Тема 7. Адресация элементов массива (разделы Ахо 6.4.3, 6.4.4)

7.1. Общие сведения

Быстрый доступ к элементам массива легко осуществляется, когда элементы хранятся в блоке из последовательных ячеек памяти. Размер каждого элемента массива обозначим через d, тогда i-й элемент массива A начинается с адреса

$$base + (i - low) \times d, \qquad (1)$$

где low – нижняя граница индекса, а $base$ – относительный адрес памяти, выделенной под массив, т. е. относительный адрес $A[low]$. Выражение (1) может быть частично вычислено во время компиляции, если переписать его как

$$i \times d + (base - low \times d).$$

Подвыражение $c = base - low \times d$ можно вычислить в момент обработки объявления массива. Полагая, что величина c хранится в записи таблицы символов для A, относительный адрес $A[i]$ мы получим просто прибавлением $i \times d$ к c.

Предвычисления в процессе компиляции могут быть применены и для вычисления адресов элементов многомерных массивов. Двумерный массив обычно хранится либо по строкам, либо по столбцам. В случае двумерного массива, хранимого по строкам, относительный адрес $A[i_1, i_2]$ может быть вычислен по формуле

$$base + ((i_1 - low_1) \times n_2 + i_2 - low_2) \times d,$$

где low_1 и low_2 – нижние границы значений i_1 и i_2, а n_2 – число значений, которые может принимать i_2. Таким образом, если $high_2$ – верхняя граница значения i_2, то $n_2 = high_2 - low_2 + 1$. Полагая, что в процессе компиляции неизвестны только i_1 и i_2, можно переписать приведенное выше выражение следующим образом.

$$((i_1 \times n_2) + i_2) \times d + (base - ((low_1 \times n_2) + low_2) \times d) \qquad (2)$$

Второе слагаемое этого выражения (выделено красным цветом) можно вычислить в процессе компиляции.

Можно обобщить хранение по строкам на случай многомерного массива. Обобщение состоит в хранении элементов массива таким образом, что при последовательном сканировании блока памяти самый правый индекс изменяется наиболее быстро. Выражение (2) обобщается в этом случае в следующее выражение для относительного адреса $A[i_1, i_2, \ldots, i_k]$.

$$((\ldots((i_1 \times n_2 + i_2) \times n_3 + i_3)\ldots) \times n_k + i_k) \times d +$$
$$+ base - ((\ldots((low_1 \times n_2 + low_2) \times n_3 + low_3)\ldots) \times n_k + low_k) \times d \qquad (3)$$

Поскольку для всех j $n_j = high_j - low_j + 1$ считается фиксированным, второе слагаемое в (3) (выделено красным цветом) может быть вычислено компилятором и сохранено в записи таблицы символов для A.

Обозначим второе слагаемое через c:

$$c = base - ((\ldots((low_1 \times n_2 + low_2) \times n_3 + low_3)\ldots) \times n_k + low_k) \times d.$$

Следует заметить, что если $low = 0$ для всех измерений массива (как, например, в языке программирования C), то $c = base$.

7.2. Объявление массива

Вернемся к материалу из раздела 5.4 (СУО для проверки типов) в части объявления массива.

Рассмотрим продукции, используемые для объявления массива (для удобства нетерминал *SmpType* переименовал в *Ind* и терминал **rng** обозначил через **..**):

ArrType → **arr [** *LstInd* **] of** *Type*
LstInd → *Ind* | *Ind* **,** *LstInd*
Ind → **id** | **num .. num**

Эти продукции не подходят для применения конструктора типа *array*(*I*, *T*), определяющего массив с элементами типа *T* и множеством индексов *I*, представляющим собой диапазон целых чисел, а также для вычисления размера массива (путем умножения размера элемента на количество элементов) и числа измерений. Проблема решается, если представить указанные продукции следующим образом:

ArrType → **arr [** *LstInd*
LstInd → *Ind* **] of** *Type* | *Ind* **,** *LstInd*
Ind → **id** | **num .. num**

Немного модифицировал СУО (учел указанные выше переименования, а также убрал семантические правила, связанные с проверкой типов для продукций 4 и 5). Это СУО для массива определяет тип (*type*), размер типа (*width*), число измерений (*ndim*) как синтезируемые атрибуты нетерминалов:

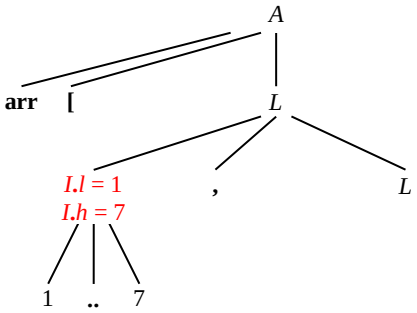
Продукция	Семантические правила
1) *ArrType* → **arr [** *LstInd*	*ArrType*.*type* := *LstInd*.*type*; *ArrType*.*width* := *LstInd*.*width*; *ArrType*.*ndim* := *LstInd*.*ndim*
2) *LstInd* → *Ind* **]** **of** *Type*	*m* := *Ind*.*low*; *n* := *Ind*.*high*; *LstInd*.*type* := *array*(*m*..*n*, *Type*.*type*); *LstInd*.*width* := (*n* – *m* + 1) * *Type*.*width*; **if** *Type*.*type* = *arr* **then** //если тип массив *LstInd*.*ndim* := *Type*.*ndim* + 1 **else** *LstInd*.*ndim* := 1
3) *LstInd* → *Ind* **,** $LstInd_1$	*m* := *Ind*.*low*; *n* := *Ind*.*high*; *LstInd*.*type* := *array*(*m*..*n*, $LstInd_1$.*type*); *LstInd*.*width* := (*n* – *m* + 1) * $LstInd_1$.*width*; *LstInd*.*ndim* := $LstInd_1$.*ndim* + 1
4) *Ind* → **id**	*Ind*.*low* := *GetLow*(**id.***pnt*); *Ind*.*high* := *GetHigh*(**id.***pnt*);
5) *Ind* → $\mathbf{num}_1$ **..** $\mathbf{num}_2$	*Ind*.*low* := $\mathbf{num}_1$.*val*; *Ind*.*high* := $\mathbf{num}_2$.*val*

Желтым цветом выделены правила, которые легко можно включить в конструктор типа *array*. Здесь они показаны, чтобы акцентировать на них внимание. Более того, если атрибут *type* сделать как запись (структура), включив туда поля для атрибутов *width* и *ndim* и возложив их вычисление на конструктор *array*, СУО существенно упростится.

Синтезируемые атрибуты *low* и *high* нетерминала *Ind* предназначены для хранения соответственно нижней и верхней границ диапазона. Функции *GetLow* и *GetHigh* предоставляют соответственно значения нижней и верхней границ диапазона из записи таблицы символов, на которую указывает **id**.*pnt*. Терминал **num** имеет синтезируемый атрибут *val*, представляющий собой значение числовой константы (доступ к типу и значению константы обеспечивается по значению атрибута токена, указывающего на соответствующую запись в таблице символов).

Рассмотрим пример аннотированного дерева разбора для массива (размер целого типа – 4 байта) **arr [** 1 **..** 7 **,** 5 **..** 9 **,** 3 **..** 5 **] of** int

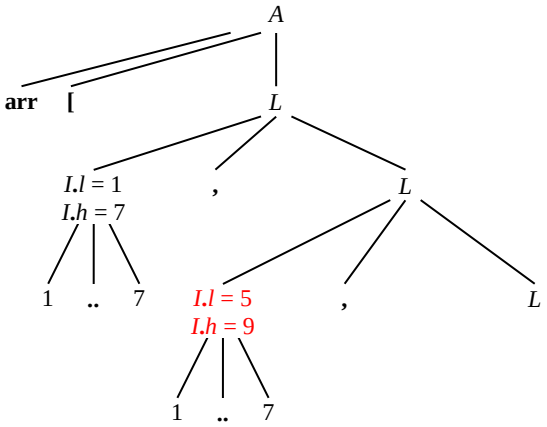
На аннотированном дереве разбора нетерминалы и атрибуты для удобства обозначены: *ArrType – A, LstInd – L, Ind – I, Type – T, type – t, width – w, ndim – nd*, конструктор типа массив *array – a*, целый тип int – *i*.



arr [1 **rng** 7 **,** 5 **rng** 9 **,** 3 **rng** 5 **] of** int

Продукция	Семантические правила
1) *ArrType* → **arr [** *LstInd*	*ArrType*.*type* := *LstInd*.*type*; *ArrType*.*width* := *LstInd*.*width*; *ArrType*.*ndim* := *LstInd*.*ndim*
3) *LstInd* → *Ind* **,** $LstInd_1$	*l* := *Ind*.*low*; *h* := *Ind*.*high*; *LstInd*.*type* := *array*(*l*..*h*, $LstInd_1$.*type*); *LstInd*.*width* := (*h* – *l* + 1) * $LstInd_1$.*width*; *LstInd*.*ndim* := $LstInd_1$.*ndim* + 1
5) *Ind* → $\mathbf{num}_1$ **..** $\mathbf{num}_2$	*Ind*.*low* := $\mathbf{num}_1$.*val*; *Ind*.*high* := $\mathbf{num}_2$.*val*

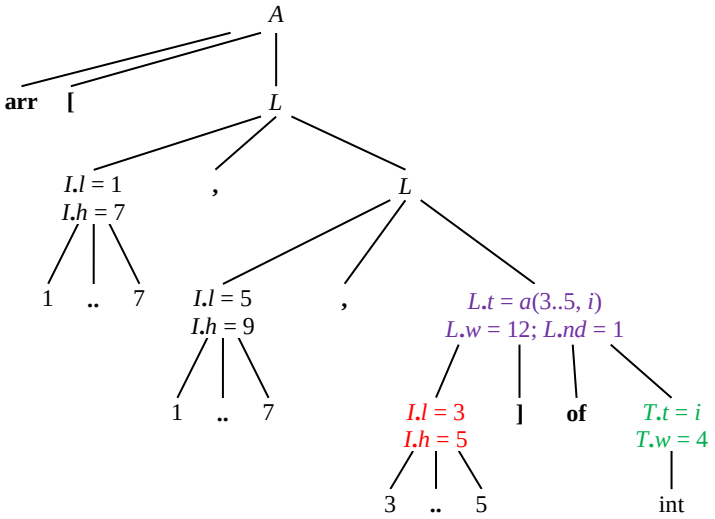
В соответствии с продукцией 5 устанавливаются значения *Ind*.*low* = 1 и *Ind*.*high* = 7.



