Języki formalne i kompilatory

Gramatyki atrybutowane

Przykład translacji

Przykład 1 – tłumaczenie $\{a^n\}$ na $\{a^{n^2}\}$

Wejście: słowo należące do $\{a^n \mid n > 0\}$

Wyjście: odpowiadające mu słowo z języka $\{a^{n^2} \mid n > 0\}$

Przykład:

słowo wejściowe: $aaa = a^3$

słowo wyjściowe: $aaaaaaaaa = a^9$

Przykład 1 – tłumaczenie $\{a^n\}$ na $\{a^{n^2}\}$

Wejście: słowo należące do $\{a^n \mid n > 0\}$

Wyjście: odpowiadające mu słowo z języka $\{a^{n^2} \mid n > 0\}$

Przykład:

słowo wejściowe: *aaa* = a^3

słowo wyjściowe: *aaaaaaaaa* = a^9

Założenie: parsing odbywa się metodą *bottom-up*

Należy wykorzystać mechanizm gramatyk S-atrybutowanych

Przykład 1 – tłumaczenie $\{a^n\}$ na $\{a^{n^2}\}$

Wejście: słowo należące do $\{a^n \mid n > 0\}$

Wyjście: odpowiadające mu słowo z języka $\{a^{n^2} \mid n > 0\}$

Przykład:

słowo wejściowe: *aaa* = a^3

słowo wyjściowe: *aaaaaaaaa* = a^9

Założenie: parsing odbywa się metodą *bottom-up*

Należy wykorzystać mechanizm gramatyk S-atrybutowanych

Ile atrybutów należy przewidzieć?

Przykład 1 – tłumaczenie $\{a^n\}$ na $\{a^{n^2}\}$

Wejście: słowo należące do $\{a^n \mid n > 0\}$

Wyjście: odpowiadające mu słowo z języka $\{a^{n^2} \mid n > 0\}$

Przykład:

słowo wejściowe: *aaa* = a^3

słowo wyjściowe: *aaaaaaaaa* = a^9

Założenie: parsing odbywa się metodą *bottom-up*

Należy wykorzystać mechanizm gramatyk S-atrybutowanych

Niezbędna wiedza: $\boldsymbol{(k+1)^2 = k^2 + 2k + 1}$

Przykład 1 – tłumaczenie $\{a^n\}$ na $\{a^{n^2}\}$

Wejście: słowo należące do $\{a^n \mid n > 0\}$

Wyjście: odpowiadające mu słowo z języka $\{a^{n^2} \mid n > 0\}$

Założenie: parsing odbywa się metodą *bottom-up*

Należy wykorzystać mechanizm gramatyk S-atrybutowanych

Niezbędna wiedza: $(k+1)^2 = k^2 + 2k + 1$

Ilu atrybutów potrzebujemy?

Przykład 1 – tłumaczenie $\{a^n\}$ na $\{a^{n^2}\}$

Wejście: słowo należące do $\{a^n \mid n > 0\}$

Wyjście: odpowiadające mu słowo z języka $\{a^{n^2} \mid n > 0\}$

Założenie: parsing odbywa się metodą *bottom-up*

Należy wykorzystać mechanizm gramatyk S-atrybutowanych

Niezbędna wiedza: $(\boldsymbol{k+1})^{\boldsymbol{2}} = \boldsymbol{k^2 + 2k + 1}$

Ilu atrybutów potrzebujemy?
Dwóch. Jeden będzie faktycznie przeczytanym do danego momentu łańcuchem liter *a* (odpowiednik ***k***), drugi będzie łańcuchem liter *a* o długości równej kwadratowi liczby dotychczas przeczytanych liter *a* (odpowiednik $\boldsymbol{k^2}$).

Przykład 1 – tłumaczenie $\{a^n\}$ na $\{a^{n^2}\}$

Wejście: słowo należące do $\{a^n \mid n > 0\}$

Wyjście: odpowiadające mu słowo z języka $\{a^{n^2} \mid n > 0\}$

Założenie: parsing odbywa się metodą *bottom-up*

Należy wykorzystać mechanizm gramatyk S-atrybutowanych

Jaką gramatykę zaproponować, aby najłatwiej wykorzystać mechanizm gramatyk S-atrybutowanych przy parsingu bottom-up? (Aby móc czytając słowo litera po literze od razu „liczyć” i wyznaczać wartości atrybutów)

Przykład 1 – tłumaczenie $\{a^n\}$ na $\{a^{n^2}\}$

Wejście: słowo należące do $\{a^n \mid n > 0\}$

Wyjście: odpowiadające mu słowo z języka $\{a^{n^2} \mid n > 0\}$

Jaką gramatykę zaproponować, aby najłatwiej wykorzystać mechanizm gramatyk S-atrybutowanych przy parsingu bottom-up? (Aby móc czytając słowo litera po literze od razu „liczyć” i wyznaczać wartości atrybutów)

$S \rightarrow L$

$L \rightarrow L\ a$

$L \rightarrow a$

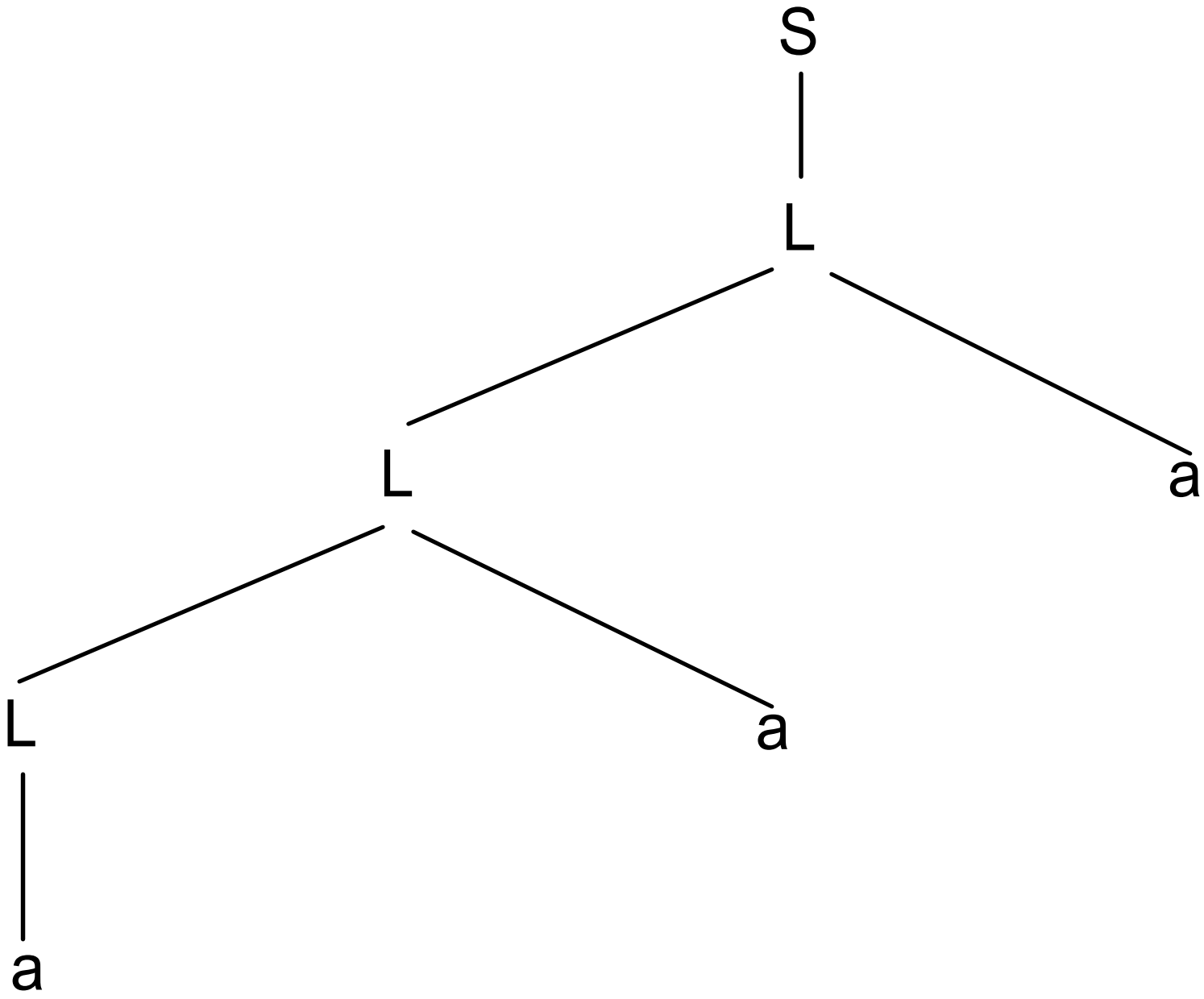
Przykład 1 – tłumaczenie $\{a^n\}$ na $\{a^{n^2}\}$

