Поляков В. М., канд. техн. наук, доц.,
Рязанов Ю. Д., доц.
Белгородский государственный технологический университет им. В.Г. Шухова

АЛГОРИТМ ПОСТРОЕНИЯНЕРЕКУРСИВНЫХ ПРОГРАММ-РАСПОЗНАВАТЕЛЕЙ ЛИНЕЙНОЙ СЛОЖНОСТИ ПО ДЕТЕРМИНИРОВАННЫМ СИНТАКСИЧЕСКИМ ДИАГРАММАМ

Ryazanov.iurij@yandex.ru

Рассматриваются вопросы использования синтаксических диаграмм для автоматизации проектирования трансляторов. Предложен алгоритм построения нерекурсивных программ-распознавателей линейной сложности по табличному представлению детерминированных синтаксических диаграмм. Алгоритм может быть использован в системах автоматизированного построения трансляторов на основе синтаксических диаграмм.

Ключевые слова: *транслятор, детерминированная синтаксическая диаграмма, множество выбора, программа-распознаватель.*

Одним из удобных способов описания формальных языков является синтаксическая диаграмма (СД). Этот графический, оперирующий образами способ ориентирован на человека и в основном применялся в руководствах по языкам программирования [1]. Сейчас известны примеры использования СД при инженерном, ручном проектировании трансляторов [2 — 6]. В системах автоматизированного построения трансляторов [7 — 10] СД не применяются. Как правило, спецификация транслятора в этих системах представляется в нотации РБНФ, выполняется классическое проектирование транслятора [11] и его реализация с использованием алгоритмов линейной сложности, если это позволяет исходная РБНФ, либо с использованием универсальных алгоритмов более высокой сложности, таких как GLR, GLL или алгоритмов с возвратами.

Здесь дается формальное определение СД, позволяющее использовать табличное представление СД, удобное для автоматической обработки, и предлагается алгоритм преобразования таблиц детерминированных СД в нерекурсивную программу-распознаватель линейной сложности.

Табличный способ представления СД и предложенные алгоритмы ориентированы на их использование в системах автоматизированного построения трансляторов.

Синтаксическую диаграмму будем задавать четверкой $D = (T, N, G, S)$, где

T — конечное множество терминалов;

N — конечное множество нетерминалов;

$S \in N$ — начальный нетерминал;

$G = (V, E)$ — ориентированный граф, где

$V = V_T \cup V_N \cup V_u \cup V_{вход} \cup V_{выход}$, где

$V_{вход}$ — множество точек входа, $|V_{вход}| = |N|$;

V_T — множество терминальных вершин;

V_N — множество нетерминальных вершин;

V_u — конечное множество узлов;

$V_{выход}$ — множество точек выхода, $|V_{выход}| = |N|$;

$E = E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5$, где

$E_1 \subseteq \{(a,b) \mid a \in V_{вход}, b \in V_u\}$— множество входных дуг;

$E_2 \subseteq \{(a,b) \mid a \in V_u, b \in V_{выход}\}$ — множество выходных дуг;

$E_3 \subseteq \{(a,b) \mid a \in V_u, b \in V_T \cup V_N\}$ — множество дуг, выходящихиз узлов;

$E_4 \subseteq \{(a,b) \mid a \in V_T \cup V_N, b \in V_u\}$ — множество дуг, входящих в узлы;

$E_5 \subseteq \{(a,b) \mid a \in V_u, b \in V_u\}$ — множество ε-дуг, соединяющих узлы.

Каждому нетерминалу соответствует связная компонента графа. Компонента именуется соответствующим нетерминалом, имеет только одну точку входа и одну точку выхода и конечное множество вершин других типов. Точки входа и выхода на диаграмме компоненты не изображаются. Нетерминальная вершина изображается прямоугольником, в который вписан терминальный символ. Узел изображается на диаграмме жирной точкой. В точку входа не входит ни одна дуга и выходит конечное множество дуг (входные дуги компоненты). Узлы, в которые входят входные дуги, называются начальными. Из точки выхода не выходит ни одна дуга и входит конечное множество дуг (выходные дуги компоненты). Узлы, из которых выходят выходные дуги, называются заключительными. Каждая дуга, за исключением входных и выходных дуг, может выходить из узла и входить в терминальную или нетерминальную вершину или другой узел, либо выходить из терминальной или нетерминальной вершины и

входить в узел. В каждую терминальную и нетерминальную вершину входит только одна дуга и выходит только одна дуга. На количество дуг, входящих в узлы и выходящих из них, ограничений нет. На рис. 1 приведен пример синтаксической диаграммы.

