Тишин В.В.

Алгоритмически неразрешимые проблемы.

Рассмотрим следующую проблему: существует ли алгоритм, который для любой машины Тьюринга T и произвольного слова B выясняет вопрос: применима машина T к слову B или нет? Сформулируем *проблему применимости*:

Проблема применимости:

Существует ли машина Тьюринга L, применимая ко всем словам вида $N(T)\lambda B$, где $N(T)$ - код произвольной машины Тьюринга T, B – произвольное слово, такая, что в случае, если T применима к слову B, то заключительная конфигурация машины L имеет вид $\underset{s_0}{1}$, а если T не применима к слову B, то заключительная конфигурация машины L имеет вид $\underset{s_0}{\lambda}$?

Теорема об алгоритмической неразрешимости проблемы применимости.

Не существует машины Тьюринга L, решающей проблему применимости.

Доказательство.

1. Решим вспомогательную задачу: построить машину Тьюринга T_1, применимую ко всем словам вида $x_1x_2...x_n, \quad x_i \in \{a,b\}, \; i \in \{1,2,...,n\}$, преобразующие их в слово $x_1x_2...x_n\lambda x_1x_2...x_n$, по окончании работы переводящая считывающее устройство машины Тьюринга T_1 на начало полученного слова.

Можно предложить следующее решение этой задачи. Пусть дано некоторое слово $x_1x_2...x_n$ в алфавите $\{a,b\}$. Работа машины T_1 будет состоять из n циклов. К началу i – го цикла конфигурация такова, что продублировано i -1 букв исходного слова и считывающее устройство рассматривает букву x_i:

$$x_1x_2...x_{i-1}\underset{s_1}{x_i}\,x_{i+1}...x_n\lambda x_1x_2...x_{i-1}$$

Очередной цикл будет проходить в три этапа:

I – запоминание буквы (при этом она отмечается);

II – поиск правой крайней пустой клетки, в которую записывается x_i;

III – возвращение считывающей головки машины влево к помеченной букве, стирание метки, сдвиг головки на одну клетку вправо.

Рассмотрим два случая.

1) $x_i = a$.

Первый этап осуществляется следующими командами:

$as_1 \to a' П s_2$ (a'- помеченная буква *a*, s_2 - состояние запоминания *a*).

Поиск крайней правой пустой клетки обеспечивается командами:

$bs_2 \to bПs_2$; $as_2 \to aПs_2$; $\lambda s_2 \to \lambda Пs_3$; $bs_3 \to bПs_3$; $as_3 \to aПs_3$.

Запись в пустую ячейку символа *a*: $\lambda s_3 \to aЛs_4$.

Возвращение влево к метке a': $as_4 \to aЛs_4$; $bs_4 \to bЛs_4$; $\lambda s_4 \to \lambda Лs_4$.

Стирание метки и сдвиг головки вправо: $a's_4 \to aПs_1$.

2) $x_i = b$.

Первый этап осуществляется следующими командами:

$bs_1 \to b' П s_5$ (b'- помеченная буква *b*, s_5 - состояние запоминания *b*).

Поиск крайней правой пустой клетки:

$bs_5 \to bПs_5$; $as_5 \to aПs_5$; $\lambda s_5 \to \lambda Пs_6$; $bs_6 \to bПs_6$; $as_6 \to aПs_6$.

Запись в пустую ячейку символа *b*: $\lambda s_6 \to bЛs_4$.

Возвращение влево к метке b': $as_4 \to aЛs_4$; $bs_4 \to bЛs_4$; $\lambda s_4 \to \lambda Лs_4$.

Стирание метки и сдвиг головки вправо: $b's_4 \to bПs_1$.

Если при сдвиге вправо головка при внутреннем состоянии s_1 встретит символ λ - это означает, что всё слово уже продублирова-

но и нужно двигаться на его начало: $\lambda s_1 \to \lambda Л s_7$; $a s_7 \to a Л s_7$; $a s_7 \to a Л s_7$.

