## Тема 8. Записи (6.3.6 в Ахо)

Вернемся к *L*-атрибутному СУО для объявления типа из раздела 5.4 (см. табл. 6). Дополним продукциями для списка объявлений *LstD* и синтезируемым атрибутом *width* (размер типа) нетерминалов *L* и *T*. Процедура *AddType* сохраняет как тип, так и размер в соответствующей записи таблицы символов.

| Продукция | Семантические правила |
|---|---|
| 1) $LstD \to D$ | |
| 2) $LstD \to LstD_1\ D$ | |
| 3) $D \to$ **id** $L$ **;** | *AddType* (**id.***pnt*, *L***.***type*, *L***.***width*) |
| 4) $L \to$ **,** **id** $L_1$ | *AddType* (**id.***pnt*, $L_1$***.type***, $L_1$***.width***);<br>*L***.***type* := $L_1$***.type***<br>*L***.***width* := $L_1$***.width*** |
| 5) $L \to$ **:** $T$ | *L***.***type* := *T***.***type***<br>*L***.***width* := *T***.***width*** |

Тип запись можно представить продукцией

$T \rightarrow$ **record** *LstD* **end**

Здесь нетерминал *LstD* определяет последовательность объявлений полей записи. Должны выполняться следующие требования:

- имена полей должны быть различны;
- относительный адрес имени поля отсчитывается относительно начала области данных, выделенных для записи.

Тип записи имеет вид *record*(*t*), где *record* – конструктор типа, *t* – ссылка на таблицу символов, хранящую информацию о полях данного типа записи.

Поскольку предполагается использование нескольких таблиц символов, необходимо знать, в какой таблице содержится требуемая запись. Возможны следующие варианты решения:

1. Можно нетерминалам *V* и *E* добавить синтезируемый атрибут, значением которого будет указатель на соответствующую таблицу символов.

2. Атрибут *addr* представить как двойку (*p*, *ad*), где *p* – указатель на таблицу символов, *ad* – указатель на запись в этой таблице символов.

3. Применение специального глобального стека (назовем его *ST*), в вершине которого находится указатель на текущую таблицу символов.

В рассматриваемом СУО применен третий вариант.

СУО для объявления записи можно представить следующим образом:

| Продукция | Семантические правила |
|---|---|
| 6) $T \rightarrow M$ **record** *LstD* **end** | *p* := *Pop* (*ST*);<br>*T*.*type* := *record*(*p*);<br>*T*.*width* := *offset*;<br>*offset* := *Pop* (*SO*) |
| 7) $M \rightarrow \varepsilon$ | *pNew* := *NewTbl*();<br>*Push* (*pNew*, *ST*);<br>*Push* (*offset*, *SO*);<br>*offset* := 0 |

В продукции 6 используется нетерминал-маркер *M*, с которым связана продукция $M \rightarrow \varepsilon$. Этот маркер фиксирует момент, когда необходимо создать новую таблицу символов для полей записи. При преобразовании СУО в СУТ это действие должно выполняться непосредственно перед терминалом **record**.

В семантических правилах продукции 7 функция *NewTbl*() создает новую таблицу символов и помещает указатель на нее в стек *ST*. В стеке *SO* сохраняется текущее значение *offset* (указатель на первую ячейку свободной памяти) и переменной *offset* присваивается нулевое значение (*SO* и *offset* – глобальные переменные). Объявления, сгенерированные *LstD*, сохраняют типы и относительные адреса в новой таблице символов. Функция *Top* (*ST*) возвращает значение элемента из вершины стека *ST* без его исключения.

**Обратите внимание,** что маркер *M* стоит перед **record**. В оригинале (т. е. в Ахо) маркер *M* стоит после ключевого слова **record** непосредственно перед *LstD*. Для выбранного *LR*(1)-метода синтаксически управляемой трансляции это приведет к неправильному результату трансляции. Рассмотрим подробнее причину.