arr [1 **rng** 7 **,** 5 **rng** 9 **,** 3 **rng** 5 **] of** int

Продукция	Семантические правила
3) $LstInd \rightarrow Ind$ **,** $LstInd_1$	$l := Ind.low$; $h := Ind.high$; $LstInd.type := array(l..h, LstInd_1.type)$; $LstInd.width := (h - l + 1) * LstInd_1.width$; $LstInd.ndim := LstInd_1.ndim + 1$
5) $Ind \rightarrow$ **num**$_1$ **..** **num**$_2$	$Ind.low :=$ **num**$_1$.*val*; $Ind.high :=$ **num**$_2$.*val*

В соответствии с продукцией 5 устанавливаются значения *Ind.low* = 5 и *Ind.high* = 9.



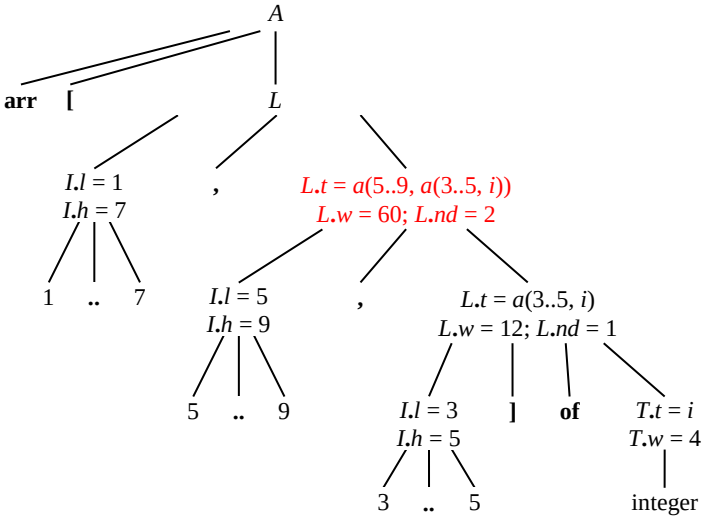
arr [1 **rng** 7 **,** 5 **rng** 9 **,** 3 **rng** 5 **] of** int

Продукция	Семантические правила
2) *LstInd* → *Ind* **]** **of** *Type*	*l* := *Ind*.*low*; *h* := *Ind*.*high*; *LstInd*.*type* := *array*(*l*..*h*, *Type*.*type*); *LstInd*.*width* := (*h* – *l* + 1) * *Type*.*width*; **if** *Type*.*type* = *arr* **then** //если тип массив *LstInd*.*ndim* := *Type*.*ndim* + 1 **else** *LstInd*.*ndim* := 1
5) *Ind* → **num**$_1$ **..** **num**$_2$	*Ind*.*low* := **num**$_1$.*val*; *Ind*.*high* := **num**$_2$.*val*

В соответствии с продукцией 5 устанавливаются значения *Ind*.*low* = 3 и *Ind*.*high* = 5.

В соответствии с продукцией *T* → **id** (**id** – имя типа, в СУО продукция не показана) устанавливаются значения *Type*.*type* = int и *Type*.*width* = 4.

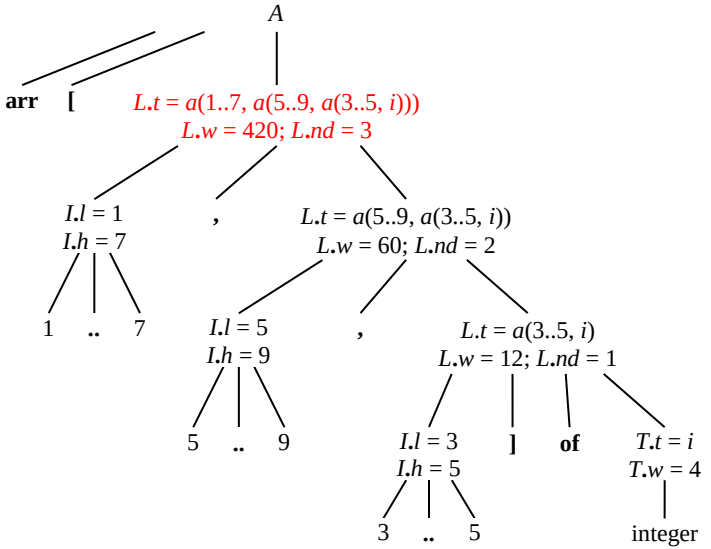
В соответствии с продукцией 2 конструктор типа устанавливает тип *LstInd*.*type* = *array*(3..5, int), вычисляется *LstInd*.*width* = (5 – 3 + 1) * 4 = 12 и устанавливается *LstInd*.*ndim* = 1.



