Тема 6. Генерация промежуточного кода

6.2. Трансляция арифметических выражений

В процессе трансляции выражения в промежуточный код происходит его преобразование в трехадресный код, в котором каждая из команд имеет не более чем одну операцию (см. рис. 8, *а*).

СУО (табл. 9) формирует трехадресный код для оператора присваивания и арифметического выражения.

Таблица 9

СУО для трансляции оператора присваивания
и арифметического выражения

Продукция	Семантические правила
$S \to \mathbf{id} := E$	$S.code := E.code \parallel Gen(\mathbf{id}.pnt ':=' E.addr)$
$E \to E_1 + T$	$E.addr := NewTemp()$ $E.code := E_1.code \parallel T.code \parallel Gen(E.addr ':=' E_1.addr '+' T.addr)$
$E \to T$	$E.addr := T.addr$ $E.code := T.code$
$E \to - T$	$E.addr := NewTemp()$ $E.code := T.code \parallel Gen(E.addr ':=' '-' T.addr)$
$T \to T_1 * F$	$T.addr := NewTemp()$ $T.code := T_1.code \parallel F.code \parallel Gen(T.addr ':=' T_1.addr '*' F.addr)$
$T \to F$	$T.addr := F.addr$ $T.code := F.code$
$F \to (E)$	$F.addr := E.addr$ $F.code := E.code$
$F \to \mathbf{id}$	$F.addr := \mathbf{id}.pnt$ $F.code := ''$

Синтезируемый атрибут *addr* является указателем на запись в таблице символов, где имеется вся необходимая информация об объекте (переменная, константа или временная переменная), вплоть до адреса размещения объекта в памяти. В синтезируемом атрибуте *code* формируется трехадресный код для соответствующего нетерминала. Функция *NewTemp*() создает новую временную переменную и возвращает указатель на соответствующую запись в таблице (в зависимости от реализации это может быть общая таблица символов или специальная таблица временных имен). Формирование трехадресной команды для удобства показывается процедурой *Gen*($x\ ':='\ y\ '+'\ z$), представляющей команду $x := y + z$ (при реализации вместо символов операций в команду записываются их коды). В результате выполнения правила $F.addr := \mathbf{id}.pnt$ для продукции $F \to \mathbf{id}$ атрибут $F.addr$ указывает на запись в таблице символов для данного экземпляра **id**. Символ || обозначает операцию конкатенации строк.

Поскольку атрибуты *code* могут быть довольно длинными строками, чаще применяют *инкрементную* трансляцию. Она заключается в том, что формируется единый поток генерируемых трехадресных команд в некотором глобальном массиве или файле. Процедура *Gen* при этом не только представляет трехадресную команду, но и инкрементно добавляет новую команду к последовательности сформированных к данному моменту команд. Реализация инкрементной трансляции СУО, генерирующей тот же код, что и СУО в табл. 9, представлена в табл. 10.

Таблица 10

Инкрементная трансляция арифметического выражения

Продукция	Семантические правила		Не инкрементная
$S \to \mathbf{id} := E$	*Gen* (**id.***pnt* ′:=′ *E.addr*)		*S.code* := *E.code* || *Gen* (**id.***pnt* ′:=′ *E.addr*)
$E \to E_1 + T$	*E.addr* := *NewTemp* () *Gen* (*E.addr* ′:=′ E_1*.addr* ′+′ *T.addr*)		*E.addr* := *NewTemp* () *E.code* := E_1*.code* || *T.code* || *Gen* (*E.addr* ′:=′ E_1*.addr* ′+′ *T.addr*)
$E \to T$	*E.addr* := *T.addr*		*E.addr* := *T.addr* *E.code* := *T.code*
$E \to - T$	*E.addr* := *NewTemp* () *Gen* (*E.addr* ′:=′ ′–′ *T.addr*)		*E.addr* := *NewTemp* () *E.code* := *T.code* || *Gen* (*E.addr* ′:=′ ′–′ *T.addr*)
$T \to T_1 * F$	*T.addr* := *NewTemp* () *Gen* (*T.addr* ′:=′ T_1*.addr* ′*′ *F.addr*)		*T.addr* := *NewTemp* () *T.code* := T_1*.code* || *F.code* || *Gen* (*T.addr* ′:=′ T_1*.addr* ′*′ *F.addr*)
$T \to F$	*T.addr* := *F.addr*		*T.addr* := *F.addr* *T.code* := *F.code*
$F \to (E)$	*F.addr* := *E.addr*		*F.addr* := *E.addr* *F.code* := *E.code*
$F \to \mathbf{id}$	*F.addr* := **id.***pnt*		*F.addr* := **id.***pnt* *F.code* := ′′

