**Левая и правая рекурсия**

Рассмотрение грамматик с левой и правой рекурсией позволит нам получить некоторые признаки, помогающие определить наличие или отсутствие LL(1) свойства у КС-грамматик.

Если в грамматике G существует нетерминал

 +

 A, для которого A $⇒$ Аа, где а-непустая цепочка, грамматика содержит левую рекурсию.

*Грамматика, содержащая левую рекурсию, не может быть LL( 1) грамматикой.*

Если в грамматике G существует нетерминал

 +

A, для которого A $⇒$аА, где а-непустая цепочка, грамматика содержит правую рекурсию.

*Леворекурсивная грамматика всегда может быть преобразована в эквивалентную праворекурсивную.*

**Синтаксический анализ арифметических выражений**

Рассмотрим арифметические выражения, синтаксис которых задается грамматикой

G8: E→T|E+T|E-T

 T→M|T\*M|T/M

 M→a|b|c|( E)

Эта КС-грамматика, построенная нами раньше, обладает рядом достоинств.

Она однозначна и ассоциирует операнды в соответствии с общепринятым порядком выполнения операций, когда вначале выполняются умножение и деление, затем — сложение и вычитание.

Однако нетрудно видеть, что G8 леворекурсивна.

Действительно, для нетерминала Е справедливо: Е → Е + Т. Наличие левой рекурсии препятствует использованию рекурсивного спуска.

G8 может быть заменена эквивалентной (порождающей тот же язык) праворекурсивной грамматикой:

G13: E→T|T+E|T-E

 Т→М|М\*Т|М/Т

 М→а |b|с|(Е)

Хотя левой рекурсии больше нет, грамматика G13 не является LL(1)-грамматикой.

Чтобы убедиться в этом, достаточно обратиться к правилам для нетерминала Е:

E→T|T+E|T-E

Напомним, что эту строку следует рассматривать как сокращенную запись трех альтернативных правил для нетерминала Е. Очевидно, что выбор одного из этих трех правил на основе анализа одного входного символа невозможен, поскольку правая часть каждого правила начинается одним и тем же нетерминалом Т.

На рис. 24 показана синтаксическая диаграмма нетерминала Е грамматики G13, рассмотрение которой также убеждает, что требование детерминированного распознавания не выполнено: множества направляющих символов всех трех ветвей совпадают: first1 = first2 = first3 = {a, b, c, (}.



Рис.24 Диаграмма нетерминала E грамматики G13

Грамматика G13 и соответствующие синтаксические диаграммы могут быть преобразованы так, чтобы использование рекурсивного спуска стало возможным.

Особенно легко увидеть возможность преобразования диаграмм.

Действительно, достаточно вынести блок T из трех параллельных ветвей и поместить его в общую ветвь, как требование детерминированного распознавания для диаграммы нетерминала E будет соблюдено (рис. 25).



Рис.25 Преобразованная диаграмма нетерминала E грамматики G13

Аналогичное изменение может быть выполнено и для диаграммы слагаемого (нетерминал Т).

Менее очевиден способ преобразования грамматики. Потребуется два дополнительных нетерминала. Обозначим их А и В. Тогда новая грамматика может быть записана так:

G14: Е→ТА

 А→ε| + Е| - Е

 Т→МВ

 В→ε| \* Т| / Т

 M→a|b|c| (E)

Содержательно А и В можно трактовать как «выражение без первого слагаемого» и «слагаемого без первого множителя» соответственно.

G14 – это LL(1)-грамматика для арифметических выражений. Но и она не вполне может нас устроить. Дело в том, что G14, как и G13 (но не G8), связывает операнды нескольких идущих подряд операций одного приоритета неподходящим образом, группируя их справа налево.

На рис. 26,*а* показано дерево вывода выражения

а - b - с в грамматике G14.

Устранив из этого синтаксического дерева нетерминалы (и ε) и перенося знаки операций во внутренние вершины, получим семантическое дерево выражения (рис. 26, б), которое, увы, не соответствует правильному порядку его вычисления:

оно подразумевает группировку операндов а - (b - с), в то время как а - b - с = (а - b) - с. Правильное семантическое дерево можно видеть на рис. 2.27, в.



Рис.26 Дерево выражения a-b-c

Модернизируем G14 с целью обеспечить правильную ассоциацию операций и операндов. Получаем грамматику G15:

G15: E→TA

 A→ε|+TA|-TA

 T→MB

 В→ε|\*МВ|/МВ

 M→a|b|c| (E)

Эта однозначная праворекурсивная LL( 1 )-грамматика приписывает выражению правильную структуру. По ней без труда может быть написан синтаксический анализатор, работающий по алгоритму рекурсивного спуска.

 Интересно отметить, что программа-распознаватель, написанная по этой грамматике, будет рекурсивной (что, как известно, эквивалентно итерации), но не будет содержать циклов.

Недостатками такой грамматики является некоторая ее громоздкость и наличие нетерминалов A и В с неочевидным смыслом.

