Эквивалентные преобразования рекурсивных функций, описанных на языке РЕФАЛ*

В.Ф.Турчин

Киев-Алушта, 1972

Если наложить на язык РЕФАЛ [1] некоторые ограничения, то можно построить чрезвычайно мощную систему эквивалентных преобразований алгоритмов (точнее - рекурсивных функций), описанных на этом языке. Ограничения таковы:

1) Исключаются символы обмена и связанные с ними возможности менять состояние поля памяти рефал-машины в процессе ее работы,

2) Исключаются свободные переменные терма.

3) Исключаются рекурсивные переменные[1].

4) Запрещается использование открытых и повторных переменных выражения.

Определенный таким образом язык получил название *ограниченного рефала.* Не имея возможности в рамках настоящего доклада подробно описывать систему эквивалентных преобразований в ограниченном рефале, мы лишь сформулируем без доказательства основные теоремы и правила преобразований, после чего приведем пример их использования.

Для сокращения записи исключим малые латинские буквы из числа возможных объектных знаков и резервируем их для собственных знаков рефала. Вместо $\underline{\mathtt{K}}$, $\underline{\mathtt{S}}$, $\underline{\mathtt{E}}$ будем писать, соответственно, k, s, e. Идентификатор свободной переменной превратим в нижний индекс: s_1, s_2, e_1, e_a и т. д. (в индексе можно использовать и малые буквы). Предложения будем всегда начинать с новой строки, с абзацным отступом; знак $\underline{\S}$ опускается. Правая функциональная скобка $\underline{.}$ (подчеркнутая точка) стилизуется в знак $\perp$. В качестве метасимволов для изображения рефал-объектов будем использовать рукописные заглавные латинские буквы.

*В.Ф.Турчин, Эквивалентные преобразования рекурсивных функций, описанных на языке РЕФАЛ. В сб.: Труды симпозиума “Теория языков и методы построения систем программирования”, Киев-Алушта: 1972. Стр. 31-42.

[1]Из соображений экономии места мы описываем упрощенный вариант системы эквивалентных преобразований. В полном варианте разрешается использовать некоторые рекурсивные переменные символа.

1 Основные понятия

Типовое выражение называется L-*выражением*, если:

1) ни одно его подвыражение не содержит более чем одну свободную переменную выражения, не входящую в скобки,

2) ни одна свободная переменная выражения не входит в него более одного раза.

Примеры L-выражений:

$$ABC$$

$$'RS'\,(s_1\,e_2\,(A\,e_3\,B))\,s_1$$

Примеры типовых выражений, не являющихся L -выражениями:

$$e_1\;+\;e_2$$

$$s_1\,e_2\,(s_3\,e_2)$$

В ограниченном рефале левые части предложений могут быть только L-выражениями.

Теорема 1 . *Каждое подвыражение L-выражения является L-выражением.*

Теорема 2 . *Пусть $\mathcal{E}_l$ - L-выражение, которое при замене свободных переменных на некоторые значения переходит в объектное выражение $\mathcal{E}_0$. Тогда не существует другого набора значений, используя который можно перевести $\mathcal{E}_l$ в $\mathcal{E}_0$.*

Каждому типовому выражению $\mathcal{E}_t$ можно сопоставить множество всех тех объектных выражений, которые могут быть отождествлены как $\mathcal{E}_t$. Это множество мы назовем *классом*, изображаемым выражением $\mathcal{E}_t$, или просто, классом $\mathcal{E}_t$. Класс, изображаемый L -выражением, назовем L -классом. Отношения включения и равенства между множествами объектных выражений будем записывать с помощью знаков $\subset$ и $=$, а операции пересечения и соединения - с помощью знаков $\cap$ и $\cup$. Знак $\emptyset$ будет изображать пустое множество. Объектное выражение, (рассматриваемое как частный случай типового) изображает класс, состоящий из единственного элемента. Запись $\mathcal{E}_0 \subset \mathcal{E}_t$, где $\mathcal{E}_0$ - объектное выражение, означает отождествимость $\mathcal{E}_0$ как $\mathcal{E}_t$.

Обобщим понятие подстановки на случай, когда значениями переменных могут быть типовые выражения. Будем называть *правильной* подстановкой в $\mathcal{E}_t$ замену всех свободных переменных, входящих в $\mathcal{E}_t$ на некоторые выражения, называемые их *значениями*, при соблюдении следующих правил:

1) Значением свободной переменной символа может быть только символ или свободная переменная символа.

2) Значения всех вхождений одной переменной должны совпадать.

