УДК 519.06

Г. Ф. ДЮБКО, канд. техн. наук, ФУДЖУ ХАЛЛИД ИССА

СИНТАКСИЧЕСКИЙ АНАЛИЗ ЯЗЫКОВЫХ КОНСТРУКЦИЙ, ОПИСЫВАЮЩИХ ЦИФРОВЫЕ УСТРОЙСТВА

Введение

Экспертные системы создаются для решения широкого круга проблем, в частности для решения задач проектирования – нахождения конфигурации компонентов системы, которая удовлетворяет целевым условиям и множеству проектных ограничений, а также задач планирования – разработки последовательности действий для достижения множества целей при данных начальных условиях и временных ограничениях. Одной из технологий, решающих проблему построения экспертной системы (ЭС), является технология на основе знаний, в которой знания составляют ядро системы. Традиционно ЭС строится на базе некоторой модели знаний: продукционной, логической, сетевой, фреймовой. Предлагается для построения экспертных систем применять трёхкомпонентную модель знаний, одной из компонент которой является грамматика. Применение трёхкомпонентной модели знаний позволяет повысить как эффективность функционирования самой системы, так и эффективность процесса её создания. Манипулирование знаниями, представленными продукциями грамматики, производится с помощью синтаксического анализа. Целью работы является рассмотрение методов синтаксического анализа и отработка методики построения синтаксического анализатора для решения задач по созданию программных моделей для тестирования физических устройств (средств вычислительной техники).

Постановка и решение задачи

Синтаксический анализ – это процесс обработки ИЯК с помощью продукций грамматики, результатом которого является некоторая иерархическая структура, представленная обычно деревом. Часто синтаксический анализ называют разбором (parsing), а дерево – деревом разбора. Разбор осуществляется при помощи последовательности шагов, каждый из которых требует применения конкретной продукции грамматики. Последовательность шагов, приводящая к созданию цепочки символов, называется порождением ИЯК. Задача синтаксического анализа есть нахождение порождения ИЯК или указания, что его не существует. Во втором случае ИЯК не принадлежит языку, описанному грамматикой, использованной в анализе. В порождении на каждом шаге из имеющейся цепочки получается следующая цепочка путем замены какого-то нетерминала имеющейся цепочки на правую часть от той продукции, где этот нетерминал является левой частью. Если на каждом шаге порождения заменяется крайний левый нетерминал, то порождение называется левым, если правый нетерминал, то – правым. Существуют порождения, которые не являются ни левыми, ни правыми. Для различных порождений одной и той же цепочки существует инвариант – дерево разбора.

Различают различные стратегии синтаксического анализа: развертку, свертку, смешанную. При развертке рассматривается порождение от аксиомы к ИЯК, при свертке – от ИЯК к аксиоме, в смешанной стратегии комбинируется свертка и развертка. Развертка обладает большей степенью наглядности, чем свертка, однако, свертка обладает большей общностью и более мощной инструментальной поддержкой.

Важным фактором при выполнении очередного шага порождения является способ выбора продукции, применяемой на этом шаге. Существуют специальные типы грамматик, где выбор продукции обуславливается наличием и выполнением определенных признаков и условий, обеспечивающих высокую эффективность алгоритма нахождения порождения.

Так, например, при развертке используют LL(1) – грамматику, которая позволяет однозначно выбрать продукцию на очередном шаге порождения, используя только два условия –

нетерминальный символ из цепочка порождения и текущий терминальный символ из входной цепочки. Однако, чтобы грамматика была LL(1), на ее продукции накладываются жесткие ограничения, которые часто невозможно удовлетворить. Кроме того, при преобразовании продукций к форме LL(1) разрываются семантические правила, что приводит к еще одному виду нестыковок. При программной реализации синтаксического анализа используется стек, в котором хранится текущая цепочка, полученная на текущем шаге порождения. При развертке самый левый нетерминал, находящийся в вершине стека, заменяется цепочкой (правой частью продукции). При свертке выделяется цепочка в вершине стека, которая является правой частью некоторой продукции, и заменяется на нетерминал. Таким образом, в алгоритме синтаксического анализа использованы две структуры данных – стек и входная строка.

В развертке появление терминала на вершине стека влечет сравнение этого терминала с текущим терминалом входной строки и при их совпадении удаление терминала из стека и перемещение на один символ по входной строке. Такой подход позволяет получить пустую цепочку на последнем шаге порождения в стеке и во входной строке, что является признаком успешности порождения.