arr [1 **rng** 7 **,** 5 **rng** 9 **,** 3 **rng** 5 **] of** int

Продукция	Семантические правила
3) *LstInd* → *Ind* **,** *LstInd*$_1$	*l* := *Ind*.*low*; *h* := *Ind*.*high*; *LstInd*.*type* := *array*(*l*..*h*, *LstInd*$_1$.*type*); *LstInd*.*width* := (*l* – *h* + 1) * *LstInd*$_1$.*width*; *LstInd*.*ndim* := *LstInd*$_1$.*ndim* + 1

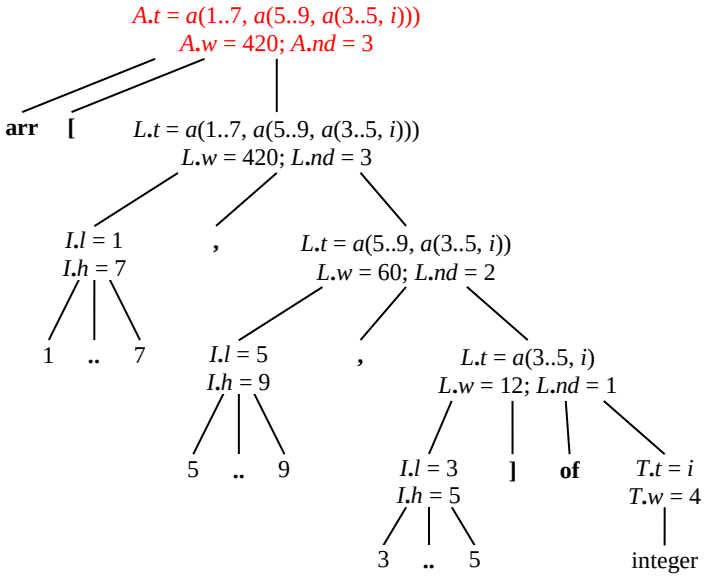
В соответствии с продукцией 3 конструктор типа устанавливает тип *LstInd*.*type* = *array*(5..9, *array*(3..5, int)), вычисляется *LstInd*.*width* = (9 – 5 + 1) * 12 = 60 и устанавливается *LstInd*.*ndim* = 1 + 1 = 2.



arr [1 **rng** 7 **,** 5 **rng** 9 **,** 3 **rng** 5 **] of** int

Продукция	Семантические правила
3) *LstInd* → *Ind* **,** *LstInd*$_1$	*l* := *Ind*.*low*; *h* := *Ind*.*high*; *LstInd*.*type* := *array*(*l*..*h*, *LstInd*$_1$.*type*); *LstInd*.*width* := (*l* – *h* + 1) * *LstInd*$_1$.*width*; *LstInd*.*ndim* := *LstInd*$_1$.*ndim* + 1

В соответствии с продукцией 3 конструктор типа устанавливает тип *LstInd*.*type* = *array*(1..7, *array*(5..9, *array*(3..5, int))), вычисляется *LstInd*.*width* = (7 – 1 + 1) * 60 = 420 и устанавливается *LstInd*.*ndim* = 2 + 1 = 3.



arr [1 **rng** 7 **,** 5 **rng** 9 **,** 3 **rng** 5 **] of** int

Продукция	Семантические правила
1) *ArrType* → **arr [** *LstInd*	*ArrType*.*type* := *LstInd*.*type*; *ArrType*.*width* := *LstInd*.*width*; *ArrType*.*ndim* := *LstInd*.*ndim*

В соответствии с продукцией 1 типом массива *ArrType*.*type* становится *LstInd*.*type* = *array*(1..7, *array*(5..9, *array*(3..5, int))), размером типа *ArrType*.*width* становится *LstInd*.*width* = 420, числом измерений *ArrType*.*ndim* становится *LstInd*.*ndim* = 3.

Для завершения объявления массива осталось вычислить

$c = base - ((\ldots((low_1 \times n_2 + low_2) \times n_3 + low_3)\ldots) \times n_k + low_k) \times d.$

Например, если имеется объявление массивов a1**,** a2 **:** *ArrType*, то для этих массивов вычитаемое в приведенном выражении будет одним и тем же. Отличие только в адресе памяти *base*, выделенной под массив (адрес первого элемента массива). Поэтому завершить объявление массива можно, вычислив только вычитаемое (обозначим его через *s*), а затем в процессе распределения памяти установить значения $c = base - s$ для каждого массива из списка идентификаторов и поместить в соответствующие записи в таблице символов. Для этого семантическое правило для вычисления *s* как синтезируемого атрибута нетерминала *ArrType* следует добавить в продукцию *ArrType* → **arr [** *LstInd* рассмотренного выше СУО (функция *CalcS* возвращает значение вычитаемого *s*):

Продукция	Семантические правила
1) *ArrType* → **arr [** *LstInd*	*ArrType*.*type* := *LstInd*.*type*; *ArrType*.*width* := *LstInd*.*width*; *ArrType*.*ndim* := *LstInd*.*ndim*; *ArrType*.*s* := *CalcS*()

Рассмотрим вычисление $s = ((\ldots((low_1 \times n_2 + low_2) \times n_3 + low_3)\ldots) \times n_k + low_k) \times d$.