Как видно в рассмотренном примере, преобразование СУО для применения инкрементной трансляции особых проблем не вызывает (хотя и могут быть определенные трудности). Поэтому в большинстве СУО будем использовать атрибут *code*, поскольку он дает более наглядное представление о последовательности формирования трехадресного кода.

Рассмотренные выше СУО легко можно приспособить для соответствующих представлений трехадресных команд. Для четверок семантические правила практически не требуют изменений (в процедуре *Gen* следует уточнить порядок заполнения полей). Для троек и косвенных троек вместо *NewTemp* в атрибуты *addr* следует записывать номер (метку) текущей команды, для тернарных операций предусмотреть формирование двух троек. Для косвенных троек, кроме формирования самих троек, дополнительно следует создавать список указателей на тройки.

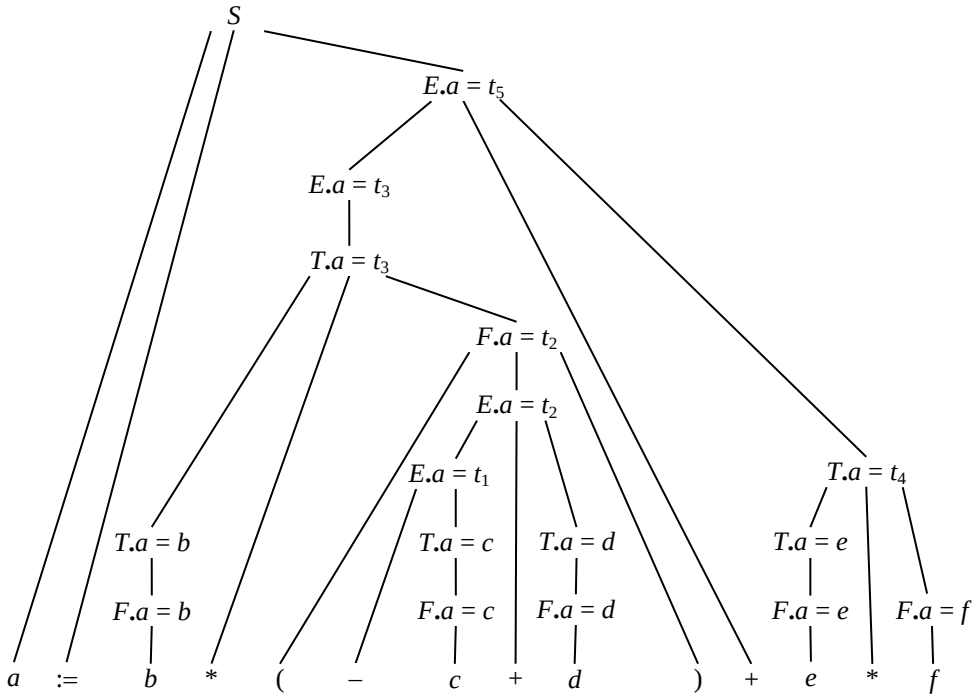
Инкрементная трансляция арифметического выражения