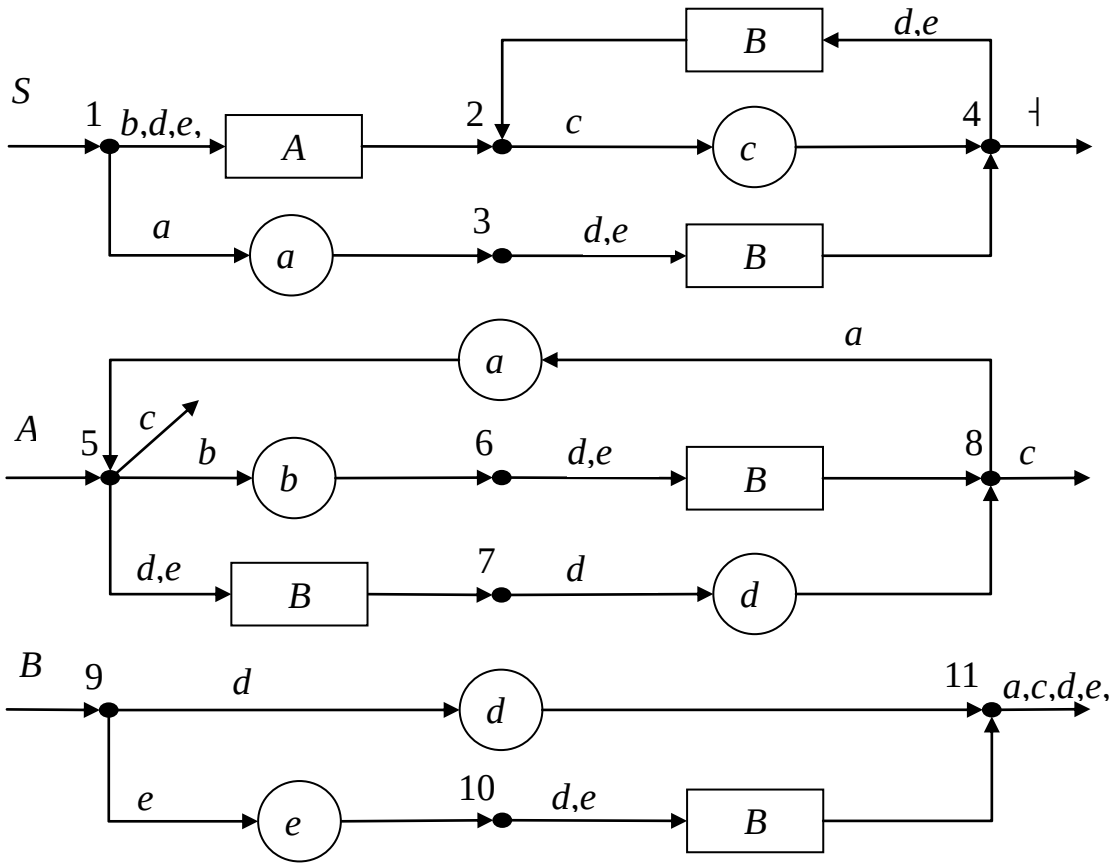

Рис. 1. Синтаксическая диаграмма

Синтаксическую диаграмму D можно представить множеством таблиц T, по одной для каждой компоненты.

Определим функцию R: $U \rightarrow \{1,2,\ldots,|V_u|\}$ разметки узлов, которая каждому узлу синтаксической диаграммы ставит во взаимнооднозначное соответствие элемент из множества $\{1,2,\ldots,|V_u|\}$.

Компоненту A синтаксической диаграммы будем представлять одноименной таблицей A, содержащей m строк и n столбцов, где n — количество узлов в компоненте A, $m=|T|+|N|+1$. Столбцы таблицы соответствуют узлам и отмечаются метками в соответствии с функцией R. Столбцы, соответствующие начальным узлам, отмечаются стрелочками. Столбцы, соответствующие заключительным узлам, отмечаются символом 1. Строки отмечаются терминалами, нетерминалами и символом ε. Каждый элемент таблицы содержит некоторое, возможно пустое, множество меток.