В свертке символы из входной строки переносятся в вершину стека, накапливая там (с уже имеющимися) правую часть некоторой продукции. Процесс свертывания в общем случае является недетерминированным, так как в вершине стека могут иметься несколько цепочек, являющихся правыми частями разных продукций, среди которых нужно выбрать одну. Для разрешения недетерминизма (конфликтов) в свертке используют так же специальные типы грамматик. Грамматика, все конфликты которой, возникающие при свертке слева направо, могут быть разрешены с использованием фиксированного объема информации, касающейся уже проведенного анализа и конечного числа символов предпросмотра, называется LR(k)-грамматикой. Здесь L означает чтение исходной строки слева (left) направо, R – правые порождения (Rightmost), а k – означает количество символов предпросмотра. Если требуется только один символ предпросмотра, то грамматика относится к классу LR(1), что обеспечивает высокую скорость анализа.

Рассмотрим более подробно свертку и алгоритм синтаксического анализа сверткой, в котором превалируют две операции: перенос, свертка, которые дали название алгоритму «алгоритм типа перенос – свертка».

Дадим формальное определение этого алгоритма [1].

Пусть G=(V_T, V_N, P, S) – КС-грамматика, продукции которой занумерованы целыми числами от 1 до n. Алгоритмом типа перенос – свертка для грамматики G называется пара функций (f, g), где f называется функцией переноса, а g – функцией свертки. Определим функции f и g:

– f отображает пару $V^* \times (V_T^* \cup \{\#\})$ в множество {**перенос, свертка, ошибка, допуск**}, где $V = V_N \cup V_T$, а # – концевой маркер цепочки (V – алфавит стека);

– g отображает пару $V^* \times (V_T^* \cup \{\#\})$ в множество {1, 2, …, n, **ошибка**}при условии, что если g (α, ω) = i, то правая часть i-го правила является суффиксом цепочки α.

Обозначим через (α, ω, π) состояние алгоритма типа «перенос – свертка», в котором α – содержимое стека, ω – непрочитанная часть входной цепочки, π – последовательность номеров продукций, участвующих в порождении; а через => – один шаг алгоритма из этого состояния. Один шаг алгоритма можно описать так:

– если f $(\alpha, a\omega)$ – **перенос**, то $(\alpha, a\omega, \pi) \Rightarrow (\alpha a, \omega, \pi)$ для $\alpha \in V^*, \omega \in V_T^* \cup \{\#\}$ и $\pi \in \{1,\ldots,n\}^*$;

– если f $(\alpha\beta, \omega)$=**свертка**, g $(\alpha\beta, \omega)$=i и $A \to \beta$ – продукция с номером i, то $(\alpha\beta, \omega, \pi) \Rightarrow (\alpha A, \omega, \pi i)$;

– если f (α,ω)=**допуск**, то $(\alpha,\omega,\pi)\Rightarrow$ **допуск**;

– в остальных случаях $(\alpha,\omega,\pi)\Rightarrow$ **ошибка.**

Алгоритм типа «перенос – свертка» в общем случае неэффективен ввиду наличия недетерминизма (какую функцию f или g выбрать; или какой суффикс β_i из множества $\{\alpha\beta_1,\alpha\beta_2,...,\alpha\beta_k\}$, где $A_1\rightarrow\beta_1$, ..., $A_k\rightarrow\beta_k$, выбрать для свертки).

Критерий принятия решений относительно предпринимаемого действия – переноса или свертки, или выбора правильного суффикса – может содержаться в таблице, называемой таблицей синтаксического анализа. Алгоритм типа «перенос – свертка» может быть реализован с помощью автомата с магазинной памятью, одной из компонент которого является множество состояний, выраженных целыми числами. Каждому состоянию анализатора соответствует одна строка в таблице синтаксического анализа (двумерная матрица), а каждому терминалу и нетерминалу грамматики – один столбец. Каждый шаг анализа определяется позицией таблицы, соответствующей **текущему состоянию и входным символам.** Позиция таблицы может принадлежать к одному из двух типов:

– позиция **переноса** вида fi, вынуждающая анализатор выполнить действие переноса и изменить текущее состояние на состояние i;

– позиция **свертки** вида gi, вынуждающая анализатор выполнить действие свертки, используя продукцию j.

Позиции таблицы, не означенные fi или gi, должны быть означены символом **ошибка**.

