Тема 6. Генерация промежуточного кода

6.6. Метод обратных поправок для управляющих операторов

Метод обратных поправок можно использовать и для однопроходной инкрементной трансляции управляющих операторов. Соответствующее СУО представлено в табл. 15. Как и в СУО в табл. 14, логические выражения *B* имеют два списка переходов *B.truelist* и *B.falselist*, соответствующие истинным и ложным значениям *B*. Нетерминалы *L* (список операторов) и *S* (оператор) имеют атрибут *nextlist*, представляющий собой указатель на список команд переходов к команде, идущей непосредственно за кодом *L* или *S*. В эти списки группируются команды с временно неопределенными переходами.

Примечание. В рассматриваемом СУО для простоты в управляющих операторах синтаксически допустим только один оператор. Если необходима последовательность операторов, то достаточно ввести понятие составного оператора с помощью, например, продукции вида $S \rightarrow$ **{** *L* **}** (или $S \rightarrow$ **begin** *L* **end**) с семантическим правилом *S.nextlist* := *L.nextlist*.

Таблица 15

СУО для трансляции управляющих операторов методом обратных поправок

Продукция	Семантические правила
1) $L \to L_1$; M S	*BackPatch* (*L*$_1$.*nextlist*, *M*.*instr*) *L*.*nextlist* := *S*.*nextlist*
2) $L \to S$	*L*.*nextlist* := *S*.*nextlist*
3) $S \to$ **id** := B	см. табл. 14 (СУО для логических выражений)
4) $S \to$ **id** := E	см. табл. 10 (СУО для арифметических выражений) для оператора присваивания необходимо добавить семантическое правило *S.nextlist* := **null**
5) $S \to$ **if** B **then** M S_1	*BackPatch* (*B*.*truelist*, *M*.*instr*) *S*.*nextlist* := *Merge* (*B*.*falselist*, *S*$_1$.*nextlist*)
6) $S \to$**if** B **then** M_1 S_1 N **else** M_2 S_2	*BackPatch* (*B*.*truelist*, *M*$_1$.*instr*); *BackPatch* (*B*.*falselist*, *M*$_2$.*instr*) *tmp* := *Merge* (*S*$_1$.*nextlist*, *N*.*nextlist*) *S*.*nextlist* := *Merge* (*tmp*, *S*$_2$.*nextlist*)
7) $S \to$ **while** M_1 B **do** M_2 S_1	*BackPatch* (*S*$_1$.*nextlist*, *M*$_1$.*instr*); *BackPatch* (*B*.*truelist*, *M*$_2$.*instr*) *S*.*nextlist* := *B*.*falselist* *Gen* (′goto′ *M*$_1$.*instr*)
8) $S \to$ **repeat** M_1 S_1 **until** M_2 B	*BackPatch* (*S*$_1$.*nextlist*, *M*$_2$.*instr*); *BackPatch* (*B*.*falselist*, *M*$_1$.*instr*) *S*.*nextlist* := *B*.*truelist*
9) $M \to \varepsilon$	*M*.*instr* := *nextinstr*
10) $N \to \varepsilon$	*N*.*nextlist* := *MakeList* (*nextinstr*) *Gen* (′goto ?′)

В продукции для оператора **if-then**, если выражение B истинно, то следует перейти к вычислению S_1, поэтому целевой меткой переходов из $B.truelist$ устанавливается метка первой команды S_1. Эта метка получается с помощью синтезируемого атрибута $M.instr$ маркера M. Если B ложно, необходимо обеспечить пропуск оператора S_1 и переход к первой команде, непосредственно следующей за S (эта команда является также непосредственно следующей за S_1). Поэтому выполняется объединение списков $B.falselist$ и $S_1.nextlist$ в список $S.nextlist$.

При трансляции оператора **if-then-else** выполняется обратная поправка для переходов из списка $B.truelist$ установкой целевой метки $M_1.instr$, представляющей начало кода S_1 и из списка $B.falselist$ установкой целевой метки $M_2.instr$, представляющей начало кода S_2. В конец кода S_1 необходимо добавить безусловный переход для пропуска кода S_2 и передачи управления команде, непосредственно следующей за S. Позиция этой команды в данный момент неизвестна. Поэтому с помощью маркера N с продукцией $N \rightarrow \varepsilon$ формируется команда безусловного перехода и создается одноэлементный список $N.nextlist$ с индексом этой команды. Затем списки $S_1.nextlist$, $N.nextlist$ и $S_2.nextlist$ объединяются в список $S.nextlist$ (для объединения используется локальная переменная tmp).