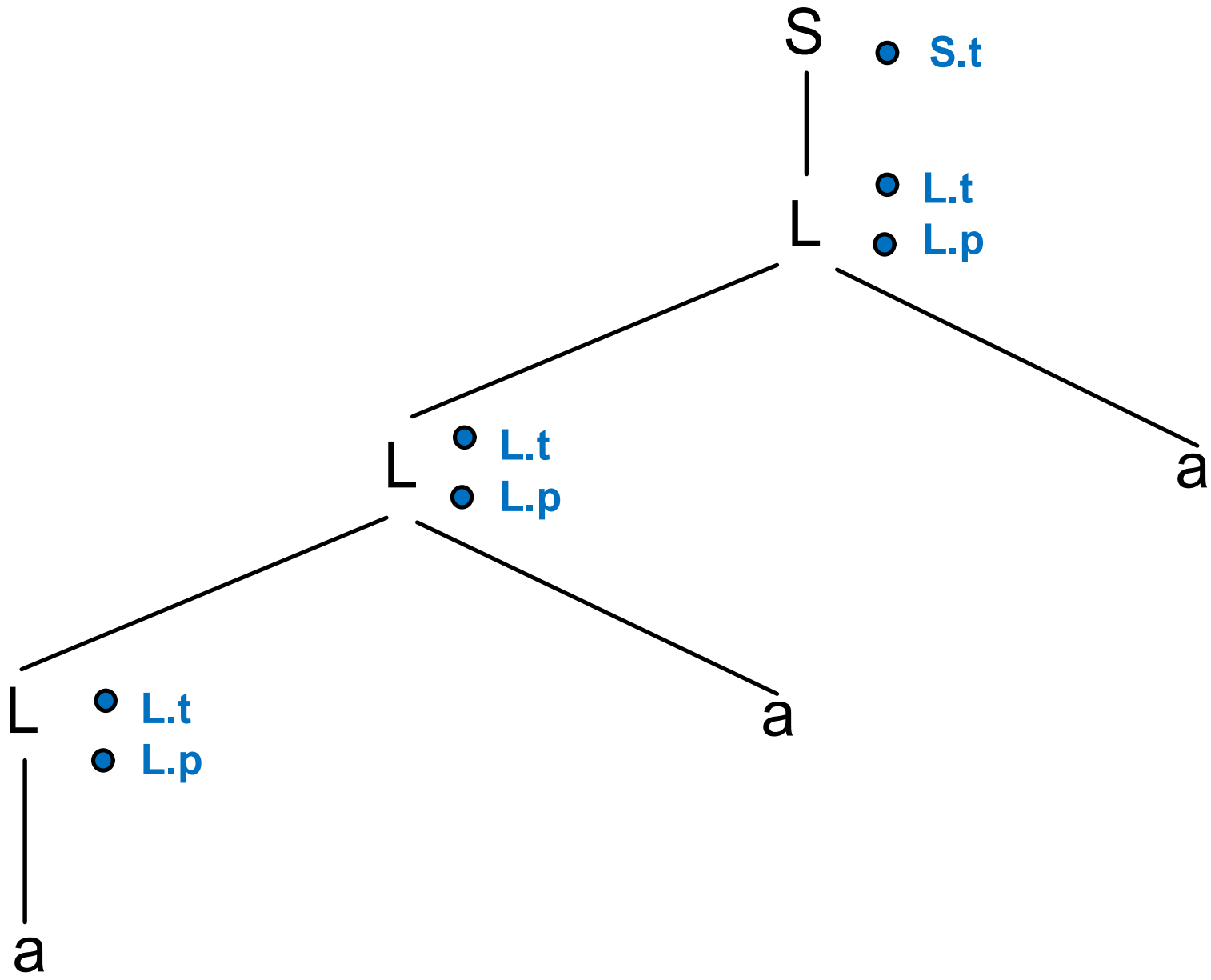
Wejście: słowo należące do $\{a^n \mid n > 0\}$
Wyjście: odpowiadające mu słowo z języka $\{a^{n^2} \mid n > 0\}$