Таблица синтаксического анализа представляет зависимую от входного языка часть синтаксического анализатора. Остальная часть анализатора является универсальной и, интерпретируя данные в таблице синтаксического анализа, выполняет действия,определенные в формализме «перенос – свертка». В начале синтаксического анализа анализатор находится в состоянии 1, а входной символ – это первый символ анализируемой входной цепочки. Вместо таблицы синтаксического анализа можно использовать ее эквивалент в форме характеристического конечного автомата (ХКА).

Опишем теперь процесс создания ХКА из контекстно-свободной грамматики путем аннотирования грамматики [2].

Аннотирование грамматики происходит следующим образом. Конфигурацией называется позиция в правой части продукции перед первым символом, после последнего символа или между двумя любыми символами. В конфигурации может быть несколько позиций.

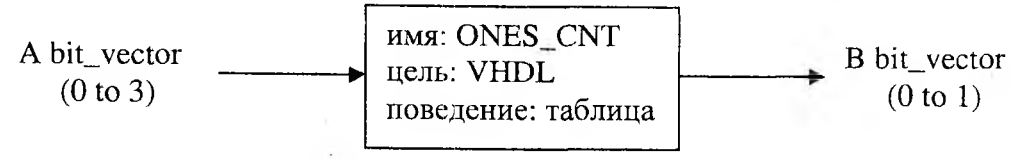
Начинается аннотирование введением состояния 1 (позиция перед первым символом правой части продукции, где левая часть есть аксиома), состояние 2 ставится перед вторым символом этой правой части и т.д.; пусть получено соединение ${}_iX$, где i – состояние, X – символ грамматики. Если X – нетерминальный символ, то состояние i появляется в начале всех правых частей продукций, где X есть левая часть. Если же X есть терминал, то за ним ставится следующее по порядку состояние ${}_iX_{i+1}$. Конфигурации в различных продукциях, соответствующие одному состоянию, неразличимы с точки зрения синтаксического анализатора. Предположим, что мы получили ${}_{i1,i2,i3}X_{j1,j2}$, где X – нетерминальный символ в одной из продукций, и i1 соответствует j1 (т.е. возможен переход из i1 в j1), а i2, i3 – j2.

В другой продукции этот же символ аннотирован как ${}_{i1}X_{j3}$ ($j1,j2<j3$). Тогда в первой продукции аннотация символа X должна быть дополнена как ${}_{i1,i2,i3}X_{j1,j2,\,j3}$, и i1 соответствует j1, j3, a, i2, i3 – j2.

Состояние j, находящееся в конце продукции, соответствует свертке. Если между переносом и сверткой имеется конфликт в состоянии j, он разрешается следующим образом. Когда из состояния j имеется переход в состоянии j1 с терминальным символом x, и текущий символ входной цепочки есть x, осуществляется перенос. Иначе это должна быть свертка. Конфликт сверток между собой разрешается состоянием, так как каждое из «сверточных» состояний связано строго с одной продукцией.

Продемонстрируем построение грамматики, ее аннотирование и создание ХКА для примера, который можно считать базовым при проектировании цифровых устройств.

Пусть имеется цифровой модуль с одним входом и одним выходом, задающими сигналы типа bit_vector, поведение которого описано таблицей истинности, представленный так, как на рис. 1.



Поведение	A [8, 3]	(0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1);	B [8, 2]	(0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1)

Рис. 1

Описание модуля, выполненное в визуальной форме можно представить графовой продукцией.

VHDL_таблица ONES_CNT (A bit_vector in; bit_vector out) ::
поведение in A [8, 3] (0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1);
out B [8, 2] (0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1) (1)

Преобразование рис. 1 в строку (1) производится автоматически (вид лексического преобразования). Разрабатываемая грамматика должна дать возможность выявить имя блока, интерфейс блока и его спецификацию, вид поведения, цель (программа на VHDL), таблицы, задающие поведение.

Грамматика должна описывать синтаксис строк типа (1). Грамматика, удовлетворяющая этим требованиям, представлена своими продукциями здесь ниже.

1. <схема>→ **VHDL_таблица имя** (<интерфейс>) **::** <поведение>
2. <интерфейс>→**имя** <тип>**in; имя** <тип> **out**
3. <тип>→ **bit_vector (число to число)**
4. <поведение>→ **поведение in имя [число, число]** (<список данных>);
   out имя [число, число] (<список данных>)
5. <список данных>→<список данных>**,** <данное>
6. <список данных>→ <данное>
7. <данное>→0
8. <данное>→1

Для аннотирования грамматики сократим ее запись, введя следующие обозначения.