В продукции для оператора цикла **while** используются маркеры M_1 для первой команды кода B и M_2 для первой команды кода S_1. Необходимость маркера M_1 объясняется тем, что в конец кода S_1 добавляется безусловный переход для передачи управления на первую команду кода B, индекс которой должен быть известен. При трансляции выполняется обратная поправка для переходов из списка S_1**.***nextlist* установкой целевой метки M_1**.***instr*, представляющей начало кода B и из списка B**.***truelist* установкой целевой метки M_2**.***instr*, представляющей начало кода S_1. Если B ложно, необходимо обеспечить пропуск оператора S_1 и переход к первой команде, непосредственно следующей за S. Поэтому списком S**.***nextlist* становится список B**.***falselist*. Добавляется команда безусловного перехода для передачи управления первой команде кода B, индекс которой сохранен в M_1**.***instr*.

В продукции для оператора цикла **repeat** используются маркеры M_1 для первой команды кода S_1 и M_2 для первой команды кода B. Необходимость маркера M_1 объясняется тем, что в коде B для всех команд из списка $B.falselist$ необходимо реализовать передачу управления на первую команду кода S_1, индекс которой должен быть известен. При трансляции выполняется обратная поправка для переходов из списка $B.falselist$ установкой целевой метки $M_1.instr$, представляющей начало кода S_1 и из списка $S_1.nextlist$ установкой целевой метки $M_2.instr$, представляющей начало кода B. Если B истинно, необходимо обеспечить переход к первой команде, непосредственно следующей за S. Поэтому списком $S.nextlist$ становится список $B.truelist$.

Последовательность операторов представляется продукцией $L \rightarrow L_1 \ ; \ M \ S$. Выполняется обратная поправка для переходов из списка $L_1.nextlist$ установкой метки $M.instr$, представляющей начало кода S. Списком $L.nextlist$ становится список $S.nextlist$.

Для продукции $L \rightarrow S$ список $L.nextlist$ совпадает $S.nextlist$.

Обратите внимание, что для реализации обратных поправок нетерминал S (оператор) имеет атрибут *nextlist*. Поэтому для продукции $S \rightarrow$ **id** := E (E – арифметическое выражение) в соответствующее СУО необходимо добавить семантическое правило $S.nextlist$:= **null.**

В рассмотренном СУО трехадресная команда генерируется только в продукциях для операторов **if-then-else** (через маркер N и продукцию $N \rightarrow \varepsilon$) и **while**. Весь остальной код формируется в семантических правилах, связанных с операторами присваивания и выражениями.

В рассмотренных выше СУО, использующих метод обратных поправок, все семантические правила выполняются в конце правых частей продукций. Это означает, что это практически готовые схемы трансляции для восходящего разбора. Достаточно решить вопросы хранения атрибутов и детализировать действия соответствующими операциями со стеком.

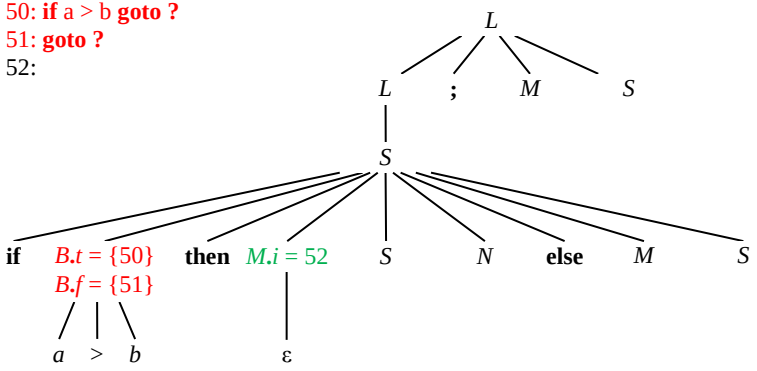
В заключение следует отметить, что в лекциях основное внимание было уделено методам построения СУО для решения задач проверки типов и генерации промежуточного кода. Поэтому, чтобы упростить изложение этих методов, не рассматривались структурированные типы данных (массивы, записи и т.п.) и применение конструкторов типа. Варианты СУО и СУТ, обеспечивающих проверку типов, распределение памяти и генерацию промежуточного кода для различных структурированных типов данных, указателей, операторов языков программирования для самостоятельной проработки можно найти в работах [1; 2]. Рекомендации по практическому применению методов для реализации различных фаз компиляции можно посмотреть также в работе [9].

Рассмотрим пример аннотированного дерева разбора для последовательности операторов

if *a* > *b* **then** *c* := *a* **else** *c* := *b* **;**
while *c* < *d* **do** *c* := *c* + *k*

Отсчет номеров позиций команд начнем с 50. На аннотированном дереве разбора атрибуты для удобства обозначены: *truelist – t, falselist – f, instr – i, nextlist – n*, значения атрибутов *truelist, falselist* и *nextlist* показываются как содержимое списков.