Пустая таблица A заполняется по следующим правилам. Если в диаграмме компоненты из узла с меткой i существует путь в узел с меткой j, проходящий только через одну терминальную вершину с терминалом $a \in T$, то в элемент таблицы $A_{a,i}$ добавляем метку j. Если из узла с меткой i существует путь в узел с меткой j, проходящий только через одну нетерминальную вершину с нетерминалом $B \in N$, то в элемент таблицы $A_{B,i}$ добавляем метку j. Если существует ε-дуга из узла i в узел j, то в элемент таблицы $A_{\varepsilon,i}$ добавляем метку j. Синтаксическая диаграмма (см. рис. 1) в табличной форме представлена в табл. 1.

Цепочку языка можно получить, «двигаясь» по дугам СД от точки входа начальной компоненты к ее точке выхода. При этом если дуга идет в терминальную вершину, то вписанный в нее символ добавляем в цепочку, если дуга идет в нетерминальную вершину, то переходим в соответствующую компоненту и движемся по ней аналогичным образом до точки выхода, после чего возвращаемся в предыдущую компоненту и продолжаем движение. После прохождения выходной дуги начальной компоненты в цепочку добавляем концевой маркер (⊣).

Таблица 1

Табличное представление синтаксической диаграммы

S	↓			1
	1	**2**	**3**	**4**
a	3			
b				
c		4		
d				
e				
S				
A	2			
B			4	2

A	↓ 1			1
	5	**6**	**7**	**8**
a				5
b	6			
c				
d			8	
e				
S				
A				
B	7	8		

B	↓		1
	9	**10**	**11**
a			
b			
c			
d	11		
e	10		
S			
A			
B		11	

Символ *x*, который может быть добавлен в цепочку после прохождения выходящей из узла дуги *e*, принадлежит множеству выбора дуги *e*. Узел *u* называется детерминированным, если множества выбора любых двух дуг, выходящих из узла *u*, не пересекаются. СД является детерминированной, если в ней все узлы детерминированные. В работе [12] описаны алгоритмы вычисления множеств выбора дуг, выходящих из узлов, и определения принадлежности СД классу детерминированных СД.

Определим интуитивно множества выбора дуг, выходящих из узлов, в СД на рис. 1 и припишем их к соответствующим дугам. Множества выбора дуг, входящих в терминальную вершину, очевидно, содержат только терминал, записанный в этой вершине. Множество выбора дуг, входящих в вершину с нетерминалом *B*, равно {*d, e*}, т. к. из начального узла 9 компоненты *B* дуги идут только в вершины с терминалами *d* и *e*. Множество выбора дуги (1, *A*) равно {*b, d, e, c*}. Принадлежность символа *b* этому множеству объясняется дугой (5, *b*), символов *d* и *e* — дугой (5, *B*) и дугами, выходящими из узла 9, символа *c* — дугами (5, выход) и (2, *c*). Множество выбора дуги (4, выход) содержит только концевой маркер {┤}, т. к. после прохождения этой дуги цепочка заканчивается. Множество выбора дуги (8, выход) содержит только символ *c* потому, что после выхода из вершины с нетерминалом *A* (дуга (*A*, 2)) попадаем в узел 2, из которого выходит единственная дуга (2, *c*). Множество выбора дуги (11, выход) равно {*a, c, d, e,* ┤}. Это объясняется последовательностями дуг ((*B*, 8), (8, *a*)), ((*B*, 2), (2, *c*)), ((*B*, 4), (4, *B*), (9, *d*) и (9, *e*)) и ((*B*, 4), (4, выход)). Выполнив анализ множеств выбора дуг, выходящих из узлов, можно сделать вывод, что СД детерминированная.

Множества выбора дуг, выходящих из узлов, можно представить таблицами (табл. 2), аналогичными по структуре таблицам СД.

Таблица 2

Множества выбора для дуг синтаксической диаграммы

S	↓			1
	1	**2**	**3**	**4**
a	*a*			
b				
c		*c*		
d				
e				
S				
A	*b,c,d,e*			
B			*d,e*	*d,e*
выход				┤

A	↓ 1			1
	5	**6**	**7**	**8**
a				*a*
b	*b*			
c				
d			*d*	
e				
S				
A				
B	*d,e*	*d,e*		
выход	*3*			*3*

B	↓		1
	9	**10**	**11**
a			
b			
c			
d	*d*		
e	*e*		
S			
A			
B		*d,e*	
выход			*a,c,d,e,*┤

Теперь рассмотрим алгоритм построения нерекурсивных программ-распознавателей по таблицам детерминированных синтаксических диаграмм.

