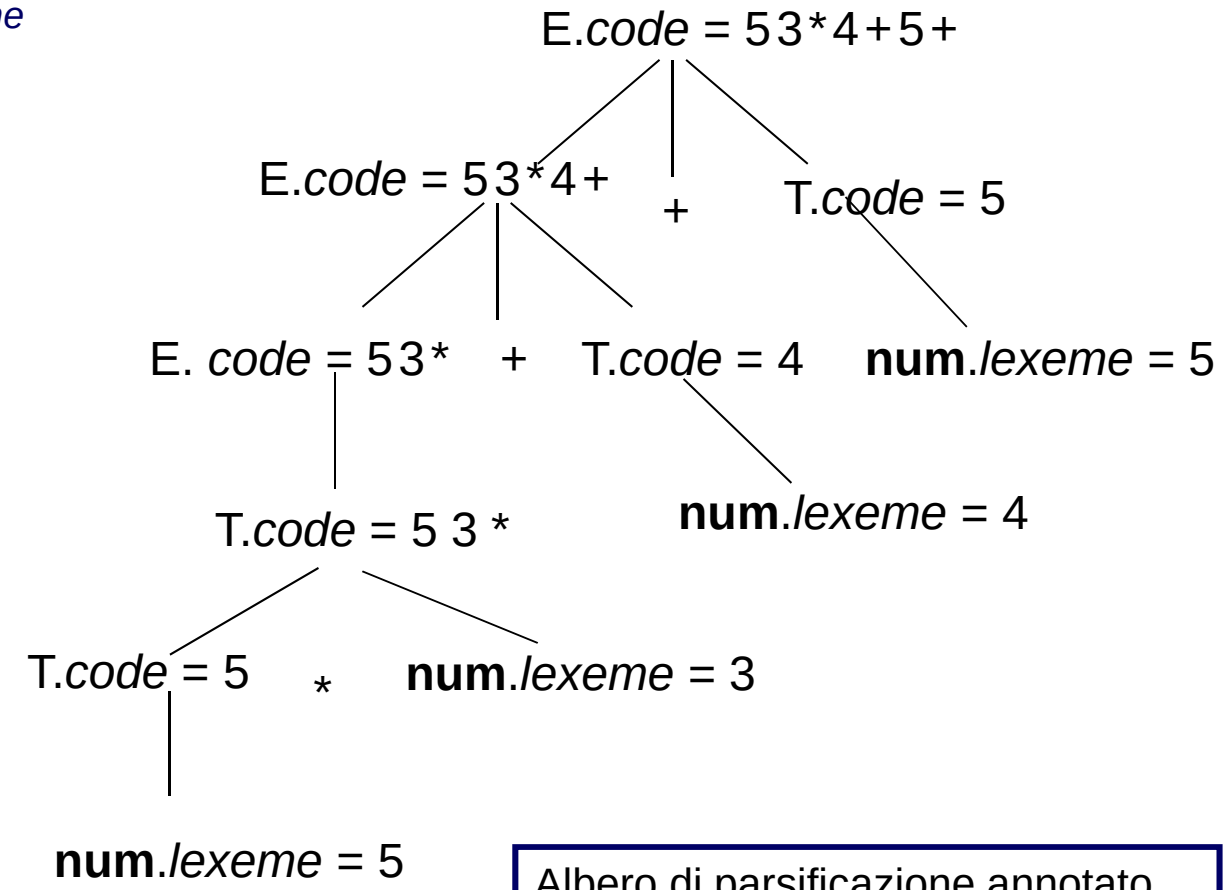
Generazione di output "al volo" (on-the-fly)

Esempio: Notazione infissa → Notazione postfissa III

$E \rightarrow E_1 + T$	{E.*code* = E_1.*code* || T.*code* || '+'}
$E \rightarrow T$	{E.*code* = T.*code*}
$T \rightarrow T$***mun**	{T.*code* = | T.*code* || **num**.lexeme || '*'}
$T \rightarrow$ **num**	{T.*code* = **num**.*lexeme*



Albero di parsificazione annotato per la stringa 5*3-4+5

Notiamo che in ogni nodo l'attributo .*code* è formato concatenando l'attributo .*code* dei figli (nell'ordine di occorrenza e aggiungendo eventualmente un operatore)

Traduzione L-attribuita con grammatica LL

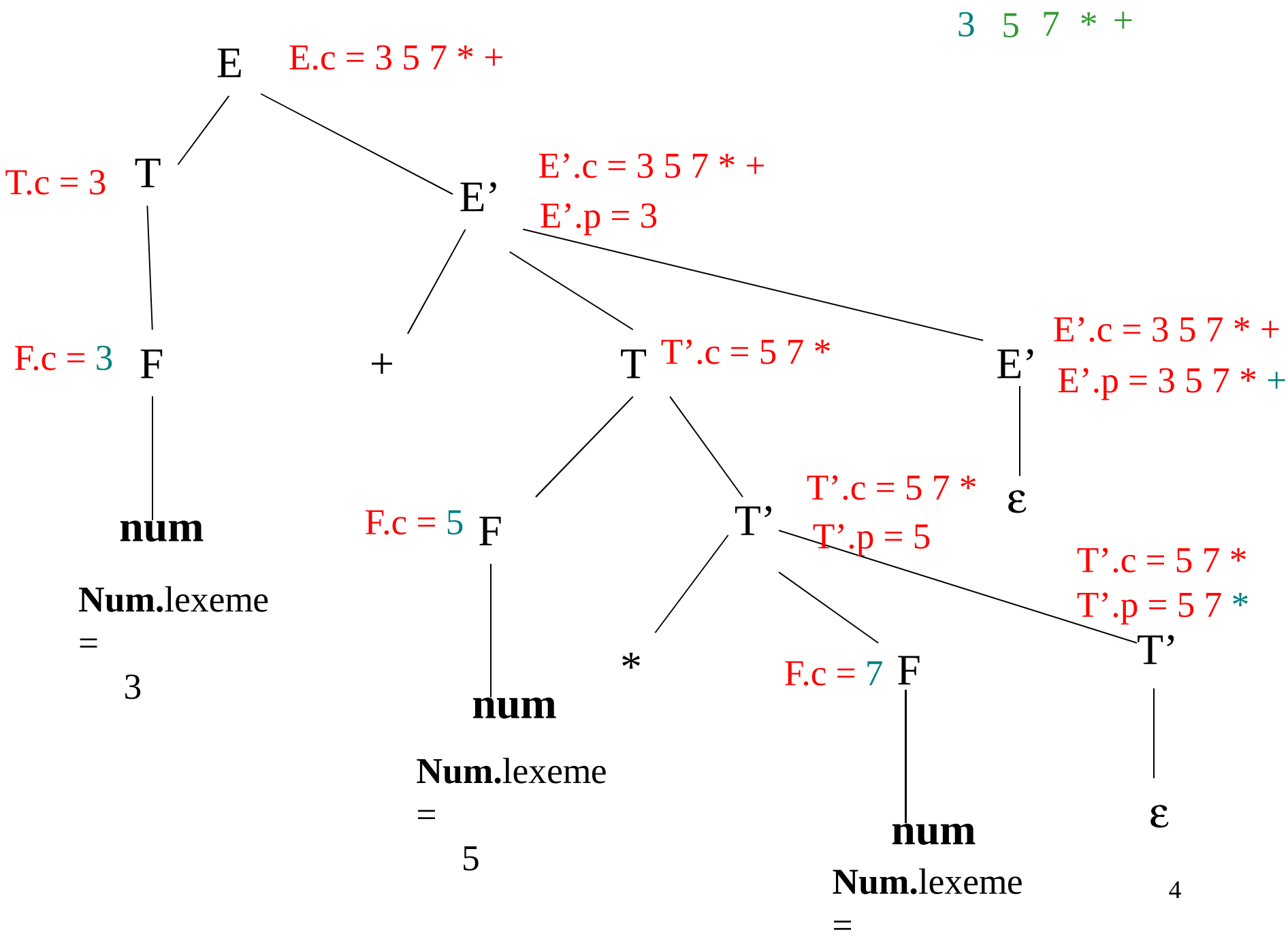
E → T {E'.prec= T.code
E' {E.code= E'.code}