50: **if** a > b **goto ?**
51: **goto ?**
52:



if $a > b$ **then** $c := a$ **else** $c := b$ **;**

Продукция	Семантические правила
1) $L \to L_1$ **;** $M\ S$	*BackPatch* (L_1.*nextlist*, M.*instr*) L.*nextlist* := S.*nextlist*
2) $L \to S$	L.*nextlist* := S.*nextlist*
6) $S \to$ **if** B **then** $M_1\ S_1\ N$ **else** $M_2\ S_2$	*BackPatch* (B.*truelist*, M_1.*instr*) *BackPatch* (B.*falselist*, M_2.*instr*) *tmp* := *Merge* (S_1.*nextlist*, N.*nextlist*) S.*nextlist* := *Merge* (*tmp*, S_2.*nextlist*)
9) $M \to \varepsilon$	M.*instr* := *nextinstr*

Из СУО для логических выражений

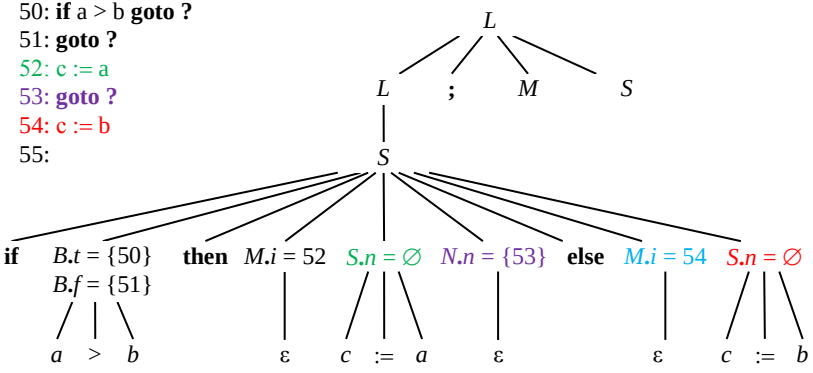
Продукция	Семантические правила
5) $B \to E_1$ **rel** E_2	B.*truelist* := *MakeList* (*nextinstr*) B.*falselist* := *MakeList* (*nextinstr* + 1) *Gen* ('if' E_1.*addr* **rel**.*op* E_2.*addr* 'goto ?') *Gen* ('goto ?')

В соответствии с продукцией 5 из СУО для логических выражений формируются две команды:

```
50: if a > b goto ?
51: goto ?
```

С помощью маркера M_1 в атрибуте M_1.*instr* сохраняется текущее значение *nextinstr*, равное 52.

50: **if** a > b **goto ?**
51: **goto ?**
52: c := a
53: **goto ?**
54: c := b
55:



if *a* > *b* **then** *c* := *a* **else** *c* := *b* **;**

Продукция	Семантические правила
6) $S \rightarrow$ **if** B **then** M_1 S_1 N **else** M_2 S_2	*BackPatch* (*B.truelist*, M_1*.instr*) *BackPatch* (*B.falselist*, M_2*.instr*) *tmp* := *Merge* (S_1*.nextlist*, *N.nextlist*) *S.nextlist* := *Merge* (*tmp*, S_2*.nextlist*)
9) $M \rightarrow \varepsilon$	*M.instr* := *nextinstr*
10) $N \rightarrow \varepsilon$	*N.nextlist* := *MakeList* (*nextinstr*) *Gen* ('goto ?')

Из СУО для арифметических выражений (в нем не было атрибута *S.nextlist*, надо добавить)

Продукция	Семантические правила
$S \rightarrow$ **id** := E	*Gen* (**id**.*pnt* ':=' *E.addr*) *S.nextlist* := **null**

В соответствии с продукцией $S \rightarrow$ **id** := E из СУО для арифметических выражений формируется команда:

```
52: c := a
```

С помощью маркера N в атрибуте *N.nextlist* сохраняется текущее значение *nextinstr*, равное 53 и формируется команда

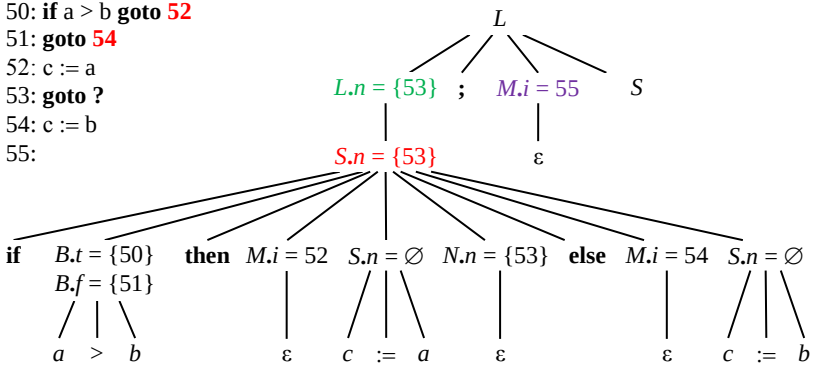
```
53: goto ?
```

С помощью маркера M_2 в атрибуте M_2*.instr* сохраняется текущее значение *nextinstr*, равное 54.