Не ориентируясь на конкретный язык программирования, будем представлять программу-распознаватель на псевдокоде. Программы-распознаватели будут использовать магазин для хранения узлов синтаксических диаграмм. Для выполнения операций над магазином будем использовать следующие команды псевдокода:

1) *инициализация магазина* — при выполнении этой команды создается пустой магазин;

2) *магазин пуст* — проверка пустоты магазина;

3) *втолкнуть (m)* — втолкнуть узел *m* в магазин;

4) *m* := *вытолкнуть* — вытолкнуть верхний магазинный символ (узел СД) и присвоить его переменной *m*;

Опишем другие команды и элементы псевдокода, используемые в программе-распознавателе:

1) *Программа* и *Конец программы* — начало и конец программы;

2) *читать (x)* — чтение очередного символа входной цепочки в переменную *x* (при первом выполнении читается первый символ цепочки);

3) 1, 2, 3 … — метки, соответствующие узлам СД;

4) логические выражения

— *x* = *t*, где *t* — терминал или концевой маркер (┤); истина, если значение переменной *x* равно *t*;

— *x* ∈ <подмножество терминалов, которое может содержать концевой маркер>;

— *x* = ┤ *и магазин пуст;*

5) *если* <логическое выражение> *то* <Оператор1> *иначе* <Оператор2> — условный оператор, где <Оператор1> может представлять собой один из четырех вариантов:

— *читать (x), переход на* <метка>;

— *втолкнуть (*<метка>*), переход на* <метка>;

— *m* := *вытолкнуть , переход на m*;

— *Допустить, конец*;

<Оператор2> — условный оператор или *Отвергнуть, конец.*

В программе-распознавателе инициализируется магазин, читается первый символ входной цепочки и далее последовательно описываются все узлы СД. Описание узла отмечается одноименной меткой и представляет собой последовательность описаний дуг, выходящих из узла. Дуга описывается условным оператором, в котором определяется принадлежность обрабатываемого символа множеству выбора дуги. Если символ принадлежит множеству выбора дуги и дуга идет в терминальную вершину, то читается следующий символ входной цепочки и выполняется переход на метку, соответствующую узлу, в который идет дуга из терминальной вершины. Если символ принадлежит множеству выбора и дуга идет в нетерминальную вершину с нетерминалом *A*, то нужно перейти на начальный узел компоненты *A*, предварительно запомнив узел *u*, в который идет дуга из нетерминальной вершины, в магазине. Из начального узла компоненты *A* начинается «движение» по узлам компоненты, и, как только попадем в заключительный узел и обрабатываемый символ будет принадлежать множеству выбора выходной дуги, нужно будет перейти в узел *u*, который будет находиться на верху магазина. Из магазина этот узел извлекается. Если символ непринадлежит множеству выбора и дуга не последняя, то описывается следующая дуга, выходящая из рассматриваемого узла, а если дуга последняя, то цепочка отвергается. Выходная дуга заключительного узла, идущая в точку выхода, описывается последней. Если символ принадлежит множеству выбора, то узел, на который следует перейти, извлекается из магазина и выполняется переход, иначе цепочка отвергается. Цепочка допускается, если при посещении заключительного узла начальной компоненты все ее символы обработаны (обрабатываемый символ — концевой маркер) и магазин пуст.

Текст программы-распознавателя, соответствующей СД (см. рис. 1, табл. 1 и 2) представлен ниже на псевдокоде:

Программа
инициализация магазина;
читать (x);
1: *если x = a то читать (x), переход на* 3 *иначе*
если x ∈ {b, c, d, e} то втолкнуть(2), переход на 5
иначе Отвергнуть, конец.
2: *если x = c то читать (x), переход на* 4
иначе Отвергнуть, конец.
3: *если x ∈ {d, e} то втолкнуть(4), переход на* 9
иначе Отвергнуть, конец.
4: *если x =* ┤ *и магазин пуст, то Допустить, конец иначе*
если x ∈ {d, e} то втолкнуть(2), переход на 9 *иначе*

если x ∈ {┤ } *то m*: = *вытолкнуть , переход на m иначе*
Отвергнуть, конец.

5: *если x = b то читать (x), переход на* 6 *иначе*
если x ∈ {*d, e*} *то втолкнуть(7), переход на* 9 *иначе*
если x ∈ {*c*} *то m* := *вытолкнуть , переход на m иначе*
Отвергнуть, конец.
6:если x ∈ {*d, e*} *то втолкнуть(8), переход на* 9
иначе Отвергнуть, конец.
7: *если x = d то читать (x), переход на* 8
иначе Отвергнуть, конец.
8: *если x = a то читать (x), переход на* 5 *иначе*
если x∈ {*c*} *то m* := *вытолкнуть , переход на тиначе*
Отвергнуть, конец.

