

Тема 2. Схемы трансляции

Синтаксически управляемая схема трансляции (СУТ) представляет собой контекстно-свободную грамматику, дополненную программными фрагментами (*семантическими действиями*), вставленными в правые части продукций. Действия обычно заключаются в фигурные скобки и могут располагаться в любой позиции правой части продукции.

Отличие СУТ от СУО заключается в том, что в СУТ явно определен порядок вычисления семантических правил, задаваемый порядком обхода дерева разбора, который, в свою очередь, определяется методом синтаксического анализа. Кроме того, семантические действия в СУТ обычно более детализированы, чем семантические правила в СУО.

2.1. Преобразование *L*-атрибутного СУО в СУТ

Преобразование *L*-атрибутного СУО в СУТ для его реализации в процессе синтаксического анализа выполняется в соответствии со следующими правилами [1]:

- а) действие, которое вычисляет наследуемый атрибут нетерминала *A*, необходимо поместить непосредственно перед вхождением *A* в правую часть продукции;
- б) действие, вычисляющее синтезируемый атрибут нетерминала в левой части продукции, следует разместить в конце правой части продукции.

Построим СУТ для L -атрибутного СУО (см. табл. 2):

СУО с наследуемым атрибутом $L.inh$

Продукция	Семантические правила
1) $D \rightarrow T L$	$L.inh := T.type$
2) $T \rightarrow \mathbf{int}$	$T.type := \mathbf{integer}$
3) $T \rightarrow \mathbf{char}$	$T.type := \mathbf{char}$
4) $L \rightarrow L_1, \mathbf{id}$	$L_1.inh := L.inh$ $AddType(\mathbf{id}.ns, L.inh)$
5) $L \rightarrow \mathbf{id}$	$AddType(\mathbf{id}.ns, L.inh)$

- 1) $D \rightarrow T \{L.inh := T.type\} L$
- 2) $T \rightarrow \mathbf{int} \{T.type := \mathbf{integer}\}$
- 3) $T \rightarrow \mathbf{char} \{T.type := \mathbf{char}\}$
- 4) $L \rightarrow \{L_1.inh := L.inh\} L_1, \mathbf{id} \{AddType(\mathbf{id}.ns, L.inh)\}$
- 5) $L \rightarrow \mathbf{id} \{AddType(\mathbf{id}.ns, L.inh)\}$.

В продукции 1 действие $\{L.inh := T.type\}$ размещено непосредственно перед L , поскольку в нем вычисляется наследуемый атрибут $L.inh$ нетерминала L . По аналогичной причине в продукции 4 действие $\{L_1.inh := L.inh\}$ помещено перед L_1 . Остальные действия вычисляют синтезируемый атрибут $T.type$ или являются контролируемым побочным действием $AddType$, поэтому они размещены в конце правых частей продукций.

Когда семантические действия включают в себя множество различных операций, можно в СУТ для компактности записи использовать вместо действий их обозначения с соответствующей расшифровкой в виде алгоритмов выполнения действий. Обозначим действия в рассматриваемой СУТ следующим образом:

$A1: L.inh := T.type$

$A2: T.type := integer$

$A3: T.type := char$

$A4: L_1.inh := L.inh$

$A5: AddType(id.ns, L.inh).$

Тогда СУТ можно записать в виде

1) $D \rightarrow T \{A1\} L$

2) $T \rightarrow \mathbf{int} \{A2\}$

3) $T \rightarrow \mathbf{char} \{A3\}$

4) $L \rightarrow \{A4\} L_1, \mathbf{id} \{A5\}$

5) $L \rightarrow \mathbf{id} \{A5\}.$

Для СУТ символы действий включаются в дерево разбора как его узлы. Действия выполняются в порядке, соответствующем прохождению дерева в глубину. При необходимости дерево разбора можно аннотировать. Для рассматриваемой СУТ дерево разбора строки **char id₁, id₂** представлено на рис. 4.

- 1) $D \rightarrow T \{A1\} L$
- 2) $T \rightarrow \mathbf{int} \{A2\}$
- 3) $T \rightarrow \mathbf{char} \{A3\}$
- 4) $L \rightarrow \{A4\} L_1, \mathbf{id} \{A5\}$
- 5) $L \rightarrow \mathbf{id} \{A5\}$.