В соответствии с продукцией $S \rightarrow$ **id** := E из СУО для арифметических выражений формируется команда:

```
54: c := b
```

50: **if** a > b **goto** 52
51: **goto** 54
52: c := a
53: **goto ?**
54: c := b
55:



if *a* > *b* **then** *c* := *a* **else** *c* := *b* **;**

Продукция	Семантические правила
1) $L \to L_1$ **;** $M\ S$	*BackPatch* (L_1.*nextlist*, *M*.*instr*) *L*.*nextlist* := *S*.*nextlist*
2) $L \to S$	*L*.*nextlist* := *S*.*nextlist*
6) $S \to$ **if** B **then** $M_1\ S_1\ N$ **else** $M_2\ S_2$	*BackPatch* (*B*.*truelist*, M_1.*instr*) *BackPatch* (*B*.*falselist*, M_2.*instr*) *tmp* := *Merge* (S_1.*nextlist*, *N*.*nextlist*) *S*.*nextlist* := *Merge* (*tmp*, S_2.*nextlist*)
9) $M \to \varepsilon$	*M*.*instr* := *nextinstr*

К данному моменту завершено формирование кода для условного оператора. Атрибуты имеют следующие значения: *B*.*truelist* = {50}, *B*.*falselist* = {51}, M_1.*instr* = 52, *N*.*nextlist* = {53}, M_2.*instr* = 54, списки S_1.*nextlist* и S_2.*nextlist* пустые. В результате выполнения процедуры *BackPatch* ({50}, 52) команда 50 получит целевую метку 52.

`50: if a > b goto 52`

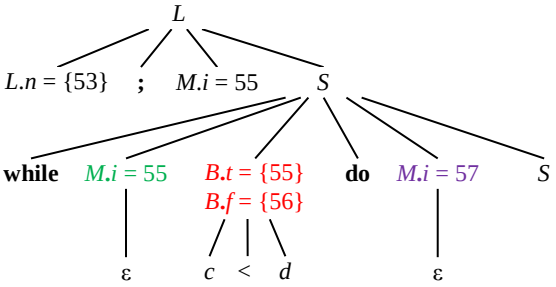
В результате выполнения *BackPatch* ({51}, 54) команда 51 получит целевую метку 54.

`51: goto 54`

Список *S*.*nextlist* = {53} образуется в результате объединения
Merge (S_1.*nextlist*, *N*.*nextlist*, S_2.*nextlist*).

В соответствии с продукцией 2 ($L \to S$) списком *L*.*nextlist* становится список *S*.*nextlist* = {53}.

С помощью маркера *M* в атрибуте *M*.*instr* (продукция 1) сохраняется текущее значение *nextinstr*, равное 55.



50: **if** a > b **goto 52**
51: **goto 54**
52: c := a
53: **goto ?**
54: c := b
55: **if** c < d **goto ?**
56: **goto ?**
57:

Рассмотрим теперь оператор цикла
while $c < d$ **do** $c := c + k$

Продукция	Семантические правила
7) $S \to$ **while** M_1 B **do** M_2 S_1	*BackPatch* (S_1.*nextlist*, M_1.*instr*) *BackPatch* (*B.truelist*, M_2.*instr*) *S.nextlist* := *B.falselist* *Gen* ('goto' M_1.*instr*)
9) $M \to \varepsilon$	*M.instr* := *nextinstr*

Из СУО для логических выражений

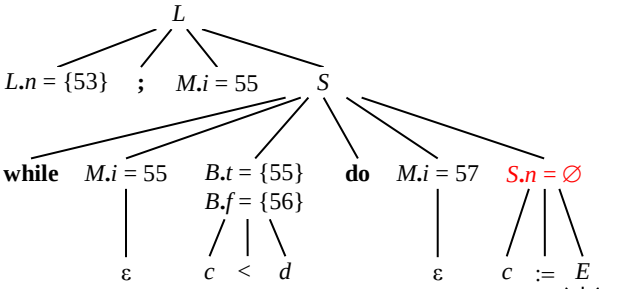
Продукция	Семантические правила
5) $B \to E_1$ **rel** E_2	*B.truelist* := *MakeList* (*nextinstr*) *B.falselist* := *MakeList* (*nextinstr* + 1) *Gen* ('if' E_1.*addr* **rel.***op* E_2.*addr* 'goto ?') *Gen* ('goto ?')