3) Совокупность всех значений не должна содержать двух переменных с одинаковыми идентификаторами, но разными указателями.

Результат применения правильной подстановки Δ к выражению $\mathcal{E}$ будем изображать в виде $\Delta//\mathcal{E}$.

Теорема 3 . *Если $\mathcal{E}_t$ - типовое выражение, а Δ - правильная подстановка, то $\Delta//\mathcal{E}_t \subset \mathcal{E}_t$.*

Класс $\mathcal{E}'_t = \Delta//\mathcal{E}_t$ будем называть *подклассом* класса $\mathcal{E}_t$, или его *сужением*. Процедуру выделения подкласса из класса $\mathcal{E}_t$ мы также будем называть сужением $\mathcal{E}_t$. Заметим, что подкласс L -класса не является, вообще говоря, L-классом. Так, любой класс есть подкласс L-класса e_1.

2 Обобщенный алгоритм проектирования

Пусть $\mathcal{E}_l$ - L-выражение, а $\mathcal{E}_t$ - произвольное типовое выражение. Поставим, задачу отыскания пересечения $\mathcal{E}_l \cap \mathcal{E}_t$. Очевидно, это есть обобщение задачи отождествления. Если $\mathcal{E}_t$ - объектное выражение, то ответ находится путем проектирования $\mathcal{E}_l$ на $\mathcal{E}_t$ (см. [1]), которое в данном случае (вследствие ограничений, наложенных на язык) производится весьма просто. Если $\mathcal{E}_t$ содержит свободные переменные, необходимо обобщение алгоритма проектирования, которое мы и опишем ниже. Суть его состоит в том, что $\mathcal{E}_t$ теперь представляет *множество* объектных выражений, и в процессе проектирования мы будем, если потребуется, сужать это множество, выделяя из него те подмножества, на которые проектирование $\mathcal{E}_l$ возможно.

Результатам проектирования $\mathcal{E}_l$ на $\mathcal{E}_t$ является совокупность r подстановок $\Delta^t_1, \ldots, \Delta^t_r$ в выражении $\mathcal{E}_t$, таких что $\Delta^t_i//\mathcal{E}_t \subset \mathcal{E}_l$. Эти r сужений $\mathcal{E}_t$ исчерпывают пересечение $\mathcal{E}_l$ и $\mathcal{E}_t$, то есть

$$\mathcal{E}_l \cap \mathcal{E}_t = \Delta^t_1//\mathcal{E}_t \cup \ldots \cup \Delta^t_r//\mathcal{E}_t$$

В частности, если $r = 0$, то $\mathcal{E}_l \cap \mathcal{E}_t = \emptyset$.

Каждое сужение Δ^t_i формируется путем ряда последовательных δ-сужений, затрагивающих не более одной переменной. Мы будем записывать δ-сужения с помощью стрелки, например: $e_x \to A\,e_x$. При этом остальные переменные переходят, сами в себя. В состав Δ^t-сужения может войти несколько δ-сужений, затрагивающих одну и ту же переменную. Так как на некоторых шагах алгоритма представляются два возможных δ-сужения, порождение Δ^t-сужений есть процесс с ветвлением, который и приводит в конечном счете к появлению r последовательностей δ-сужений, имеющих общее начало и успешно доведенных до точки "отождествление закончено".

Каждой подстановке Δ_i^t ставится в соответствие подстановка Δ_i^l в $\mathcal{E}_l$, такая что

$$\Delta_i^l//\mathcal{E}_l = \Delta_i^t//\mathcal{E}_t;$$

иначе говоря, Δ_i^l дает те значения свободных переменных, которые им надо присвоить для отождествления $\Delta_i^t//\mathcal{E}_t$ как $\mathcal{E}_l$. Эти подстановки также формируются последовательно, и мы будем записывать их с помощью стрелки, направленной влево; например, $e_x \leftarrow s_1\, e_y\, A$, что означает: переменной e_x из $\mathcal{E}_l$ присваивается значение $s_1\, e_y\, A$, где s_1 и e_y - свободные переменные из $\Delta_i^t//\mathcal{E}_t$.

Алгоритм проектирования определяется приводимыми ниже правилами P1-P4. (Идентификаторы x, y и z надо понимать в дальнейшем как произвольные идентификаторы свободных переменных).

