

Тема 3. Восходящий синтаксический анализ

3.2. L-атрибутные СУО на основе LL-грамматики

L-атрибутные СУО на основе *LL*-грамматики обычно ориентированы на нисходящий синтаксический анализ. Однако их можно адаптировать и для реализации в процессе восходящего синтаксического анализа, поскольку *LL*-грамматики являются истинным подмножеством *LR*-грамматик.

Поскольку для восходящей трансляции все действия должны находиться в конце продукции, построение СУТ для реализации *L*-атрибутного СУО на основе *LL*-грамматики в процессе *LR*-разбора предполагает следующее [1]:

1. Строится СУТ в соответствии с правилами из подразд. 2.1, согласно которым действия по вычислению наследуемых атрибутов нетерминала вставляются перед этим нетерминалом, а действия по вычислению синтезируемых атрибутов размещаются в конце продукции.

2. В грамматику добавляются нетерминалы-маркеры вместо каждого вставленного действия. Каждое вставленное действие заменяется отдельным маркером M ; для каждого маркера M существует только одна продукция вида $M \rightarrow \varepsilon$.

3. Модифицируется действие a , заменяемое маркером M в некоторой продукции $A \rightarrow \alpha\{a\}\beta$, и с продукцией $M \rightarrow \varepsilon$ связывается действие a' :

а) копирующее в качестве наследуемых атрибутов M любые атрибуты A или символов из α , которые требуются действию a ;

б) вычисляющее атрибуты таким же способом, что и a , но делающее эти атрибуты синтезируемыми атрибутами M .

Заметим, что вместо наследуемых атрибутов M можно использовать промежуточные переменные, включив в действие a' побочные действия, копирующие в эти переменные необходимые атрибуты. Это связано с тем, что время их жизни завершается сразу же после вычисления синтезируемых атрибутов M .

Пусть имеется продукция СУТ на основе LL -грамматики:
 $A \rightarrow B \{ C.inh := f(A.inh, B.syn) \} C$.

В действии наследуемый атрибут $C.inh$ вычисляется как функция наследуемого атрибута $A.inh$ и синтезируемого атрибута $B.syn$. Добавим маркер M с синтезируемым атрибутом $M.syn$ (копия $C.inh$). Тогда фрагмент СУТ для этой продукции примет следующий вид (t_1 и t_2 – промежуточные переменные):

$$A \rightarrow B M C$$

$$M \rightarrow \varepsilon \{ t_1 := A.inh; t_2 := B.syn; M.syn := f(t_1, t_2) \}.$$

На первый взгляд подобное преобразование некорректно, поскольку действие в продукции $M \rightarrow \varepsilon$ должно иметь доступ к атрибутам, связанным с символами грамматики, которые отсутствуют в данной продукции. Однако если реализовать действия с использованием стека, необходимые атрибуты всегда будут доступны с помощью известных позиций в стеке. При этом не понадобятся наследуемые атрибуты M (или промежуточные переменные). Поэтому предварительно лучше детализировать действия с помощью стековых операций (этим как бы снимается зависимость атрибутов от символов продукций), а затем выполнить рассмотренное преобразование.

Рассмотрим СУТ для вычисления арифметического выражения, основанную на $LL(1)$ -грамматике (о преобразовании грамматики СУО в $LL(1)$ -форму см. разд. 4):

$$S \rightarrow E \perp \{Print(E.val)\}$$

$$E \rightarrow T \{X.inh := T.val\} X \{E.val := X.syn\}$$

$$X \rightarrow + T \{X_1.inh := X.inh + T.val\} X_1 \{X.syn := X_1.syn\}$$

$$X \rightarrow \varepsilon \{X.syn := X.inh\}$$

$$T \rightarrow F \{Y.inh := F.val\} Y \{T.val := Y.syn\}$$

$$Y \rightarrow * F \{Y_1.inh := Y.inh + F.val\} Y_1 \{Y.syn := Y_1.syn\}$$

$$Y \rightarrow \varepsilon \{Y.syn := Y.inh\}$$

$$F \rightarrow (E) \{F.val := E.val\}$$

$$F \rightarrow \mathbf{num} \{F.val := GetVal(\mathbf{num.pnt})\}.$$

После детализации действий соответствующими операциями со стеком (используется стек атрибутов) с учетом предотвращения хранения в стеке копий значений атрибутов для действий копирования СУТ примет следующий вид:

$$S \rightarrow E \perp \{Print(Pop(St))\}$$

$$E \rightarrow T X$$

$$X \rightarrow + T \{t_2 := Pop(St); t_1 := Pop(St); Push(t_1 + t_2, St)\} X_1$$

$$X \rightarrow \varepsilon$$

$$T \rightarrow F Y$$

$$Y \rightarrow * F \{t_2 := Pop(St); t_1 := Pop(St); Push(t_1 * t_2, St)\} Y_1$$

$$Y \rightarrow \varepsilon$$

$$F \rightarrow (E)$$

$$F \rightarrow \mathbf{num} \{Push(GetVal(\mathbf{num.pnt}), St)\}.$$

Для вставленных действий добавим маркеры M и N и продукции $M \rightarrow \varepsilon$ и $N \rightarrow \varepsilon$ с соответствующими действиями:

$$S \rightarrow E \perp \{Print(Pop(St))\}$$

$$E \rightarrow T X$$

$$X \rightarrow + T M X_1$$

$$X \rightarrow \varepsilon$$

$$T \rightarrow F Y$$

$$Y \rightarrow * F N Y_1$$

$$Y \rightarrow \varepsilon$$

$$F \rightarrow (E)$$

$$F \rightarrow \mathbf{num} \{Push(GetVal(\mathbf{num.pnt}), St)\}.$$

$$M \rightarrow \varepsilon \{t_2 := Pop(St); t_1 := Pop(St); Push(t_1 + t_2, St)\}$$

$$N \rightarrow \varepsilon \{t_2 := Pop(St); t_1 := Pop(St); Push(t_1 * t_2, St)\}.$$