С помощью маркера M_1 в атрибуте M_1.*instr* (продукция 7) сохраняется текущее значение *nextinstr*, равное 55.

В соответствии с продукцией 5 из СУО для логических выражений формируются две команды:

```
55: if c < d goto ?
56: goto ?
```

С помощью маркера M_2 в атрибуте M_2.*instr* (продукция 7) сохраняется текущее значение *nextinstr*, равное 57.



50: **if** a > b **goto 52**
51: **goto 54**
52: c := a
53: **goto ?**
54: c := b

55: **if** c < d **goto ?**
56: **goto ?**
57: t_1 := c + k
58: c := t_1
59:

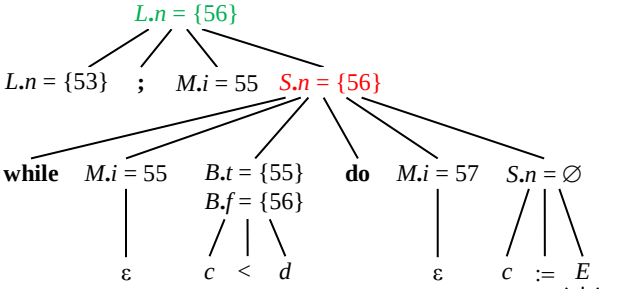
while *c* < *d* **do** *c* := *c* + *k*

Из СУО для арифметических выражений (в нем не было атрибута *S.nextlist*, надо добавить)

Продукция	Семантические правила
$S \to$ **id** := E	*Gen* (**id**.*pnt* ':=' *E.addr*) *S.nextlist* := **null**
$E \to E_1 + T$	*E.addr* := *NewTemp* () *Gen* (*E.addr* ':=' E_1.*addr* '+' *T.addr*)
$E \to T$	*E.addr* := *T.addr*
$T \to F$	*T.addr* := *F.addr*
$F \to$ **id**	*F.addr* := **id**.*pnt*

В соответствии с СУО для арифметических выражений формируются две команды (промежуточные замены типа $E \Rightarrow T \Rightarrow F \Rightarrow$ **id** в дереве не показаны):

```
57: t1 := c + k
58: c := t1
```



50: **if** a > b **goto 52**
51: **goto 54**
52: c := a
53: **goto 55**
54: c := b
55: **if** c < d **goto 57**

56: **goto ?**
57: t_1 := c + k
58: c := t_1
59: **goto 55**
60:

while *c* < *d* **do** *c* := *c* + *k*

Продукция	Семантические правила
1) $L \to L_1$ **;** $M\ S$	*BackPatch* (L_1.*nextlist*, *M*.*instr*) *L*.*nextlist* := *S*.*nextlist*
7) $S \to$ **while** $M_1\ B$ **do** $M_2\ S_1$	*BackPatch* (S_1.*nextlist*, M_1.*instr*) *BackPatch* (*B*.*truelist*, M_2.*instr*) *S*.*nextlist* := *B*.*falselist* *Gen* ('goto' M_1.*instr*)
9) $M \to \varepsilon$	*M*.*instr* := *nextinstr*

К данному моменту завершено формирование кода S_1. Атрибуты имеют следующие значения: M_1.*instr* = 55, *B*.*truelist* = {55}, *B*.*falselist* = {56}, M_2.*instr* = 57, список S_1.*nextlist* пустой.

В результате выполнения процедуры *BackPatch* (∅, 55) ничего не происходит. В результате выполнения *BackPatch* ({55}, 57) команда 55 получит целевую метку 57.

```
55: if c < d goto 57
```

Списком *S*.*nextlist* становится список *B*.*falselist* = {56}.

Формируется команда

```
59: goto 55
```

Завершено формирование кода для оператора цикла. Выполняются действия для продукции 1. В результате выполнения процедуры *BackPatch* ({53}, 55) команда 53 получит целевую метку 55.

```
53: goto 55
```

Списком *L*.*nextlist* становится список *S*.*nextlist* = {56}.