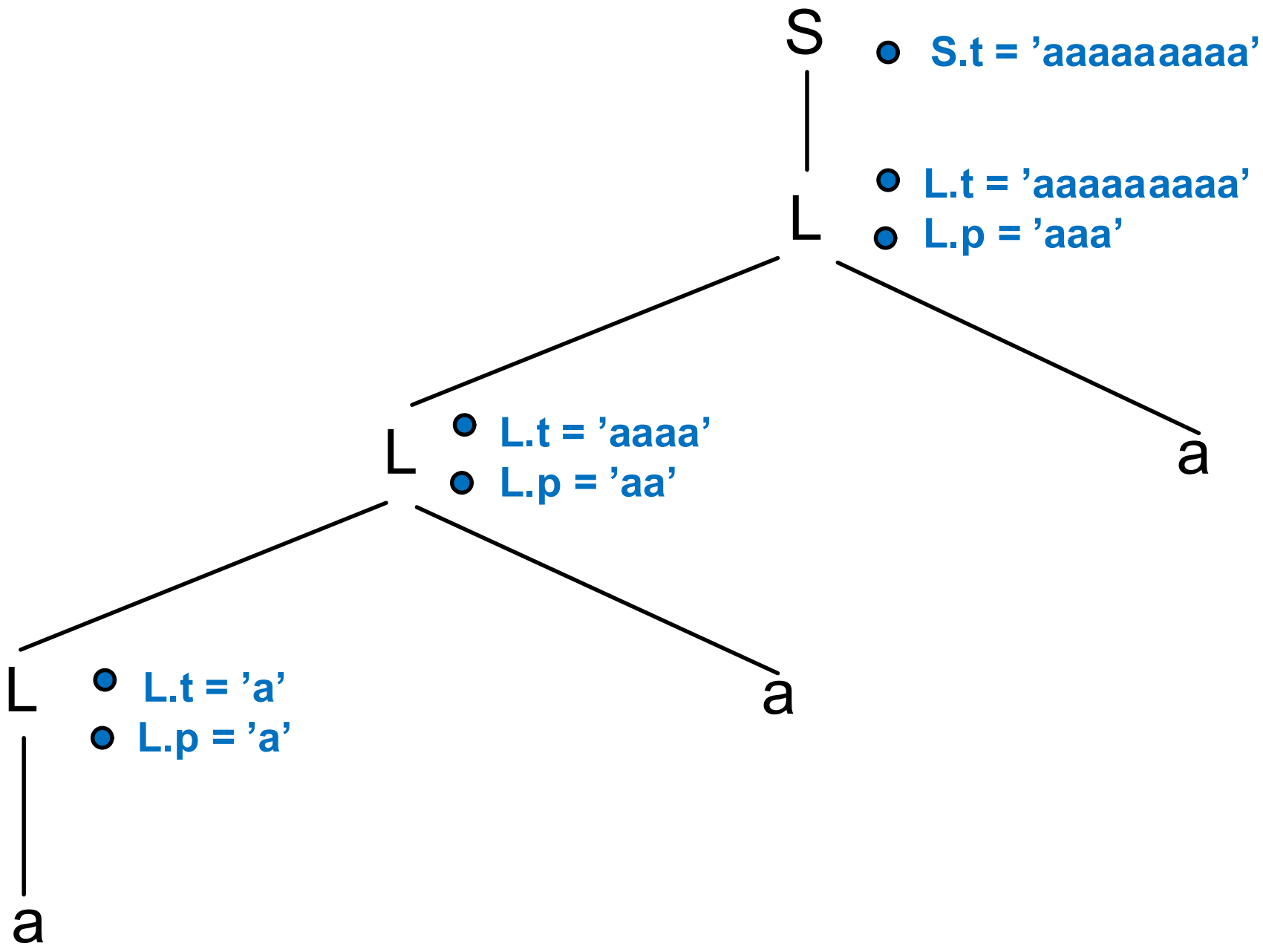
Jaką gramatykę zaproponować, aby najłatwiej wykorzystać mechanizm gramatyk S-atrybutowanych przy parsingu bottom-up? (Aby móc czytając słowo litera po literze od razu „liczyć” i wyznaczać wartości atrybutów)

$S \rightarrow L$
$L \rightarrow L_1\ a$
$L \rightarrow a$

Trzeba odróżnić od siebie wystąpienia poszczególnych symboli nieterminalnych w poszczególnych produkcjach


S
L
L
a
L
a
a

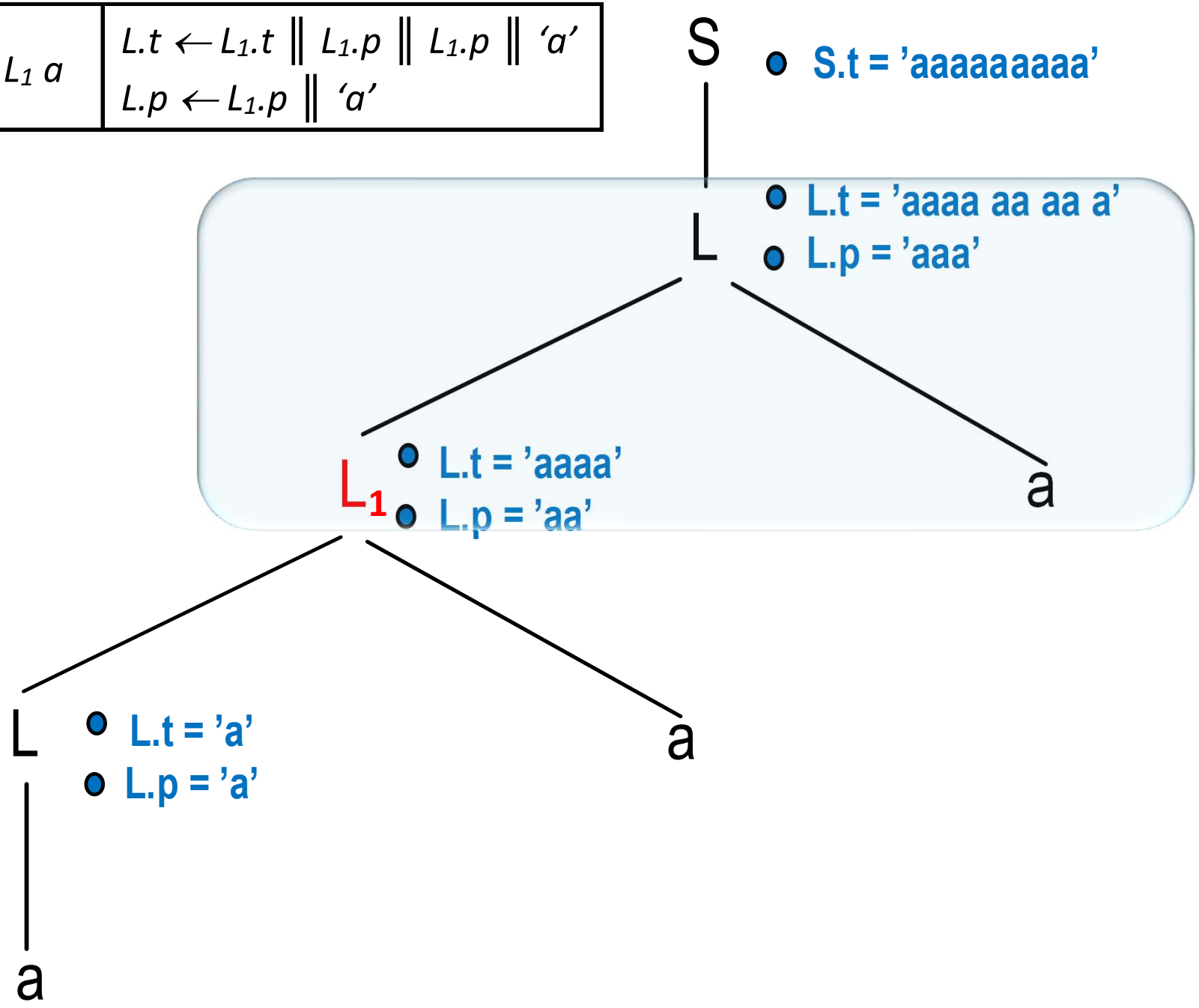

S
S.t
L
L.t
L.p
L
L.t
L.p
a
L
L.t
L.p
a
a


S
S.t = ’aaaaaaaaa’
L
L.t = ’aaaaaaaaa’
L.p = ’aaa’
L
L.t = ’aaaa’
L.p = ’aa’
a
L
L.t = ’a’
L.p = ’a’
a
a