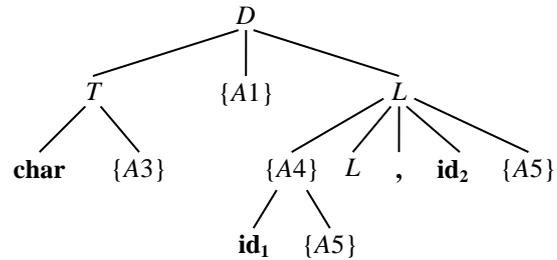


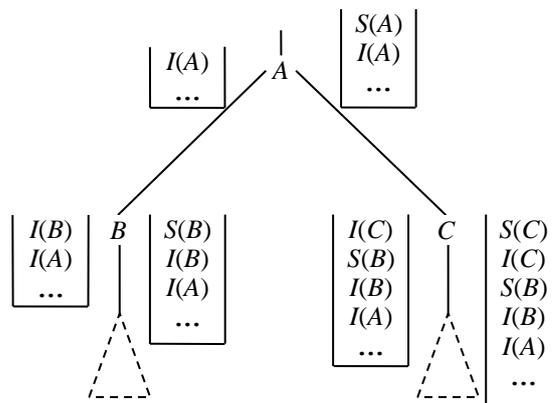
Рис. 4. Дерево разбора строки **char id₁, id₂** для СУТ

2.2. Память для хранения атрибутов

Важным является назначение памяти для хранения значений атрибутов в процессе трансляции. При заданном порядке вычисления атрибутов (зависит от порядка обхода дерева разбора) *время жизни* атрибута начинается, когда атрибут впервые вычисляется, и заканчивается, когда вычислены все атрибуты, зависящие от него. Для экономии памяти значения атрибутов сохраняются только на протяжении их времени жизни. Значения атрибутов помещаются в стек. Число и размер атрибутов символов грамматики зафиксировано, поэтому на каждом шаге процесса синтаксического анализа известно, в какой позиции стека находится интересующий атрибут. Можно разместить атрибуты в стеке синтаксического анализатора (расширив соответствующим образом структуру элемента стека) или использовать специальный стек или несколько стеков (например, отдельные стеки для синтезируемых и наследуемых атрибутов) для хранения значений атрибутов в течение времени их жизни.

Время жизни атрибута достаточно просто определяется по грамматике. Пусть имеется специальный стек для хранения атрибутов. Рассмотрим прохождение в глубину дерева разбора в соответствии с процедурой *DepthFirst* (см. подразд. 1.3). Обозначим наследуемые и синтезируемые атрибуты символа X грамматики как $I(X)$ и $S(X)$ соответственно.

Для продукции $A \rightarrow BC$ процесс обхода начинается в узле A . К этому моменту в родительском по отношению к A узле вычислены наследуемые атрибуты $I(A)$, которые находятся в верхней части стека. Вычисляются и заносятся в стек значения наследуемых атрибутов $I(B)$. После завершения обхода поддерева с корнем B в стеке над наследуемыми атрибутами $I(B)$ будут находиться синтезируемые атрибуты $S(B)$. Аналогично процесс повторяется для поддерева с корнем C , т. е. в стек заносятся наследуемые атрибуты $I(C)$ и после завершения обхода поддерева с корнем C в стек будут занесены синтезируемые атрибуты $S(C)$. Таким образом, к моменту возврата к узлу A в стеке будут находиться значения атрибутов $I(A)$, $I(B)$, $S(B)$, $I(C)$, $S(C)$. Все атрибуты, необходимые для вычисления синтезируемых атрибутов A , в данный момент находятся в верхней части стека, их число и позиции в стеке известны. После вычисления синтезируемых атрибутов A время жизни атрибутов $I(B)$, $S(B)$, $I(C)$, $S(C)$ заканчивается, поэтому обход поддерева с корнем A завершается со стеком, содержащим в верхней части $I(A)$, $S(A)$.



Рассмотрим хранение атрибутов при обходе поддерева для продукции

$$L \rightarrow \{L_1.inh := L.inh\} L_1, \mathbf{id} \{AddType(\mathbf{id}.ns, L.inh)\}$$

из СУТ из подразд. 2.1. Изменение содержимого стека атрибутов в процессе обхода узлов дерева разбора представлено на рис. 5. Слева от узла показано содержимое стека до посещения узла, а справа – после посещения.

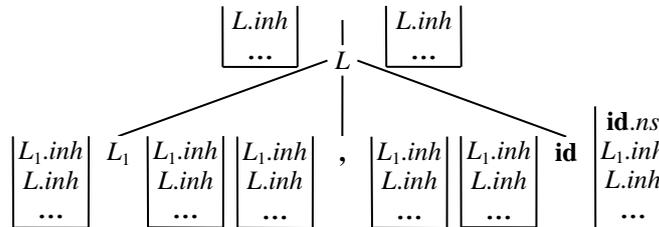


Рис. 5. Содержимое стека атрибутов в процессе обхода узлов

Непосредственно перед посещением узла L в вершине стека находится значение его наследуемого атрибута $L.inh$. До посещения узла L_1 вычисляется его наследуемый атрибут $L_1.inh$ согласно действию $L_1.inh := L.inh$ и заносится в стек. После обхода поддерева, корнем которого является L_1 , следовало бы занести значения его синтезируемых атрибутов, но, поскольку их нет, содержимое стека остается неизменным. Терминал "," не имеет атрибутов, поэтому содержимое стека не изменяется. Терминал \mathbf{id} имеет только синтезируемый атрибут $\mathbf{id}.ns$, поэтому он добавляется в стек после посещения этого узла. После обхода поддерева с корнем L выполняется действие $AddType(\mathbf{id}.ns, L.inh)$, при этом время жизни атрибутов $L_1.inh$

и $\mathbf{id}.ns$ завершается (они исключаются из стека). Поскольку у L нет синтезируемых атрибутов, в вершине стека остается наследуемый атрибут $L.inh$.

