1. 语言和文法

文法（grammar）

表达语言构成规则的形式化方法

$G=(V_N, V_T, S, P)$

V_N:非终结符集

V_T:终结符集

S:文法开始符号

P:产生式　　$A \to \alpha$

由文法产生语言（3型）

Regular Grammar

例：设文法G_1＝({S},{a,b}, S,P)，其中P为:

(0) $S \rightarrow aS$

(1) $S \rightarrow a$

(2) $S \rightarrow b$

答：$L(G_1) = \{a^i (a \mid b) \mid i >= 0\}$

由文法产生语言（2型）

例：设文法G_2＝({S},{a,b},P,S)，其中P为:

(0) $S \rightarrow aSb$

(1) $S \rightarrow ab$

答：$L(G_2)=\{a^nb^n|n>=1\}$

由语言构造文法 - 题型

- 构造形如$a^{mi}b^{ni}$的语言的文法

  $S \rightarrow \underbrace{a\cdots a}_{m个a}S\underbrace{b\cdots b}_{n个b} \mid \varepsilon$

- $a^i b^j$, $(i \geq 2j,\ j \geq 1)$

  $a^{i-2j}a^{2j}b^j$

由语言构造文法（续）

例：设$L_2=\{a^i b^j c^k \mid i, j, k>=1$ 且$a, b, c \in V_T\}$，试构造生成L_2的文法G_2。

答：

(0) $S \rightarrow aS \mid aB$

(1) $B \rightarrow bB \mid bC$

(2) $C \rightarrow cC \mid c$ L2R

(0) $S \rightarrow ABC$

(1) $A \rightarrow aA \mid a$

(2) $B \rightarrow bB \mid b$

(3) $C \rightarrow cC \mid c$ T2B

构造无ε产生式的上下文无关文法

- 无ε产生式的上下文无关文法要满足条件
  - 若P中含$S \rightarrow \varepsilon$，则S不出现在任何产生式右部，其中S为文法的开始符号;
  - P中不再含有其它任何ε产生式。

例题

设G_1=({S},{a,b},P,S),其中

P: (0) $S \to \varepsilon$ (1) $S \to aSbS$ (2) $S \to bSaS$

答案:

(1) $V_0 = \{S\}$

(2) P':

$S \to abS \mid aSbS \mid aSb \mid ab$

$S \to baS \mid bSaS \mid bSa \mid ba$

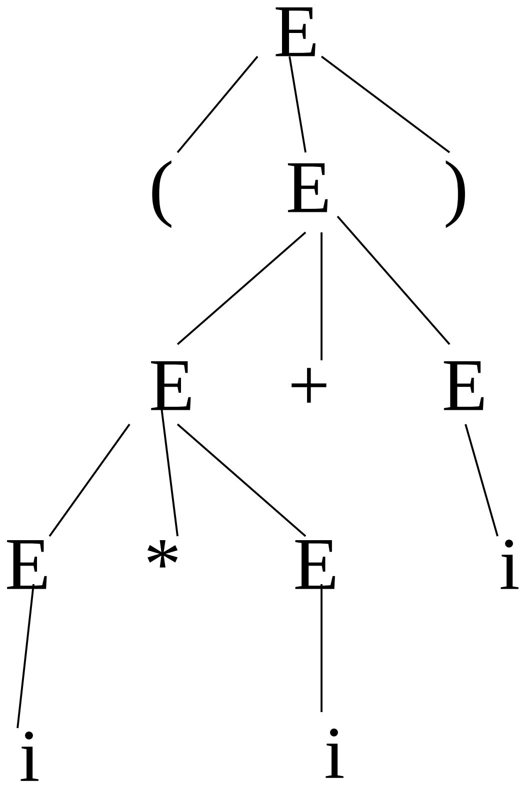
$S' \to \varepsilon \mid S$

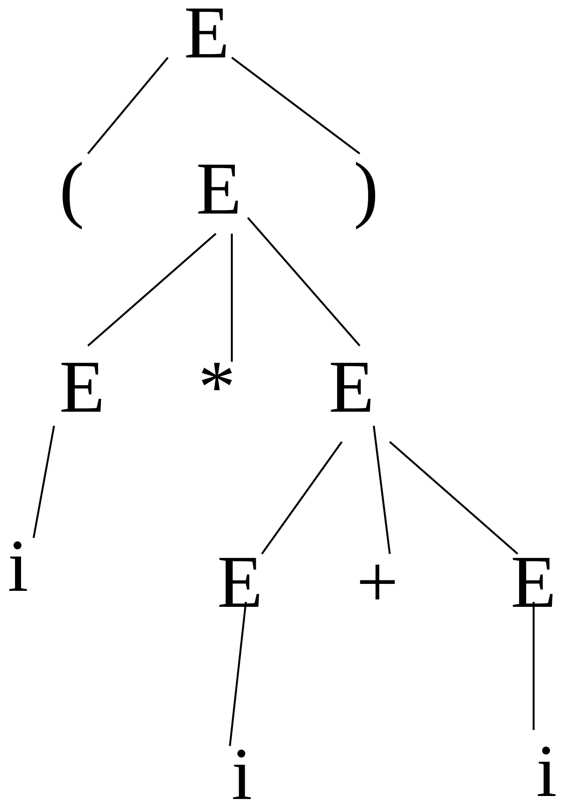
(3) $G_1' = (\{S', S\}, \{a, b\}, P', S')$