Продукция	Семантические правила
1) $S \to$ **id** $:= E$	*Gen* (**id.***pnt* $':='$ *E.addr*)
2) $E \to E_1 + T$	*E.addr* := *NewTemp* () *Gen* (*E.addr* $':='$ E_1*.addr* $'+'$ *T.addr*)
3) $E \to T$	*E.addr* := *T.addr*
4) $E \to - T$	*E.addr* := *NewTemp* () *Gen* (*E.addr* $':='$ $'-'$ *T.addr*)
5) $T \to T_1 * F$	*T.addr* := *NewTemp* () *Gen* (*T.addr* $':='$ T_1*.addr* $'*'$ *F.addr*)
6) $T \to F$	*T.addr* := *F.addr*
7) $F \to (E)$	*F.addr* := *E.addr*
8) $F \to$ **id**	*F.addr* := **id.***pnt*

$a := b * (- c + d) + e * f$

На рис. вместо **id**.*pnt* указан сам идентификатор, атрибут *addr* обозначен как *a*.

$t_1 := - c$
$t_2 := t_1 + d$
$t_3 := b * t_2$
$t_4 := e * f$
$t_5 := t_3 + t_4$
$a := t_5$



Рассмотрим подробнее процесс аннотирования.

Инкрементная трансляция арифметического выражения

Продукция	Семантические правила
1) $S \to \mathbf{id} := E$	*Gen* (**id**.*pnt* ':=' *E*.*addr*)
2) $E \to E_1 + T$	*E*.*addr* := *NewTemp* () *Gen* (*E*.*addr* ':=' E_1.*addr* '+' *T*.*addr*)
3) $E \to T$	*E*.*addr* := *T*.*addr*
4) $E \to - T$	*E*.*addr* := *NewTemp* () *Gen* (*E*.*addr* ':=' '–' *T*.*addr*)
5) $T \to T_1 * F$	*T*.*addr* := *NewTemp* () *Gen* (*T*.*addr* ':=' T_1.*addr* '*' *F*.*addr*)
6) $T \to F$	*T*.*addr* := *F*.*addr*
7) $F \to (E)$	*F*.*addr* := *E*.*addr*
8) $F \to \mathbf{id}$	*F*.*addr* := **id**.*pnt*

$a := b * (- c + d) + e * f$

Согласно продукциям 8 и 6 соответствующие значения принимают *T*.*addr*.

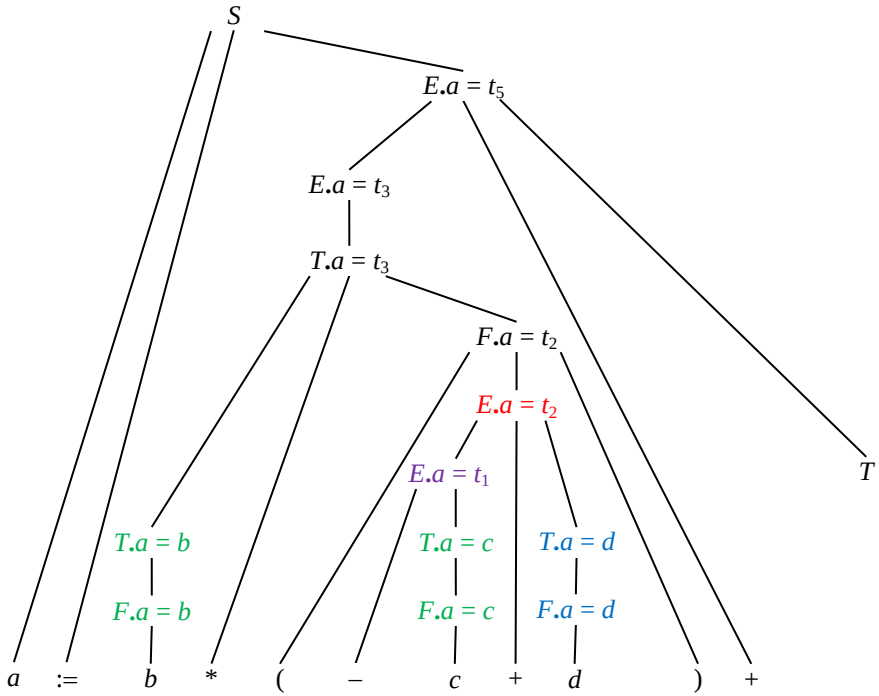
В соответствии с продукцией 4 создается новая переменная t_1, *E*.*addr* принимает значение t_1 и формируется команда

$t_1 := - c$

Согласно продукциям 8 и 6 соответствующее значение принимает *T*.*addr* = *d*.