P1. Если $\mathcal{E}_l$ пусто, то отождествление возможно, если $\mathcal{E}_t$ либо пусто, либо есть последовательность свободных переменных выражения $e_x \ldots e_z$. В последнем случае производится сужение

$$e_x \to <\text{пусто}>, \ldots, e_z \to <\text{пусто}>$$

P2. Если $\mathcal{E}_l$ есть e_x, то отождествление успешно заканчивается присваиванием $e_x \leftarrow \mathcal{E}_t$.

P3. Пусть $\mathcal{E}_l$ имеет вид $\mathcal{T}_l\mathcal{E}_l'$, где $\mathcal{T}_l$ - терм, не являющийся свободной переменной выражения. Если $\mathcal{E}_t$ пусто, отождествление невозможно. В противном случае представим его в виде $\mathcal{T}_t\mathcal{E}_t'$, где $\mathcal{T}_t$ - терм. Затем производится проектирование $\mathcal{T}_l$ на $\mathcal{T}_t$ по правилам, описанным ниже. Если оно невозможно, то невозможно и проектирование $\mathcal{E}_l$ на $\mathcal{E}_t$. Если оно возможно, то учитываются сужения и присваивания, сделанные при проектировании $\mathcal{T}_l$ на $\mathcal{T}_t$ и продолжается проектирование $\mathcal{E}_l'$ на $\mathcal{E}_t'$. Если образовалось более одной ветви, то каждая из них обрабатывается независимо.

Правила проектирования термов таковы (в первых фразах сокращенно записано условие):

P3.1 $\mathcal{T}_l = \mathcal{S}$, $\mathcal{T}_t = \mathcal{S}$. ($\mathcal{S}$ - произвольный символ). Никаких действий.

P3.2 $\mathcal{T}_l = \mathcal{S}$, $\mathcal{T}_t = s_x$. $s_x \to \mathcal{S}$.

P3.3 $\mathcal{T}_l = \mathcal{S}$, $\mathcal{T}_t = e_x$. Ветвление

$$e_x \to <\text{пусто}>$$

$$e_x \to \mathcal{S}e_x$$

P3.4 $\mathcal{T}_l = s_y$, $\mathcal{T}_t = \mathcal{S}$. $s_y \leftarrow \mathcal{S}$.

P3.5 $\mathcal{T}_l = s_y$, $\mathcal{T}_t = s_x$. $s_y \leftarrow s_x$. Кроме того, в $\mathcal{E}_l'$ производим замену s_y на a_x (читать: *чужая* переменная s_x).

P3.6 $\mathcal{T}_l = s_y$, $\mathcal{T}_t = e_x$. Ветвление:

$$e_x \to <\text{пусто}>$$

$$e_x \to s_z e_x$$

Идентификатор z должен быть отличен от всех остальных идентификаторов (*новый* идентификатор).

P3.7 $\mathcal{T}_l = (\mathcal{E}_l^0)$, $\mathcal{T}_t = (\mathcal{E}_t^0)$. Проектируется $\mathcal{E}_l^0$ на $\mathcal{E}_t^0$.

P3.8 $\mathcal{T}_l = (\mathcal{E}_l^0)$, $\mathcal{T}_t = e_x$. Ветвление:

$$e_x \to <\text{пусто}>$$

$$e_x \to (e_z)e_x$$

(Идентификатор z - новый).

P3.9 $\mathcal{T}_l = a_x$, $\mathcal{T}_t = \mathcal{S}$. $s_x \to \mathcal{S}$, $a_x \to \mathcal{S}$.

P3.10 $\mathcal{T}_l = a_x$, $\mathcal{T}_t = s_y$. $s_y \to s_x$.

P3.11 $\mathcal{T}_l = a_x$, $\mathcal{T}_t = e_y$. Ветвление:

$$e_y \to <\text{пусто}>$$

$$e_y \to s_x e_y$$

В остальных случаях проектирование $\mathcal{T}_l$ на $\mathcal{T}_t$ невозможно.

P4. Пусть $\mathcal{E}_l$ имеет вид $e_x \mathcal{E}_l' \mathcal{T}_l$, Если $\mathcal{E}_t$ пусто, отождествление невозможно. В противном случае представим $\mathcal{E}_t$ в виде $\mathcal{E}_t' \mathcal{T}_l$ и будем поступать аналогично пункту P3.

Подстановки Δ_i^t, которые получаются в результате применения алгоритма проектирования, назовем *коноригинальными*.

Теорема 4 . *Пусть Δ_i - коноригинальные подстановки, а $\mathcal{E}$ - некоторое выражение. Тогда $\Delta_i // \mathcal{E}$ - попарно непересекающиеся L-классы.*

Следствие. Пересечение двух классов, построенное с помощью алгоритма проектирования, является объединением попарно непересекающихся L-классов.