Зависание над пустой ячейкой в состоянии s_7 будет означать, что считывающее устройство дошло до начало полученного слова. Сдвигаем головку вправо и останавливаем работу. $\lambda s_7 \to \lambda П s_0$.

Запишем программу машины T_1:

	s_1	s_2	s_3	s_4	s_5	s_6	s_7
λ	$\lambda Л s_7$	$\lambda П s_3$	$a Л s_4$	$\lambda Л s_4$	$\lambda П s_6$	$b Л s_4$	$\lambda П s_0$
a	$a' П s_2$	$a П s_2$	$a П s_3$	$a Л s_4$	$a П s_5$	$a П s_6$	$a Л s_7$
b	$b' П s_5$	$b П s_2$	$b П s_3$	$b Л s_4$	$b П s_5$	$b П s_6$	$b Л s_7$
a'	-	-	-	$a П s_1$	-	-	-
b'	-	-	-	$b П s_1$	-	-	-

В клетки таблицы, помеченные прочерком, можно писать любые команды, так как до исполнения этих команд дело никогда не дойдёт.

Задание для самостоятельной работы: проверить работу машины Тьюринга T_1 над словом *aba*.

2. Докажем теорему об алгоритмической неразрешимости проблемы применимости от противного.

Будем считать, что в алфавите машины T_1, $\{a,b\}=\{*,1\}$. Допустим, что существует машина Тьюринга L, решающая проблему применимости. Построим на базе этой машины новую машину Тьюринга L' следующим образом: начальным состоянием машины L' объявляется начальное состояние машины T_1, все внутренние состояния и команды машин T_1 и L объявляются также внутренними состояниями и командами L', заключительное состояние машины T_1 отождествляется с начальным состоянием машины L, заключительное состояние машины L объявляется заключительным состоянием машины L'.

Рассмотрим произвольную машину Тьюринга T и запишем на ленте её код $N(T)$. Запустим машину L'. Сначала, применяя команды машины T_1, будет продублирован через пробел код $N(T$ получено слово $N(T)\lambda N(T)$, затем будут исполняться команды машины L, и, в зависимости от T, возможны случаи:

а) Машина T самоприменима.

Тогда T применима к слову $N(T)$ и, выполняя команды машины L, в заключительном состоянии получаем конфигурацию $\underset{s_0}{1}$.

б) Машина T несамоприменима.

Тогда T не применима к слову $N(T)$ и, выполняя команды машины L, в заключительном состоянии получаем конфигурацию $\underset{s_0}{\lambda}$.

Таким образом, построенная машина L' решает проблему самоприменимости. Но, согласно теореме об алгоритмической неразрешимости проблемы самоприменимости, это невозможно. Полученное противоречие показывает, что сделанное допущение о существовании машины Тьюринга L, решающей проблему применимости, не существует. Теорема доказана. ■

Нормальные алгоритмы.

Алфавитом будем называть множество из конечного числа различных символов, называемых *буквами*.

Например, алфавитами являются {0;1;2}, {&;^;%;$;$a$}.

Словом в данном алфавите называется горизонтальный ряд конечного числа букв данного алфавита.

Например, словами в алфавите $\{a;b\}$ являются $a, abba, bababb$.

Пустым словом называется слово, не содержащее букв, и будем считать, что любой алфавит содержит пустое слово.

Будем говорить, что *слово* P *входит в слово* Q, если Q имеет вид $Q = Q_1 P Q_2$. (Может быть, или Q_1, или Q_2 являются пустыми словами.

Примеры. Пусть в алфавите $\{a,b\}$ имеем слово $Q=ababbabb$. Тогда $P=aba$ входит в слово Q, так как $Q=Pbbabb$. Слово $R=bab$ также входит в слово Q, так как $Q=aRbabb=ababRb$. Заметим, что в этом примере слово R входит в слово Q не единственным способом.

В дальнейшем будем находить первые вхождения данных слов в другие слова и заменять их на другие слова.

Пусть A – алфавит, не содержащий в качестве букв символов $\cdot$ и $\rightarrow$. *Обычной формулой подстановки* в алфавите A называется слово $P\rightarrow Q$, где P и Q- некоторые слова в алфавите A.