Нетерминалы: <схема> – H, <интерфейс> – I,<поведение> – C, <тип> – T, <список данных> – L, <данное> – D;

Терминалы: VHDL_таблица–v, :: -:, имя–i, in–e, out–s, bit_vector–b, число–n, to–t.

Вышеприведенная грамматика в этих обозначениях имеет вид (включая аннотирование):

1. $H \rightarrow {}_1V_2\, i_3\, (_4I_5)_{6:7}\, C_8$
2. $I \rightarrow {}_4\, i_9 T_{10}\, e_{11};\ {}_{12}\, i_{13}\, T_{14}\, S_{15}$
3. $T \rightarrow {}_{9,13}\, b_{16}\, (_{17}\, n_{18}\, t_{19}\, n_{20})_{21}$
4. $C \rightarrow {}_7\, P_{22}\, e_{23}\, i_{24}[_{25}\, n_{26},\ {}_{27}n_{28}]_{29}\, (_{30}\, L_{31})_{32};$
   $_{33}\, S_{34}\, i_{35}\, [_{36}\, n_{37},\ {}_{38}\, n_{39}]_{40}\, (_{41}\, L_{42})_{43}$
5. $L \rightarrow {}_{30,41}\, L_{44},\ {}_{45}\, D_{46}$
6. $L \rightarrow {}_{30,41}\, D_{47}$

7. $D \rightarrow_{30,41,45} 0_{48}$
8. $D \rightarrow_{30,41,45} 1_{49}$

Состояния, в которых производится свертка: 8, 15, 21, 43, 46, 47, 48, 49. Конфликтов между переносом и сверткой нет.

Характеристический конечный автомат, построенный в соответствии с аннотированной грамматикой, приведен на рис. 2.

Далее сформулируем алгоритм синтаксического анализа PARSING (PR, KA, BX), аргументами которого являются PR – продукции грамматики; KA – характеристический конечный автомат; BX – анализируемая цепочка.

В этом алгоритме использованы следующие структуры данных и функции:

BX – массив входных данных;

CTE – стек состояний;

CTS – стек символов;

PR – множество продукций грамматики;

KA – характеристический конечный автомат;

XKA (X, Y, Z) – функция, возвращающая состояние по
X – имя структуры, содержащей автомат,
Y – символ, который читает автомат,
Z – состояние, в котором происходит чтение;

XKAN (X, Y) – функция, возвращающая номер продукции, по которой производится свертка X при указанном состоянии Y;

PRODL (X, Y) – возвращает левую часть продукции, содержащейся в X и имеющей номер Y;

PRODR (X, Y) – возвращает правую часть продукции с номером Y из X;

PRODn (X, Y) – возвращает количество символов, содержащихся в правой части продукции с номером Y из X.

```
    PARSING (PR, KA, BX)
1.  i←1
2.  k←1
3.  j←0
4.  CTE[K] ←'1'
5.  инициализация листьев дерева
6.  while BX[i] ≠ '#' and "анализ не закончен"
7.      do if CTE[K] – свертка
8.          then NP←XKAN (KA, CTE[K])
9.                   DELETEE (CTE, PRODn (PR, NP))
10.                  DELETES (CTS, PRODP (PR,NP))
11.                  j←j+1
12.                  L←PRODL (PR, NP)
13.                  CTS [j] ←L
14.                  k←k+1
15.                  STE[K] ←XKA (KA, L, STE[K–1])
16.                  построение текущего узла дерева разбора
17.         else k←k+1
18.                  STE[K] ←XKA (KA, BX[i], CTE[K–1])
19.                  j←1+1
20.                  CTS [j] ← BX[i]
21.                  i←i+1
```