E' → + T {E_1'.prec = E'.prec || T.code ||'+'}
E' {E'.code = E_1'.code}
E' → ε {E'.code= E'.prec}

T → F {T'.prec = F.code}
T' {T.code = T'.code}

T' → * F {T_1'.prec = T'.prec || F.code|| '*'}
T_1' {T'.code = T_1'.code}
T' → ε {T'.code= T'.prec}

F → (E) {F.code = E.code}
F → **num** {F.code= **num**.lexeme}


3 5 7 * +
E
E.c = 3 5 7 * +
T.c = 3
T
E’
E’.c = 3 5 7 * +
E’.p = 3
F.c = 3
F
+
T
T’.c = 5 7 *
E’
E’.c = 3 5 7 * +
E’.p = 3 5 7 * +
num
Num.lexeme
=
3
F.c = 5
F
T’
T’.c = 5 7 *
T’.p = 5
ε
T’.c = 5 7 *
T’.p = 5 7 *
T’
*
F.c = 7
F
num
Num.lexeme
=
5
num
Num.lexeme
=
ε

Generazione di output ‘’on-the fly’’

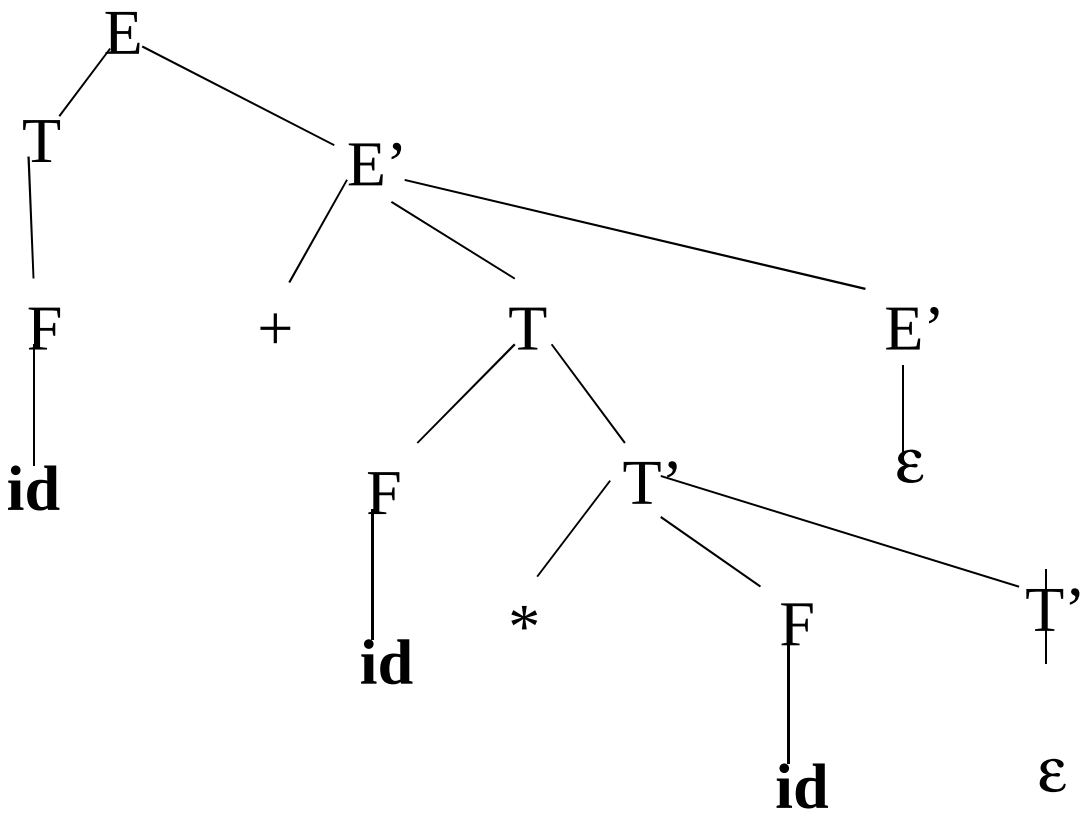
Un caso di particolare interesse si verifica quando in una traduzione un file di output (tipicamente il codice in un compilatore/traduttore) viene generato seguendo l’ordine di lettura *in profondità, da sinistra a destra,* dei nodi dell’albero sintattico.

In questo caso si puo’ sostituire la valutazione di attributi tipo code con la inserendo negli schemi comandi per la generazione diretta del codice nei punti opportuni.

Esempio:

traduzione in notazione *postfissa* di espressioni aritmentiche con parser LL

Esempio di albero di derivazione



Il seguente schema realizza la traduzione da un espressione aritmetica da una forma infissa ad una postifissa
L'output viene costruito su un scritto sequenzialmente su un file.
La funzione *emit* aggiunge un elemento all'output.

