1973 год ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ Выпуск 57

УДК 681.142.2

КЛАССЫ TR-ГРАММАТИК

В.И.Константинов

Выделяются интересные для практических приложений классы TR -грамматик, допускающих беступиковые схемы синтаксического анализа. Рассматриваются вопросы машинной реализации TR-грамматик на основе TR-таблиц. Приводятся алгоритмы работы синтаксических анализаторов для выделенных классов TR-грамматик.

I. Рассмотрим TR-грамматику без леворекурсивных элементов. По определению [I], с каждой TR-грамматикой связано множество R имен некоторых конечных подмножеств правил вида (α, τ, p), $\alpha \in A \cup V_N$, $\tau \in R$, $p \in P$. Сопоставим каждому подмножеству правил τ_k множество терминальных символов, которое назовем о п о р н ы м для подмножества τ_k и обозначим $O(\tau_k)$. Множество $O(\tau_k)$ строится следующим образом:

а) если $\tau_k = \{a_1 \to \tau_{i_1}, \ldots, a_n \to \tau_{i_n}\}$, $a_i \in A$, $1 \le i \le n$, то $O(\tau_k) = \{a_1, \ldots, a_n\}$, причем некоторые из a_i могут совпадать;

б) если $\tau_k = \{a_1 \to \tau_{i_1}, \ldots, a_n \to \tau_{i_n}, S_1 \to \tau_{j_1}, \ldots, S_m \to \tau_{j_m}\}$, $a_i \in A$, $1 \le i \le n$, $S_j \in V_N$, $1 \le j \le m$, то $O(\tau_k) = \{a_1, \ldots, a_n\} \cup O(S_1) \cup \ldots \cup O(S_m)$, $O(S_j) = \{a \in A / S_j \Rightarrow a\psi, \psi \in (A \cup V_N)^*\}$.

Таким образом, $O(\tau_k)$ - это множество первых терминальных элементов цепочек, выводимых из правил τ_k .

ЛЕММА I. В г р а м м а т и к а х б е з л е в о р е к у р с и в н ы х э л е м е н т о в м н о ж е с т в о $O(\tau_k)$ д л я к а ж д о г о τ_k м о ж е т б ы т ь п о с т р о е н о з а к о н е ч н о е ч и с л о ш а г о в.

Действительно, каждая грамматика содержит конечное число нетерминальных символов $S_1, \ldots, S_n$. Любое из $O(\tau_k)$ будет построено за конечное число шагов, если за конечное число шагов будут построены $O(S_j)$, S_j -нетерминальные левые части правил из τ_k . Покажем, как за конечное число шагов строится $O(S_j)$ для каждого $S_j \in V_N$. Определяя $O(S_j)$ как множество терминальных первых элементов цепочек, выводимых из S_j , под выводом из S_j будем понимать вывод из аксиомы $\tau_0^{G(S_j)}$, соответствующей S_j .

Если $\tau_0^{G(S_j)}$ состоит из правил с терминальными левыми частями, заносим их в $O(S_j)$ и на этом процесс построения $O(S_j)$ заканчиваем. Пусть среди правил, принадлежащих $\tau_0^{G(S_j)}$, есть правила с нетерминальной левой частью и пусть левые части этих правил - $S_1, \ldots, S_k$, Очевидно, что ни одно из $S_1, \ldots, S_k$ не может совпадать с S_j , иначе элемент S_j был бы леворекурсивным. Рассмотрим аксиомы $\tau_0^{G(S_1)}, \ldots, \tau_0^{G(S_k)}$. Терминальные левые части входящих в них правил заносим в $O(S_j)$, для нетерминальных этот процесс продолжаем. Поскольку на каждом шаге число нетерминальных элементов, которые могут быть первыми элементами порождаемых S_j цепочек, уменьшается хотя бы на единицу и нетерминальных элементов - конечное число, приходим в конце концов к цепочкам, начинающимся только с терминальных символов, т.е. за конечное число шагов $O(S_j)$ действительно построено.

Лемма доказана.

Пусть дана TR-грамматика G , образованная атомными грамматиками $G(S_i)$, $G = \bigcup_{i=0}^{n} G(S_i)$. Назовем грамматику $G(S_i)$ г р а м м а т и к о й п е р в о г о т и п а , если всё её заключительные правила - первого типа [I], иначе $G(S_i)$ назовем г р а м м а т и к о й в т о р с г о т и п а .

Свяжем с каждой атомной грамматикой $G(S_i)$ второго типа некоторое множество терминальных элементов, обозначаемое $P(S_i)$ и строящееся следующим образом.