В результате для последовательности команд

if $a > b$ **then** $c := a$ **else** $c := b$ **;**

while $c < d$ **do** $c := c + k$

формируется следующий трехадресный код:

```
50: if a > b goto 52
51: goto 54
52: c := a
53: goto 55
54: c := b
55: if c < d goto 57
56: goto ?
57: t1 := c + k
58: c := t1
59: goto 55
60:
```

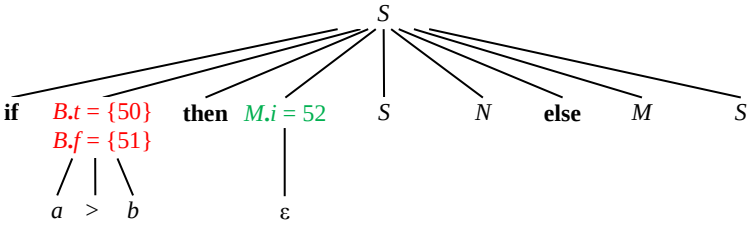
В команде 56 еще нет целевой метки, поскольку соответствующий код еще не сформирован!

Рассмотрим еще один пример:

if $a > b$ **then repeat** $c := c + k$ **until** $c < d$
else $c := b$

Отсчет номеров позиций трехадресных команд начнем с 50. На аннотированном дереве разбора атрибуты для удобства обозначены: *truelist – t, falselist – f, instr – i, nextlist – n*, значения атрибутов *truelist, falselist* и *nextlist* показываются как содержимое списков.

Поскольку у нас один оператор, в дереве не показаны фрагменты для продукций $L \to L_1$ **;** $M\ S$ и $L \to S$, а сразу начинаем построение с нетерминала S.



50: **if** a > b **goto ?**
51: **goto ?**
52:

if $a > b$ **then repeat** $c := c + k$ **until** $c < d$
else $c := b$

Продукция	Семантические правила
6) $S \rightarrow$ **if** B **then** M_1 S_1 N **else** M_2 S_2	*BackPatch* (*B.truelist*, M_1*.instr*) *BackPatch* (*B.falselist*, M_2*.instr*) *tmp* := *Merge* (S_1*.nextlist*, *N.nextlist*) *S.nextlist* := *Merge* (*tmp*, S_2*.nextlist*)
9) $M \rightarrow \varepsilon$	*M.instr* := *nextinstr*

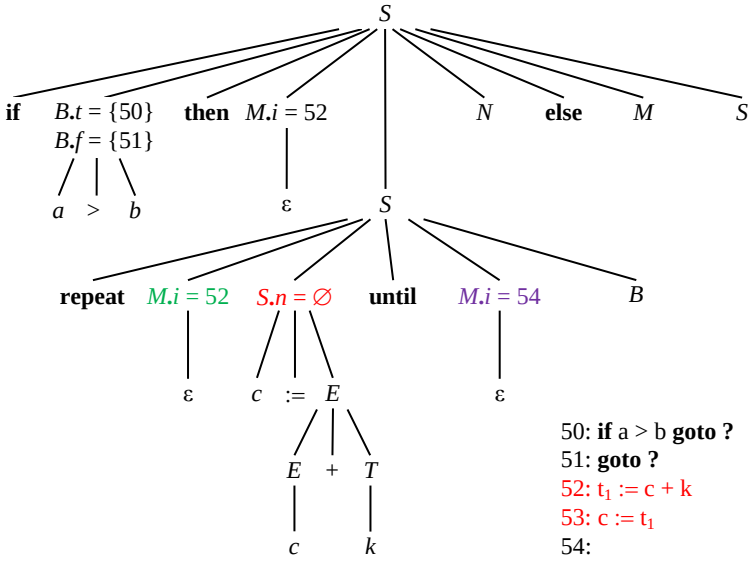
Из СУО для логических выражений

Продукция	Семантические правила
5) $B \rightarrow E_1$ **rel** E_2	*B.truelist* := *MakeList* (*nextinstr*) *B.falselist* := *MakeList* (*nextinstr* + 1) *Gen* ('if' E_1*.addr* **rel.***op* E_2*.addr* 'goto ?') *Gen* ('goto ?')

В соответствии с продукцией 5 из СУО для логических выражений формируются две команды:

```
50: if a > b goto ?
51: goto ?
```

С помощью маркера M_1 в атрибуте M_1*.instr* сохраняется текущее значение *nextinstr*, равное 52.



if $a > b$ **then repeat** $c := c + k$ **until** $c < d$
else $c := b$

Продукция	Семантические правила
8) $S \to$ **repeat** $M_1\ S_1$ **until** $M_2\ B$	*BackPatch* (S_1.*nextlist*, M_2.*instr*) *BackPatch* (*B.falselist*, M_1.*instr*) *S.nextlist* := *B.truelist*
9) $M \to \varepsilon$	*M.instr* := *nextinstr*

Из СУО для арифметических выражений (в нем не было атрибута *S.nextlist*, надо добавить)