$S \rightarrow L$	$S.t \leftarrow L.t$
$L \rightarrow L_1\ a$	$L.t \leftarrow L_1.t \parallel L_1.p \parallel L_1.p \parallel$ ‘a’ $L.p \leftarrow L_1.p \parallel$ ‘a’
$L \rightarrow a$	$L.t \leftarrow$ ‘a’ $L.p \leftarrow$ ‘a’

gdzie: $\parallel$ - operator konkatenacji tekstów

$L \rightarrow L_1\ a$	$L.t \leftarrow L_1.t \parallel L_1.p \parallel L_1.p \parallel$ 'a' $L.p \leftarrow L_1.p \parallel$ 'a'



Dana jest gramatyka opisująca język wyrażeń (wykorzystujących operatory: mnożenia (*) i dodawania (+) oraz operandy *a, b* i *c*) zapisanych w odwrotnej notacji polskiej:

$$S \rightarrow SS+ \mid SS* \mid a \mid b \mid c$$

Zbudować parser LL(1) dla języka generowanego przez powyższą gramatykę. Gramatykę trzeba będzie odpowiednio przekształcić przed rozpoczęciem budowy parsera. Przeprowadzić symulację działania parsera dla słowa:

$$abc+a*+$$

oraz narysować drzewo wyprowadzenia.

Musimy najpierw wykonać usuwanie lewostronnej rekurencji, a potem dokonać lewostronnej faktoryzacji

$$S \rightarrow SS+ \mid SS* \mid a \mid b \mid c$$

Po usunięciu lewostronnej rekurencji:

$$S \rightarrow aR \mid bR \mid cR$$
$$R \rightarrow S+R \mid S*R \mid \varepsilon$$

Po lewostronnej faktoryzacji:

$$S \rightarrow aR \mid bR \mid cR$$
$$R \rightarrow SV \mid \varepsilon$$
$$V \rightarrow +R \mid *R$$

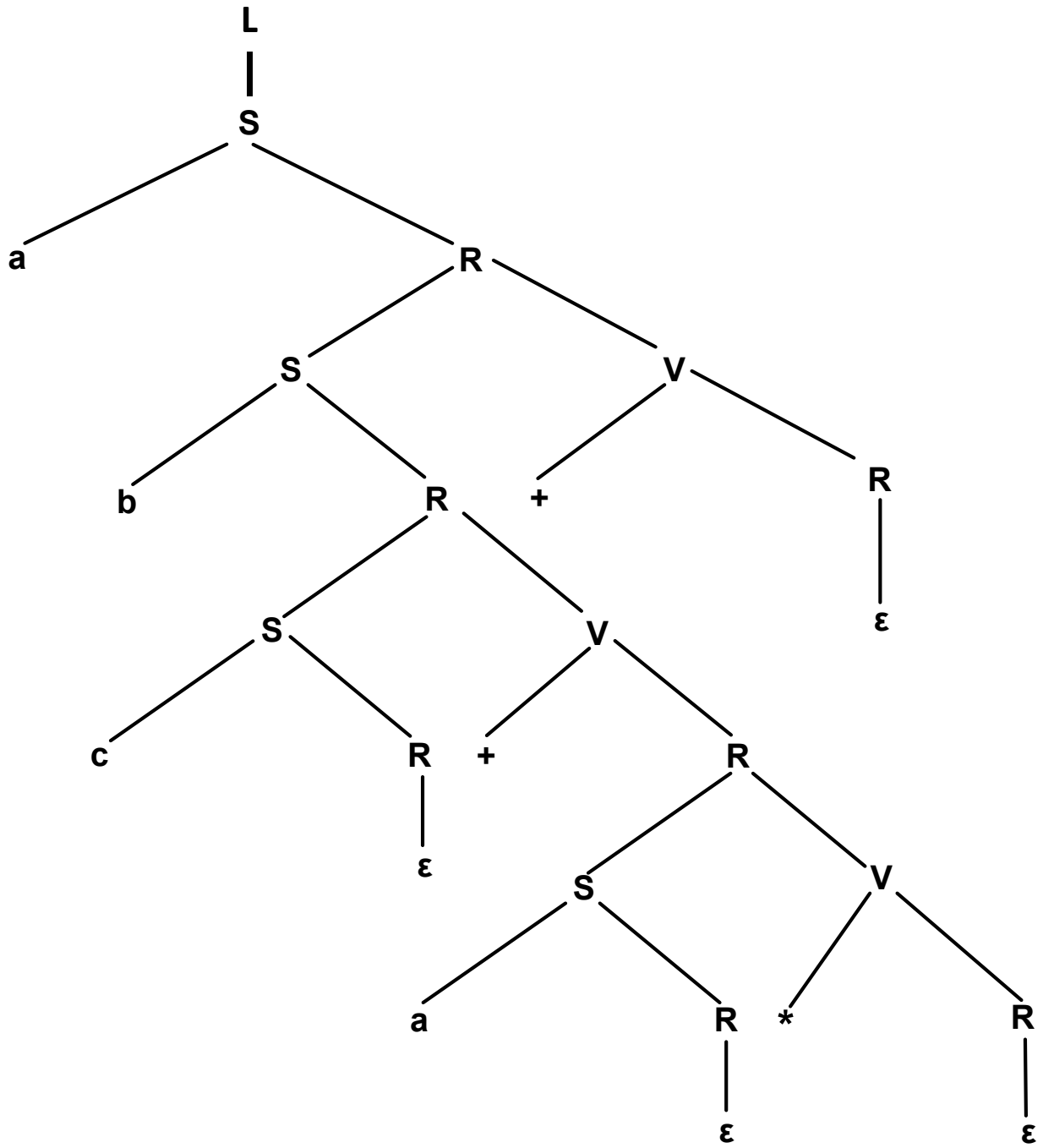
	First	*Follow*	+	*	*a*	*b*	*c*	$
S	*a, b, c*	$,+,*			*aR*	*bR*	*cR*	
R	*a, b, c, ε*	$,+,*	*ε*	*ε*	*SV*	*SV*	*SV*	*ε*
V	+,*	$,+,*	*+R*	**R*				

Dodajmy jeszcze jedną produkcję, bo się później przyda

$$
\begin{aligned}
L &\rightarrow S \\
S &\rightarrow aR \mid bR \mid cR \\
R &\rightarrow SV \mid \varepsilon \\
V &\rightarrow +R \mid *R
\end{aligned}
$$

Rozważamy wejście:

