделаем формулу подстановки *|→|[?}* предпоследней в схеме разрабатываемого алгоритма. Продолжение начатого выше примера:

bbab→|bbab→|[?}bbab

3. Вопрошающая головка определяет, имеется ли справа от нее буква. Если буква есть, то она перебрасывается через стенку влево, а вопрошающая головка превращается в «головку, бегущую вправо» *[>}*. Эти действия реализуются следующими двумя формулами подстановок:

|[?}a→a|[>}

|[?}b→b|[>}

Если справа от вопрошающей головки буквы не окажется (а это возможно, когда входное слово является пустым или когда все сохраняемые буквы уже переброшены за стенку влево, а все удаляемые буквы уже удалены), то стенка и вопрошающая головка исчезают, а работа алгоритма завершается. Эти действия обеспечиваются наличием следующей завершающей формулы подстановки:

|[?}→.

Продолжение начатого выше примера:

bbab→|bbab→|[?}bbab→b|[>}bab

4. Головка, бегущая вправо, перепрыгивает через все буквы, стремясь достичь конца слова, что гарантировано наличием таких двух формул подстановок:

[>}a→a[>}

[>}b→b[>}

Когда головка, бегущая вправо, достигает конца слова, она превращается в «затирающую головку» *{x]*, благодаря наличию формулы подстановки *[>}→{x]*. Начатый выше пример продолжается так:

bbab→|bbab→|[?}bbab→b|[>}bab→b|b[>}ab→b|ba[>}b→b|bab[>}→
→b|bab{x]

5. Затирающая головка уничтожает находящуюся слева от нее букву и сама тоже исчезает, что обеспечивается следующими формулами подстановок:

a{x]→

b{x]→

А теперь, когда все головки исчезли, разблокированной оказывается стенка, немедленно порождающая вопрошающую головку. Так в цикле все сохраняемые буквы перебрасываются влево через стенку, а все удаляемые буквы удаляются. Причем, переброс через стенку каждой буквы левой части слова согласован с удалением затирающей головкой одной буквы правой части слова. Вот продолжение начатого выше примера:

bbab→...→b|bab{x]→b|ba→b|[?}ba→bb|[>}a→bb|a[>}→bb|a{x]→
→bb|→bb|[?}

6. Теперь срабатывает завершающая формула подстановки *|[?}→.*, и работа завершается.

Таким образом, получается следующая схема нормального (марковского) алгоритма:

$B:$ $|[?}a\to a|[>\}$ (R_1)
$|[?}b\to b|[>\}$ (R_2)
$|[?}\to .$ (R_3)
$[>\}a\to a[>\}$ (R_4)
$[>\}b\to b[>\}$ (R_5)
$[>\}\to \{x]$ (R_6)
$a\{x]\to$ (R_7)
$b\{x]\to$ (R_8)
$|\to |[?\}$ (R_9)
$\to |$ (R_{10})

Корректность работы этого алгоритма на непустом входном слове уже фактически проверена при описании процесса конструирования. Проверим правильность его работы на пустом входном слове:

$B()\to R_{10}\to \perp \to R_9\to \underline{|[?\}}\to R_3\to \Lambda$

Результат вполне корректен: после удаления правой половины пустого слова (которая является пустым подсловом) остается левая половина, которая сама есть пустое слово.