$E \rightarrow T\ E'$

$E' \rightarrow +\ T\ \{emit(\text{'+'})\}\ E'$

$E' \rightarrow \varepsilon$

$T \rightarrow F\ T'$

$T' \rightarrow *\ F\ \{emit(\text{'*'})\ T'$

$T' \rightarrow \varepsilon$

$F \rightarrow \mathbf{id}\ \{emit(\mathbf{id}.\text{lexeme})\}$

$F \rightarrow (E)$

Esempi di funzione

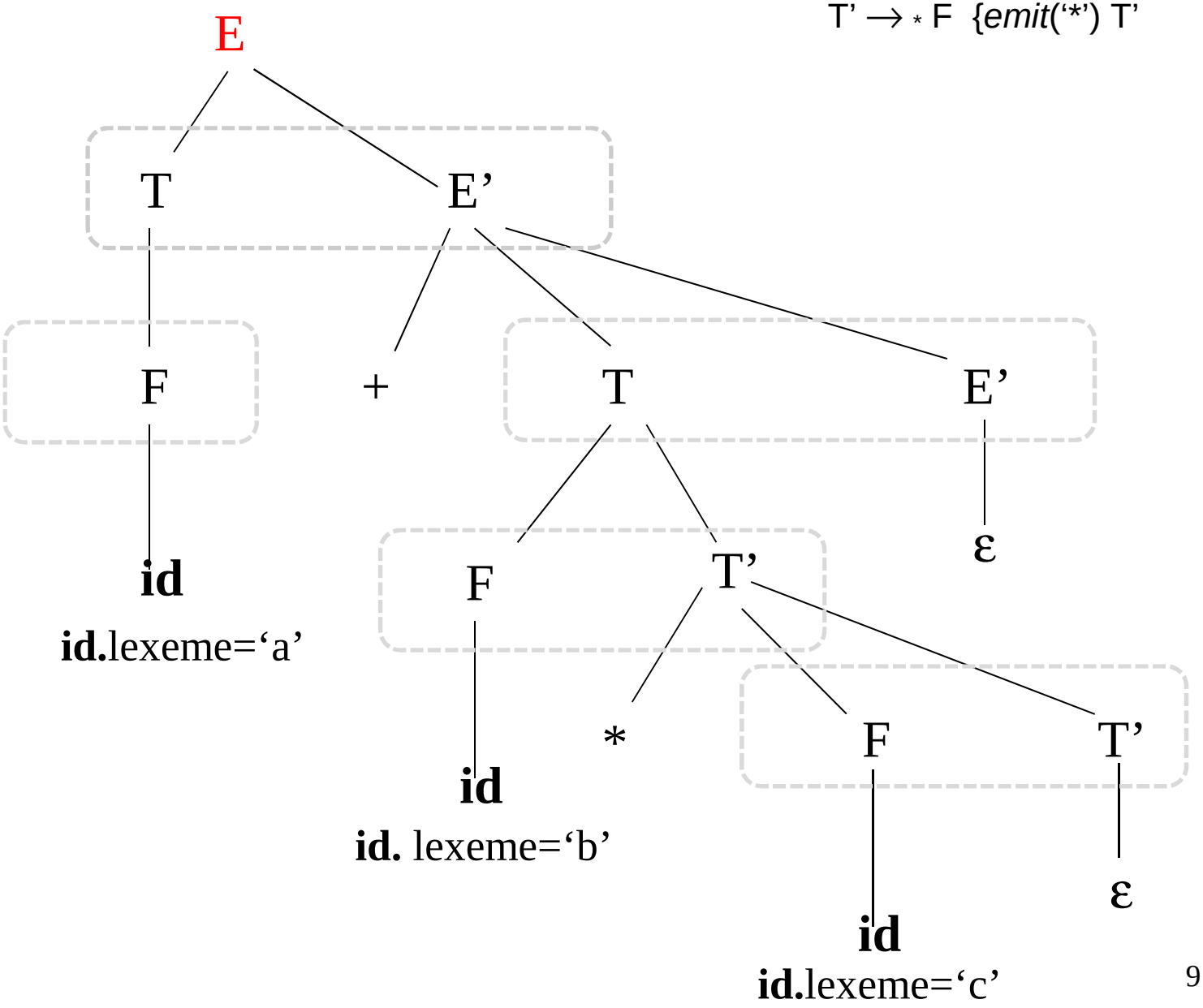
```
function E()
        begin
                if cc = '(' or cc = $  then T()
                   E'()
                   return
        end
function E'()
          begin  if (cc = '+')  then
                        cc ← PROSS
                        T()
                        emit('+')
                        E'()
                        return
                    else if (cc = ')' or cc=$ then
                         return
                    else ERRORE (…)
           end
```

Esercizio: scrivere le altre funzioni

Traduzione di a+b*c in a b c * +

E' → + T {*emit*('+')} E'

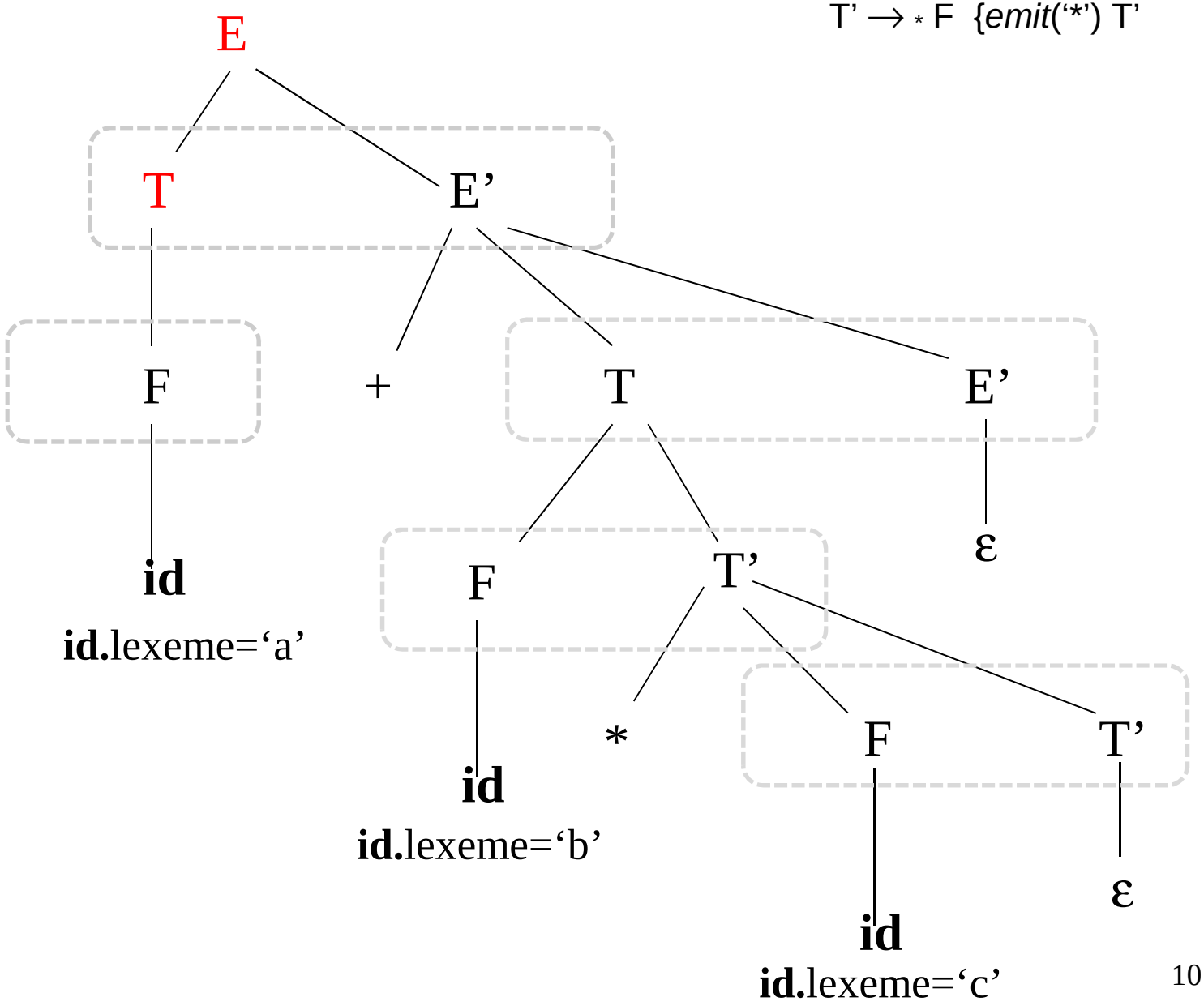
T' → * F {*emit*('*') T'



Traduzione di a+b*c in a b c * +

E' → + T {*emit*('+')} E'