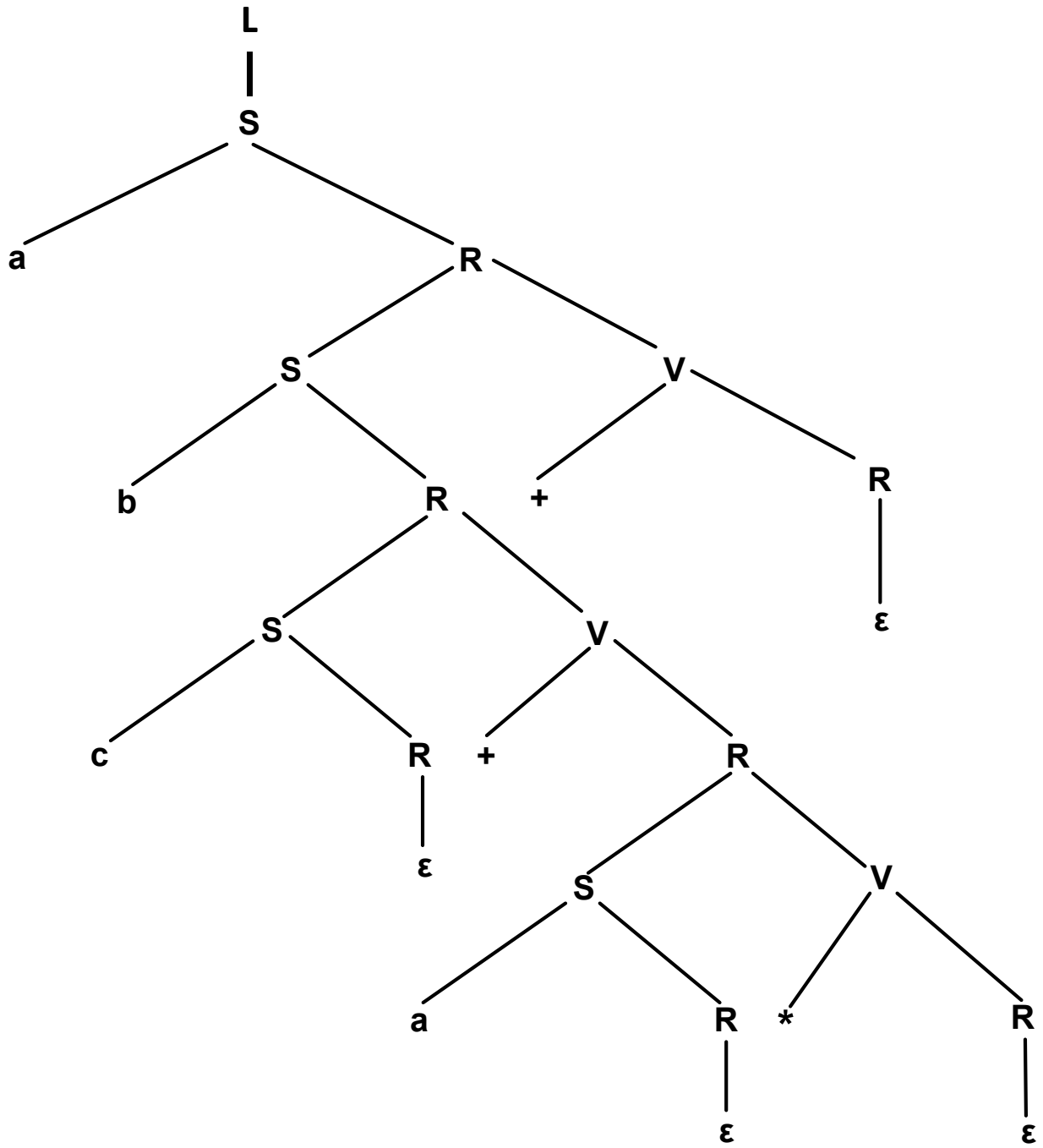
$abc+a*+$


L
S
a
R
S
V
b
R
+
R
ε
S
V
c
R
+
R
ε
S
V
a
R
*
R
ε
ε

Teraz będziemy chcieli przetłumaczyć nasze wejście do kodu trójadresowego.

Czyli naszym zadaniem jest stworzenie gramatyki L-atrybutowanej, umożliwiającej tłumaczenie testu w odwrotnej notacji polskiej do kodu trójadresowego, równocześnie z parsingiem, przy wykorzystaniu technologii top-down oraz zapisanie gramatyki atrybutowanej w postaci schematu tłumaczenia.

W których miejscach drzewa trzeba będzie drukować tłumaczenie?


L
S
a
R
S
V
b
R
+
R
ε
S
V
c
R
+
R
ε
S
V
a
R
*
R
ε
ε

Teraz będziemy chcieli przetłumaczyć nasze wejście do kodu trójadresowego.

W których miejscach drzewa trzeba będzie drukować tłumaczenie?

Wejście: ***abc+a*+***

…czyli w notacji infiksowej: ***a+(b+c)*a***

Tłumaczenie powinno wyglądać tak:

Teraz będziemy chcieli przetłumaczyć nasze wejście do kodu trójadresowego.

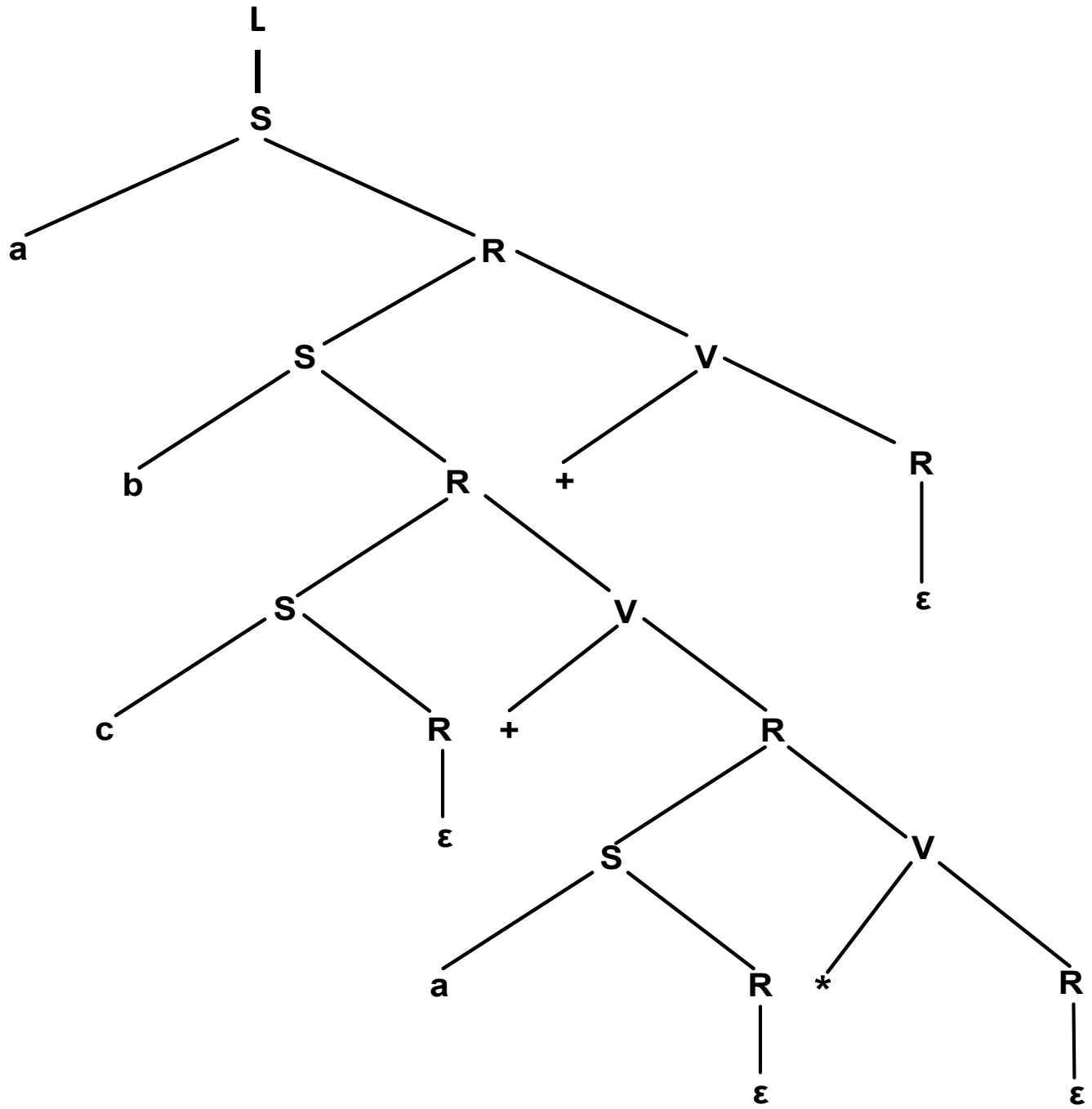
W których miejscach drzewa trzeba będzie drukować tłumaczenie?