二义性文法

$E \rightarrow E + E \mid E * E \mid (E) \mid i$

$(i * i + i)$





非二义性文法

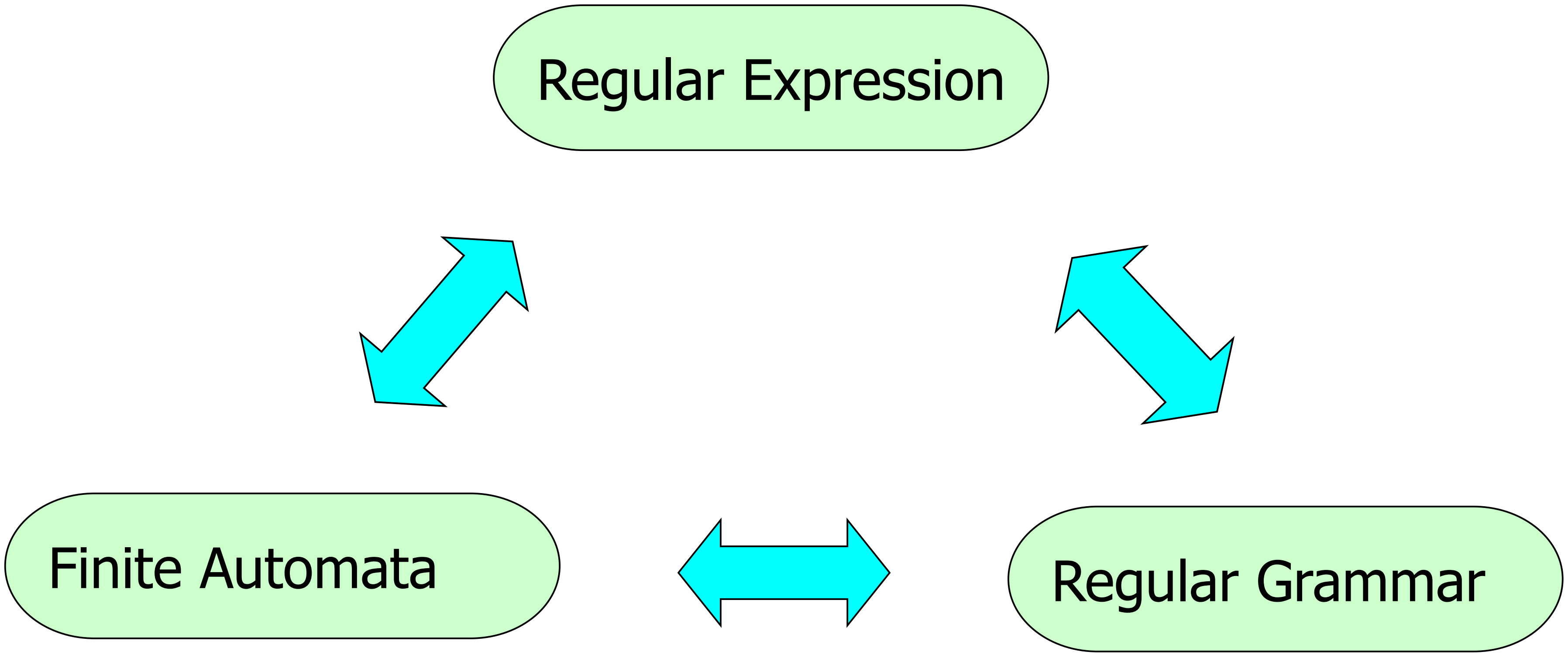
$E \rightarrow T \mid E + T$

$T \rightarrow F \mid T * F$

$F \rightarrow (E) \mid i$

2. 词法分析

Theme of this Chapter



NFA→DFA→DFA_{min}

- Subset Construction algorithm

(1)$I_0 = \varepsilon$-closure(S_0), $I_0 \in Q$

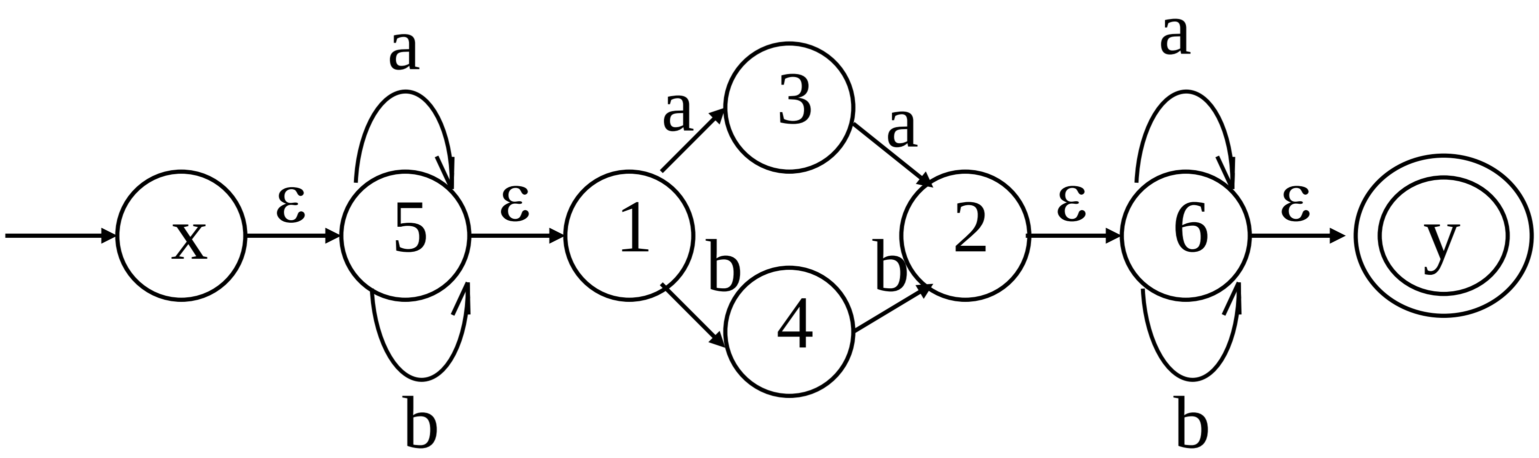
(2)For each I_i , $I_i \in Q$,

let $I_t = \varepsilon$-closure(move(I_i,a))

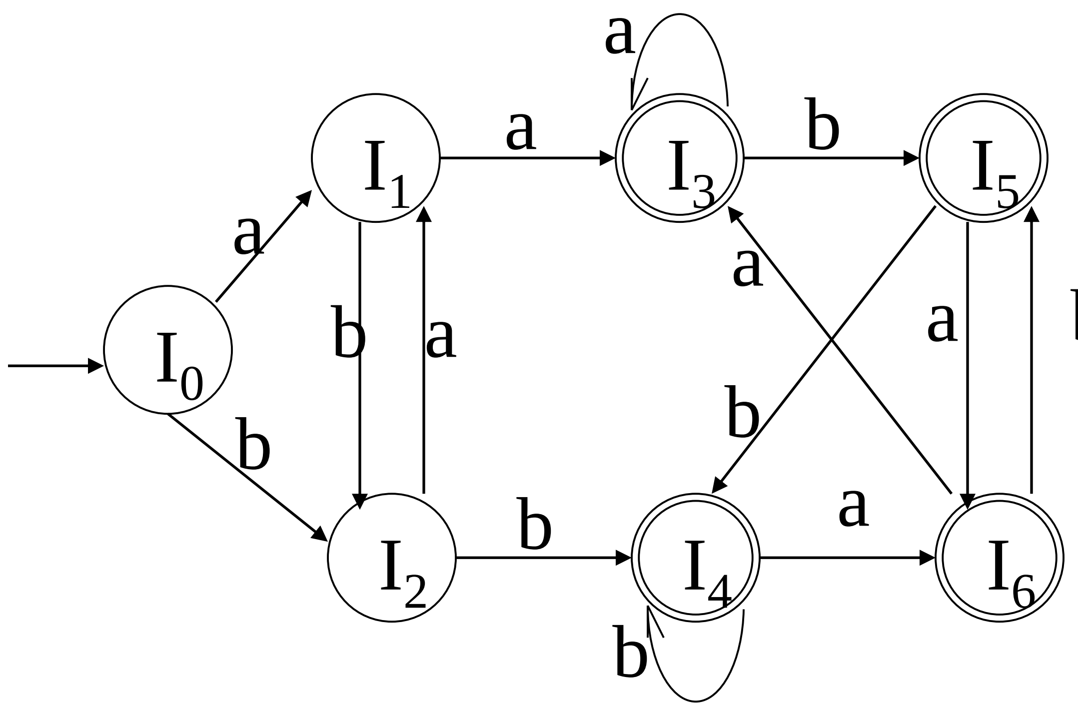
if $I_t \notin Q$, then put I_t into Q

(3)Repeat step (2), until there is no new state to put into Q

(4)Let $F=\{I \mid I \in Q, \text{and } I \cap Z <> \Phi\}$

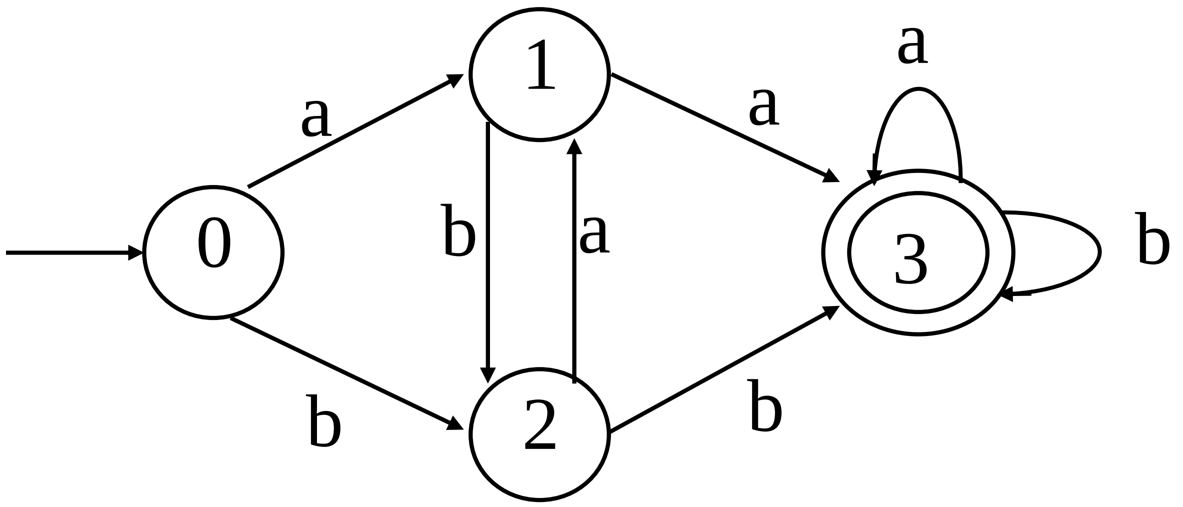


I	a	b
$I_0=\{x,5,1\}$	$I_1=\{5,3,1\}$	$I_2=\{5,4,1\}$
$I_1=\{5,3,1\}$	$I_3=\{5,3,2,1,6,y\}$	$I_2=\{5,4,1\}$
$I_2=\{5,4,1\}$	$I_1=\{5,3,1\}$	$I_4=\{5,4,1,2,6,y\}$
$I_3=\{5,3,2,1,6,y\}$	$I_3=\{5,3,2,1,6,y\}$	$I_5=\{5,1,4,6,y\}$
$I_4=\{5,4,2,1,6,y\}$	$I_6=\{5,3,1,6,y\}$	$I_4=\{5,4,1,2,6,y\}$
$I_5=\{5,1,4,6,y\}$	$I_6=\{5,3,1,6,y\}$	$I_4=\{5,4,1,2,6,y\}$
$I_6=\{5,3,1,6,y\}$	$I_3=\{5,3,2,1,6,y\}$	$I_4=\{5,1,4,6,y\}$



$\Pi_0 = \{ \{0,1,2\},\{3,4,5,6\} \}$

$\Pi_1 = \{ \{1\},\ \{0\},\ \{2\},\ \{3,4,5,6\} \}$



FA

Write grammer for the following languages over the alphabet Σ={0, 1}:

(1) All strings that contain an odd number of 1's.