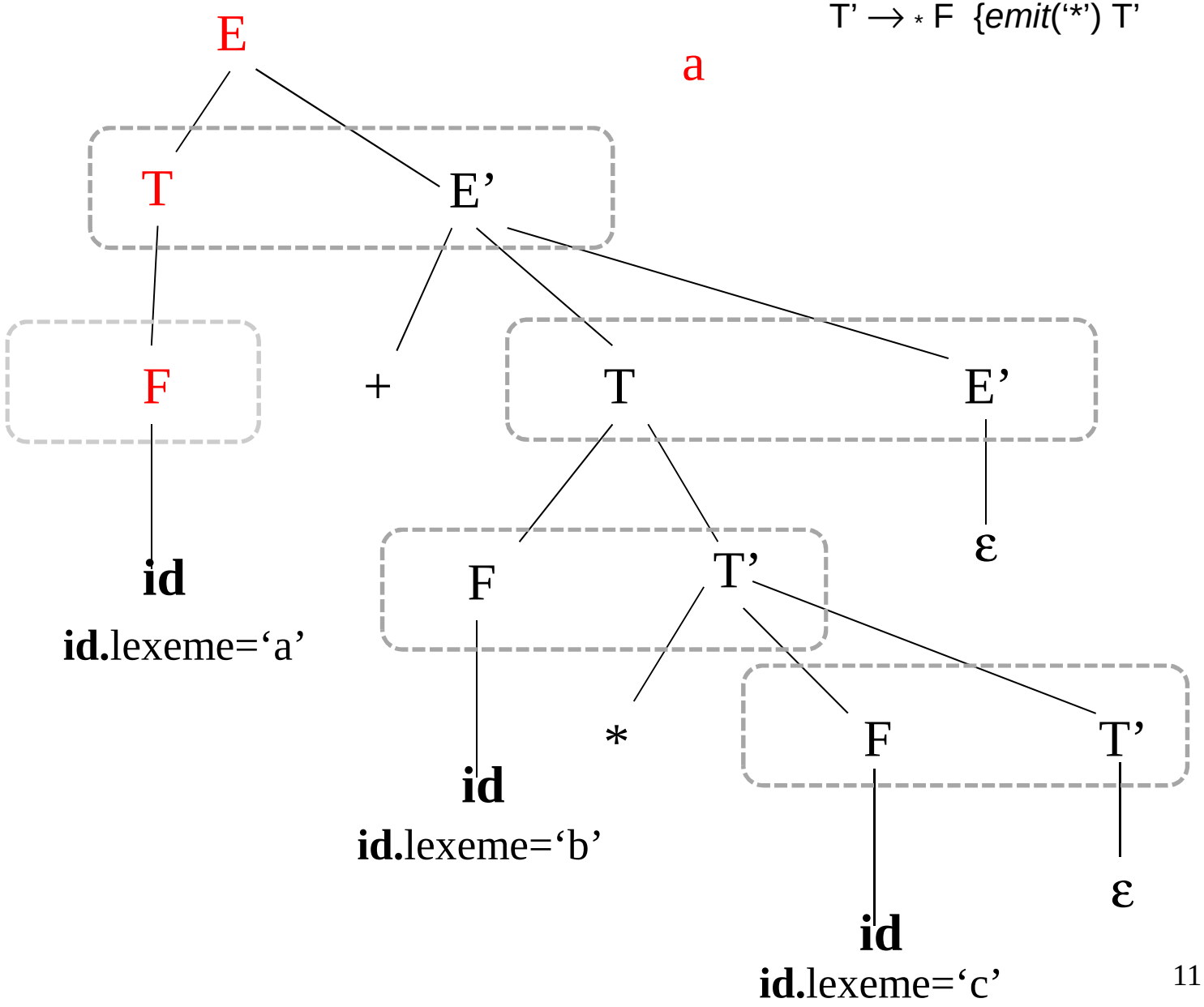
T' → * F {*emit*('*') T'



Traduzione di a+b*c in a b c * +

E' → + T {*emit*('+')} E'

T' → * F {*emit*('*') T'

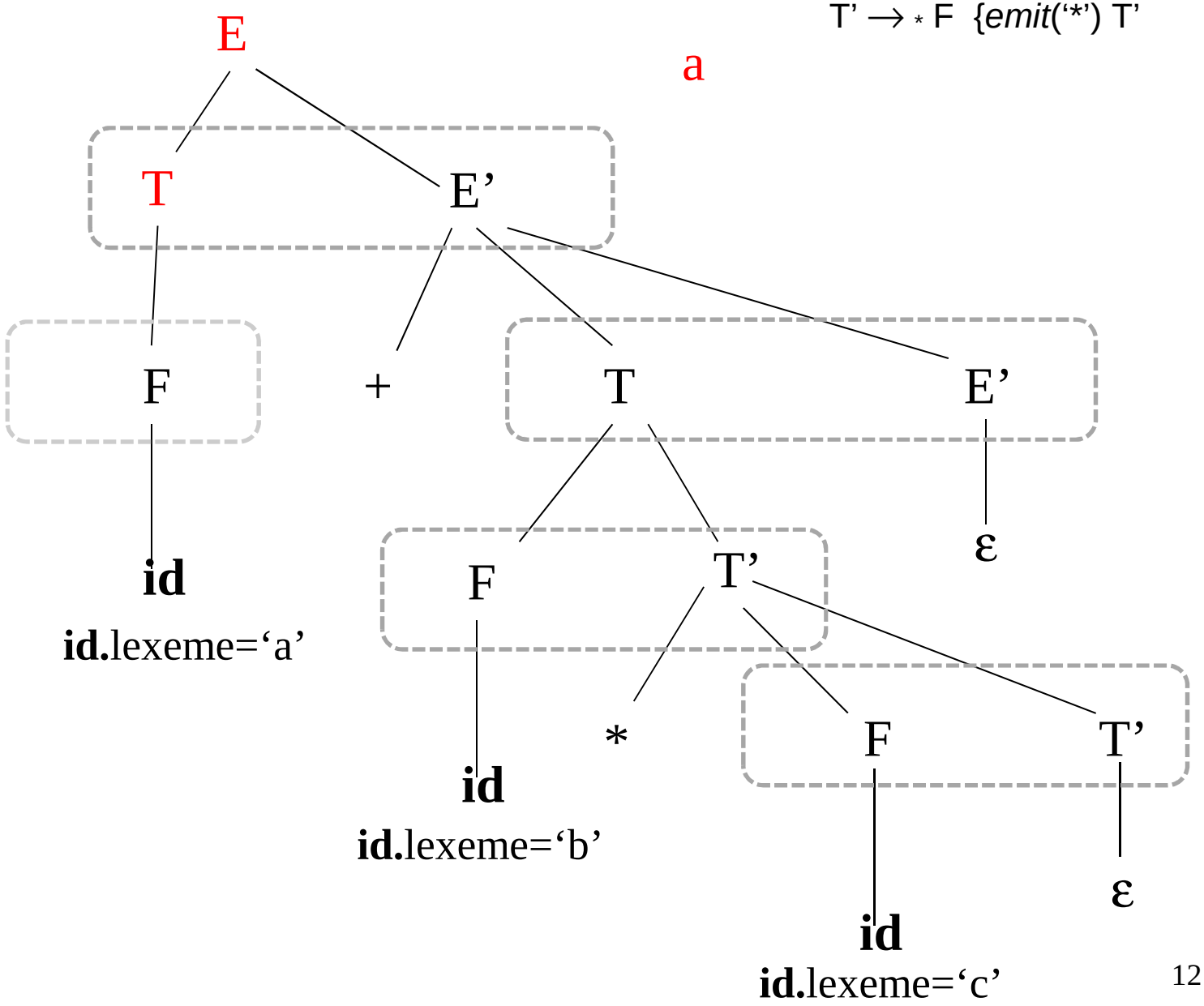


a

Traduzione di a+b*c in a b c * +

E' → + T {*emit*('+')} E'