Пусть продукция имеет вид $T \rightarrow$ **record** $M$ *LstD* **end.** В соответствии с продукциями грамматики имеем *Follow* $(M) = \{$**id**$\}$. Напомним, что функция *Follow* определяет множество терминалов, которые могут следовать непосредственно за нетерминалом $M$ в какой-либо сентенциальной форме. Значения данной функции используются для реализации свертки, т. е. для *SLR*(1)-пункта $[M \rightarrow \bullet]$ свертка выполняется, если очередным токеном, сформированным сканером, является **id** $\in$ *Follow* $(M)$. Таким образом, к моменту свертки (а семантические правила выполняются в процессе свертки) идентификатор **id** , который является именем поля записи, уже добавлен сканером в таблицу символов предыдущего уровня. Это неправильно, поскольку имя поля должно быть добавлено в новую таблицу символов для полей записи, создаваемой функцией *NewTbl* () в семантических правилах продукции $M \rightarrow \varepsilon$. Поэтому правильно будет $M$ поставить перед **record**. Тогда *Follow* $(M) = \{$**record**$\}$, свертка будет выполнена по терминалу **record**, в семантических правилах продукции $M \rightarrow \varepsilon$ будет создана новая таблица символов для полей записи и имя поля записи попадет именно в эту новую таблицу.

Другой вариант решения проблемы – изменить грамматику, добавив ключевое слово или специальный символ после нетерминала *M*, например, следующим образом (добавлено ключевое слово **of**):

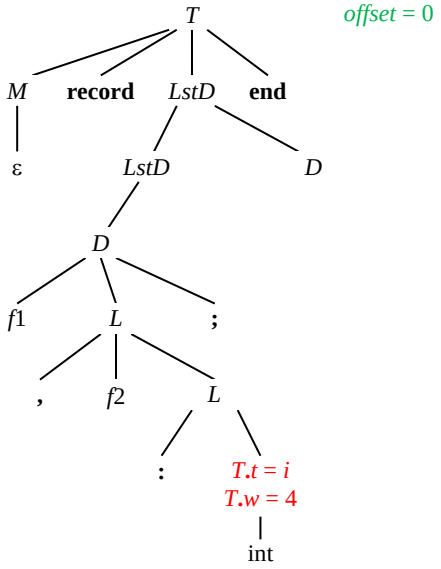
*T* → **record** *M* **of** *LstD* **end**

Теперь *Follow* (*M*) = {**of**}, свертка будет выполнена по терминалу **of**, в семантических правилах продукции *M* → ε будет создана новая таблица символов для полей записи и имя поля записи попадет именно в эту новую таблицу.

Семантические правила продукции 6 устанавливают в атрибуте ***T.type*** тип записи и в атрибуте ***T.width*** размер записи, после чего восстанавливают сохраненные в стеках *ST* и *SO* таблицу символов и смещение *offset*. Этим завершается трансляция данного типа записи.

Рассмотрим пример аннотированного дерева разбора для записи (размер целого типа – 4 байта, вещественного – 8 байт) **record** f1 **,** f2 **:** int **;** f3 **:** real **; end**

На аннотированном дереве разбора атрибуты для удобства обозначены: *type* – *t*, *width* – *w*, целый тип int – *i*, вещественный тип real – *r*.

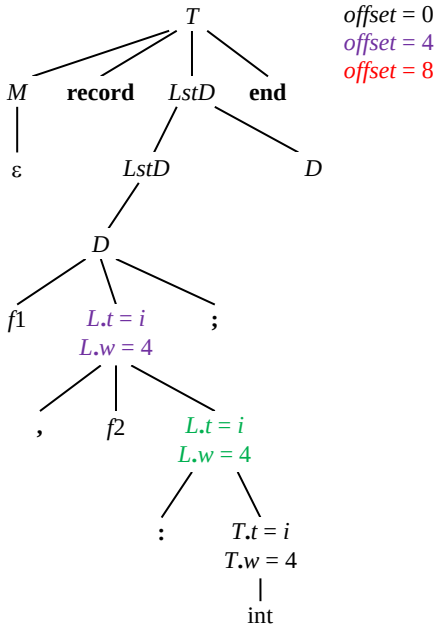



**record** f1 **,** f2 **:** int **;** f3 **:** real **; end**

| Продукция | Семантические правила |
|---|---|
| 1) *LstD* → *D* | |
| 2) *LstD* → $LstD_1$ *D* | |
| 3) *D* → **id** *L* **;** | *AddType* (**id.***pnt*, *L.type*, *L.width*) |
| 4) *L* → **,** **id** $L_1$ | *AddType* (**id.***pnt*, $L_1$*.type*, $L_1$*.width*);<br>*L.type* := $L_1$*.type*<br>*L.width* := $L_1$*.width* |
| 5) *L* → **:** *T* | *L.type* := *T.type*<br>*L.width* := *T.width* |
| 6) *T* → *M* **record** *LstD* **end** | *p* := *Pop* (*ST*);<br>*T.type* := *record*(*p*);<br>*T.width* := *offset*;<br>*offset* := *Pop* (*SO*) |
| 7) *M* → ε | *pNew* := *NewTbl*();<br>*Push* (*pNew*, *ST*);<br>*Push* (*offset*, *SO*);<br>*offset* := 0 |