Конструктор *array* (*I*, *T*) определяет массив элементов типа *T* и множество индексов *I*. Для массива **arr [** 1 **..** 7 **,** 5 **..** 9 **,** 3 **..** 5 **] of** int множество индексов со всеми параметрами можно представить следующим образом (*i* – номер индекса, *low* и *high* – соответственно нижняя и верхняя границы диапазона, *n* – число элементов в массиве, *width* – размер массива, *elwidth* – размер элемента массива):

i	*low*	*high*	*n*	*width*	*elwidth*
1	1	7	7	420	60
2	5	9	5	60	12
3	3	5	3	12	4

Вместо поля *elwidth* лучше сделать поле типа элемента массива (например, *eltype*), из которого будут доступны все данные о типе элемента (включая и *elwidth*).

Тогда алгоритм вычисления величины *s* функцией *CalcS* можно представить следующим образом (*ndim* – число измерений):

$s := A[1].low$
for $i := 2$ **to** $ndim$ **do**
  $s := s \times A[i].n + A[i].low$
$s := s \times A[ndim].elwidth$

Для нашего примера функция *CalcS* вернет значение *s* = 132.

Следует обратить внимание на то, что рассмотренное выше СУО формирует это множество индексов в обратном порядке. Это надо учесть при практической реализации вычисления *s*!

7.3. Схема трансляции для адресации элементов массива

Основная проблема при генерации кода для работы с элементами массива состоит в том, чтобы связать данное вычисление с грамматикой. Пусть имеются следующие продукции для работы с элементами массива (*E* – нетерминал для выражения)

$L \rightarrow \mathbf{id}\ [\ Elist\ ] \mid \mathbf{id}$

$Elist \rightarrow Elist\ ,\ E \mid E$

Для того чтобы при группировании выражений индексов в *Elist* были допустимы различные пределы n_j у разных размерностей массива, следует переписать указанные продукции следующим образом.

$L \rightarrow Elist] \mid \mathbf{id}$

$Elist \rightarrow Elist\ ,\ E \mid \mathbf{id}\ [\ E$

Таким образом, имя массива в процессе формирования *L* присоединяется к левому выражению индекса, а не к *Elist*. Эти продукции позволяют передать в качестве синтезируемого атрибута *Elist.array* указатель на запись в таблице символов для имени массива.

Нетерминал *Elist* имеет также атрибут *Elist.ndim* для записи количества размерностей в *Elist*. Функция *limit*(*array*, *j*) возвращает n_j, количество элементов *j*-й размерности массива, на запись которого в таблице символов указывает *array*. И наконец, *Elist.addr* обозначает временную переменную, хранящую значение, вычисленное по индексному выражению в *Elist*.

Elist, порождающий первые *m* индексов ссылки на элемент *k*-мерного массива $A[i_1, i_2, \ldots, i_k]$, будет генерировать трехадресный код для вычисления

$$(\ldots((i_1 n_2 + i_2)n_3 + i_3)\ldots)n_m + i_m, \qquad (4)$$

используя рекуррентные соотношения

$$e_1 = i_1$$

$$e_m = e_{m-1} \times n_m + i_m \qquad (5)$$

Таким образом, когда $m = k$, все, что нужно для вычисления члена в первой строке (3), – это умножение на размер *d*. Заметим, что здесь i_j представляет значение выражения, и код для его вычисления внедрен в код для вычисления (4).

Нетерминал *L* имеет два атрибута – *L.addr* и *L.offset* (*offset* – смещение). Если *L* представляет собой простое имя, *L.addr* является указателем на запись в таблице символов для этого имени, а *L.offset* имеет значение **null**, указывающее, что *L* – простое имя, а не ссылка на элемент массива.

Семантические действия будут добавляться к следующей грамматике.

(1) $S \to L := E$ (5) $L \to Elist$]

(2) $E \to E + E$ (6) $L \to$ **id**

(3) $E \to (E)$ (7) $Elist \to Elist , E$

(4) $E \to L$ (8) $Elist \to$ **id** [E

Используется процедура *Gen* для формирования трехадресной команды.

Мы генерируем обычное присвоение, если L представляет собой простое имя, и индексированное присвоение по адресу, обозначенному L, в противном случае.

(1) $S \to L := E$ {**if** *L.offset* = **null then** /* *L* – простое имя */
Gen(*L.addr* ′:=′ *E.addr*)
else *Gen*(*L.addr* ′ [′ *L.offset* ′]′ ′:=′ *E.addr*)}

Код для арифметических выражений:

(2) $E \to E_1 + E_2$ {*E.addr* := *NewTemp*()
Gen(*E.addr* ′:=′ E_1*.addr* ′+′ E_2*.addr*)}

(3) $E \to (E_1)$ {*E.addr* := E_1*.addr*}

Когда ссылка на массив *L* свертывается в *E*, требуется значение *L* как значение элемента массива. Таким образом, необходимо использовать индексацию для получения содержимого ячейки памяти *L.addr*[*L.offset*].