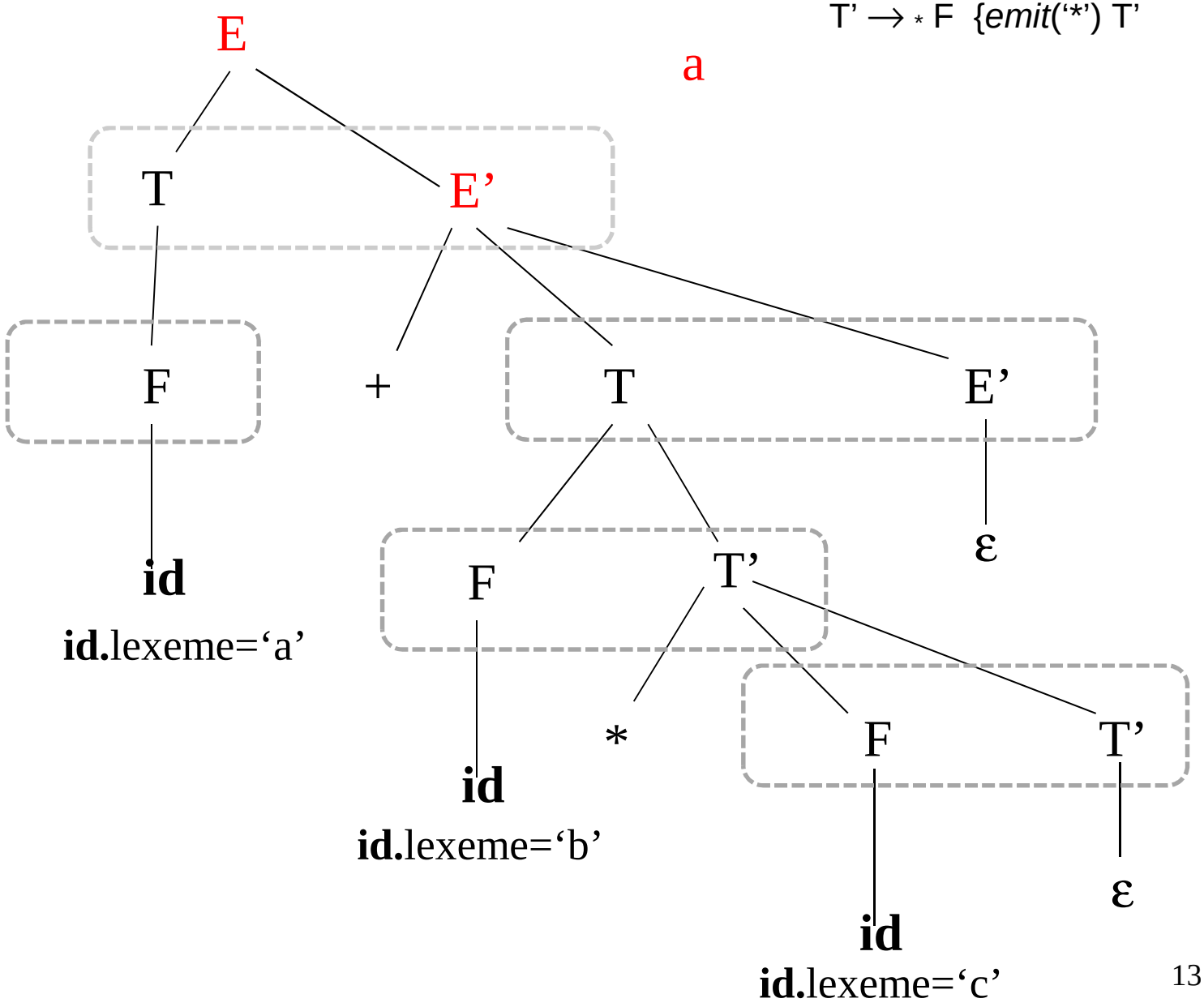
T' → * F {*emit*('*') T'



Traduzione di a+b*c in a b c * +

E' → + T {*emit*('+')} E'

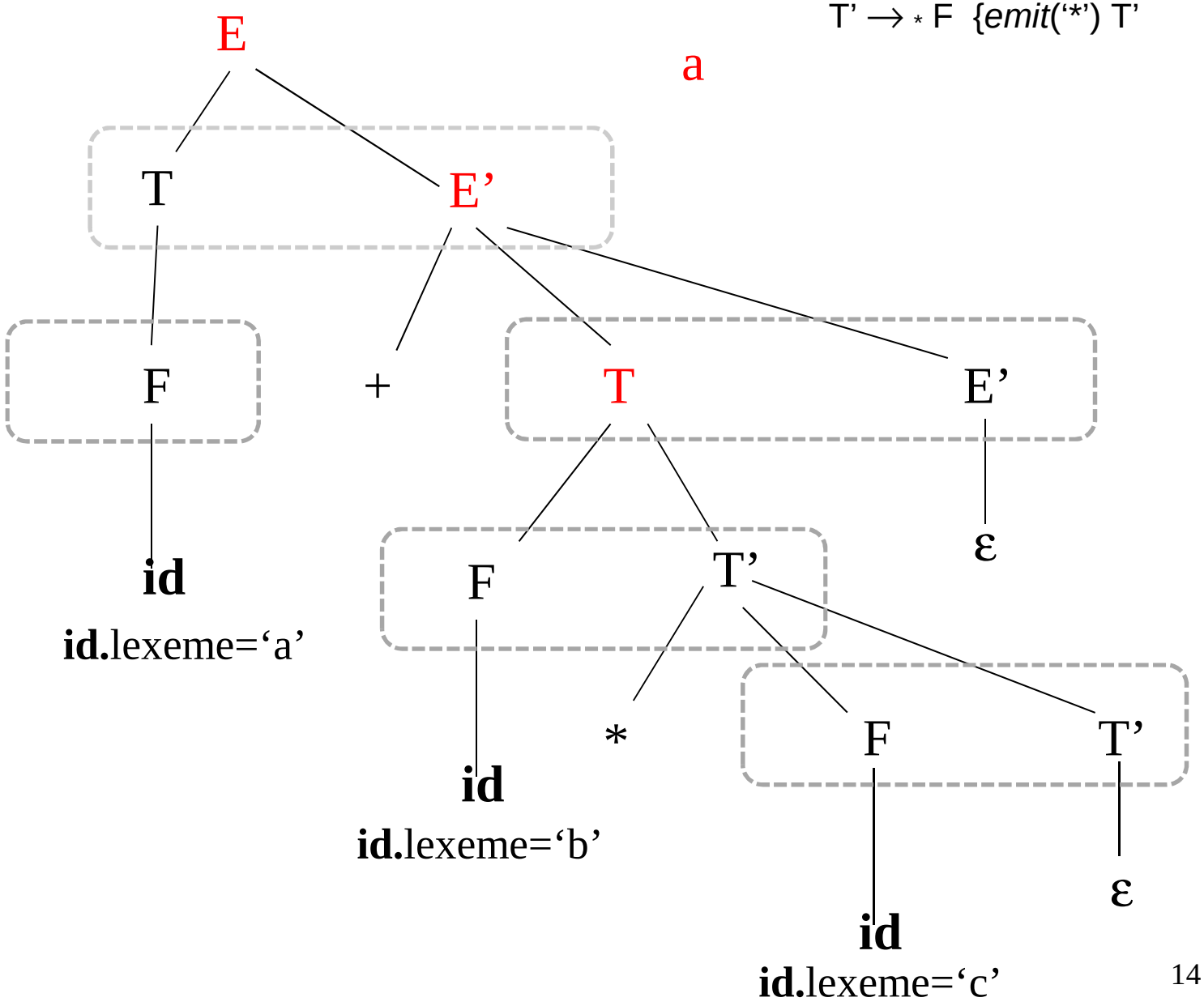
T' → * F {*emit*('*') T'



Traduzione di a+b*c in a b c * +

E' → + T {*emit*('+')} E'