(2) All strings which do not contain the substring 01.

3. 语法分析

To Avoid Backtracking

- No left recursion (direct & indirect)

  $P \rightarrow P\alpha|\beta$ =>

  $P \rightarrow \beta P'$ $P' \rightarrow \alpha P'|\varepsilon$

- No common prefixes

  extract common left factor

- No ambiguity

  rewrite the grammar

  operator: precedence & associativity

  else dangling

Compute First & Follow Set

- FIRST(α)

  $\alpha=\alpha_1\alpha_2\ldots\alpha_n$

  FIRST(α) = FIRST(α_1), $\varepsilon \notin$ FIRST(α_1)

  (FIRST(α_1)-{ε}) $\cup$ (FIRST(α_2)-{ε}) …

  $\cup$FIRST(α_k)

  $\varepsilon \in$ FIRST(α_i) ($1 \le i < k$) , $\varepsilon \notin$ FIRST(α_k)

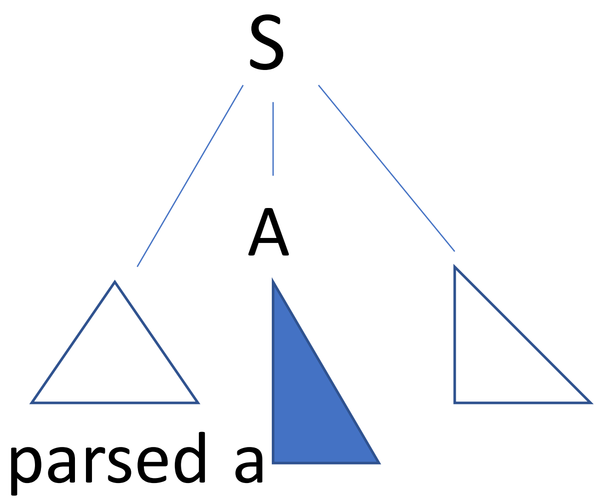
  $\varepsilon \in$ FIRST(α) if $\varepsilon \in$ FIRST(α_i) ($1 \le i \le n$)

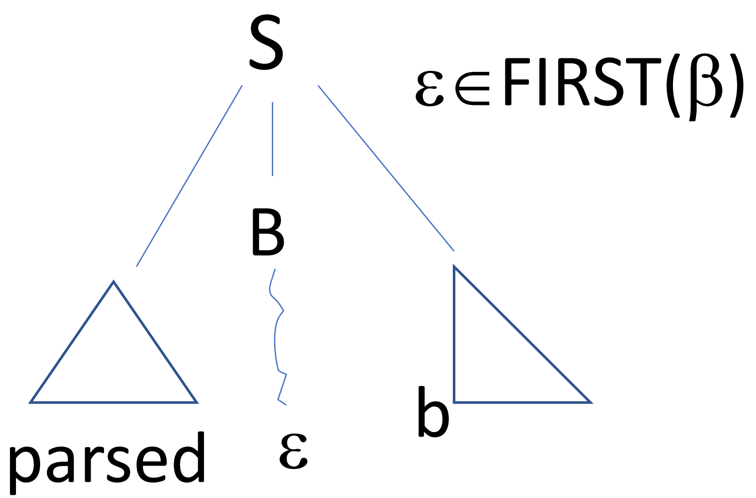
- FOLLOW(N)

  A->αNβ Add FIRST(β)-{ε}

  If $\varepsilon \in$ FIRST(β), or A->αN, Add FOLLOW(A)

Construct Predictive Parsing Table





$A \rightarrow \alpha$

$a \in FIRST(\alpha)$

	a
A	$A \rightarrow \alpha$

$B \rightarrow \beta$

$b \in FOLLOW(B)$

	b
B	$B \rightarrow \varepsilon$

LL(1) Grammar?

- Construct parsing table

  look for multidefined entries
- Look at FIRST & FOLLOW sets

  G is LL(1) $\Leftrightarrow$

  whenever there exists A->$\alpha|\beta$ in G,

  (1) FIRST(α)$\cap$ FIRST(β)=$\varnothing$

  (2) At most one of α and β derive the ε,

  if $\beta \Rightarrow \varepsilon$, then FIRST($\alpha$)$\cap$ FOLLOW(A)=$\varnothing$

  // Left recursion, common prefixes,

  ambiguious grammar

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid id$

1. $E \rightarrow TE'$
2. $E' \rightarrow +TE'$
3. $E' \rightarrow \varepsilon$
4. $T \rightarrow FT'$
5. $T' \rightarrow *FT'$
6. $T' \rightarrow \varepsilon$
7. $F \rightarrow id$
8. $F \rightarrow (E)$

First(E)=First(T)=First(F)={(, id}

First(E`)={+, ε}

First(T`)={*, ε}

Follow(E)= Follow(E`)={),$}

Follow(T)= Follow(T`)={+,),$}

Follow(F)={*,+,),$}

	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E’		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T’		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$		

LR Parsing

SLR(1)：FOLLOW

LR(1):

LALR(1)

Closure(I)

repeat {

for each item $[A \to \alpha \bullet B\beta, a]$ in I

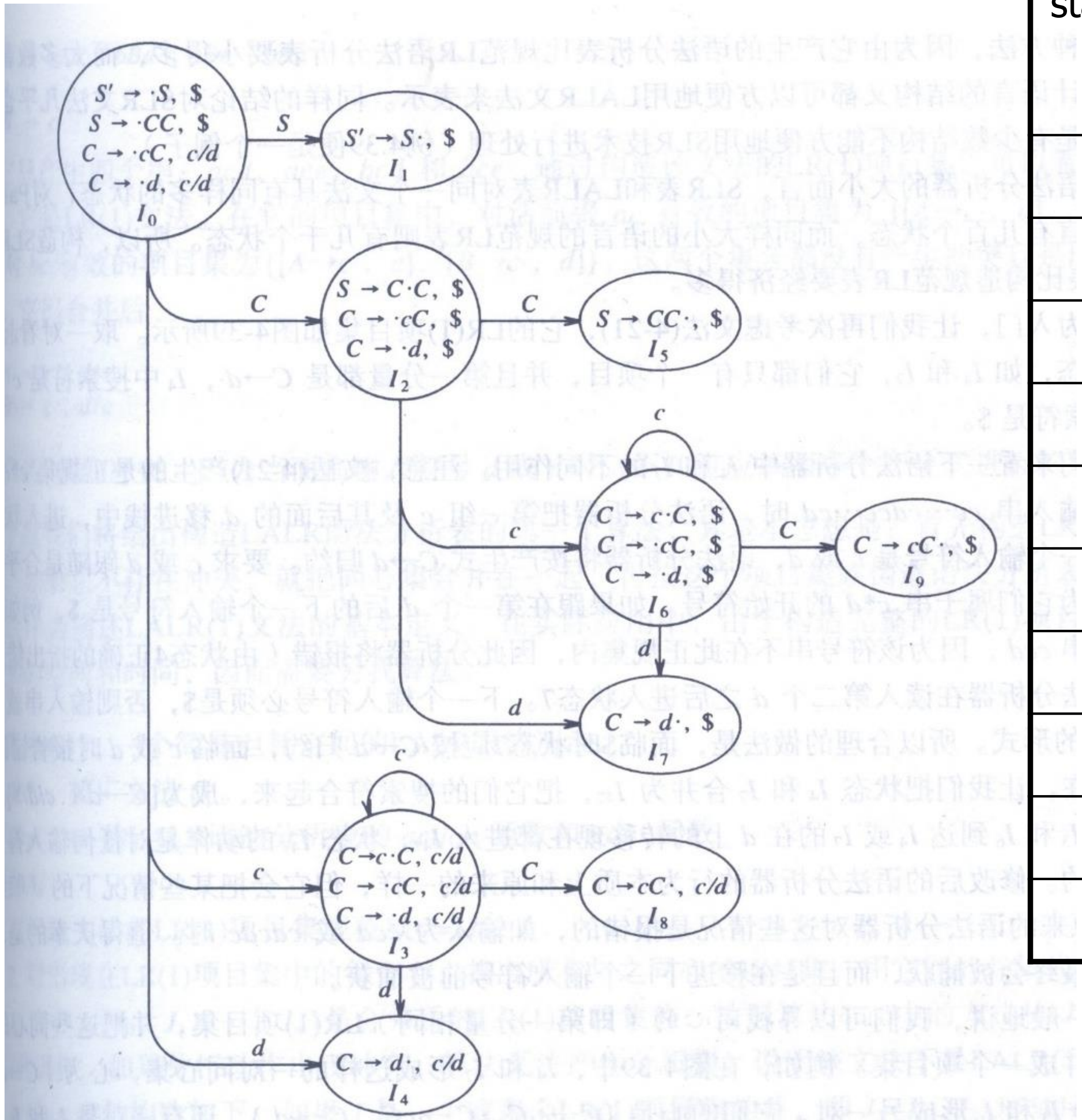
add all $[B \to \bullet\, \gamma\, , b]$ for all $b \in FIRST(\beta a)$ to I