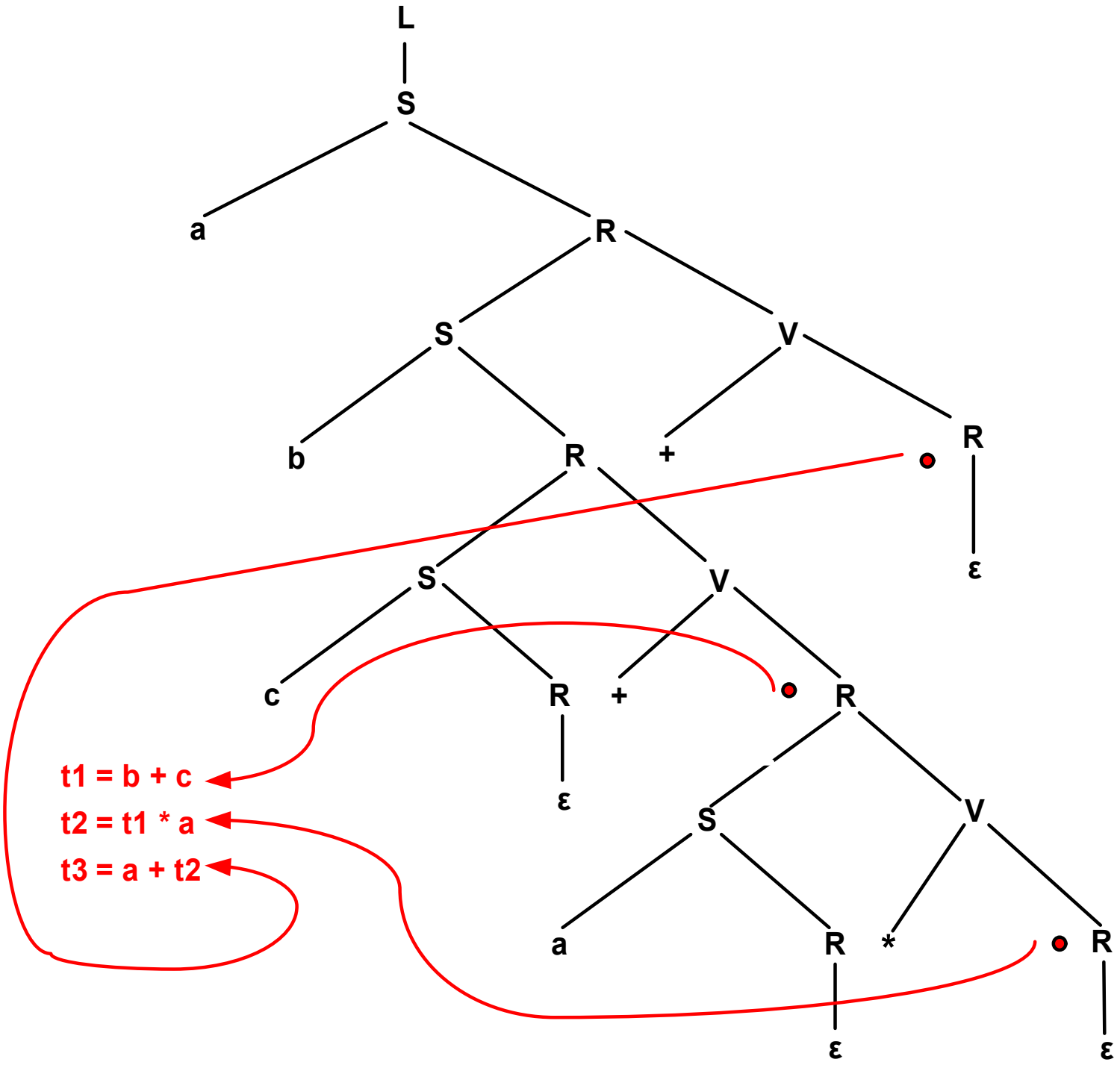
Wejście: ***abc+a*+***

…czyli w notacji infiksowej: ***a+(b+c)*a***

Tłumaczenie powinno wyglądać tak:

t1 = b+c
t2 = t1*a
t3 = a+t2


L
S
a
R
S
V
b
R
+
R
ε
S
V
c
R
+
R
ε
S
V
a
R
*
R
ε
ε


L
S
a
R
S
V
b
R
+
R
ε
S
V
c
R
+
R
ε
S
V
a
R
*
R
ε
ε
t1 = b + c
t2 = t1 * a
t3 = a + t2