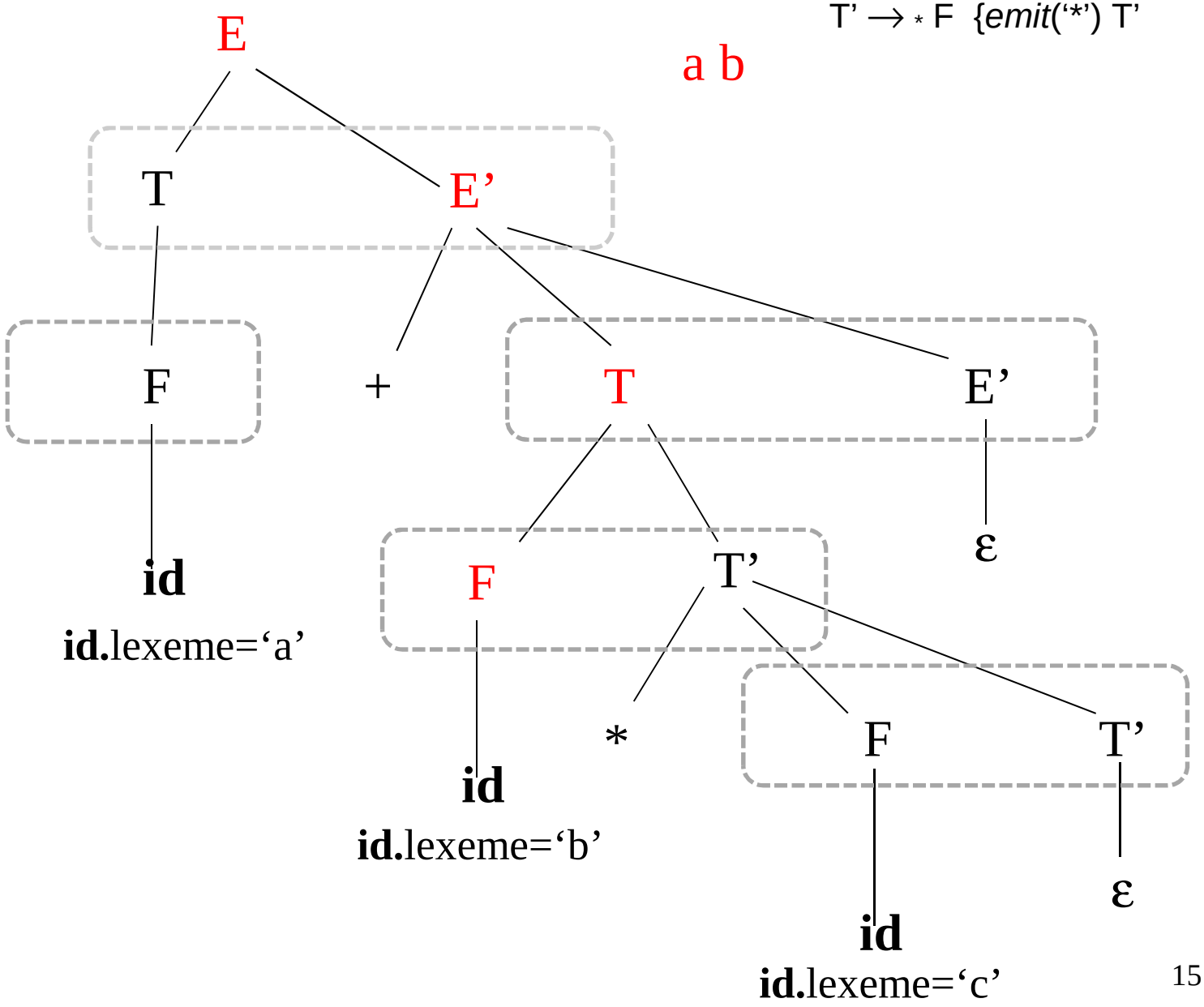
T' → * F {*emit*('*') T'



Traduzione di a+b*c in a b c * +

E' → + T {*emit*('+')} E'

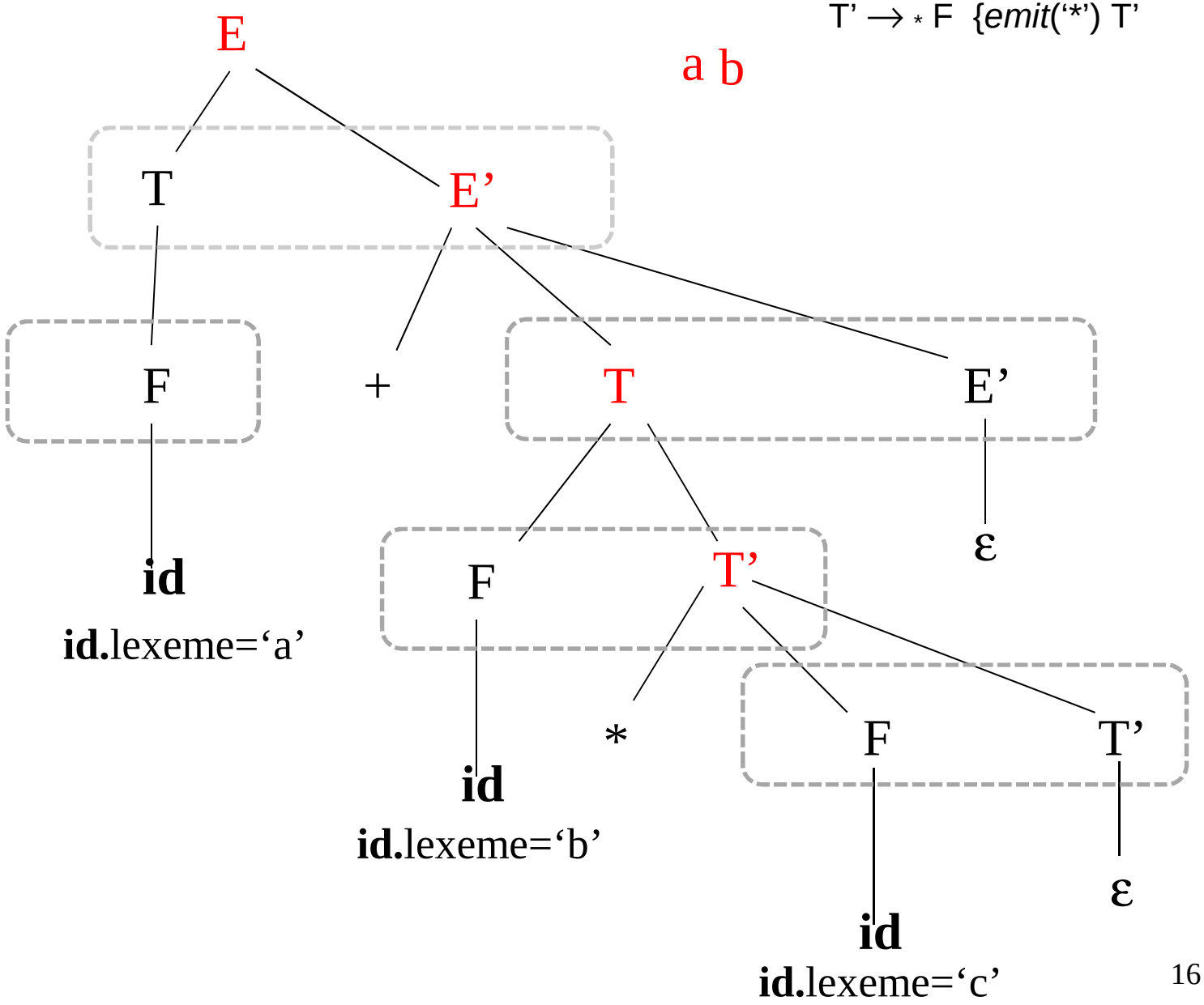
T' → * F {*emit*('*') T'



Traduzione di a+b*c in a b c * +

E' → + T {*emit*('+')} E'