В соответствии с продукцией 7 создается новая таблица символов и устанавливается *offset* = 0.

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



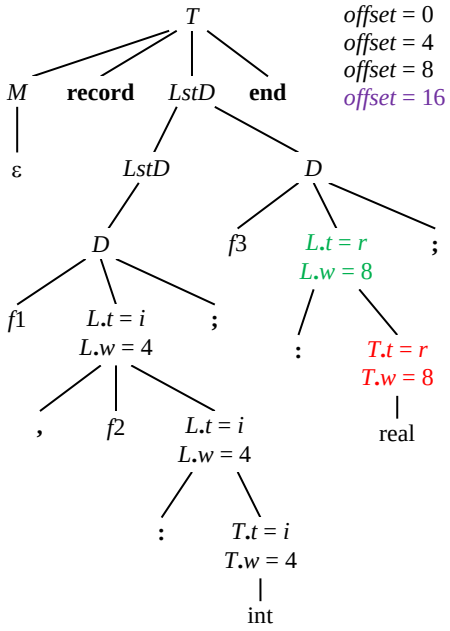

**record** f1 **,** f2 **:** int **;** f3 **:** real **; end**

| Продукция | Семантические правила |
|---|---|
| 3) $D \rightarrow$ **id** $L$ **;** | *AddType* (**id.***pnt*, *L.type*, *L.width*) |
| 4) $L \rightarrow$ **,** **id** $L_1$ | *AddType* (**id.***pnt*, $L_1$*.type*, $L_1$*.width*);<br>*L.type* := $L_1$*.type*<br>*L.width* := $L_1$*.width* |
| 5) $L \rightarrow$ **:** $T$ | *L.type* := *T.type*<br>*L.width* := *T.width* |

В соответствии с продукцией 5 атрибутами *L.type* и *L.width* становятся соответственно атрибуты *T.type* = int и *T.width* = 4.

В соответствии с продукцией 4 процедура *AddType* сохраняет тип *L.type* = int и размер *L.width* = 4 для поля *f*2 в соответствующей записи таблицы символов. Переменная *offset* принимает значение, равное 4. Атрибутами *L.type* и *L.width* становятся соответственно атрибуты $L_1$*.type* = int и $L_1$*.width* = 4.

В соответствии с продукцией 3 процедура *AddType* сохраняет тип *L.type* = int и размер *L.width* = 4 для поля *f*1 в соответствующей записи таблицы символов. Переменная *offset* принимает значение, равное 8.



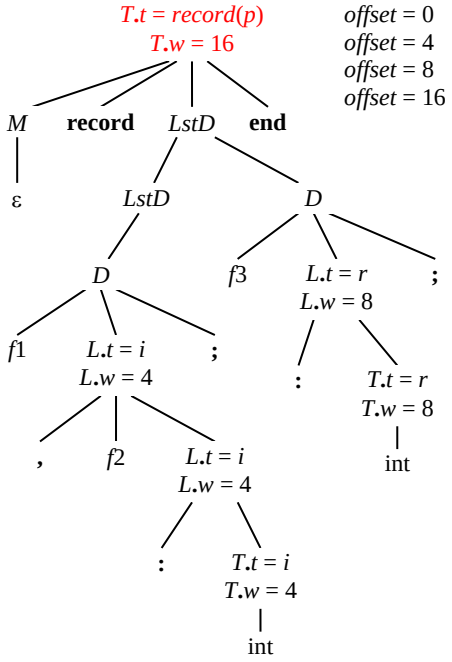

**record** f1 **,** f2 **:** int **;** f3 **:** real **; end**

| Продукция | Семантические правила |
|---|---|
| 3) $D \rightarrow$ **id** $L$ **;** | *AddType* (**id.***pnt*, *L.type*, *L.width*) |
| 5) $L \rightarrow$ **:** $T$ | *L.type* := *T.type*<br>*L.width* := *T.width* |

В соответствии с продукцией $T \rightarrow$ **id** (**id** – имя типа, в СУО продукция не показана) устанавливаются значения *Type.type* = real и *Type.width* = 8.