Если $a \to \tau$ - заключительное правило грамматики $G(S_i)$, то

I) при $a \in A$ в $P(S_i)$ попадают элементы из $O(\tau) = O(\beta_1) \cup \ldots \cup O(\beta_k)$, $\beta_1, \ldots, \beta_k \in A \cup V_N$ - левые части правил из τ , $O(\beta) = \beta$, $\beta \in A$;

2) при $a \in V_N$

$$P(S_i)=\begin{cases}P(a)\cup O(z), & a\neq S_i,\\ O(z), & a=S_i.\end{cases}$$

Если $G(a)$ - грамматика первого типа, то $P(a)=\phi$ и процесс построения $P(S_i)$ на этом заканчиваем. В противном случае анализируем заключительные правила второго типа грамматики $G(a)$ и т.д. В получаемую в результате такого процесса цепочку из $P(S_i)$ и $O(z_k)$ заносим те из $P(S_i)$, которые ранее в ней не рассматривались. Если в грамматике $G(S_i)$ заключительных правил второго типа несколько, то $P(S_i)$ строим для каждого такого правила отдельно, а затем берем их объединение. Очевидно, что множества $P(S_i)$ могут быть построены за конечное число шагов.

Атомную TR-грамматику $G(S_i)$, $S_i\in V_N$, назовем грамматикой с неопределенным самопересечением, если она содержит хотя бы одно правило $S_k\to z_j$ (либо $S_k\to z_j$) такое, что $G(S_k)$ - атомная грамматика второго типа и $P(S_k)\cap O(z_j)\neq\phi$. Содержательно это определение поясним следующим примером. Пусть в грамматике $G(S)$ можно вывести цепочку $\alpha=\varphi S_1S_2\psi$; $S_1,S_2\in V_N$, $\varphi,\psi\in(A\cup V_N)^*$ и пусть порождаемые S_1 цепочки имеют вид $\theta\{\mu\}$, а S_2 - соответственно $\mu\gamma$, $\theta,\mu,\gamma\in(A\cup V_N)^*\setminus\varepsilon$, ε - пустая цепочка. Тогда в момент анализа первого элемента цепочки μ неизвестно, следует ли распознавать его в $G(S_2)$ или остаться в $G(S_1)$.

TR - грамматику назовем детерминированной, если опорное множество любого из подмножеств правил, её образующих, не содержит совпадающих элементов и ни одна из образующих её атомных грамматик не является грамматикой с неопределенным самопересечением.

Выделение класса детерминированных TR-грамматик объясняется тем, что они допускают или отвергают входные цепочки быстрее, чем недетерминированные. Из леммы I и определения атомной TR-грамматики с неопределенным самопересечением следует:

ЛЕММА 2. Свойство TR-грамматики быть детерминированной проверяется за конечное число шагов.

В классе детерминированных TR-грамматик выделим подкласс, образованный мононетерминальными грамматиками. По определению, грамматика является мононетерминальной, если любое из подмножеств правил, её образующих, содержит не более одного правила с нетерминальной левой частью. Особенностью этого подкласса является более высокая скорость работы анализатора, что показано ниже.

П. При представлении TR-грамматик в памяти электронной вычислительной машины (ЭВМ) каждому правилу TR-грамматик отведем строку TR-таблицы. Подмножеству правил будет при этом соответствовать уровень - последовательность рядом стоящих строк TR-таблицы. Роль имени подмножества правил выполняет адрес первого правила уровня. Для подмножеств правил, являющихся аксиомами, эти адреса соответствуют нетерминальным символам. Атомные TR-грамматики можно располагать в памяти ЭВМ в любом порядке.

Строка TR-таблицы имеет следующую структуру:

T_1	T_2	T_3	T_4	T_5	T_6	T_7

T_1 - указатель конца уровня,

$$T_1=\begin{cases}1 - \text{данное правило, последнее в уровне;}\\ 0 - \text{в противном случае;}\end{cases}$$

T_2 - указатель вида правила,

$$T_2=\begin{cases}1 - \text{правило с нетерминальной левой частью,}\\ 0 - \text{правило с терминальной левой частью;}\end{cases}$$

T_3 - код символа из A либо адрес строки TR-таблицы для правил с нетерминальной левой частью;

T_4 - адрес множества глобальных правил-преемников;

T_5 - выделитель разомкнутых правил,

$$T_5=\begin{cases}1 - \text{правило входит в подмножество, имя которого принадлежит } R^*,\\ 0 - \text{в противном случае;}\end{cases}$$

T_6 - указатель типа правил,

$$T_6=\begin{cases}1 - \text{правило второго типа,}\\ 0 - \text{в противном случае;}\end{cases}$$

T_7 - выходной символ (для преобразующих TR-грамматик).

Рассмотрим алгоритм работы синтаксического анализатора для детерминированных и мононетерминальных грамматик. Этот практически важный класс (он включает языки АЛГОЛ-60, ФОРТРАН, BASIC и др.) допускает беступиковые схемы синтаксического анализа. Последнее означает, что никакая подстрока входной строки не анализируется более одного раза. Это позволяет вести синтаксический анализ за время, пропорциональное n-длине входной строки, тогда

как для КС-грамматик [2] это время в общем случае пропорционально n^3 . Описание алгоритмов будет вестись применительно к таблицам. Это не уменьшает общности, т.к. небольшая модификация делает их пригодными и для грамматик, представленных в виде подмножеств правил. Для этого необходимо вместо адресов памяти рассматривать имена подмножеств правил и номера правил в подмножестве, их содержащем.