Заключительной формулой подстановки в алфавите A называется слово $P\rightarrow .Q$, где P и Q- слова в алфавите A.

P называется *левой частью* формулы подстановки $P\rightarrow aQ$, а Q- её *правой частью*, где a есть . или пустое слово.

Нормальной схемой подстановок в алфавите A называется конечная упорядоченная система подстановок $S=\begin{cases} P_1\rightarrow a_1Q_1 \\ P_2\rightarrow a_2Q_2 \\ \ldots\ldots\ldots \\ P_k\rightarrow a_kQ_k \end{cases}$, где a_i есть точка . или пустое слово, Q_i и P_i - слова в алфавите A, $i-1,2,...,k$.

Нормальным алгоритмом над алфавитом A называется пара (B,S), где B – алфавит, включающий в себя алфавит A и не содержащий букв . и $\rightarrow$, а S - нормальная схема подстановок над алфавитом B.

Нормальный алгоритм над алфавитом A преобразует слова в A следующим образом.

Работа данного нормального алгоритма над словом R состоит из отдельных шагов, в результате которых получается последовательность слов $R=R_1,R_2,R_3,...,R_i,R_{i+1},...$ **(1)**

Слово R_{i+1} получается из слова R_i следующим образом.

Просматривается нормальная схема подстановок и из неё выбирается самая верхняя формула, в левую часть которой входит слово R_i. Пусть это формула $P_j \to a_j Q_j$, где a_j - либо . либо пустое слово. Затем первое вхождение P_j в слово R_i заменяется на слово Q_j, что и даёт слово R_{i+1}.

Работа нормального алгоритма над словом R *заканчивается* в двух случаях:

1) существует такое слово R_i из последовательности (1), что слово R_{i+1} получается из слова R_i с помощью заключительной формулы подстановки;

2) существует такое слово R_j из последовательности (1), что ни одна левая часть формул подстановок из нормальной схемы не имеет вхождений в слово R_j.

В первом случае результатом работы нормального алгоритма над словом R объявляется слово R_{i+1}, во втором - R_j.

В этих случаях говорят, что данный нормальный алгоритм *применим к слову* R.

В остальных случаях работа нормального алгоритма над словом R не заканчивается (последовательность (1) будет бесконечной), тогда будем говорить, что данный нормальный алгоритм *неприменим к слову*.

Примеры.

1) Построить тождественный нормальный алгоритм, т.е. алгоритм, применимый к каждому слову в любом алфавите A и результатом работы которого было бы то же самое слово.

Такой нормальный алгоритм может быть задан нормальной схемой $\{\to .$, т.е. в этой схеме имеется только одна формула подстановки, являющаяся заключительной, с пустой левой и правой частью.

2) Построить нормальный алгоритм, применимый ко всем словам вида $x_1x_2...x_n$ в алфавите $\{a,b\}$ и преобразующий их в слово

$$\alpha = \begin{cases} x_n, \text{если } x_{n-1} = a \\ b^{n-1}x_n, \text{если } x_{n-1} = b, \quad n > 1. \end{cases}$$

Последней в схеме подстановок запишем формулу $\to \alpha$, тогда слово $x_1x_2...x_n$ перейдёт в $\alpha x_1x_2...x_n$ Затем с помощью формул подстановок $\alpha a \to a\alpha,\ \alpha b \to b\alpha$ символ α прейдёт на конец слова: $x_1x_2...x_n\alpha$

Для разбора варианта $x_{n-1}=a$ введём формулы подстановок $ab\alpha \to \beta b,\ aa\alpha \to \beta a,\ a\beta \to \beta,\ \beta \to .$

Для разбора случая $x_{n-1}=b$ введём формулы подстановок

$bb\alpha \to \gamma bb,\ ba\alpha \to \gamma ba,\ a\gamma \to \gamma b,\ b\gamma \to \gamma b,\ \gamma \to .$