В соответствии с продукцией 5 атрибутами *L.type* и *L.width* становятся соответственно атрибуты *T.type* = real и *T.width* = 8.

В соответствии с продукцией 3 процедура *AddType* сохраняет тип *L.type* = real и размер *L.width* = 8 для поля *f*3 в соответствующей записи таблицы символов. Переменная *offset* принимает значение, равное 16.




**record** f1 **,** f2 **:** int **;** f3 **:** real **; end**

| Продукция | Семантические правила |
|---|---|
| 6) $T \rightarrow M$ **record** *LstD* **end** | *p* := *Pop* (*ST*);<br>*T.type* := *record*(*p*);<br>*T.width* := *offset*;<br>*offset* := *Pop* (*SO*) |

Семантические правила продукции 6 устанавливают в атрибуте *T.type* тип записи *record*(*p*) и в атрибуте *T.width* размер записи *offset* = 16, после чего восстанавливают сохраненные в стеках *ST* и *SO* таблицу символов и смещение *offset*. Этим завершается трансляция данного типа записи.

Обращение к полю записи можно представить следующим СУО (по сути это схема трансляции, поскольку все атрибуты синтезируемые и все правила выполняются в конце правых частей продукций):

| Продукция | Семантические правила |
|---|---|
| 1) $S \to V := E$ | *Gen*(*V.addr* ':=' *E.addr*) |
| 2) $V \to$ **id** | *V.addr* := (*Top*(*ST*), **id.***pnt*) |
| 3) $V \to Rcrd$ **. id** | *V.addr* := (*Top*(*ST*), **id.***pnt*)<br>*Pop* (*ST*) |
| 4) $Rcrd \to Rcrd_1$ **. id** | *t* := *GetType* (**id.***pnt*)<br>**if** $t \neq$ *record*(*p*) **then** *type_error*<br>*Pop* (*ST*)<br>*Push* (*p, ST*) |
| 5) *Rcrd* → **id** | *t* := *GetType* (**id**.*pnt*)<br>**if** $t \neq$ *record*(*p*) **then** *type_error*<br>*Push* (*p, ST*) |
| 6) $E \to V$ | *E.addr* := *V.addr* |

Для реализации проверки типов нетерминалы *V* (переменная в левой части оператора присваивания) и *E* (выражение) наряду с атрибутом *addr* (указатель на запись в таблице символов) должны иметь и атрибут *type* (тип). Вычисление атрибута *type* очевидно и не раз рассматривалось в предыдущих разделах, поэтому в данном СУО этот атрибут и правила для его вычисления не разбираются.

Функция *GetType* предоставляет тип из записи таблицы символов, на которую указывает **id**.*pnt*. Предполагается, что выполнение действия *type_error* (ошибка типа) приводит к формированию соответствующего сообщения об ошибке и немедленному прекращению выполнения всех остальных семантических правил.

Поскольку используется несколько таблиц символов, для обеспечения доступа к нужной на данный момент таблице применяется глобальный стек *ST*, в вершине которого находится указатель на текущую таблицу символов. Функция *Top* (*ST*) возвращает значение элемента из вершины стека *ST* без его исключения.

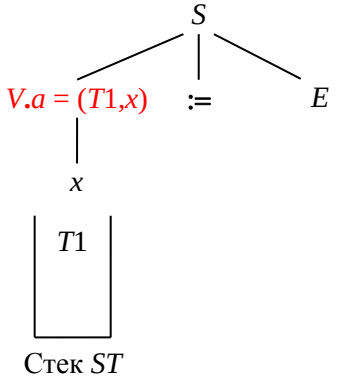
Рассмотрим аннотирование дерева разбора для присвоения *x* **:=** *r***.***y***.***z*.

Здесь *r* – имя записи, данные о которой сохранены в записи основной таблицы символов *T*1, а данные о полях записи сохранены в отдельной таблице символов *T*2. Поскольку поле *y* также является записью, данные о полях этой записи содержатся в отдельной таблице символов *T*3. По сути *T*1, *T*2 и *T*3 – это обозначения указателей на соответствующие таблицы.