(4) $E \to L$ {**if** *L.offset* = **null then** /* *L* – простое имя */
E.addr := *L.addr*
else begin
E.addr := *NewTemp*()
Gen(*E.addr* ':=' *L.addr* '[' *L.offset* ']')
end}

Ниже *L.offset* является новой временной переменной, представляющей первое слагаемое в (3); функция *width*(*Elist.array*) возвращает значение *d* в (3). *L.addr* представляет $c = base - ((\ldots((low_1 \times n_2 + low_2) \times n_3 + low_3)\ldots) \times n_k + low_k) \times d$, т. е. второе слагаемое в (3), возвращаемое функцией *c*(*Elist.array*).

$$((\ldots((i_1 \times n_2 + i_2) \times n_3 + i_3)\ldots) \times n_k + i_k) \times d +$$
$$+ base - ((\ldots((low_1 \times n_2 + low_2) \times n_3 + low_3)\ldots) \times n_k + low_k) \times d \qquad (3)$$

(5) $L \rightarrow Elist\ ]$ {*L.addr* := *NewTemp*()
L.offset := *NewTemp*()
Gen(*L.addr* ':=' *c*(*Elist.array*))
Gen(*L.offset* ':=' *Elist.addr* '*' *width*(*Elist.array*))}

Нулевое смещение указывает на простое имя.

(6) $L \rightarrow$ **id** {*L.addr* := **id.***pnt*
L.offset := **null**}

При рассмотрении следующего индексного выражения применяем рекуррентное соотношение (5).

$$e_1 = i_1$$
$$e_m = e_{m-1} \times n_m + i_m \qquad (5)$$

В следующем действии $Elist_1.addr$ соответствует e_{m-1} из (5), а $Elist.addr$ – e_m. Заметим, что если $Elist_1$ имеет $m - 1$ компонент, то $Elist$ в левой части продукции имеет m компонентов.

(7) $Elist \rightarrow Elist_1 , E$ {$t := NewTemp()$
$m := Elist_1.ndim + 1$
$Gen(t\ ':='\ Elist_1.addr\ '*'\ limit(Elist_1.array, m))$
$Gen(t\ ':='\ t\ '+'\ E.addr)$
$Elist.array := Elist_1.array$
$Elist.addr := t$
$Elist.ndim := m$}

$(\ldots((i_1 n_2 + i_2)n_3 + i_3)\ldots)n_m + i_m,$ (4)

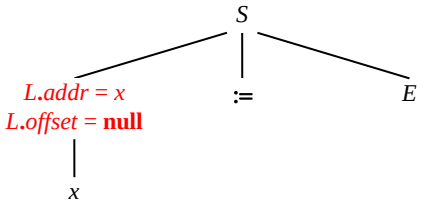
E.addr хранит как значение выражения *E*, так и значение (4) для $m = 1$ (при $m = 1$ эти значения, по сути, одинаковы).

(8) *Elist* → **id** [*E* {*Elist.array* := **id.***pnt*
Elist.addr := *E.addr*
Elist.ndim := 1}

Рассмотрим пример.

Пусть массив определен как A **: arr [** 3 **..** 6 **,** 1 **..** 7 **] of** int, т. е. *A* – массив целых чисел размером 4×7 с $low_1 = 3$, $low_2 = 1$, $n_1 = 4$, $n_2 = 7$. Примем размер *d* целого числа равным 4, тогда размер массива равен 112. Для такого массива константа $c = base_A - 88$.

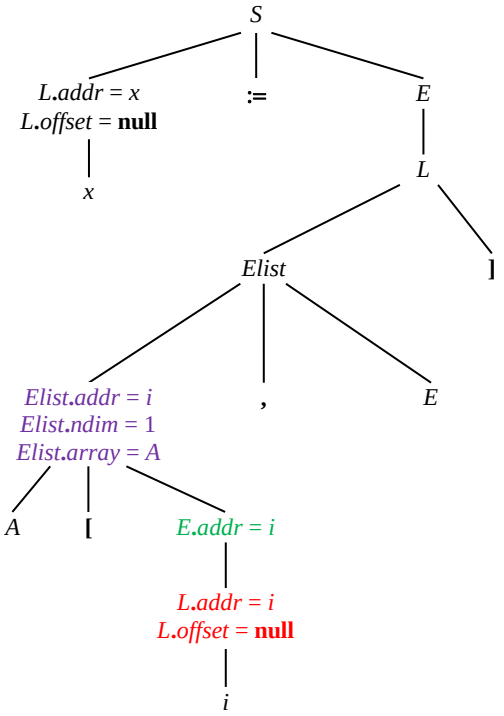
Рассмотрим аннотирование дерева разбора для присвоения $x := A[i, j]$. Отсчет номеров позиций формируемых трехадресных инструкций начнем с 50.