А теперь мы воспользуемся теми выразительными средствами, которые дают синтаксические диаграммы.

Это в первую очередь возможность использования цикла вместо рекурсии.

Вообще, можно заметить, что некоторые трудности, возникающие на практике при использовании грамматик Хомского, обусловлены тем, что с их помощью приходится выражать повторение через рекурсию.

Так, о выражении, представляющем собой просто последовательность слагаемых, мы вынуждены думать как о рекурсивной конструкции: выражение — это первое слагаемое, за которым после знака снова записано выражение (правая рекурсия, неподходящая группировка слагаемых: первое слагаемое складывается со всем остальным выражением). Или: если к выражению прибавить или от выражения отнять слагаемое, то снова получится выражение (левая рекурсия, правильная группировка слагаемых). Эти проблемы могут быть устранены добавлением в нотацию формальных грамматик явного обозначения для повторения, что и будет сделано при рассмотрении грамматик языков программирования.

Синтаксические диаграммы также позволяют обойтись без рекурсии там, где она служит всего лишь для задания повторений.

Рассмотрим диаграмму нетерминала Е (см. рис. 25). Налицо рекурсия: на диаграмме Е присутствует нетерминальный блок Е. Но это правая, концевая рекурсия. Обращение к блоку Е можно заменить переходом на вход диаграмму (рис. 27, a).



Рис.27 Синтаксическая диаграмма выражения, не содержащая рекурсии

Такую диаграмму для выражения можно было построить и без выписывания и преобразования грамматики. Суть изображенного проста: выражение — это последовательность одного или более слагаемых, между которыми записываются знаки «+» или «-». Поскольку обработка слагаемых в ходе синтаксического анализа будет происходить последовательно в цикле, не составит труда организовать трансляцию так, чтобы это соответствовало выполнению операций слева направо.

Чтобы программировать распознаватель было удобней, преобразуем диаграмму в эквивалентную, содержащую цикл с предусловием (рис. 27, б).

Аналогично строится диаграмма слагаемого (нетерминал Т) (рис. 28, а).

Диаграмма множителя соответствует правилам для нетерминала М в грамматиках G8,

G13,G14G15(рис.28,б).



Рис.28. Синтаксические диаграммы слагаемого и множителя

Теперь можно программировать анализатор. Запишем только распознающие процедуры. Константы, переменные и вспомогательные процедуры предполагаются такими же, как в распознавателе многочленов.

Начать естественно с процедуры, соответствующей начальному нетерминалу, то есть с процедуры Е:

**procedure** Е;

**begin**

 Т;

 **while** Ch **in** [' + ', '-'] **do begin**

 NextCh;

 T;

 **end;**

**end;**

Далее записываем распознаватель слагаемого:

**procedure** Т;

**begin**

 М;

 **while** Ch **in** ['\*', '/'] **do begin**

 NextCh;

 M;

 **end;**

**end;**

И, наконец, распознающую процедуру для нетерминалам— множителя:

**procedure** M;

**begin**

 **if** Ch **in** [ 'a', 'b' , 'c'] **then**

 NextCh

 **else** **if** Ch = ' (' **then begin**

 NextCh;

 E;

  **if** Ch = ') ' **then**

 NextCh

 **else**

 Error('Ожидается ")"');

 **end**

 **else**

 Error ('Ожидается a, b, с или ' (');

**end**;

Остается только один вопрос: как разместить три эти процедуры в программе? По общему правилу языка Паскаль любой идентификатор может быть использован только после его описания. Процедура Е вызывает процедуру T, поэтому должна располагаться после нее. Процедура T вызывает M, поэтому M надо разместить перед T. Получается такой порядок: M; T; E. Но из процедуры M вызывается E, в то время как описание E мы разместили после M. Выходом из этой ситуации, которая возникла из-за наличия в нашей программе косвенной рекурсии, является использование директивы **forward** для опережающего описания процедуры E.

 Структура программы будет такой:

**procedure** E; **forward**;

**procedure** M;

**procedure** Т;

**procedure** Е;

Есть и другое, весьма изящное решение проблемы: использование вложенных процедур. Такой распознаватель представлен в листинге

Листинг. Распознаватель арифметических выражений

**procedure** Е;

 **procedure** T;

 **procedure** M;

 **begin**

 **if** Ch **in** ['а', 'b', 'с'] **then**

 NextCh

 **else** **if** Ch = '(' **then begin**

 NextCh;

 E;

 **if** Ch = ') ' **then**

 NextCh

 **else**

 Error('Ожидается ")"');

 **end**

 **else**

 Error('Ожидается a, b, с или "("');

 **end** {M};

 **begin**

 M;

 **while** Ch **in** ['\*', '/']

 NextCh;

 M;

 **end;**

**end** {Tj;

**begin**

 T;

 **while** Ch **in** [ ' + ', '-'] **do begin**

 NextCh;

 T;

**end**; { E }