(if not already in I);

}until (no more items can be added to I);

1. $S` \rightarrow S$
2. $S \rightarrow CC$
3. $C \rightarrow cC|d$

I_0 $S' \rightarrow \bullet S, \$$
$S` \rightarrow \bullet S,\$$ $S \rightarrow \bullet CC,\$$ $C \rightarrow \bullet cC, c\|d$ $C \rightarrow \bullet d,c\|d$



state	Action			goto	
	c	d	$	S	C
0	S_3	S_4		1	2
1			acc		
2	S_6	S_7			5
3	S_3	S_4			8
4	r_3	r_3			
5			r_1		
6	S_6	S_7			9
7			r_3		
8	r_2	r_2			
9			r_2		

文法二义性

I0:
$E' \to \bullet E$
$E \to \bullet E + E$
$E \to \bullet E * E$
$E \to \bullet (E)$
$E \to \bullet \mathrm{id}$

I1:
$E' \to E \bullet$
$E \to E \bullet + E$
$E \to E \bullet * E$

I2:
$E \to (\bullet E)$
$E \to \bullet E + E$
$E \to \bullet E * E$
$E \to \bullet (E)$
$E \to \bullet \mathrm{id}$

I3:
$E \to \mathrm{id} \bullet$

I4:
$E \to E + \bullet E$
$E \to \bullet E + E$
$E \to \bullet E * E$
$E \to \bullet (E)$
$E \to \bullet \mathrm{id}$

I5:
$E \to E * \bullet E$
$E \to \bullet E + E$
$E \to \bullet E * E$
$E \to \bullet (E)$
$E \to \bullet \mathrm{id}$

I6:
$E \to (E \bullet)$
$E \to E \bullet + E$
$E \to E \bullet * E$

I7:
$E \to E + E \bullet$
$E \to E \bullet + E$
$E \to E \bullet * E$

I8:
$E \to E * E \bullet$
$E \to E \bullet + E$
$E \to E \bullet * E$

I9:
$E \to (E) \bullet$

FOLLOW(E')={$}
FOLLOW(E): {$,),*,+}

id+id*id
Id+id+id

SLR(1)

I_1: E'->E•
E->E• +E
E->E• *E

I_7: E->E+E•
E->E• +E
E->E• *E

I_8: E->E*E•
E->E• +E
E->E• *E

	Action						goto
	id	+	*	(	)	$	E
0	S_3			S2			1
1		S4	S5			acc	
2	S_3			S2			6
3		r_4	r4		r4	r4	
4	S_3			S2			7
5	S_3			S_2		r_1	8
6		S_4	S_5		S_9		
7		r_1/S_4	S_5/r_1		r_1	r_1	
8		r_2/S_4	r_2/S_5		r_2	r_2	
9		r_3	r_3		r_3	r_3	

id+id*id 0,1,4,7 E+E *id$

Id+id+id 0,1,4,7 E+E +id$

I_7: E->E+E•
E->E• +E
E->E• *E

I_8: E->E*E•
E->E• +E
E->E• *E

r_1/S_4	S_5/r_1
r_2/S_4	r_2/S_5

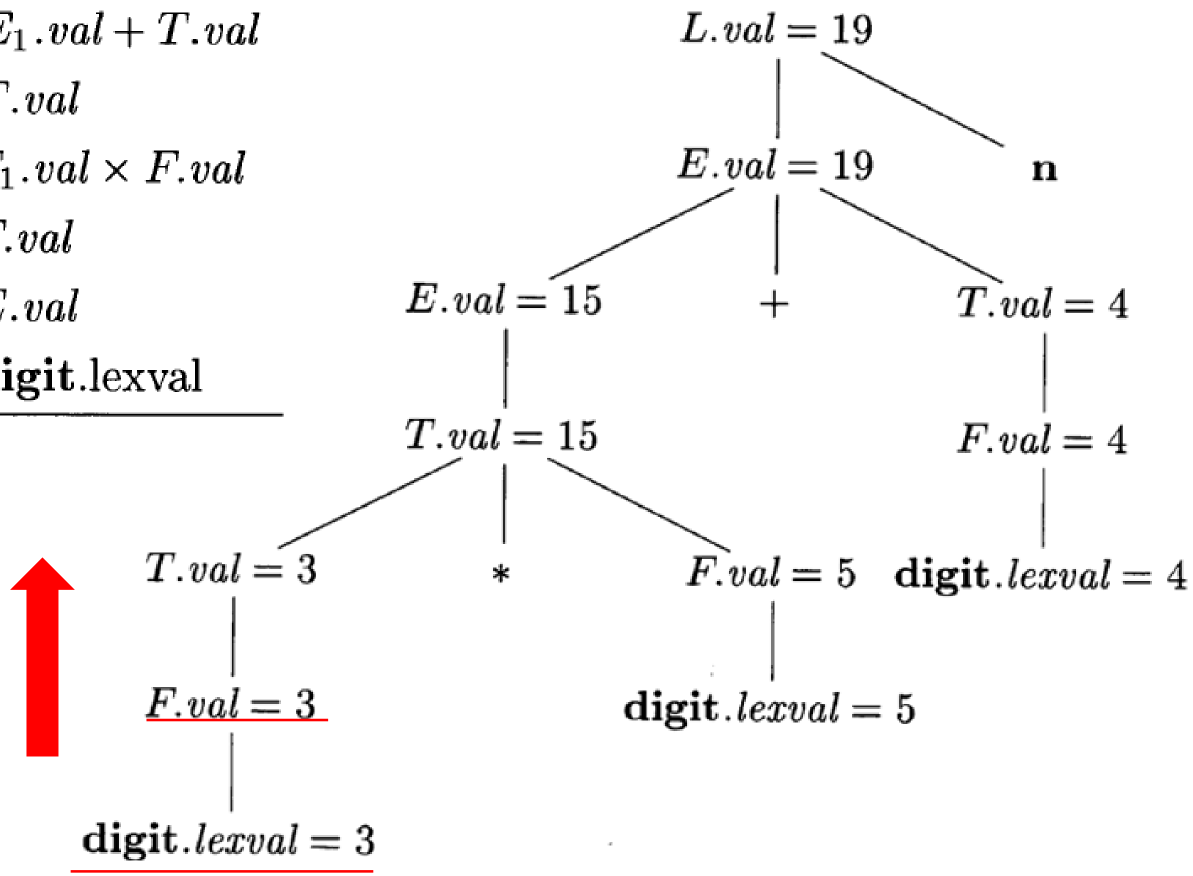
Specify precedence & associativity

	Action						goto
	id	+	*	(	)	$	E
0	S_3			S2			1
1		S4	S5			acc	
2	S_3			S2			6
3		r_4	r4		r4	r4	
4	S_3			S2			7
5	S_3			S_2		r_1	8
6		S_4	S_5		S_9		
7		r_1	S_5		r_1	r_1	
8		r_2	r_2		r_2	r_2	
9		r_3	r_3		r_3	r_3	

4. 语法制导翻译

E.g. Annotated parse tree for 3∗5+4 **n**

产生式	语义规则
1) $L \rightarrow E$ **n**	$L.val = E.val$
2) $E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
3) $E \rightarrow T$	$E.val = T.val$
4) $T \rightarrow T_1 * F$	$T.val = T_1.val \times F.val$
5) $T \rightarrow F$	$T.val = F.val$
6) $F \rightarrow (E)$	$F.val = E.val$
7) $F \rightarrow$ **digit**	$F.val =$ **digit**.lexval