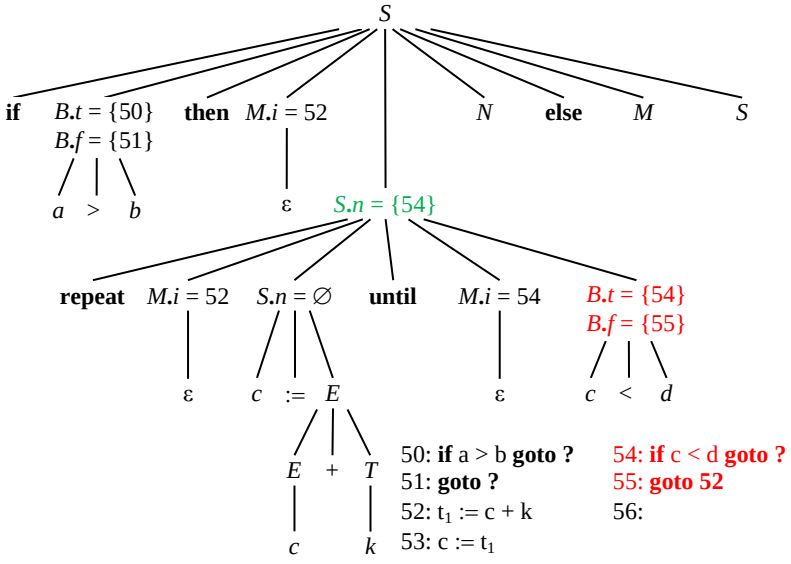
Продукция	Семантические правила
$S \to$ **id** := E	*Gen* (**id**.*pnt* ':=' *E.addr*) *S.nextlist* := **null**
$E \to E_1 + T$	*E.addr* := *NewTemp* () *Gen* (*E.addr* ':=' E_1.*addr* '+' *T.addr*)
$E \to T$	*E.addr* := *T.addr*
$T \to F$	*T.addr* := *F.addr*
$F \to$ **id**	*F.addr* := **id**.*pnt*

С помощью маркера M_1 в атрибуте M_1.*instr* сохраняется текущее значение *nextinstr*, равное 52 (продукция 8).

В соответствии с СУО для арифметических выражений формируются две команды (промежуточные замены типа $E \Rightarrow T \Rightarrow F \Rightarrow$ **id** в дереве не показаны):

```
52: t1 := c + k
53: c := t1
```

С помощью маркера M_2 в атрибуте M_2.*instr* сохраняется текущее значение *nextinstr*, равное 54 (продукция 8).



if $a > b$ **then repeat** $c := c + k$ **until** $c < d$
else $c := b$

Продукция	Семантические правила
8) $S \to$ **repeat** $M_1\ S_1$ **until** $M_2\ B$	*BackPatch* (S_1.*nextlist*, M_2.*instr*) *BackPatch* (*B.falselist*, M_1.*instr*) *S.nextlist* := *B.truelist*
9) $M \to \varepsilon$	*M.instr* := *nextinstr*

Из СУО для логических выражений

Продукция	Семантические правила
5) $B \to E_1$ **rel** E_2	*B.truelist* := *MakeList* (*nextinstr*) *B.falselist* := *MakeList* (*nextinstr* + 1) *Gen* ('if' E_1.*addr* **rel**.*op* E_2.*addr* 'goto ?') *Gen* ('goto ?')

В соответствии с продукцией 5 из СУО для логических выражений формируются две команды:

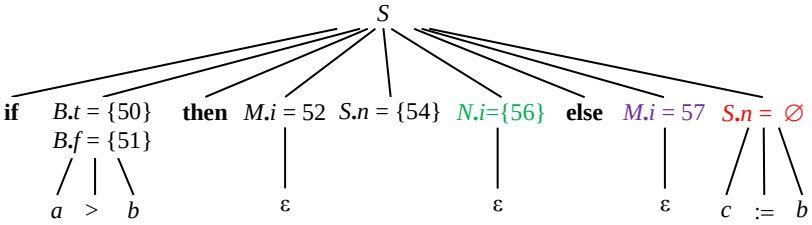
```
54: if c < d goto ?
55: goto ?
```

К данному моменту завершено формирование кода *B*. Атрибуты имеют следующие значения: M_1.*instr* = 52, *B.truelist* = {54}, *B.falselist* = {55}, M_2.*instr* = 54, S_1.*nextlist* = ∅.

В результате выполнения процедуры *BackPatch* (∅, 55) ничего не происходит. В результате выполнения *BackPatch* ({55}, 52) команда 55 получит целевую метку 52.

```
55: goto 52
```

Списком *S.nextlist* становится список *B.truelist* = {54}.