Wejście: ***abc+a*+***

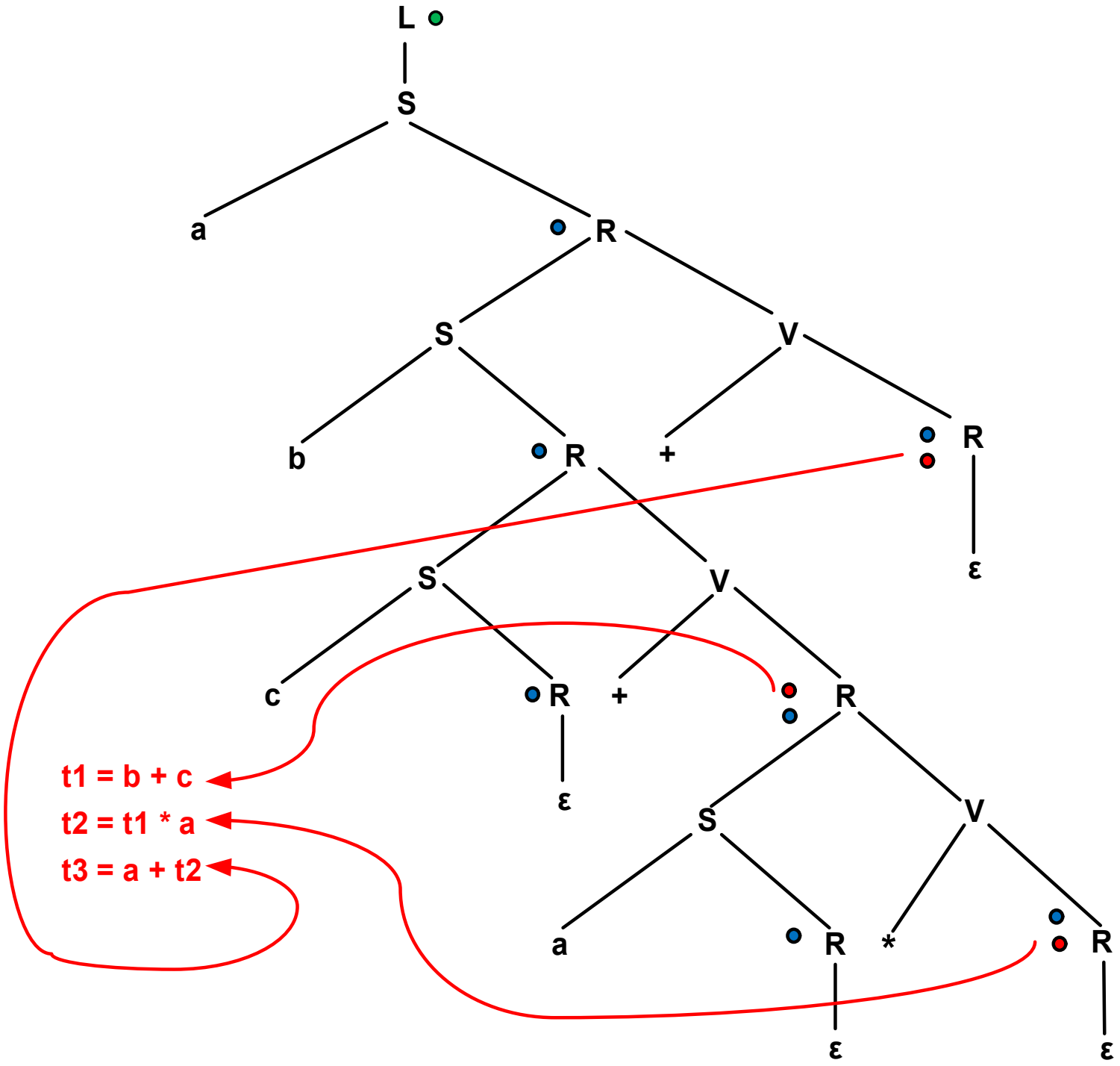
…czyli w notacji infiksowej: ***a+(b+c)*a***

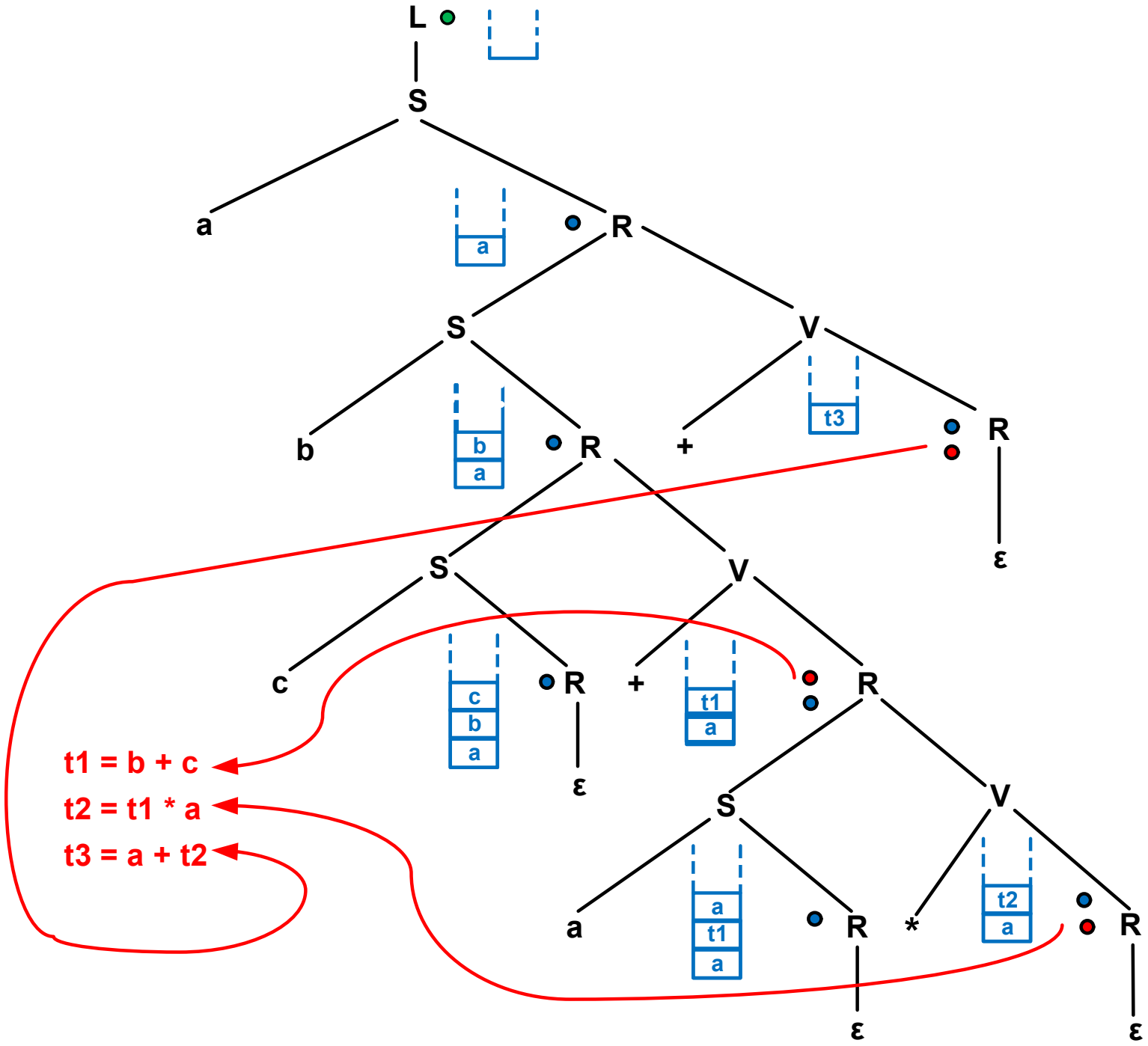
Tłumaczenie powinno wyglądać tak:

t1 = b+c
t2 = t1*a
t3 = a+t2

W jaki sposób zapewnić posiadanie odpowiedniej informacji w odpowiednim momencie przetwarzania i tłumaczenia w technologii top-down?