3 Правила эквивалентных преобразований

Функцию $\mathcal{F}'$ назовем эквивалентной функции $\mathcal{F}$, если при всяком $\mathcal{E}$, при котором существует результат конкретизации $k\mathcal{F}\,\mathcal{E}\bot$, существует и совпадает с ним результат конкретизации $k\mathcal{F}'\,\mathcal{E}\bot$. Последовательность промежуточных - шаг за шагом - выражений при конкретизации $k\mathcal{F}'\,\mathcal{E}\bot$ не обязана, вообще говоря, совпадать с аналогичной последовательностью при конкретизации $k\mathcal{F}\,\mathcal{E}\bot$; если же она совпадает, мы будем говорить, что не только *функция* $\mathcal{F}'$, но и определяющий ее *алгоритм* (набор предложений) эквивалентен алгоритму, определяющему функцию $\mathcal{F}$. Сформулируем правила эквивалентных преобразований алгоритмов (EAl - EA5) и собственно функций (EF1 - EF2).

EA1. К алгоритму можно приписать (в конце) любое предложение.

EA2. Пусть два предложения с левыми частями $\mathcal{L}_1$ и $\mathcal{L}_2$, такими что $\mathcal{L}_1 \cap \mathcal{L}_2 = \emptyset$, стоят в алгоритме рядом. Тогда их можно поменять местами.

EA3. Пусть предложение с левой частью $\mathcal{L}_1$ предшествует предложению с левой частью $\mathcal{L}_2$, такой что $\mathcal{L}_2 \subset \mathcal{L}_1$. Тогда второе предложение можно удалить.

EA4. Пусть в алгоритме стоят рядом два предложения:

$k\mathcal{L}_1 \sim \mathcal{R}_1$

$k\mathcal{L}_2 \sim \mathcal{R}_2$

причем $\mathcal{L}_1 \subset \mathcal{L}_2$, и если в $\mathcal{R}_2$ заменить все свободные переменные теми значениями, которые они принимают при отождествлении $\mathcal{L}_1$ как $\mathcal{L}_2$, то $\mathcal{R}_2$ совпадет с $\mathcal{R}_1$. Тогда первое предложение можно удалить.

EA5. Пусть алгоритм содержит предложение

$k\mathcal{L} \sim \mathcal{R}$

и пусть $\mathcal{L}_1 = \Delta//\mathcal{L}$ есть L -класс, являющийся сужением $\mathcal{L}$ путем правильной подстановки Δ. Тогда указанное предложение можно заменить на пару:

$k\mathcal{L}_1 \sim \Delta//\mathcal{R}$

$k\mathcal{L} \sim \mathcal{R}$

EF1. Пусть одно из предложений, описывающих функцию $\mathcal{F}$, имеет вид

(*) $\quad k\mathcal{L}_f \sim \mathcal{C}_1\ k\mathcal{G}\,\mathcal{E}_t\perp\,\mathcal{C}_2$

где $\mathcal{E}_t$ - типовое выражение, $\mathcal{C}_1$ и $\mathcal{C}_2$ - последовательности знаков, а $\mathcal{G}$ - функция с описанием

$k\mathcal{G}\,\mathcal{L}_g^1 \sim \mathcal{R}_g^1$

$\ldots$

$k\mathcal{G}\,\mathcal{L}_g^n \sim \mathcal{R}_g^n$

С помощью алгоритма проектирования, найдем n наборов конооригинальных подстановок $\{\Delta_1^i, \ldots, \Delta_{r_n}^i\}$, определяющих n пересечений

$$\mathcal{E}_t \cap \mathcal{L}_g^i = \bigcup_{j=1}^{r_i} \Delta_j^i//\mathcal{E}_t, \quad i = 1, \ldots, n$$

Предложение $(*)$ можно заменить на $r_1 + \ldots + r_n$ предложений, расположенных в порядке возрастания индекса i.

$k\,\Delta_j^i//\mathcal{L}_f \sim \{\Delta_j^i//\mathcal{C}_1\}\,\mathcal{R}_g^{ij}\,\{\Delta_j^i//\mathcal{C}_2\}$

где $\mathcal{R}_g^{ij}$ получается из $\mathcal{R}_g^i$ заменой свободных переменных на те значения, которые они принимают при отождествлении $\Delta_j^i//\mathcal{E}$ как $\mathcal{L}_g^i$. (Фигурные скобки ограничивают область действия знака $//$.)