В соответствии с продукцией 2 создается новая переменная t_2, *E*.*addr* принимает значение t_2 и формируется команда

$t_2 := t_1 + d$



Инкрементная трансляция арифметического выражения

Продукция	Семантические правила
1) $S \to \mathbf{id} := E$	*Gen* (**id**.*pnt* ':=' *E.addr*)
2) $E \to E_1 + T$	*E.addr* := *NewTemp* () *Gen* (*E.addr* ':=' E_1.*addr* '+' *T.addr*)
3) $E \to T$	*E.addr* := *T.addr*
4) $E \to - T$	*E.addr* := *NewTemp* () *Gen* (*E.addr* ':=' '–' *T.addr*)
5) $T \to T_1 * F$	*T.addr* := *NewTemp* () *Gen* (*T.addr* ':=' T_1.*addr* '*' *F.addr*)
6) $T \to F$	*T.addr* := *F.addr*
7) $F \to (E)$	*F.addr* := *E.addr*
8) $F \to \mathbf{id}$	*F.addr* := **id**.*pnt*

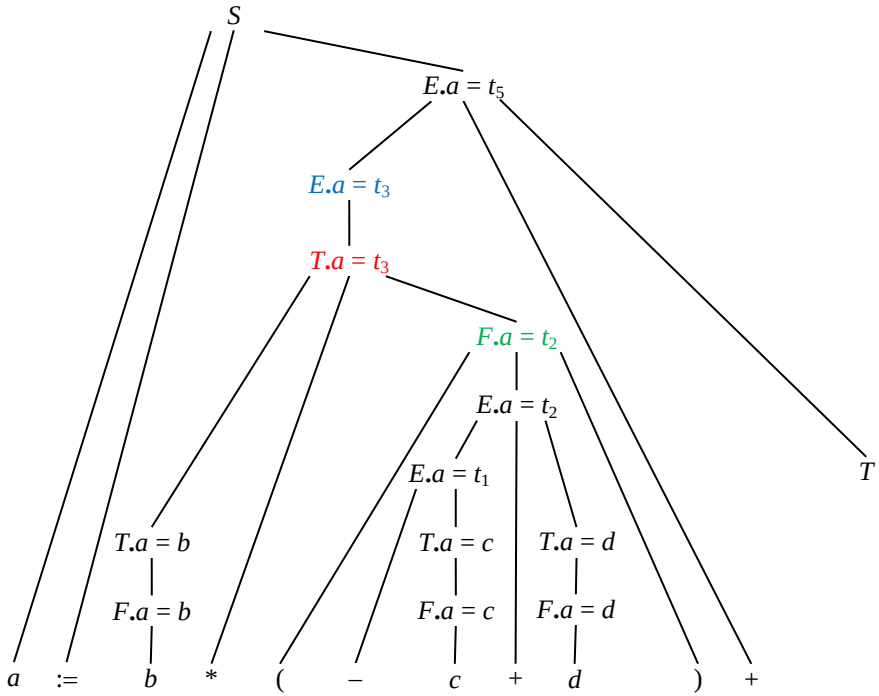
$a := b * (- c + d) + e * f$

Согласно продукции 7 получаем $F.addr = t_2$.

В соответствии с продукцией 5 создается новая переменная t_3, *T.addr* принимает значение t_3 и формируется команда

$t_3 := b * t_2$

Согласно продукции 3 получаем $E.addr = t_3$.