Пусть на вход анализатора поступила цепочка $S = a_1 a_2 \dots a_n$, $S \in A^* \setminus \varepsilon$. Введем еще один символ, #, $\# \notin (A \cup V_N)$, который назовем маркером конца входной строки. Считаем, что любая строка ограничена справа маркером # .Алгоритм работы анализатора для детерминированных TR-грамматик представим в виде записанной на языке **BASIC** программы. Поскольку изобразительные средства **BASICa** не позволяют передать все тонкости алгоритма (нельзя работать с частью ячейки), для соответствующих разрядов строки TR-таблицы введем обозначение $T_i[J], 1 \le i \le 7$.

```
 10  LET L = 0
 20  LET I,J,K = 1
 30  IF T2[J] = 0 THEN 80
 40  LET L=L+ 1
 50  LET S[L] = J
 60  LET J = T3[J]
 70  GOTO 30
 80  IF  Σ[I] = T3[J] THEN 220
 90  IF T1[J] = 1   THEN 120
100  LET J=J+ 1
110  GOTO 30
120  IF L=0 THEN 290
130  IF T5[J] = 0 THEN 190
140  LET J = S[L]
150  LET L = L- 1
160  LET V[K] = T7[J]
170  LET K = K + 1
180  GOTO 260
190  LET J = S[L]
200  LET L = L - 1
210  GOTO 90
```



```
220  LET V[K] = T7[J]
230  LET K = K+ 1
240  LET I = I+ 1
250  IF Σ[I] = # THEN 310
260  LET J = T4[J]
270  IF J ≠ 0 THEN 30
280  IF L ≠ 0 THEN 130
290  PRINT  "MISTAKE"
300  STOP
310  IF L = 0 THEN 370
320  LET J = S[L]
330  LET L = L - 1
340  LET V[K] = T7[J]
350  LET K = K + 1
360  GOTO 310
370  IF T4[J] ≠ 0  OR  T6[J] ≠ 1 THEN 290
380  PRINT  "NO MISTAKES"
390  END
```

В этой программе $\Sigma[I]$ – I-й входной символ, $S[L]$ – L-ая ячейка стека связи, $V[K]$ – K-ый выходной символ. Описание массивов $\Sigma[I]$, $V[K]$, $S[L]$, $T[J]$ в программу не включено.

Данный алгоритм непосредственно применим и для мононетерминальных грамматик. Однако, если ввести каноническую форму записи TR -грамматик, согласно которой в любом уровне сначала идут правила с терминальной левой частью, а затем – с нетерминальной, работу анализатора можно ускорить. Для этого оператор 210 заменяем на 210 GOTO 120. Это позволяет избежать проверки, является ли считанное из стека правило последнем в уровне (в случае мононетерминальных грамматик оно обязательно последнее).

Предложенный алгоритм был реализован в синтаксическом анализаторе системного транслятора для однородной вычислительной системы (ОВС) "Минск-222" [3,4]. Объем задающей его программы – порядка 50 слов ЭВМ "Минск-22", скорость работы – несколько десятков команд на анализируемый символ входной цепочки. Этот же алгоритм положен в основу синтаксического анализатора, являющегося одним из модулей программного обеспечения ОВС МИНИМАКС [5].

Литература

1. КОНСТАНТИНОВ В.И., НУРИЕВ Р.М. Метаязык трансляции контекстно-свободных языков. Настоящий сборник, с.77-83.

2. ЯНГЕР Д.Х. Распознавание и анализ контекстно-свободных языков за время n^3. - Проблемы математической логики, М., "Мир", 1970, с. 344-362.

3. ЕВРЕИНОВ Э.В., ЛОПАТО Г.П. Универсальная вычислительная система "Минск-222". -"Вычислительные системы", Новосибирск, 1966, вып. 23, с. 13-20.

4. КОНСТАНТИНОВ В.И. Транслятор для однородной вычислительной системы, -"Вычислительные системы", Новосибирск, 1972, вып. 51, с. 59-69.

5. ВИНОКУРОВ В.Г., ДИМИТРИЕВ Ю.К., ЕВРЕИНОВ Э.В., КОСТЕЛЯНСКИЙ В.М., ЛЕХНОВА Г.М., МИРЕНКОВ Н.Н., РЕЗАНОВ В.В., ХОРОШЕВСКИЙ В.Г. Однородная вычислительная система из мини-машин. -"Вычислительные системы" Новосибирск, 1972, вып. 51, с. 127-145.

Поступила в ред.-изд.отд.
6 февраля 1973 года