На аннотированном дереве разбора вместо **id.***pnt* как обычно указан сам идентификатор, атрибут *addr* обозначен как *a*.

Для наглядности атрибут *addr* представлен как двойка (*p*, *a*), где *p* – указатель на таблицу символов, *a* – указатель на запись в этой таблице символов (атрибут *addr*).

x := r.y.z

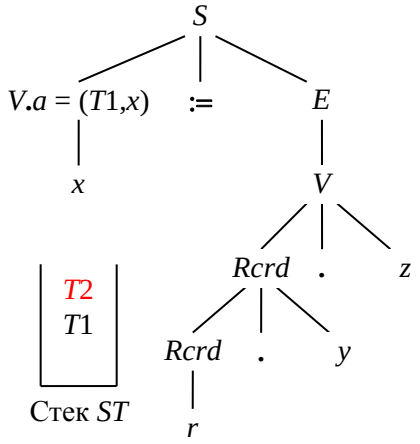



| Продукция | Семантические правила |
|---|---|
| 1) $S \rightarrow V := E$ | *Gen*(**V.***addr* ':=' **E.***addr*) |
| 2) $V \rightarrow$ **id** | **V.***addr* := (*Top*(*ST*), **id.***pnt*) |

К данному моменту в вершине стека *ST* находится указатель *T*1 – указатель на таблицу символов, в которой запись **id.***pnt* содержит всю необходимую информацию об идентификаторе *x*.

В соответствии с продукцией 2 устанавливается *V.addr* = (*T*1, *x*).

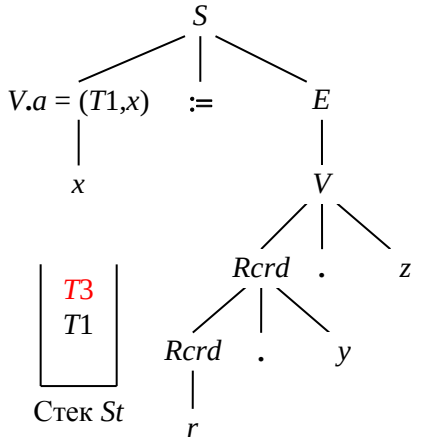
x := r.y.z




| Продукция | Семантические правила |
|---|---|
| 3) $V \rightarrow Rcrd$ **.** **id** | *V.addr* := (*Top*(*ST*), **id.***pnt*)<br>*Pop* (*ST*) |
| 4) $Rcrd \rightarrow Rcrd_1$ **.** **id** | *t* := *GetType* (**id.***pnt*)<br>**if** *t* ≠ *record*(*p*) **then** *type_error*<br>*Pop* (*ST*)<br>*Push* (*p*, *ST*) |
| 5) $Rcrd \rightarrow$ **id** | *t* := *GetType* (**id.***pnt*)<br>**if** *t* ≠ *record*(*p*) **then** *type_error*<br>*Push* (*p*, *ST*) |
| 6) $E \rightarrow V$ | *E.addr* := *V.addr* |

В соответствии с продукцией 5 из записи **id.***pnt* функция *GetType* предоставляет тип идентификатора *r*, из типа *record*(*p*) определяется указатель *p* на таблицу символов для полей записи *r*, который заносится в стек *ST*. В нашем случае *p* = *T*2.

x := r.y.z

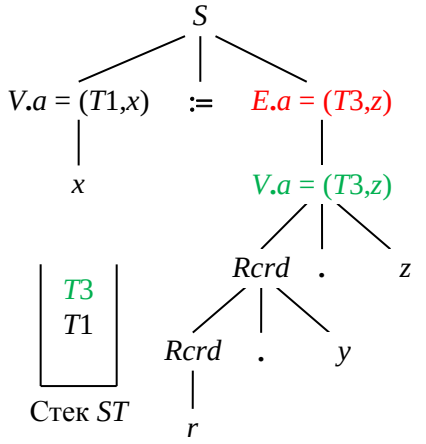



| Продукция | Семантические правила |
|---|---|
| 4) $Rcrd \rightarrow Rcrd_1$ **.** **id** | $t$ := *GetType* (**id.***pnt*)<br>**if** $t \neq$ *record*($p$) **then** *type_error*<br>*Pop* (*ST*)<br>*Push* ($p$, *ST*) |

В соответствии с продукцией 4 из записи **id.***pnt* функция *GetType* предоставляет тип идентификатора (поля) *y*, из типа *record*(*p*) определяется указатель *p* на таблицу символов для полей записи *y*. В нашем случае *p* = *T*3.