$t_1 := - c$

$t_2 := t_1 + d$

$t_3 := b * t_2$

Инкрементная трансляция арифметического выражения

Продукция	Семантические правила
1) $S \to \mathbf{id} := E$	*Gen* (**id**.*pnt* ':=' *E*.*addr*)
2) $E \to E_1 + T$	*E*.*addr* := *NewTemp* () *Gen* (*E*.*addr* ':=' E_1.*addr* '+' *T*.*addr*)
3) $E \to T$	*E*.*addr* := *T*.*addr*
4) $E \to - T$	*E*.*addr* := *NewTemp* () *Gen* (*E*.*addr* ':=' '–' *T*.*addr*)
5) $T \to T_1 * F$	*T*.*addr* := *NewTemp* () *Gen* (*T*.*addr* ':=' T_1.*addr* '*' *F*.*addr*)
6) $T \to F$	*T*.*addr* := *F*.*addr*
7) $F \to (E)$	*F*.*addr* := *E*.*addr*
8) $F \to \mathbf{id}$	*F*.*addr* := **id**.*pnt*

$a := b * (- c + d) + e * f$

Согласно продукциям 8 и 6 получаем *T.addr = e*, а согласно продукции 8 получаем *F.addr = e*.

В соответствии с продукцией 5 создается новая переменная t_4, *T.addr* принимает значение t_4 и формируется команда

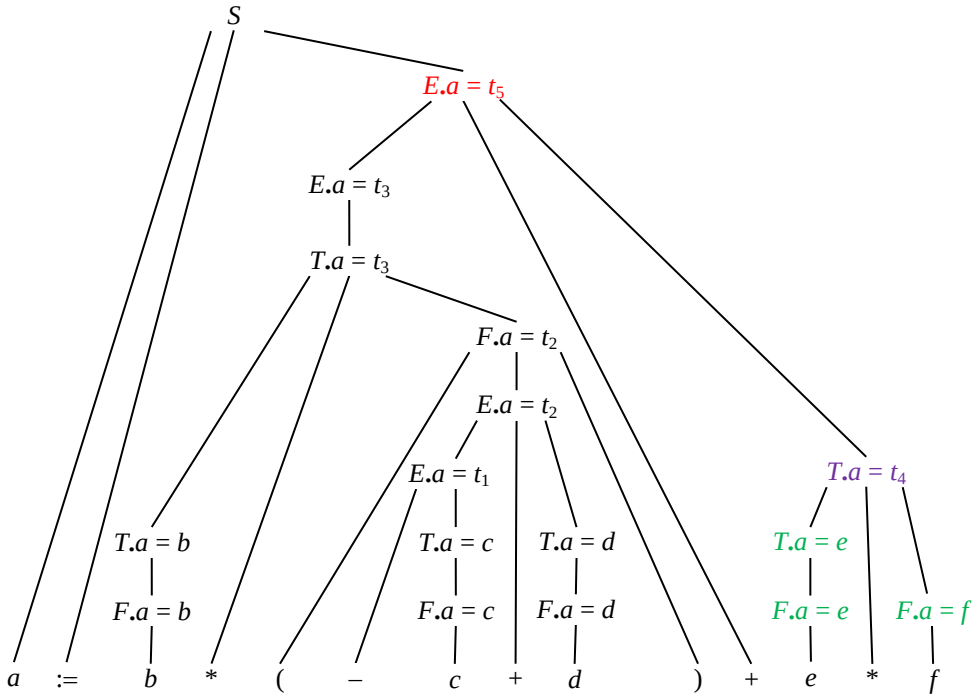
$t_4 := e * f$

В соответствии с продукцией 2 создается новая переменная t_5, *E.addr* принимает значение t_5 и формируется команда

$t_5 := t_3 + t_4$

В соответствии с продукцией 1 формируется команда

$a := t_5$



$t_1 := - c$
$t_2 := t_1 + d$
$t_3 := b * t_2$
$t_4 := e * f$
$t_5 := t_3 + t_4$
$a := t_5$