50: **if** a > b **goto ?**
51: **goto ?**
52: t_1 := c + k
53: c := t_1

54: **if** c < d **goto ?**
55: **goto 52**
56: **goto ?**
57: c := b
58:

if *a* > *b* **then repeat** *c* := *c* + *k* **until** *c* < *d*
else *c* := *b*

Продукция	Семантические правила
6) $S \rightarrow$ **if** B **then** M_1 S_1 N **else** M_2 S_2	*BackPatch* (*B.truelist*, M_1*.instr*) *BackPatch* (*B.falselist*, M_2*.instr*) *tmp* := *Merge* (S_1*.nextlist*, *N.nextlist*) *S.nextlist* := *Merge* (*tmp*, S_2*.nextlist*)
9) $M \rightarrow \varepsilon$	*M.instr* := *nextinstr*
10) $N \rightarrow \varepsilon$	*N.nextlist* := *MakeList* (*nextinstr*) *Gen* ('goto ?')

Далее фрагмент дерева для оператора цикла для компактности не будет показан.

Из СУО для арифметических выражений (в нем не было атрибута *S.nextlist*, надо добавить)

Продукция	Семантические правила
$S \rightarrow$ **id** := E	*Gen* (**id**.*pnt* ':=' *E.addr*) *S.nextlist* := **null**

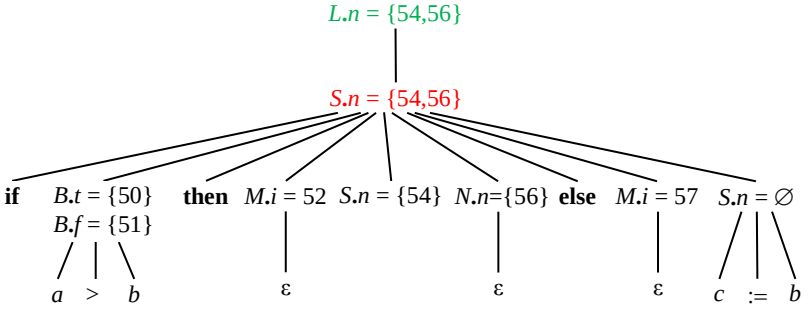
С помощью маркера N в атрибуте *N.nextlist* сохраняется текущее значение *nextinstr*, равное 56 и формируется команда

```
56: goto ?
```

С помощью маркера M_2 в атрибуте M_2*.instr* сохраняется текущее значение *nextinstr*, равное 57.

В соответствии с продукцией $S \rightarrow$ **id** := E из СУО для арифметических выражений формируется команда:

```
57: c := b
```



50: **if** a > b **goto** 52
51: **goto** 57
52: t_1 := c + k
53: c := t_1

54: **if** c < d **goto ?**
55: **goto** 52
56: **goto ?**
57: c := b
58:

if *a* > *b* **then repeat** *c* := *c* + *k* **until** *c* < *d*
else *c* := *b*

Продукция	Семантические правила
2) $L \rightarrow S$	*L*.*nextlist* := *S*.*nextlist*
5) $S \rightarrow$**if** *B* **then** M_1 S_1 *N* **else** M_2 S_2	*BackPatch* (*B*.*truelist*, M_1.*instr*) *BackPatch* (*B*.*falselist*, M_2.*instr*) *tmp* := *Merge* (S_1.*nextlist*, *N*.*nextlist*) *S*.*nextlist* := *Merge* (*tmp*, S_2.*nextlist*)

К данному моменту завершено формирование кода для условного оператора. Атрибуты имеют следующие значения: *B.truelist* = {50}, *B.falselist* = {51}, M_1.*instr* = 52, *N.nextlist* = {56}, M_2.*instr* = 57, S_1.*nextlist* = {54}, S_2.*nextlist* пустой. В результате выполнения процедуры *BackPatch* ({50}, 52) команда 50 получит целевую метку 52.

```
50: if a > b goto 52
```

В результате выполнения *BackPatch* ({51}, 57) команда 51 получит целевую метку 57.

```
51: goto 57
```

Список *S.nextlist* = {54,56} образуется в результате объединения
Merge (S_1.*nextlist*, *N.nextlist*, S_2.*nextlist*).

Списком *L.nextlist* становится список *S.nextlist* = {54,56}.

В результате для команды

if $a > b$ **then repeat** $c := c + k$ **until** $c < d$
else $c := b$

формируется следующий трехадресный код:

```
50: if a > b goto 52
51: goto 57
52: t1 := c + k
53: c := t1
54: if c < d goto ?
55: goto 52
56: goto ?
57: c := b
58:
```

В командах 54 и 56 еще нет целевых меток, поскольку соответствующий код еще не сформирован! Обратите внимание, что команда 56 избыточна и никогда не будет выполнена.