Запишем нормальную схему подстановок:

$$\begin{cases} \alpha a \to a\alpha \\ \alpha b \to b\alpha \\ a\,b\alpha \to \beta b \\ a\,a\alpha \to \beta a \\ a\beta \to \beta \\ \beta \to . \\ b\,b\alpha \to \gamma b\,b \\ b\,a\alpha \to \gamma b\,a \\ a\gamma \to \gamma b \\ b\gamma \to \gamma b \\ \gamma \to . \\ \to \alpha \end{cases}$$

Проверим работу построенного нормального алгоритма над словом $abba$:

$\_abba,\ \underline{\alpha a}bba,\ a\underline{\alpha b}ba,\ ab\underline{\alpha b}a,\ abb\underline{\alpha a},\ ab\underline{baa}\alpha,\ ab\underline{\gamma b}a,\ a\underline{\gamma b}ba,\ \underline{\gamma b}bba,$
$bbba.$

Проверим работу построенного нормального алгоритма на словом $bbaaa$:

$\_bbaaa,\ \underline{\alpha b}baaa,\ b\underline{\alpha b}aaa,\ bb\underline{\alpha a}aa,\ bba\underline{\alpha a}a,\ bbaa\underline{\alpha a},\ bba\underline{aa\alpha},$
$bb\underline{a\beta}a,\ bb\underline{\beta a},\ b\underline{\beta a},\ \underline{\beta a},\ a.$

Видим, что нормальный алгоритм работает так, как и требовалось, при различных значениях x_{n-1}.

3. Построить нормальный алгоритм удвоения – нормальный алгоритм над $A=\{a,b\}$, преобразующий каждое слово R в алфавите A в слово RR.

Зададим нормальный алгоритм удвоения нормальной схемой подстановок:

$$\begin{cases} \alpha a \to a\beta a\alpha \\ \alpha b \to b\beta b\alpha \\ \beta aa \to a\beta a \\ \beta ab \to b\beta a \\ \beta ba \to a\beta b \\ \beta bb \to b\beta b \\ \beta \to \\ \alpha \to . \\ \quad \to \alpha \end{cases}$$

Работа этого алгоритма над словом bab состоит и следующей последовательности слов:

$\_bab,\ \underline{\alpha b}ab,\ b\beta b\underline{\alpha a}b,\ b\beta ba\beta a\underline{\alpha b},\ b\underline{\beta ba}\beta ab\beta b\alpha,$
$ba\beta b\underline{\beta ab}\beta b\alpha,\ ba\underline{\beta bb}\beta a\beta b\alpha,\ bab\underline{\beta b}\beta a\beta b\alpha,$
$babb\underline{\beta a}\beta b\alpha,\ babba\underline{\beta b}\alpha,\ babbab\underline{\alpha},\ babbab.$

4. Примерами нормальных алгоритмов, неприменимых ни к одному слову над алфавитом $A=\{a,b\}$, могут быть, алгоритмы, задаваемые схемами подстановок $\begin{cases} \alpha \to b \\ b \to a \end{cases}$ или $\{ \to aab$.

Как и в случае машин Тьюринга, будем рассматривать *числовые* функции, функции вида $f: N_0^n \to N_0$, то есть функции многих переменных, где каждая переменная может принимать значения во множестве целых неотрицательных чисел, и своей область

прибытия функция также имеет множество неотрицательных целых чисел.

Каждое неотрицательное число k будем изображать словом из k+1 единицы.

Упорядоченный набор чисел $(m_1, m_2, ..., m_n)$ будем изображать словом $1^{m_1+1} * 1^{m_2+1} * ... * 1^{m_n+1}$.

Числовая функция $f(x_1, x_2, ..., x_n)$ называется *вычислимой по Маркову*, если существует нормальный алгоритм, который каждое изображение набора аргументов преобразует в значение функции $f(x_1, x_2, ..., x_n)$ на этом наборе.

Другими словами, числовая функция $f(x_1, x_2, ..., x_n)$ называется *вычислимой по Маркову*, если существует нормальный алгоритм, применимый ко всем словам вида $1^{x_1+1} * 1^{x_2+1} * ... * 1^{x_n+1}$, преобразующий их в слово $1^{f(x_1, x_2, ..., x_n)+1}$.