W których miejscach drzewa wykonywać czynności związane z przetwarzaniem informacji?


L
S
a
R
S
V
b
R
+
R
ε
S
V
c
R
+
R
ε
S
V
a
R
*
R
ε
ε
t1 = b + c
t2 = t1 * a
t3 = a + t2


L
S
a
a
R
S
V
b
b
a
R
+
t3
R
ε
S
V
c
c
b
a
R
ε
+
t1
a
R
t1 = b + c
t2 = t1 * a
t3 = a + t2
S
V
a
a
t1
a
R
ε
*
t2
a
R
ε

$$L \rightarrow S$$
$$S \rightarrow aR$$
$$S \rightarrow bR$$
$$S \rightarrow cR$$
$$R \rightarrow SV$$
$$R \rightarrow \varepsilon$$
$$V \rightarrow +R$$
$$V \rightarrow *R$$

Operowanie danymi

$L \rightarrow S$ *{L.f←[result=pop();]}*

$S \rightarrow a$ *{R.d←[push(,a');]}* R

$S \rightarrow b$ *{R.d←[push(,b');]}* R

$S \rightarrow c$ *{R.d←[push(,c');]}* R

$R \rightarrow SV$

$R \rightarrow \varepsilon$

$V \rightarrow +$ *{R.d←[arg2=pop(); arg1=pop();*
w=new_temp(); push(w);]} R

$V \rightarrow *$ *{R.d←[arg2=pop(); arg1=pop();*
w=new_temp(); push(w);]} R

Tworzenie tłumaczenia

$L \rightarrow S$ *{L.f←[result=pop();]}*

$S \rightarrow a$ *{R.d←[push(,a');]}* *{R.t←[nothing]}* R

$S \rightarrow b$ *{R.d←[push(,b');]}* *{R.t←[nothing]}* R

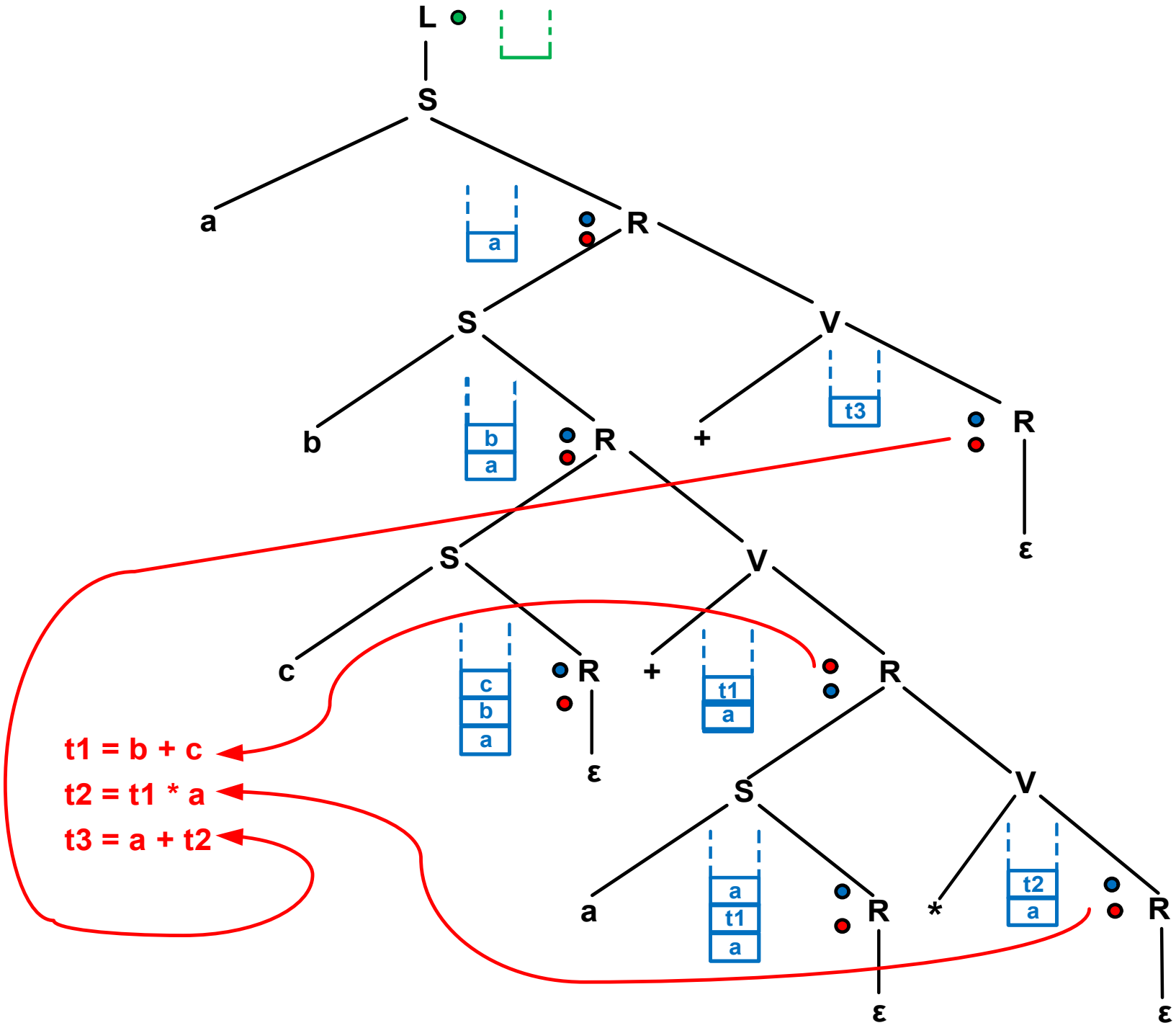
$S \rightarrow c$ *{R.d←[push(,c');]}* *{R.t←[nothing]}* R

$R \rightarrow SV$

$R \rightarrow \varepsilon$

$V \rightarrow +$ *{R.d←[arg2=pop(); arg1=pop();*
w=new_temp(); push(w);]}
{R.t←[print(w ‖ '=' ‖ arg1 ‖ '+' ‖ arg2);]} R

$V \rightarrow *$ *{R.d←[arg2=pop(); arg1=pop();*
w=new_temp(); push(w);]}
{R.t←[print(w ‖ '=' ‖ arg1 ‖ '' ‖ arg2);]}* R


L
S
a
a
R
S
V
b
b
a
R
+
t3
R
ε
S
V
c
c
b
a
R
ε
+
t1
a
R
t1 = b + c
t2 = t1 * a
t3 = a + t2
S
V
a
a
t1
a
R
ε
*
t2
a
R
ε