Заметим, что если, для какого-то i, $\mathcal{E}_t \cap \mathcal{L}_g^i = \mathcal{E}_t$, то есть $\mathcal{E}_t \subset \mathcal{L}_g^i$, то по правилу EA3 остальные предложения можно не выписывать.

EF2. Пусть одно из предложений, описывающих функцию $\mathcal{F}$, имеет вид

$$k\mathcal{F}\,\mathcal{L}_f \sim \mathcal{C}_1\; k\mathcal{G}\,\mathcal{E}\perp\, \mathcal{C}_2$$

где $\mathcal{E}$ - произвольное (общее) выражение. Заменим в выражении $\mathcal{E}$ все термы, начинающиеся со знака k, на различные *новые* свободные переменные выражения, которые назовем *неразменными* переменными. Если предложение, видоизмененное таким образом, допускает применение правила EF1 с тем ограничением, что не требуется сужений, затрагивающих неразменные переменные, то исходное предложение может быть подвергнуто аналогичному преобразованию. Для этого надо преобразовать видоизмененное предложение, а затем снова заменить неразменные переменные на те термы, которые они представляют (произведя в них необходимые подстановки).

4 Пример эквивалентного преобразования

Определим функцию сложения двоичных чисел (детерминатив $+$):

$$k + (e_a)() \sim e_a$$

$$k + ()(e_b) \sim e_b$$

$$k + (e_a s_1)(e_b 0) \sim k + (e_a)(e_b)\perp s_1$$

$$k + (e_a 0)(e_b 1) \sim k + (e_a)(e_b)\perp 1$$

$$k + (e_a 1)(e_b 1) \sim k + (k + (e_a)(1)\perp)(e_b)\perp 0$$

Поставим вопрос: каков должен быть второй аргумент, чтобы при первом аргументе 101 сумма была равно 1010? Чтобы ответить на него путем формального преобразования, введем предикат равенства, который на ограниченном рефале описывается так:

$$k = ()() \sim \text{И}$$

$$k = (e_a s_1)(e_b s_1) \sim k = (e_a)(e_b)\perp$$

$$k = e_x \sim \text{Л}$$

и предикат ω, который принимает значение И при тех и только тех аргументах, которые удовлетворяют поставленному условию:

$$k\,\omega\, e_x \sim k = (k + (101)(e_x)\perp)(1010)\perp$$

Подвергнем это предложение эквивалентному преобразованию. Сначала применим EF1 с термом $k + (101)(e_x)\bot$ в качестве выделенного терма $k\mathcal{G}\,\mathcal{E}_t\bot$. Следовательно, в первую очередь надо найти пересечение $(101)(e_x)$ и левой части $\S + .1$, то есть, $+(e_a)()$. В процессе проектирования применяем следующие правила (в квадратных скобках указывается результат применения правила): P3.1, P3.7, P2 [$e_a \leftarrow 101$], P3.7, P1 [$e_x \to$ <пусто>]. Итак, мы получили одно сужение Δ_1^1, определяемое единственным δ-сужением $e_x \to$ <пусто>. По правилу EF1 записываем предложение

$$k\,\omega \sim k = (101)(1010)\bot$$

Пересечение $+(101)(e_x)$ и левой части $\S + .2$ дает пустое множество. Пересечение с левой частью $\S + .3$, то есть $+(e_a s_1)(e_b 0)$, дает один подкласс, определяемый сужением $e_x \to e_x 0$, откуда по правилу EF1 появляется еще одно предложение:

$$k\,\omega\, e_x 0 \sim k = (k + (10)(e_x)\bot 1)(1010)\bot$$

Пересечение с левой частью $\S + .4$ дает пустое множество, а для $\S + .5$ получаем одно сужение $e_x \to e_x 1$, что дает третье (и последнее) предложение:

$$k\,\omega\, e_x 1 \sim k = (k + (k + (10)(1)\bot)(e_x)\bot 0)(1010)\bot$$

Этим применение EF1 заканчивается. Теперь преобразуем каждое из трех полученных предложений. Для первого сразу получаем

$$k\,\omega \sim \text{Л}$$

Во втором предложении объявим по правилу EF2 терм

$$k + (10)(e_x)\bot$$

неразменной переменной и воспользуемся определением равенства. Получаем:

$$k\,\omega\, e_x 0 \sim \text{Л}$$

Точно так же, используя EF2 и EF1, преобразуем третье предложение

$$k\,\omega\, e_x 1 \sim k = (k + (11)(e_x)\bot)(101)\bot$$

Теперь по правилу EA1 допишем в конце четвертое предложение:

$$k\,\omega\, e_x \sim \text{Л}$$

По правилу EA2, примененному дважды, переставим третье предложение на первое место, а два предложения с правой частью Л поглотим по правилу EA4 последним предложением. В результате получаем описание ω из двух предложений:

$$k\,\omega\,e_x 1 \sim k = (k + (11)(e_x)\perp)(101)\perp$$

$$k\,\omega\,e_x \sim \text{Л}$$

Продолжая таким образом преобразования, получим в конце концов:

$$k\,\omega\,101 \sim \text{И}$$

$$k\,\omega\,0101 \sim \text{Л}$$

$$k\,\omega\,e_x \sim \text{Л}$$

5 Алгоритм обращения алгоритмов

В приведенном выше примере, применение правил эквивалентных преобразований рефал-программы свелось, в сущности, к двоичному вычитанию столбиком. Стратегия, которую мы использовали при выборе правила и объекта, к которому оно применяется, была весьма проста и естественна; ее легко сформулировать в виде алгоритма. В результате мы получаем определенный алгоритм преобразования алгоритмов, описанных на рефале. Этот алгоритм можно описать (и он, действительно, описан) на рефале же (ограниченном). Пусть α - детерминатив соответствующей рекурсивной функции. Аргументом функции α является метакодовая запись некоторого набора предложений. Договоримся, что функция α всегда преобразует функцию (точнее, метакодовую запись алгоритма, определяющую эту функцию) с детерминативом $'OMEGA'$. Рассмотрим выражение:

$$k\,\alpha !\text{П}!K('OMEGA'!EX)!K(=(!K(+(101)(!EX)))(1010))\ldots\perp$$

Здесь непосредственно за α следует метакодовая запись предложения, определяющего функцию $'OMEGA'$. Мы не приводим описания метакода; его можно восстановить, сравнивая метакод с оригиналом на стр. 7 настоящего доклада (предложение, определяющее функцию ω). Многоточием мы заменили метакод описания функций $+$ и $=$, который нужен для функции α, но не меняется в процессе конкретизации. В результате выполнения конкретизации мы получим преобразованное описание функции $'OMEGA'$ в метакоде, то есть:

$$!\text{П}!K('OMEGA'101)\text{И}!\text{П}!K('OMEGA'0101)\text{И}$$

$$!\text{П}!K('OMEGA'!EX)\text{Л}\ldots$$

Если обозначим через β несложную функцию, которая из метакода описания функции $'OMEGA'$ извлекает первый аргумент первого предложения при условии, что правая часть есть И, то конкретизация

$$k\,\beta\,k\,\alpha\,\mathcal{E}_\omega\perp\perp$$

где $\mathcal{E}_\omega$ - тот же аргумент, что и выше, даст 101, то есть разность между 1010 и 101 - числами, входящими в аргумент.

Введем теперь функцию γ предложением:

$$k\,\gamma(e_a)(e_b) \sim k\,\beta\,k\,\alpha\,!\Pi!K('OMEGA'\,!EX)$$
$$!K(=(!K(+(e_b)(!EX)))(e_a))\ldots\perp\perp$$

Как легко понять, эта функция осуществляет двоичное вычитание из второго аргумента первого. Комбинация функций $k\,\beta\,k\,\alpha\ldots\perp\perp$ служит в качестве универсального решающего алгоритма. Для получения обратного алгоритма надо только в аргумент α ввести описание прямого алгоритма вместе с описанием предиката равенства и предиката ω, который фиксирует точную постановку задачи. В частности, алгоритм сложения столбиком в позиционной системе счисления преобразуется в алгоритм вычитания. Подчеркнем, что этот алгоритм *эффективен* (вычитание столбиком): он не требует экспоненциального перебора (универсальный алгоритм с экспоненциальным перебором написать нетрудно), а требует числа действий, пропорционального числу цифр в аргументе. Вопрос о том, при каких условиях описанный алгоритм (или его усиленный вариант) решает задачу без экспоненциального перебора, выходит за рамки настоящего доклада. Заметим только, что при обращении алгоритма умножения столбиком, мы также получаем эффективный алгоритм деления, и что класс алгоритмов, допускающих эффективное обращение этим методом, весьма широк.

ЛИТЕРАТУРА

1. В.Ф.Турчин, Программирование на языке РЕФАЛ. Препринты ИПМ АН СССР №№ 41, 43, 44, 48, 49 (1971).