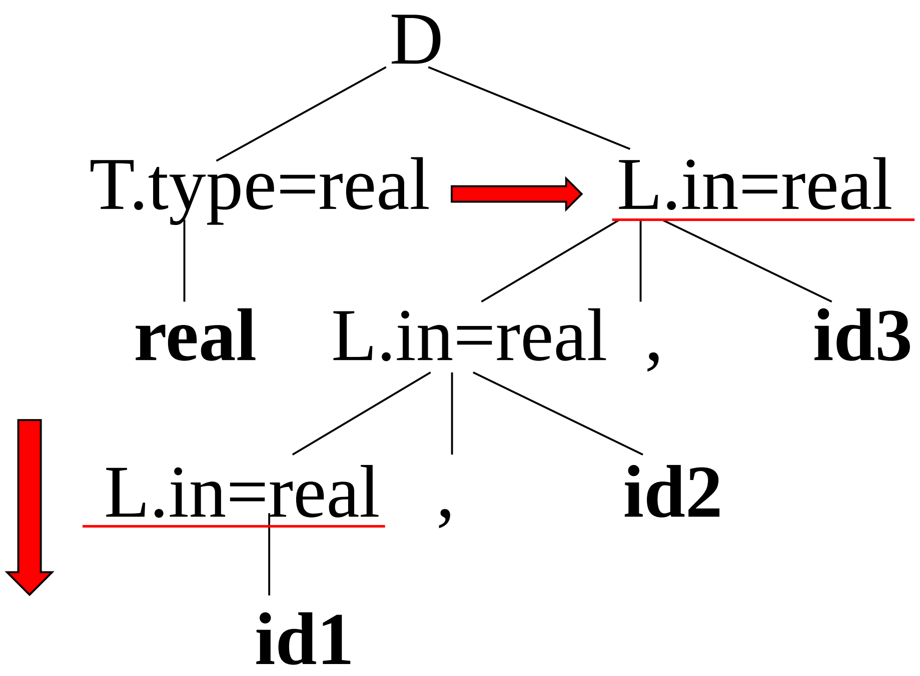
- 3∗5+4n

lexval, val： synthesized attribute

- E.g. Syntax-directed definition with inherited attribute L.in

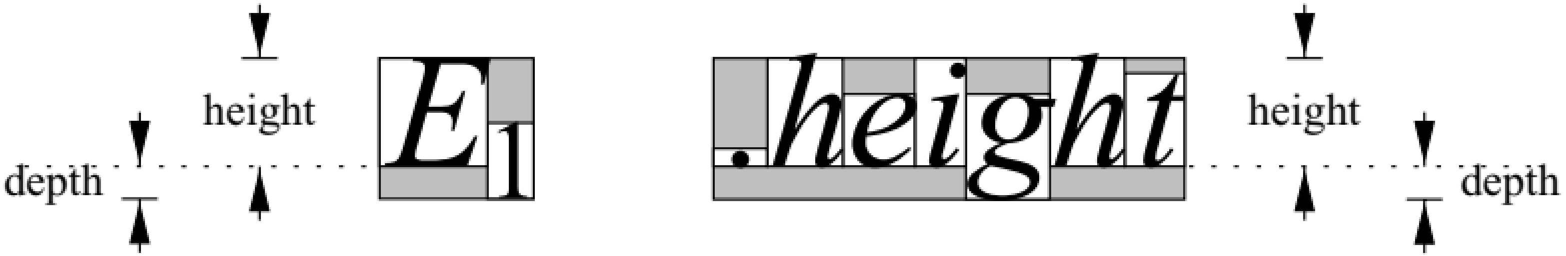
Production	Semantic rules
D $\rightarrow$T L	L.in=T.type
T $\rightarrow$**int**	T.type= integer
T $\rightarrow$**real**	T.type=real
L $\rightarrow$L $^{(1)}$,id	L $^{(1)}$.in=L.in addtype(**id**.entry, L.in)
L $\rightarrow$ **id**	addtype(**id**.entry, L.in)

real id1,id2,id3



in: inherited attribute
type: synthesized attribute

SDT for L-attributed SDD ---Typesetting Box



Production	Semantic Rules
$S \rightarrow B$	B.ps=10
$B \rightarrow B_1 B_2$	B_1.ps=B.ps
	B_2.ps=0.7*B.ps
	B.ht=max(B_1.ht,B_2.ht-0.25*B.ps)
	B.dp=max(B_1.dp,B_2.dp+0.25*B.ps)
$B \rightarrow$ text	B.ht=getHt(B.ps, text.lexval)
	B.dp=getDp(B.ps, text.lexval)

SDT

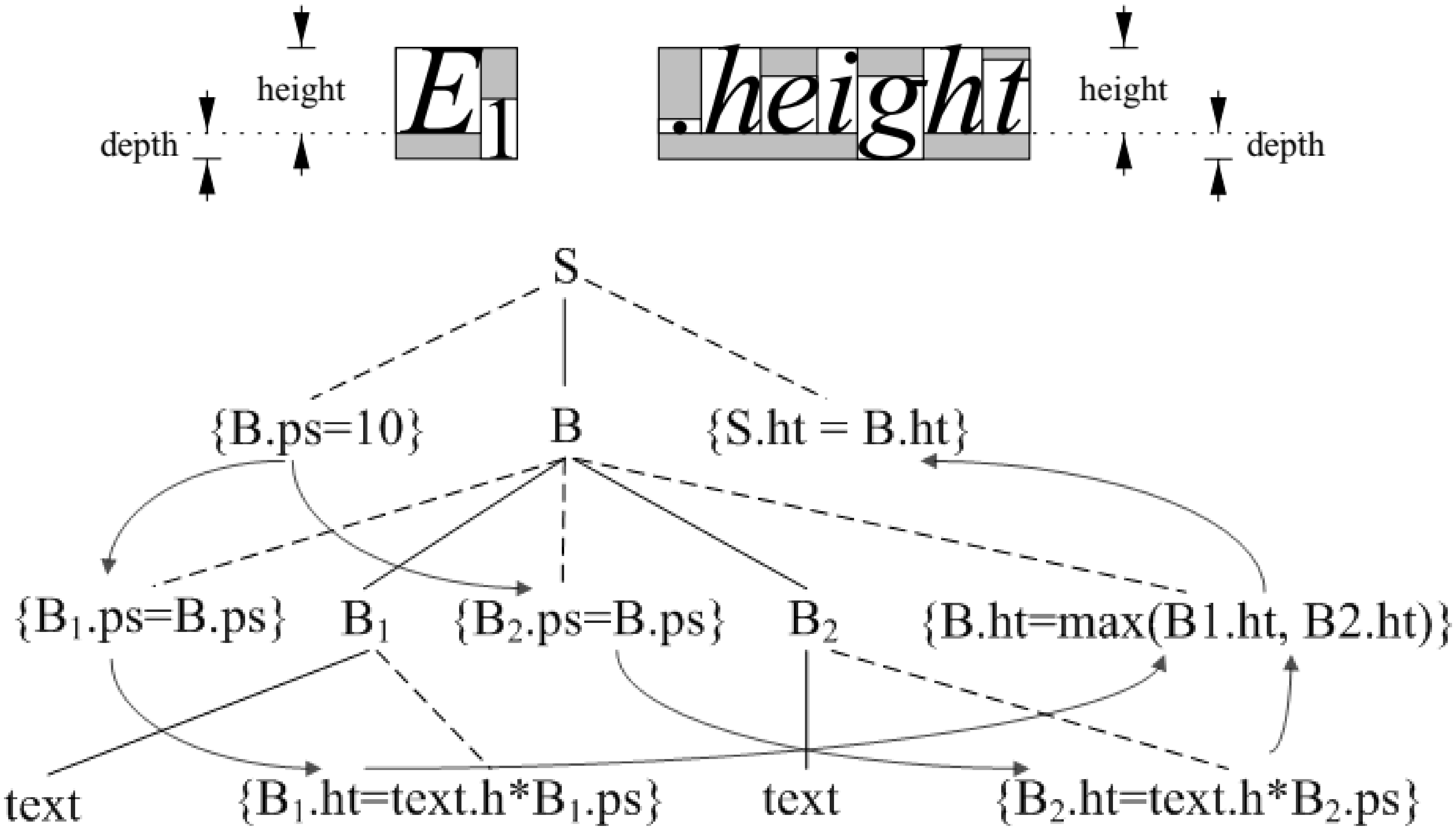
$S \to \{B.ps=10\}B\{S.ht=B.ht\}$

$B \to \{B_1.ps=B.ps\}B_1\{B_2.ps=B.ps\}B_2\{B.ht=max(B_1.ht,B_2.ht)\}$

$B \to text \ \{ B.ht=text.h*B.ps\}$

-Inherited attribute B.ps: the point size of block B
-Synthesized attribute B.ht: the height of box B
-Synthesized attribute B.dp: the depth of box B