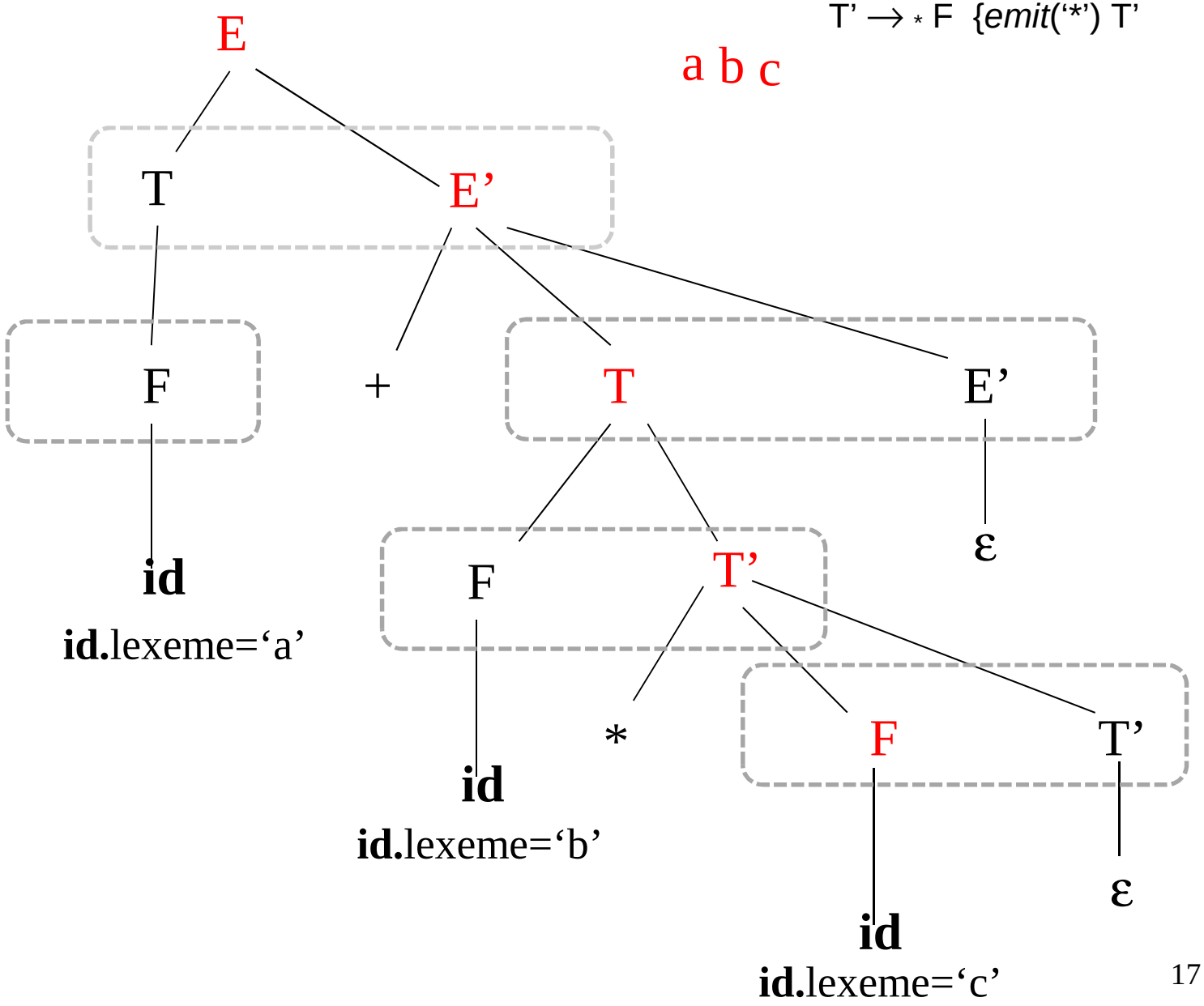
T' → * F {*emit*('*') T'



Traduzione di a+b*c in a b c * +

E' → + T {*emit*('+')} E'

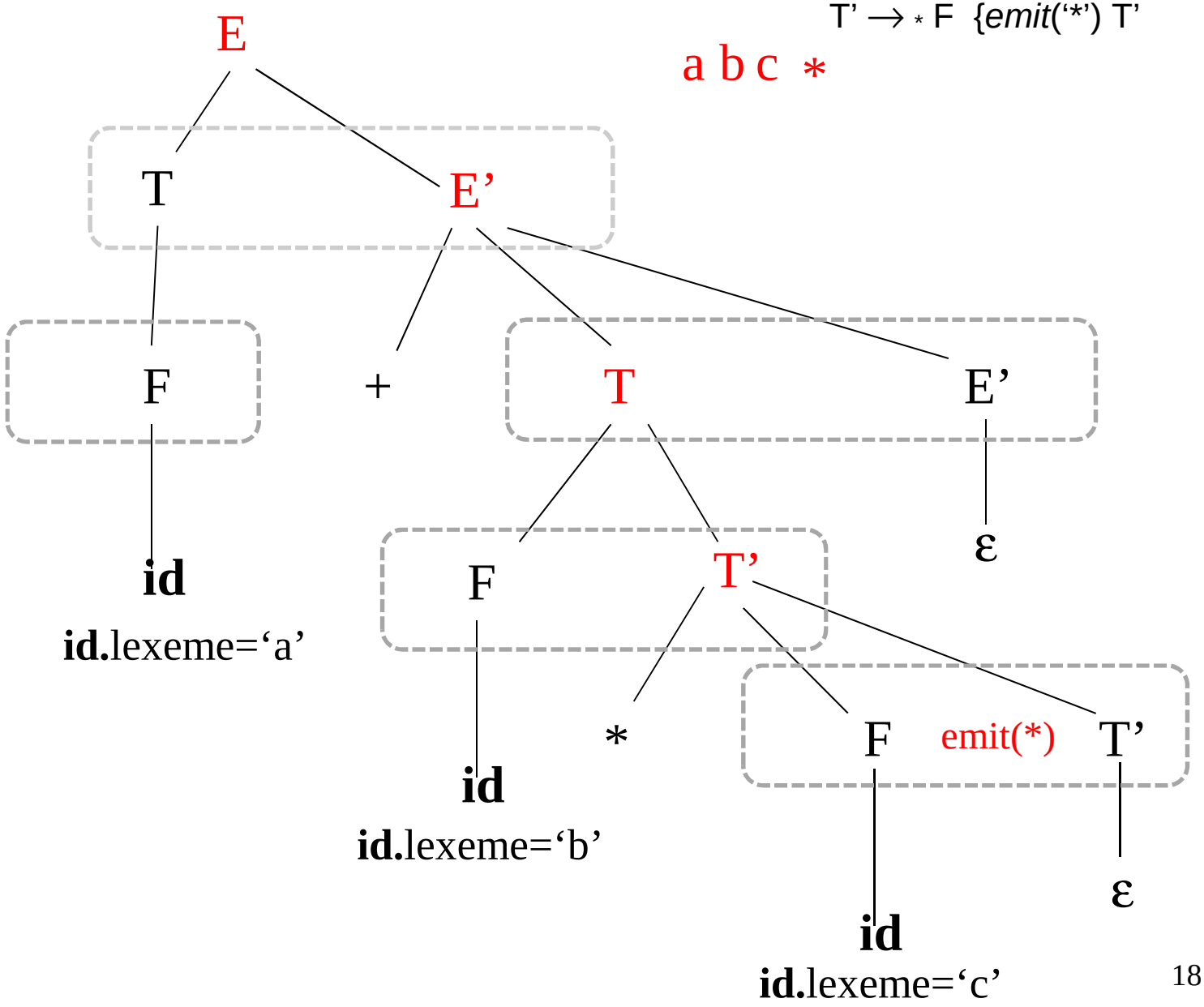
T' → * F {*emit*('*') T'



Traduzione di a+b*c in a b c * +

E' → + T {*emit*('+')} E'