Поскольку таблица *T*2 обеспечила доступ к полю *y*, больше она не нужна. Поэтому она удаляется из стека *ST*, а в стек заносится указатель *p*.

x := r.y.z




| Продукция | Семантические правила |
|---|---|
| 1) $S \rightarrow V := E$ | *Gen*(*V.addr* ':=' *E.addr*) |
| 3) $V \rightarrow Rcrd$ **. id** | *V.addr* := (*Top*(*ST*), **id.***pnt*)<br>*Pop* (*ST*) |
| 4) $Rcrd \rightarrow Rcrd_1$ **. id** | *t* := *GetType* (**id.***pnt*)<br>**if** $t \neq record(p)$ **then** *type_error*<br>*Pop* (*ST*)<br>*Push* (*p*, *ST*) |
| 6) $E \rightarrow V$ | *E.addr* := *V.addr* |

В соответствии с продукцией 3 устанавливается *V.addr* = (*T*3, *z*).

Поскольку таблица *T*3 обеспечила доступ к полю *z*, больше она не нужна. Поэтому она удаляется из стека *ST*. В результате в вершине стека *ST* будет находится указатель *T*1 (на рис. результат удаления *T*3 из стека не показан).

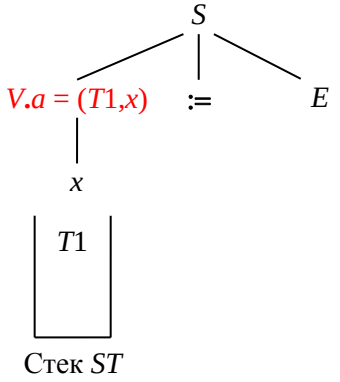
В соответствии с продукций 6 *E.addr* получает значение *V.addr* = (*T*3, *z*).

Согласно продукций 1 формируется трехадресная команда `x := z`.

Предыдущий вариант взят из 6.3.6 в Ахо. Немного подумав, предлагаю, как мне кажется, более универсальный и простой метод трансляции обращения к полю записи.

| Продукция | Семантические правила |
|---|---|
| 1) $S \to V := E$ | *Gen*(*V.addr* ':=' *E.addr*) |
| 2) $V \to$ **id** | *V.addr* := (*Top*(*ST*), **id.***pnt*)<br>*t* := *GetType* (**id.***pnt*)<br>**if** *t* = *record*(*p*) **then**<br>  *Push* (*p*, *ST*) |
| 3) $V \to V$ **. id** | *V.addr* := (*Top*(*ST*), **id.***pnt*)<br>*Pop* (*ST*)<br>*t* := *GetType* (**id.***pnt*)<br>**if** *t* = *record*(*p*) **then**<br>  *Push* (*p*, *ST*) |
| 4) $E \to V$ | *E.addr* := *V.addr* |

x := r.y.z

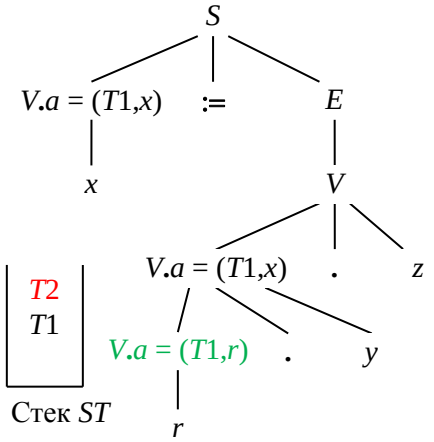



| Продукция | Семантические правила |
|---|---|
| 1) $S \rightarrow V := E$ | *Gen*(**V.***addr* ':=' **E.***addr*) |
| 2) $V \rightarrow$ **id** | **V.***addr* := (*Top*(*ST*), **id.***pnt*)<br>*t* := *GetType* (**id.***pnt*)<br>**if** *t* = *record*(*p*) **then**<br>*Push* (*p*, *ST*) |

К данному моменту в вершине стека *ST* находится указатель *T*1 – указатель на таблицу символов, в которой запись **id.***pnt* содержит всю необходимую информацию об идентификаторе *x*.