$x := A[i, j]$

(1)	$S \rightarrow L$ **:=** E	{**if** *L*.*offset* = **null then** /* *L* – простое имя */ *Gen*(*L*.*addr* ':=' *E*.*addr*) **else** *Gen*(*L*.*addr* ' [' *L*.*offset* ']' ':=' *E*.*addr*)}
(6)	$L \rightarrow$ **id**	{*L*.*addr* := **id**.*pnt* *L*.*offset* := **null**}

В соответствии с продукцией 6 устанавливаются значения атрибутов *addr* и *offset* для нетерминала *L*.



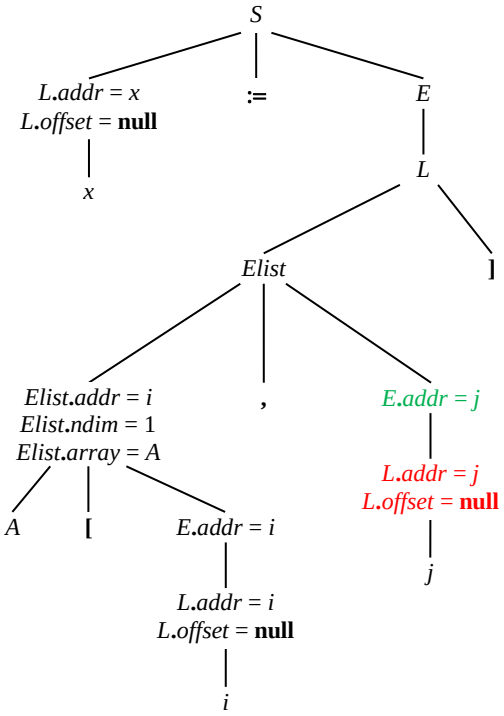
$x := A[i, j]$

(4)	$E \to L$	{**if** *L.offset* = **null then** /* *L* – простое имя */ *E.addr* := *L.addr* **else begin** *E.addr* := *NewTemp*() *Gen*(*E.addr* ':=' *L.addr* '[' *L.offset* ']') **end**}
(5)	$L \to Elist$]	
(6)	$L \to$ **id**	{*L.addr* := **id.***pnt* *L.offset* := **null**}
(7)	$Elist \to Elist_1$, E	
(8)	$Elist \to$ **id** [E	{*Elist.array* := **id.***pnt* *Elist.addr* := *E.addr* *Elist.ndim* := 1}

В соответствии с продукцией 6 устанавливаются значения атрибутов *addr* и *offset* для нетерминала *L*.

В соответствии с продукцией 4 устанавливается значение атрибута *addr* для нетерминала *E*.

В соответствии с продукцией 8 устанавливаются значения атрибутов *array*, *addr* и *ndim* для нетерминала *Elist*.

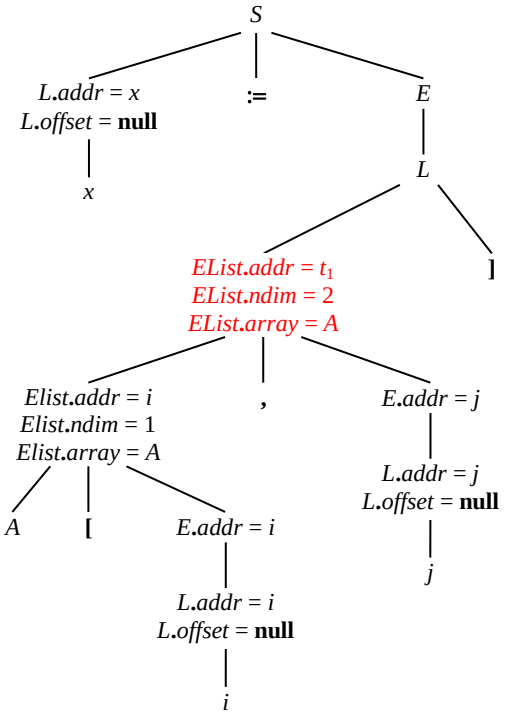


$x := A[i, j]$

(4)	$E \to L$	{**if** *L*.*offset* = **null then** /* *L* – простое имя */ *E*.*addr* := *L*.*addr* **else begin** *E*.*addr* := *NewTemp*() *Gen*(*E*.*addr* ':=' *L*.*addr* '[' *L*.*offset* ']') **end**}
(6)	$L \to$ **id**	{*L*.*addr* := **id**.*pnt* *L*.*offset* := **null**}

В соответствии с продукцией 6 устанавливаются значения атрибутов *addr* и *offset* для нетерминала *L*.

В соответствии с продукцией 4 устанавливается значение атрибута *addr* для нетерминала *E*.