A Simplified Version of Example 5.18



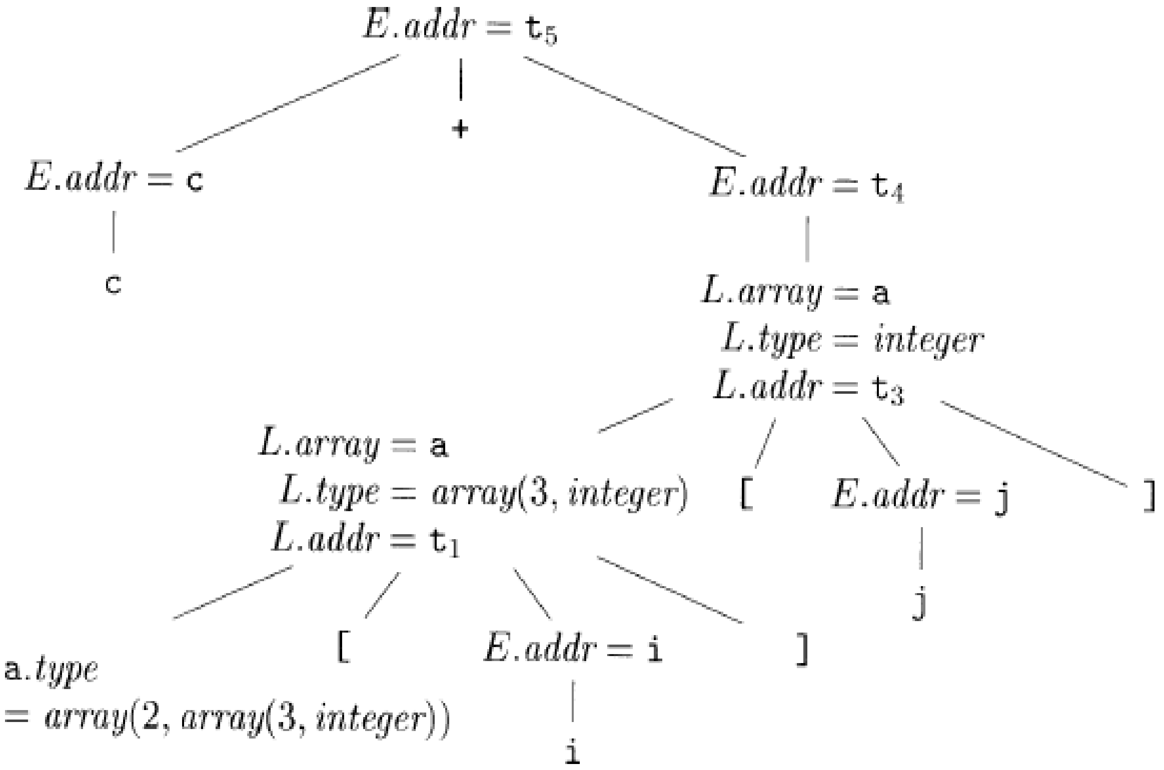
5. 中间代码生成

$$
\begin{array}{lcll}
S & \rightarrow & \textbf{id} = E \; ; & \{ \; gen(\; top.get(\textbf{id}.lexeme) \; '=' \; E.addr); \; \} \\
 & | & L = E \; ; & \{ \; gen(L.array\;.base \; '[' \; L.addr \; ']' \; '=' \; E.addr); \; \} \\
E & \rightarrow & E_1 + E_2 & \{ \; E.addr = \textbf{new} \; Temp\,(); \\
 & & & \quad gen(E.addr \; '=' \; E_1.addr \; '+' \; E_2.addr); \; \} \\
 & | & \textbf{id} & \{ \; E.addr = top.get(\textbf{id}.lexeme); \; \} \\
 & | & L & \{ \; E.addr = \textbf{new} \; Temp\,(); \\
 & & & \quad \underline{gen(E.addr \; '=' \; L.array.base \; '[' \; L.addr \; ']')}; \; \} \\
L & \rightarrow & \textbf{id} \; [\; E \;] & \{ \; L.array = top.get(\textbf{id}.lexeme); \\
 & & & \quad L.type = L.array.type.elem; \\
 & & & \quad L.addr = \textbf{new} \; Temp\,(); \\
 & & & \quad gen(L.addr \; '=' \; E.addr \; '*' \; L.type.width); \; \} \\
 & | & L_1 \; [\; E \;] & \{ \; L.array = L_1.array; \\
 & & & \quad L.type = L_1.type.elem; \\
 & & & \quad t = \textbf{new} \; Temp\,(); \\
 & & & \quad L.addr = \textbf{new} \; Temp\,(); \\
 & & & \quad gen(t \; '=' \; E.addr \; '*' \; L.type.width); \\
 & & & \quad gen(L.addr \; '=' \; L_1.addr \; '+' \; t); \; \}
\end{array}
$$

c+a[i][j]

$$
\begin{array}{lll}
L & \rightarrow \mathbf{id}\ [\ E\ ] & \{\ L.array = top.get(\mathbf{id}.lexeme); \\
 & & \quad L.type = L.array.type.elem; \\
 & & \quad L.addr = \mathbf{new}\ Temp\,(); \\
 & & \quad \underline{gen(L.addr\ '='\ E.addr\ '*'\ L.type.width)};\ \} \\
 & \mid\ L_1\ [\ E\ ] & \{\ L.array = L_1.array; \\
 & & \quad L.type = L_1.type.elem; \\
 & & \quad t = \mathbf{new}\ Temp\,(); \\
 & & \quad L.addr = \mathbf{new}\ Temp\,(); \\
 & & \quad \underline{gen(t\ '='\ E.addr\ '*'\ L.type.width)}; \\
 & & \quad gen(L.addr\ '='\ L_1.addr\ '+'\ t);\ \}
\end{array}
$$

$$
\begin{array}{lll}
E & \rightarrow E_1 + E_2 & \{\ E.addr = \mathbf{new}\ Temp\,(); \\
 & & \quad gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr);\ \} \\
 & \mid\ \mathbf{id} & \{\ E.addr = top.get(\mathbf{id}.lexeme);\ \} \\
 & \mid\ L & \{\ E.addr = \mathbf{new}\ Temp\,(); \\
 & & \quad gen(E.addr\ '='\ L.array.base\ '['\ L.addr\ ']');\ \}
\end{array}
$$




```
t1 = i * 12
t2 = j * 4
t3 = t1 + t2
t4 = a [ t3 ]
t5 = c + t4
```

图 6-24 表达式 c + a[i][j] 的三地址代码

Backpatching for Boolean Expressions

3) $B \rightarrow \;!\; B_1$ $\quad$ $\{\ B.truelist = B_1.falselist;$
$B.falselist = B_1.truelist;\ \}$

4) $B \rightarrow (\ B_1\ )$ $\quad$ $\{\ B.truelist = B_1.truelist;$
$B.falselist \ = \ B_1.falselist;\ \}$

5) $B \rightarrow E_1 \ \mathbf{rel}\ E_2$ $\quad$ $\{\ B.truelist = makelist(nextinstr);$
99 $B.falselist = makelist(nextinstr + 1);$
100: $gen('\texttt{if}'\ E_1.addr\ \mathbf{rel}.op\ E_2.addr\ '\texttt{goto}\ \_')$; 0
101 $gen('\texttt{goto}\ \_')$; 0 $\}$

nextinstr = 102

6) $B \rightarrow \mathbf{true}$ $\quad$ 99 $\{\ B.truelist = makelist(\underline{nextinstr});$ 100
100: $gen('\texttt{goto}\ \_')$; $\cancel{0}$ 100 $\}$ $\quad$ $\boxed{100} \rightarrow$

7) $B \rightarrow \mathbf{false}$ $\quad$ $\{\ B.falselist = makelist(nextinstr);$
$gen('\texttt{goto}\ \_')$; 0 $\}$

8) $M \rightarrow \epsilon$ $\quad$ $\{\ M.instr = nextinstr;\ \}$

1) $B \to B_1 \;||\; M\; B_2$ $\{$ $backpatch(B_1.falselist, M.instr);$
$B.truelist = merge(B_1.truelist, B_2.truelist);$
$B.falselist = B_2.falselist;\ \}$

B1.falselist : 100 goto 0

M.instr : 150

B1.truelist : if B1 goto 0 goto M.instr

2) $B \to B_1 \;\&\&\; M\; B_2$ $\{$ $backpatch(B_1.truelist, M.instr);$
$B.truelist = B_2.truelist;$
$B.falselist = merge(B_1.falselist, B_2.falselist);\ \}$

B2.truelist : if B2 goto 0 goto 0

$x < 100 \;||\; x > 200 \;\&\&\; x \;!= \; y$

```
100:   if x < 100 goto 0
101:   goto 0/102
102:   if x > 200 goto 0/104
103:   goto 0
104:   if x != y goto 0
105:   goto 0/103
```