В результате полученная СУТ может быть использована для реализации L -атрибутного СУО в процессе восходящего синтаксического анализа.

3.3. L-атрибутные СУО на основе LR-грамматики

Очень важно обратить внимание на то, что в общем случае *L*-атрибутное СУО на основе *LR*-грамматики, имеющее наследуемые атрибуты, нельзя реализовать в процессе восходящего синтаксического анализа [1]. Это не затрагивает *S*-атрибутные СУО (хотя они и являются истинным подмножеством *L*-атрибутных СУО), поскольку у них нет наследуемых атрибутов. Неформально это объясняется следующим образом.

Пусть в LR -грамматике имеется продукция $A \rightarrow BC$, в которой необходимо вычислить наследуемый атрибут $B.inh$, зависящий от наследуемого атрибута $A.inh$. К моменту свертки к B (после обхода всех сыновей B) во входном потоке еще не обработана подстрока, порождаемая символом C . Поэтому еще нет гарантии, что основой для свертки будет именно продукция $A \rightarrow BC$. Следовательно, и нет гарантии, что для вычисления атрибута $B.inh$ следует использовать правило, связанное именно с данной продукцией, а не с какой-то другой. Даже если подождать свертки к C , чтобы убедиться в том, что будет выполнена свертка BC к A , то все равно неизвестно значение атрибута $A.inh$ – в какой именно правой части какой продукции вычисляется $A.inh$. Такой процесс без возможности принять решение может быть продолжен до тех пор, пока не будет выполнен анализ всей входной строки. Таким образом, сначала придется построить дерево разбора, а на следующем проходе выполнить семантические правила для трансляции (возможно, через построение графа зависимостей). А это уже не является реализацией СУТ именно в процессе синтаксического анализа.

Даже при отсутствии наследуемых атрибутов не все СУТ могут быть реализованы в процессе синтаксического анализа. Для примера рассмотрим СУТ для преобразования инфиксных выражений в префиксную форму:

$$S \rightarrow E \perp$$

$$E \rightarrow \{Print(' + ') \} E_1 + T$$

$$E \rightarrow T$$

$$T \rightarrow \{Print(' * ') \} T_1 * F$$

$$T \rightarrow F$$

$$F \rightarrow (E)$$

$$F \rightarrow \mathbf{num} \{Print(GetLex(\mathbf{num.pnt}))\}.$$

Здесь функция *GetLex*(**num.pnt**) возвращает лексему числовой константы из соответствующей строки таблицы символов.

В этой СУТ нет атрибутов, используются только побочные действия. Использование нетерминалов-маркеров M и N для действий соответственно $\{Print('+')\}$ и $\{Print('*')\}$ и добавление продукций $M \rightarrow \varepsilon \{Print('+')\}$ и $N \rightarrow \varepsilon \{Print('*')\}$ приведет к тому, что при чтении входного символа **num** возникнет неразрешимый конфликт между сверткой по $M \rightarrow \varepsilon$ и по $N \rightarrow \varepsilon$.

Это связано с тем, что, например, для выполнения $Print('+')$ надо знать, что будет выполнена свертка $E_1 + T$ к E . Таким образом, имеет место неконтролируемое побочное действие, поскольку оно зависит от символов, стоящих справа от него, т. е. данная СУТ по сути не является L -атрибутной.

Для решения задачи преобразования инфиксной формы выражения в префиксную можно построить S -атрибутное СУО (с последующим преобразованием в постфиксную СУТ) на основе той же LR -грамматики, добавив соответствующие синтезируемые атрибуты и семантические правила их вычисления (табл. 5).

Таблица 5

S -атрибутное СУО преобразования инфиксной формы выражения в префиксную	
Продукция	Семантические правила
1) $S \rightarrow E \perp$	$Print(E.str)$
2) $E \rightarrow E_1 + T$	$E.str := '+' \parallel E_1.str \parallel T.str$
3) $E \rightarrow T$	$E.str := T.str$
4) $T \rightarrow T_1 * F$	$T.str := '*' \parallel T_1.str \parallel F.str$
5) $T \rightarrow F$	$T.str := F.str$
6) $F \rightarrow (E)$	$F.str := E.str$
7) $F \rightarrow \mathbf{num}$	$F.str := GetLex(\mathbf{num.pnt})$

В синтезируемом атрибуте str (типа строки, при реализации лучше использовать указатель на строку) формируется соответствующая строка префиксной формы. Символ \parallel обозначает операцию конкатенации строк.