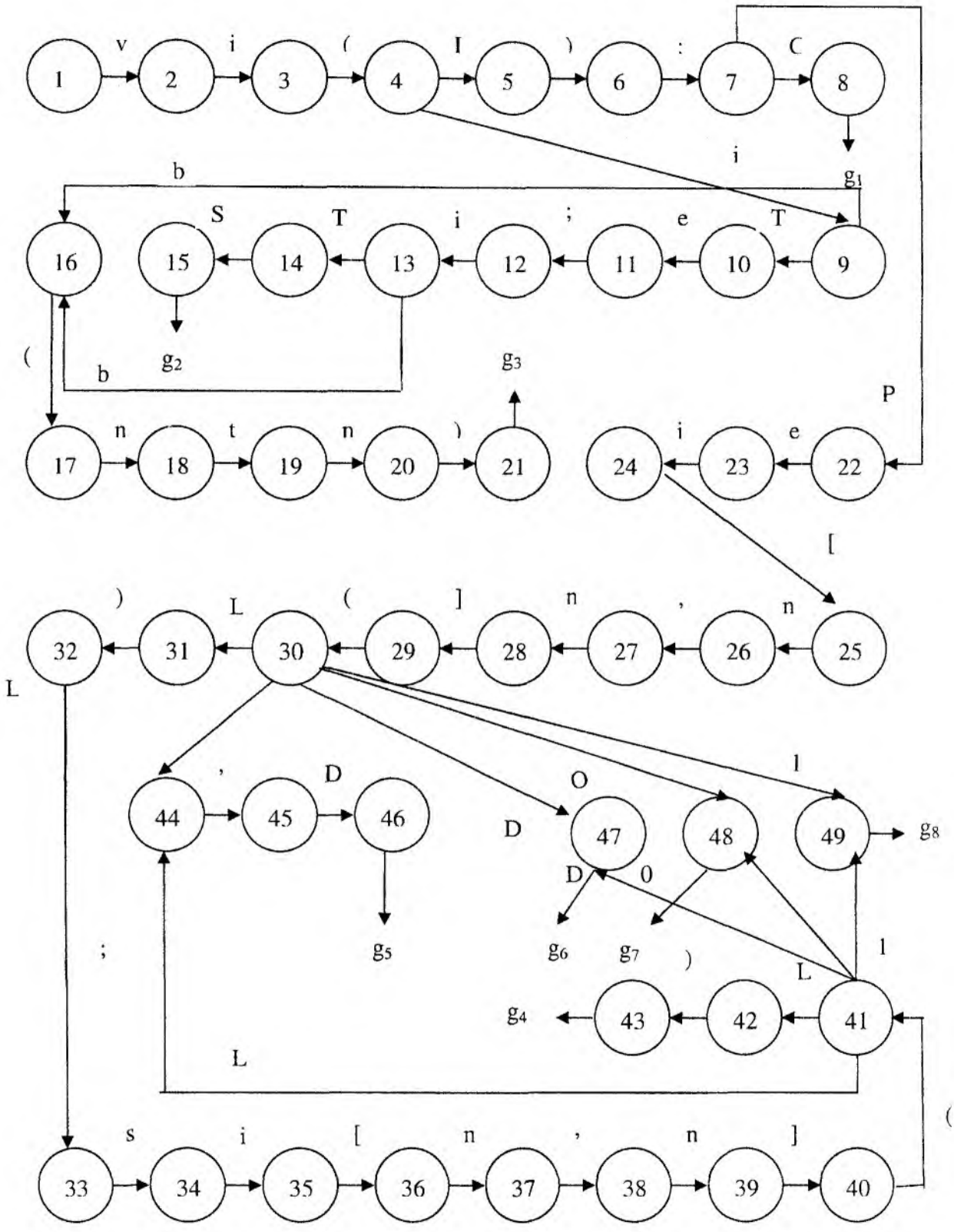

1
v
2
i
3
(
4
I
5
)
6
:
7
C
8
i
g1
b
16
S
15
T
14
i
13
;
12
e
11
T
10
9
(
b
g2
g3
P
17
n
18
t
19
n
20
)
21
24
i
23
e
22
[
32
)
31
L
30
(
29
]
28
n
27
,
26
n
25
L
44
,
45
D
46
D
47
O
48
l
49
g8
D
0
;
g5
g6
g7
)
L
l
g4
43
42
41
L
(
33
s
34
i
35
[
36
n
37
,
38
n
39
]
40

Рис. 2

Заключение

Предложенная методика построения синтаксического анализатора методом свертки в аннотированной грамматике легко поддаётся формализации, что гарантирует генерацию анализаторов в автоматическом режиме. Беступиковый анализатор весьма эффективен при манипуляции лингвистическими знаниями и может быть использован в системе автоматической генерации программных моделей, представляющих электронные устройства.

Список литературы: 1. *А. Ахо, Дж. Ульман.* Теория синтаксического анализа, перевода и компиляции. Т.1: Синтаксический анализ. М.: Мир, 1978. 612 с. 2. *Робин Хантер* Основные концепции компиляторов. М.: Издательский дом «Вильямс», 2002 . 256 с.

Харьковский национальный
университет радиоэлектроники

Поступила в редколлегию 16.03.2004