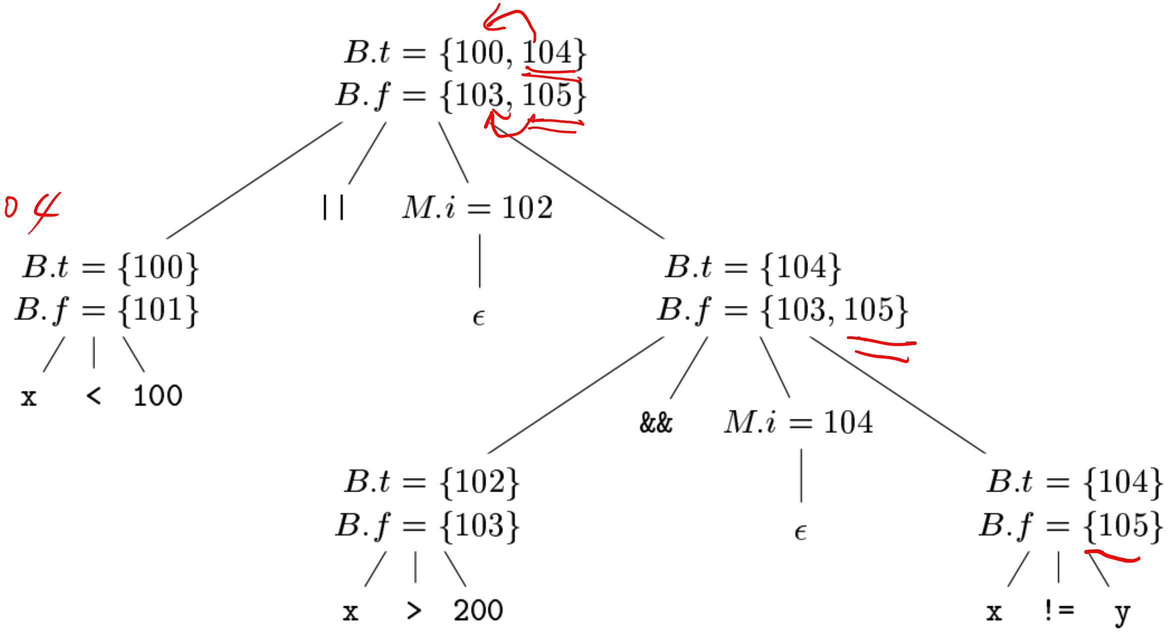


Figure 6.44: Annotated parse tree for $x < 100 \;||\; x > 200 \;\&\&\; x \;!= \; y$

1) $S \rightarrow \textbf{if}\,(\,B\,)\,M\,S_1\ \{\ backpatch(B.truelist,\ M.instr);$
$\quad S.nextlist\ =\ merge(B.falselist,\ S_1.nextlist);\ \}$

2) $S \rightarrow\ \textbf{if}\,(\,B\,)\,M_1\,S_1\,N\ \textbf{else}\ M_2\,S_2$
$\quad \{\ backpatch(B.truelist,\ M_1.instr);$
$\quad backpatch(B.falselist,\ M_2.instr);$
$\quad temp\ =\ merge(S_1.nextlist,\ N.nextlist);$
$\quad S.nextlist\ =\ merge(temp,\ S_2.nextlist);\ \}$

3) $S \rightarrow\ \textbf{while}\ M_1\,(\,B\,)\,M_2\,S_1$
$\quad \{\ backpatch(S_1.nextlist,\ M_1.instr);$
$\quad backpatch(B.truelist,\ M_2.instr);$
$\quad S.nextlist\ =\ B.falselist;$
$\quad emit(\texttt{'goto'}\ M_1.instr);\ \}$

4) $S \rightarrow \{\ L\ \}$ $\qquad \{\ S.nextlist\ =\ L.nextlist;\ \}$

5) $S \rightarrow A\ ;$ $\qquad \{\ S.nextlist\ =\ \textbf{null};\ \}$

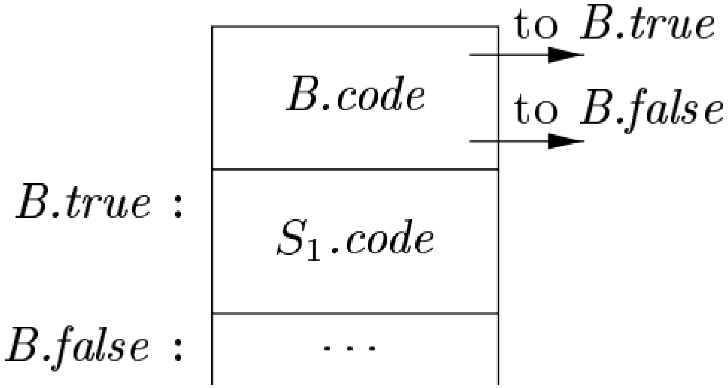
6) $M \rightarrow \epsilon$ $\qquad \{\ M.instr\ =\ nextinstr;\ \}$

7) $N \rightarrow \epsilon$ $\qquad \{\ N.nextlist\ =\ makelist(nextinstr);$
$\quad emit(\texttt{'goto \_'});\ \}$

8) $L \rightarrow L_1\ M\ S$ $\qquad \{\ backpatch(L_1.nextlist,\ M.instr);$
$\quad L.nextlist\ =\ S.nextlist;\ \}$

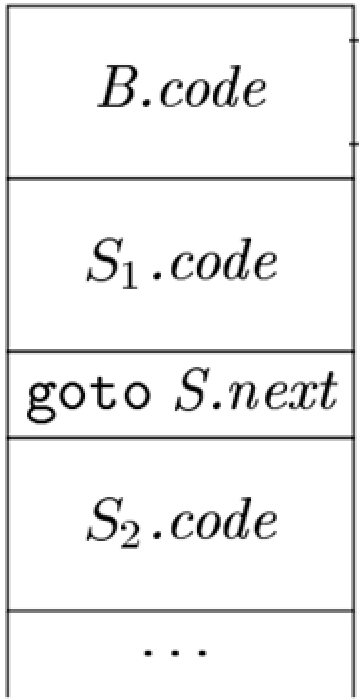
9) $L \rightarrow S$ $\qquad \{\ L.nextlist\ =\ S.nextlist;\ \}$

$$
\begin{aligned}
S \rightarrow \textbf{if}\,(\,B\,)\;M\;S_1\;\{\;& backpatch(B.truelist,\;M.instr);\\
& S.nextlist\;=\;merge(B.falselist,\;S_1.nextlist);\;\}
\end{aligned}
$$



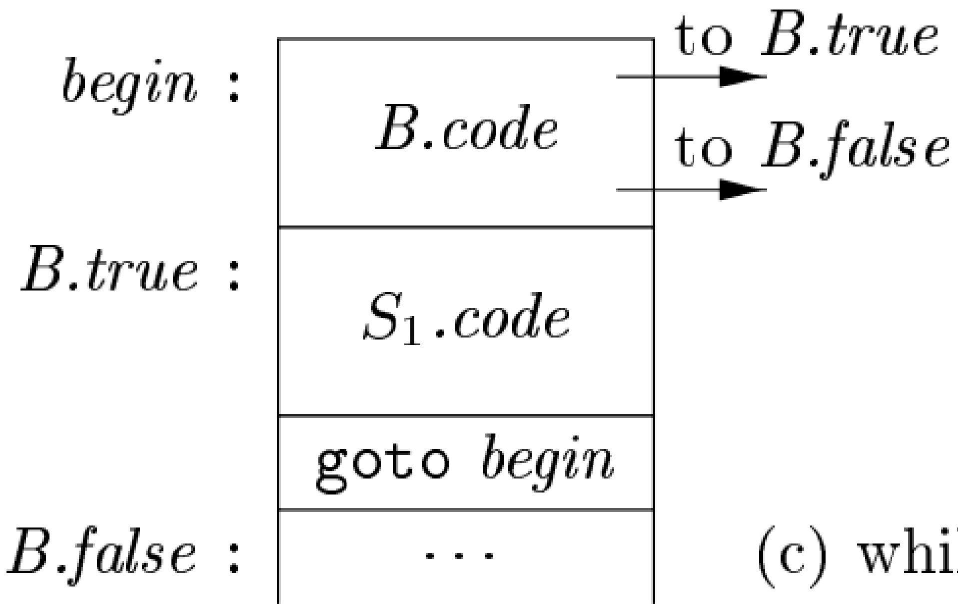
(a) if

$$
\begin{array}{ll}
S \rightarrow & \textbf{if}\,(\,B\,)\; M_1\; S_1\; N\; \textbf{else}\; M_2\; S_2 \\
& \qquad\qquad \{\; backpatch(B.truelist,\;\; M_1.instr); \\
& \qquad\qquad\;\;\, backpatch(B.falselist,\;\; M_2.instr); \\
& \qquad\qquad\;\;\, temp \;=\; merge(S_1.nextlist,\;\; N.nextlist); \\
& \qquad\qquad\;\;\, S.nextlist \;=\; merge(temp,\;\; S_2.nextlist);\; \}
\end{array}
$$



(b) if-else

$$
\begin{aligned}
S \rightarrow \quad & \textbf{while } M_1 \,(\, B \,)\, M_2 \, S_1 \\
& \qquad\qquad \{\ backpatch(S_1.nextlist,\ M_1.instr); \\
& \qquad\qquad \ \ \ backpatch(B.truelist,\ M_2.instr); \\
& \qquad\qquad \ \ \ S.nextlist \ = \ B.falselist; \\
& \qquad\qquad \ \ \ gen('\texttt{goto}'\ \underline{M_1.instr);}\ \}
\end{aligned}
$$