В СУТ время жизни некоторых атрибутов может завершиться на более ранних этапах. Поэтому нет необходимости в их сохранении в стеке после вычисления значений зависящих от них атрибутов. В нашем примере время жизни атрибута $L_1.inh$ завершается после обхода поддерева с корнем L_1 . Это объясняется тем, что от атрибута $L_1.inh$ не зависят значения атрибутов символов продукции $L \rightarrow L_1$, \mathbf{id} , стоящих справа от L_1 . Поэтому нет необходимости в его сохранении в стеке после обхода поддерева с корнем L_1 .

Заметим, что атрибут $\mathbf{id}.ns$ заносится в стек и сразу же удаляется из стека. Поэтому можно обойтись без занесения этого атрибута в стек.

Если некоторый атрибут b определяется правилом копирования вида $b := c$, а значение c находится в вершине стека, то в ряде случаев можно обойтись без размещения в стеке копии c . В нашем примере таким правилом является $L_1.inh := L.inh$, т.е. можно обойтись без занесения в стек атрибута $L_1.inh$.

Изменение содержимого стека атрибутов в процессе обхода узлов дерева разбора, учитывающее досрочное завершение времени жизни атрибутов и наличие правил копирования для той же продукции, что и на рис. 5, представлено на рис. 6.

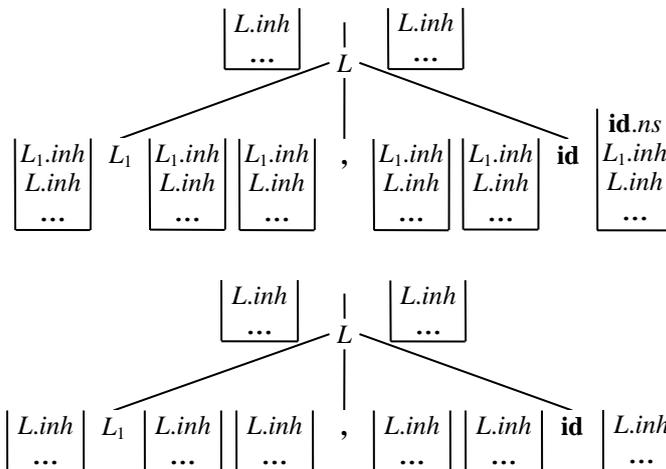


Рис. 6. Содержимое стека атрибутов с учетом досрочного завершения времени жизни и наличия правил копирования

Проведя подобный анализ для всех продукций СУТ, можно детализировать действия, связанные с вычислением или использованием атрибутов, соответствующими операциями со стеком.

После детализации действий соответствующими операциями со стеком (используется отдельный стек атрибутов) СУТ из подразд. 2.1 примет следующий вид:

- 1) $D \rightarrow T \{L.inh := T.type\} L$
- 2) $T \rightarrow \mathbf{int} \{T.type := \mathbf{integer}\}$
- 3) $T \rightarrow \mathbf{char} \{T.type := \mathbf{char}\}$
- 4) $L \rightarrow \{L_1.inh := L.inh\} L_1, \mathbf{id} \{AddType(\mathbf{id}.ns, L.inh)\}$
- 5) $L \rightarrow \mathbf{id} \{AddType(\mathbf{id}.ns, L.inh)\}$.

- 1) $D \rightarrow T L \{Pop(St)\}$
- 2) $T \rightarrow \mathbf{int} \{Push(\mathbf{integer}, St)\}$
- 3) $T \rightarrow \mathbf{char} \{Push(\mathbf{char}, St)\}$
- 4) $L \rightarrow L_1, \mathbf{id} \{AddType(\mathbf{id}.ns, StackTop(St))\}$
- 5) $L \rightarrow \mathbf{id} \{AddType(\mathbf{id}.ns, StackTop(St))\}$.

Здесь операция $Push(x, St)$ размещает значение x в стеке St , функция $Pop(St)$ исключает элемент из вершины стека и возвращает его значение, функция $StackTop(St)$ возвращает значение элемента из вершины стека St без его исключения.

В конце правой части первой продукции добавлено действие $Pop(St)$, которое после завершения разбора входной строки исключает из стека единственный оставшийся элемент и делает стек пустым. Заметим, что в исходной СУТ такого действия не было. Если нет требования, чтобы после разбора входной

строки стек атрибутов должен быть пустым, это действие можно не добавлять. Это действие не нужно также в случае использования для хранения атрибутов стека синтаксического анализатора, использующего метод синтаксического анализа, для которого одним из критериев успешного завершения разбора является пустота стека анализатора.

Использование СУТ для реализации СУО в процессе синтаксического анализа имеет ряд особенностей, зависящих от применяемого метода синтаксического анализа.