9: *если x = d то читать (x), переход на* 11 *иначе*
если x = e то читать (x), переход на 10
иначе Отвергнуть, конец.
10: *если x*∈ {*d, e*} *то втолкнуть(11), переход на* 9
иначе Отвергнуть, конец.
11: *если x*∈ {*a, c, d, e,* ┤ } *то*
m := *вытолкнуть , переход на тиначе*
Отвергнуть, конец.

Конец программы

Алгоритм построения программы-распознавателя по таблицам СД следующий:

1. Выдать «*инициализация магазина;*»
2. Выдать «*читать (x);*»
3. Для всех компонент СД сформировать фрагмент кода, используя процедуру СинтезКода.
4. Выдать «*Конец программы».*
5. Конец алгоритма.

Алгоритм процедуры СинтезКода(*A*, *MV*).
Вход: *A* — таблица компоненты *A*;
MV — таблица, хранящая множества выбора дуг, выходящих из узлов компоненты *A*.
Выход: фрагмент кода программы, соответствующий компоненте *A*.
1. Для всех узлов *u* компоненты *A* (столбцов таблицы *A*) выполнить:
1.1. Выдать метку, соответствующую узлу *u*.
Если *A* — начальная компонента и узел *u* — заключительный, то выдать
«*если x =* ┤ *и магазин пуст, то Допустить, конец иначе*»
1.2. Для всех строк-терминалов *t* выполнить:
если $A_{t,u} \neq \varnothing$, то выдать
«*если x = t то читать (x), переход на m иначе*»,
где $m = A_{t,u}$.
1.3. Для всех строк-нетерминалов *Y* выполнить:
если $A_{Y,u} \neq \varnothing$, то выдать
«*если x* ∈ $MV_{Y,u}$ *то втолкнуть*($A_{Y,u}$), *переход на m иначе*»,
где *m* — первый столбец таблицы *Y*.
1.4. Если $A_{\text{выход},u} \neq \varnothing$, то выдать
«*если x* ∈ $MV_{\text{выход},u}$ то *m* := *вытолкнуть , переход на m иначе*».
1.5. Выдать «*Отвергнуть, конец.*».
2. Конец алгоритма.

Таким образом, в статье предложен алгоритм построения нерекурсивных программ-распознавателей линейной сложности по таблицам детерминированных синтаксических диаграмм, который может быть использован в системах автоматизированного построения трансляторов на основе синтаксических диаграмм.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Йенсен К., Вирт Н. Паскаль. Руководство для пользователя и описание языка. М: Финансы и статистика, 1982. 151 с.

2. Легалов А.И.Основы разработки трансляторов. URL:

http://www.softcraft.ru/translat/lect/content.shtml (дата обращения: 15.09.2013)

3. Легалов А.И., Швец Д.А., Легалов И.А. Формальные языки и трансляторы. Красноярск: Сибирский федеральный университет, 2007. 213 с.

4. Свердлов С.З. Введение в методы трансляции. Вологда: Издательство «Русь», 1994. 80 с.

5. Свердлов С. З. Языки программирования и методы трансляции. СПб.: Питер, 2007. 638 с.

6. Карпов Ю. Г. Теория и технология программирования. Основы построения трансляторов. СПб.: БХВ-Петербург, 2005. 272 с.

7. ANTLR.URL: http://www.antlr.org/ (дата обращения: 15.09.2013)

8. ASF+SDF.URL: http://www.cwi.nl/projects/MetaEnv/ (дата обращения: 15.09.2013)

9. Bison.URL: http://www.gnu.org/software/bison/ (дата обращения: 15.09.2013)

10. Coco/R.URL: http://www.ssw.uni-linz.ac.at/Research/Projects/Coco/ (дата обращения: 15.09.2013)

11.Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. М.: Мир, 1978. т. 1, 612 с. т. 2, 487 с.

12. Рязанов Ю. Д., Севальнева М. Н. Анализ синтаксических диаграмм и синтез программ-распознавателей линейной сложности // Научные ведомости БелГУ. Сер. История. Политология. Экономика. Информатика. 2013. № 8 (151). Вып. 26/1. С. 128–136.