(c) while

$$
\begin{array}{l|l}
S \rightarrow \textbf{if} \ (\ B \ ) \ S_1 & B.true = newlabel() \\
& B.false = S_1.next = S.next \\
& S.code = B.code \ || \ label(B.true) \ || \ S_1.code \\
& \\
S \rightarrow \textbf{if} \ (\ B \ ) \ S_1 \ \textbf{else} \ S_2 & B.true = newlabel() \\
& B.false = newlabel() \\
& S_1.next = S_2.next = S.next \\
& S.code = B.code \\
& \qquad\qquad || \ label(B.true) \ || \ S_1.code \\
& \qquad\qquad || \ gen(\texttt{'goto'} \ S.next) \\
& \qquad\qquad || \ label(B.false) \ || \ S_2.code
\end{array}
$$

$$
\begin{array}{ll}
S \rightarrow \textbf{if} \ (\ B \ ) \ M \ S_1 & \{ \ backpatch(B.truelist, \ M.instr); \\
& \quad S.nextlist = merge(B.falselist, \ S_1.nextlist); \ \}
\end{array}
$$

$$
\begin{array}{l}
S \rightarrow \ \textbf{if} \ (\ B \ ) \ M_1 \ S_1 \ N \ \textbf{else} \ M_2 \ S_2 \\
\qquad\qquad \{ \ backpatch(B.truelist, \ M_1.instr); \\
\qquad\qquad \quad backpatch(B.falselist, \ M_2.instr); \\
\qquad\qquad \quad temp = merge(S_1.nextlist, \ N.nextlist); \\
\qquad\qquad \quad S.nextlist = merge(temp, \ S_2.nextlist); \ \}
\end{array}
$$

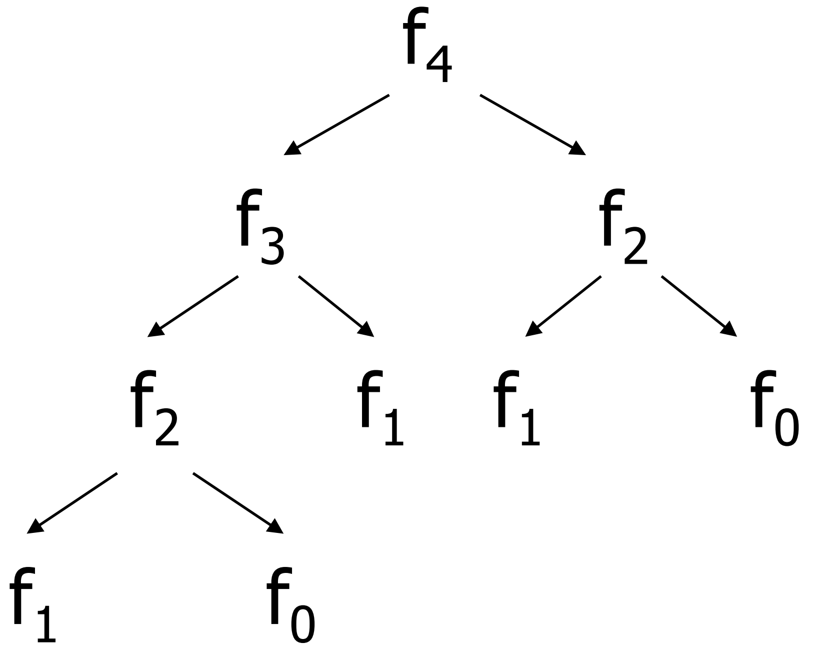
Translate the following program fragment into three-address-code using short circuit code and back-patching techniques.

```
while (a==r || a>b) {
    if (r==s)
        y=y+c;
    else
        y=y-d;
}
```

6.运行时环境

Activation Trees

A tree to depict the control flow enters and leaves activation of a procedure



$f_4()\{$
$\quad f_3;$
$\quad f_2;\}$

$f_3()\{$
$\quad f_2;$
$\quad f_1\}$

$f_2()\{$
$\quad f_1();$
$\quad f_0();\}$

An Example

```
#include <stdio.h>
  int i,j;
  int main()
   {
      int k,l;
      cin>>l; //assume l=3
      k=p(l);
      cout<<k;
   }
```

```
int p(int m)
    {
        int t;
        if (m==0 || m==1)
          return(1);
        else {
         t=m*p(m-1);
         return(t); } }
```

+19	…	
+18	K+12(SPp(2))	
+17	Returned value from P(2)	
+16	t	
+15	Formal Parameter m	2
+14	Number of Parameter	1
+13	Returned address	
+12	K+6(SPp(3))	
+11	Returned value from P(3)	
+10	t	

+9	Formal Parameter m	3
+8	Number of Parameter	1
+7	Returned address	
+6	K+2(SPmain)	
+5	Returned value from main	
+4	l	3
+3	k	
+2	K(SP0)	
+1	j	
K	i	

7.代码优化

Optimization of Basic Blocks

- Constant folding
  - Evaluate constant expressions at compile time and replace the constant expressions by their values
- Common subexpression elimination
- Copy propagation
- Dead code elimination

An Example

A source code

Pi:=3.14

A: = 2*Pi*(R+r);

B:=A;

B:=2*Pi*(R+r)*(R-r)

(1)Pi:=3.14

(2)T1:=2*Pi

(3)T2:=R+r

(4)A:=T1*T2

(5)B:=A

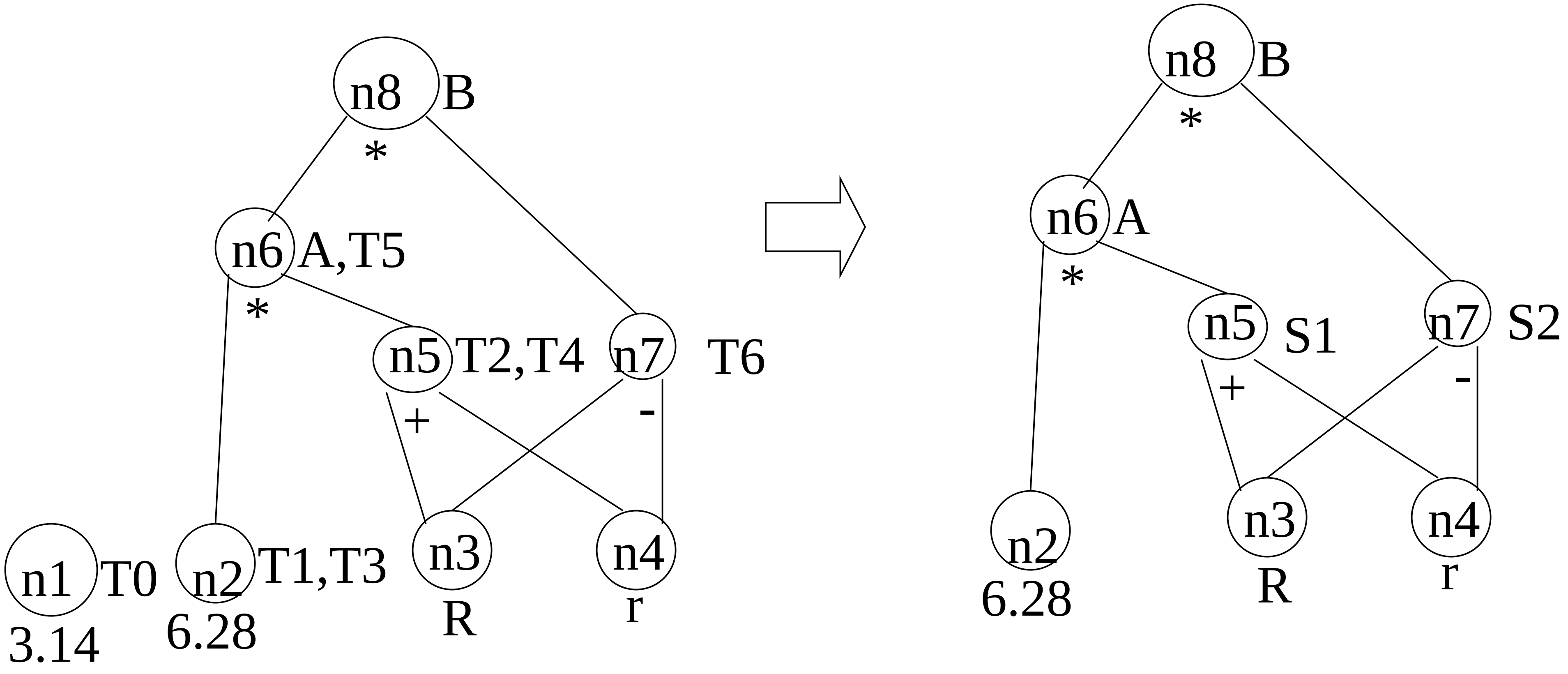
(6)T3:=2*Pi

(7)T4:=R+r

(8)T5:=T3*T4

(9)T6:=R-r

(10)B:=T5*T6



(1) S1=R+r
(2) S2=R-r
(3) A=6.28*S1
(4) B=6.28*A