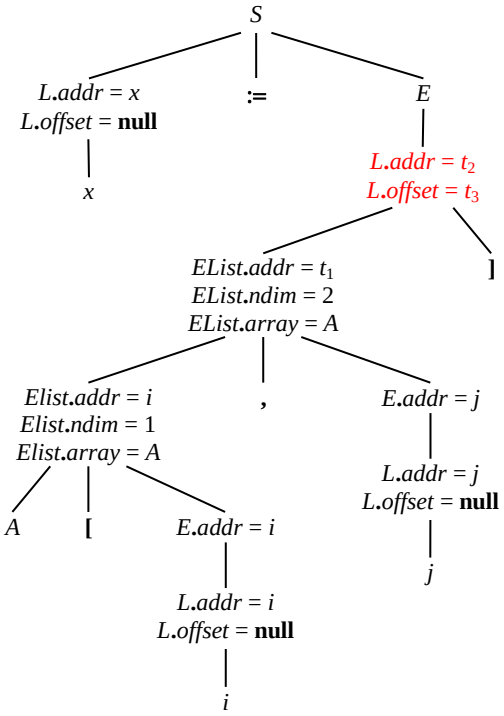
$x := A[i, j]$

(7)	$Elist \rightarrow Elist_1\ ,\ E$	{t := *NewTemp*() $m := Elist_1.ndim + 1$ *Gen*(t ':=' $Elist_1.addr$ '*' *limit*($Elist_1.array$, m)) *Gen*(t ':=' t '+' $E.addr$) $Elist.array := Elist_1.array$ $Elist.addr := t$ $Elist.ndim := m$}
(6)	$L \rightarrow$ **id**	{$L.addr :=$ **id**.*pnt* $L.offset :=$ **null**}

В соответствии с продукцией 7 создается новая переменная t_1, формируются две команды (50 и 51):

```
50: t1 := i * 7
51: t1 := t1 + j
```

Устанавливаются значения атрибутов *array*, *addr* и *ndim* для нетерминала *Elist*.

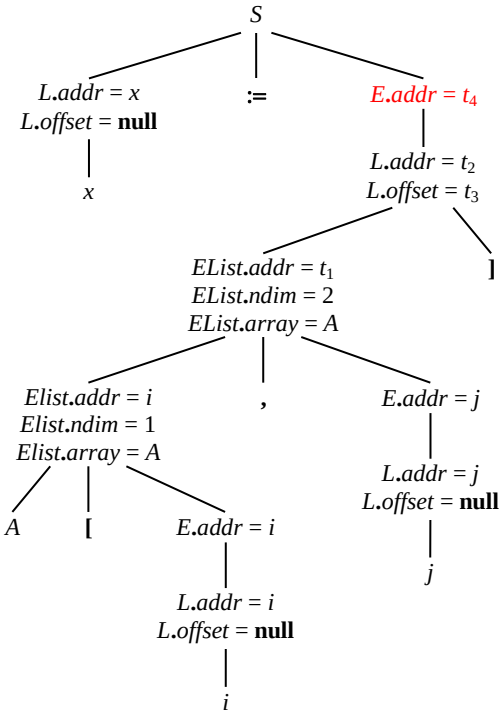


$x := A[i, j]$

(4)	$E \to L$	{**if** *L.offset* = **null then** /* *L* – простое имя */ *E.addr* := *L.addr* **else begin** *E.addr* := *NewTemp*() *Gen*(*E.addr* ':=' *L.addr* '[' *L.offset* ']') **end**}
(5)	$L \to Elist\ ]$	{*L.addr* := *NewTemp*() *L.offset* := *NewTemp*() *Gen*(*L.addr* ':=' *c*(*Elist.array*)) *Gen*(*L.offset* ':=' *Elist.addr* '*' *width*(*Elist.array*))}

В соответствии с продукцией 5 создаются две новые переменные t_2 (атрибут *L.addr*) и t_3 (атрибут *L.offset*), формируются две команды (52 и 53):

```
50: t1 := i * 7
51: t1 := t1 + j
52: t2 := c
53: t3 := t1 * 4
```



$x := A[i, j]$

(1) $S \to L := E$

{**if** *L.offset* = **null then** /* *L* – простое имя */
    Gen(*L.addr* ':=' *E.addr*)
else *Gen*(*L.addr* ' [' *L.offset* ']' ':=' *E.addr*)}

(4) $E \to L$

{**if** *L.offset* = **null then** /* *L* – простое имя */
    E.addr := *L.addr*
else begin
    E.addr := *NewTemp*()
    Gen(*E.addr* ':=' *L.addr* '[' *L.offset* ']')
end}

В соответствии с продукцией 4 создается новая переменная t_4 (атрибут *E.addr*), формируется команда 54, а в соответствии с продукцией 1 формируется команда 55:

```
50: t1 := i * 7
51: t1 := t1 + j
52: t2 := c
53: t3 := t1 * 4
54: t4 := t2[t3]
55: x := t4
```

Удостоверимся, что данная последовательность трехадресных инструкций правильно определяет адрес элемента массива. Для нашего массива $c = base - 88$. Для простоты примем $base = 0$. Обратимся к элементу $A[5, 3]$, т. е. $i = 5$, $j = 3$. Подставим эти значения в соответствующие команды:

Команды	Подстановка значений
50: t_1 := i * 7	50: t_1 := 5 * 7 = 35
51: t_1 := t_1 + j	51: t_1 := 35 + 3 = 38
52: t_2 := c	52: t_2 := −88
53: t_3 := t_1 * 4	53: t_3 := 38 * 4 = 152
54: t_4 := t_2[t_3]	54: t_4 := −88[152] /*A[5,3] находится по адресу 152 – 88 = 64*/
55: x := t_4	55: x := t_4