В соответствии с продукцией 2 устанавливается *V.addr* = (*T*1, *x*).

x := r.y.z

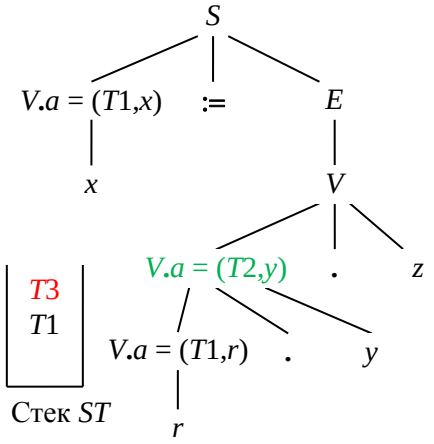



| Продукция | Семантические правила |
|---|---|
| 2) $V \rightarrow$ **id** | *V.addr* := (*Top*(*ST*), **id.***pnt*)<br>*t* := *GetType* (**id.***pnt*)<br>**if** *t* = *record*(*p*) **then**<br>*Push* (*p*, *ST*) |

В соответствии с продукцией 2 устанавливается *V.addr* = (*T*1, *r*).

Поскольку *r* имеет тип *record*(*p*), определяется указатель *p* на таблицу символов для полей записи *r*, который заносится в стек *ST*. В нашем случае *p* = *T*2.

x := r.y.z



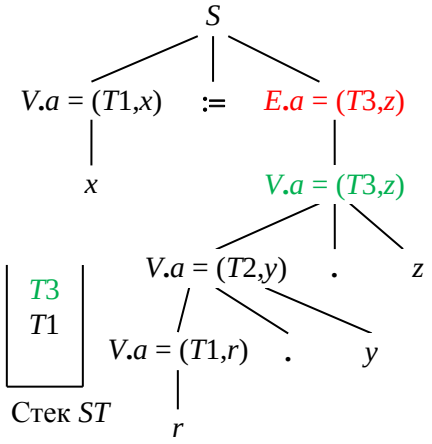

| Продукция | Семантические правила |
|---|---|
| 3) $V \rightarrow V$ **.** **id** | *V.addr* := (*Top*(*ST*), **id.***pnt*)<br>*Pop* (*ST*)<br>*t* := *GetType* (**id.***pnt*)<br>**if** *t* = *record*(*p*) **then**<br>*Push* (*p*, *ST*) |

В соответствии с продукцией 3 устанавливается *V.addr* = (*T*2, *y*).

Поскольку таблица *T*2 обеспечила доступ к полю *y*, больше она не нужна. Поэтому она удаляется из стека *ST*.

Поскольку *y* имеет тип *record*(*p*), определяется указатель *p* на таблицу символов для полей записи *y*, который заносится в стек *ST*. В нашем случае *p* = *T*3.

x := r.y.z




| Продукция | Семантические правила |
|---|---|
| 1) $S \rightarrow V := E$ | *Gen*(*V*.*addr* ':=' *E*.*addr*) |
| 3) $V \rightarrow V$ . **id** | *V*.*addr* := (*Top*(*ST*), **id**.*pnt*)<br>*Pop* (*ST*)<br>*t* := *GetType* (**id**.*pnt*)<br>**if** *t* = *record*(*p*) **then**<br>*Push* (*p*, *ST*) |
| 4) $E \rightarrow V$ | *E*.*addr* := *V*.*addr* |

В соответствии с продукцией 3 устанавливается *V*.*addr* = (*T*3, *z*). Причем *z* не является записью.

Поскольку таблица *T*3 обеспечила доступ к полю *z*, больше она не нужна. Поэтому она удаляется из стека *ST*. В результате в вершине стека *ST* будет находится указатель *T*1 (на рис. результат удаления *T*3 из стека не показан).

В соответствии с продукций 4 *E*.*addr* получает значение *V*.*addr* = (*T*3, *z*).

Согласно продукции 1 формируется трехадресная команда `x := z`.

Функции и процедуры см. 6.9 и начальные параграфы раздела 7.