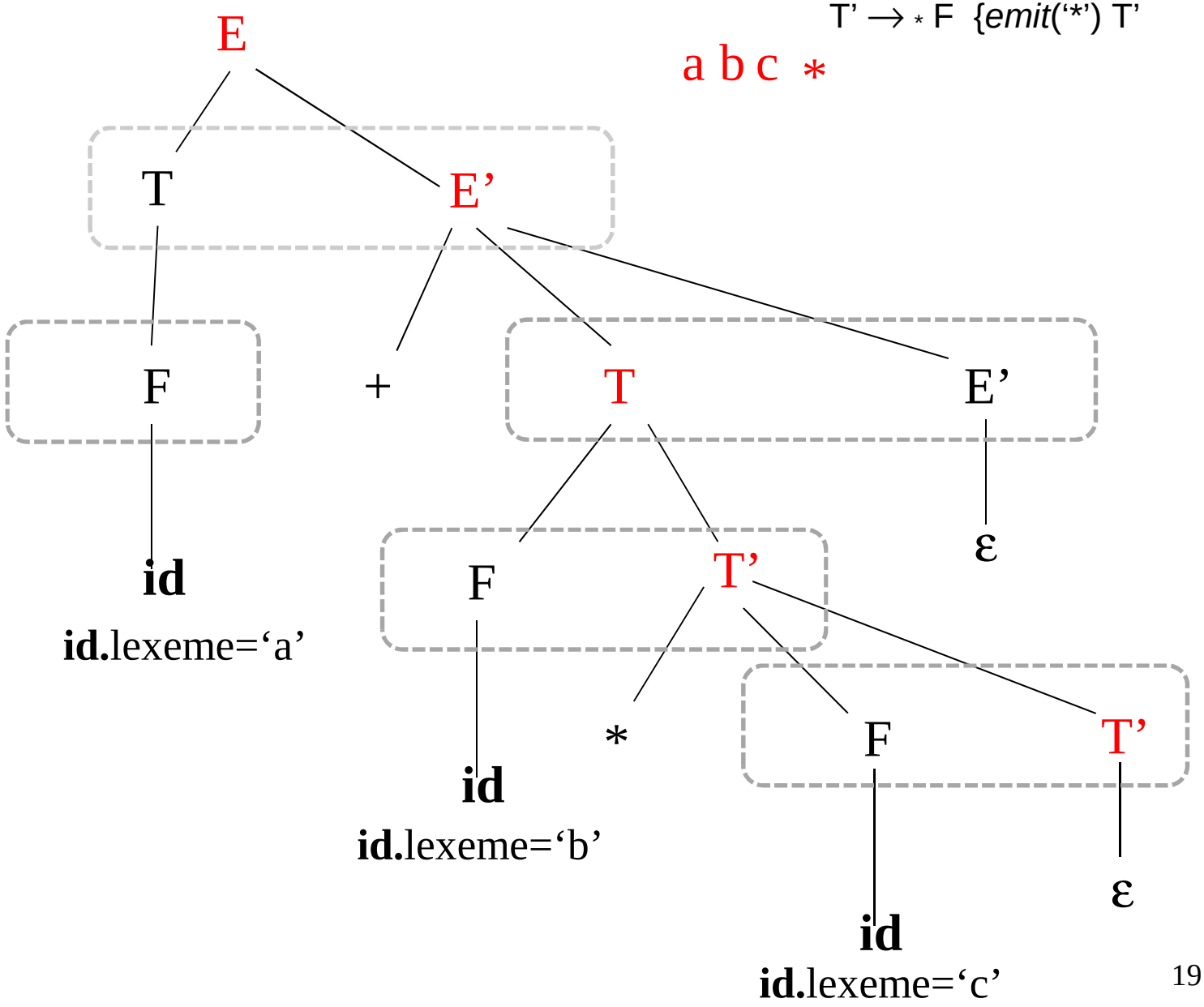
T' → * F {*emit*('*') T'



Traduzione di a+b*c in a b c * +

E' → + T {*emit*('+')} E'

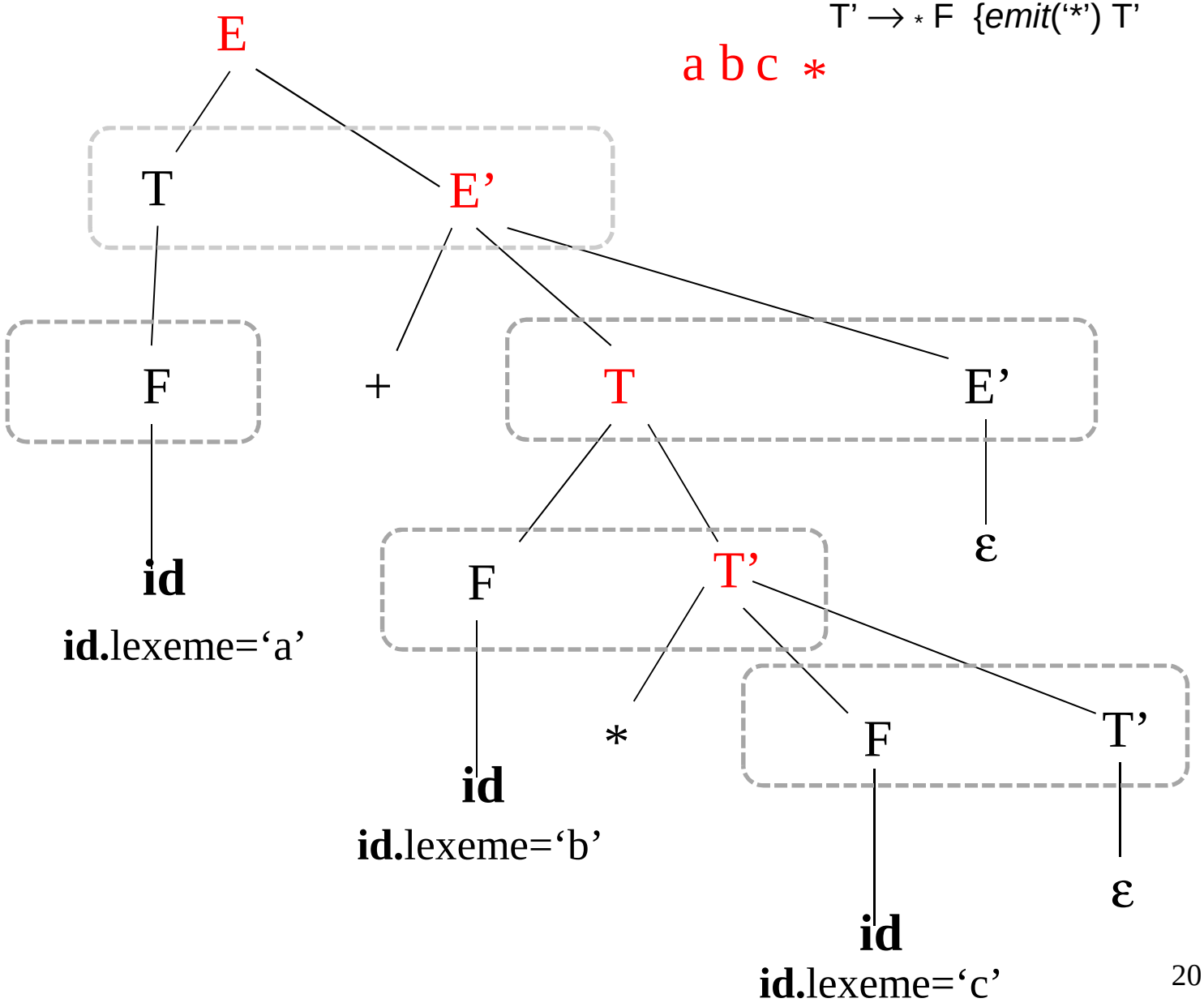
T' → * F {*emit*('*') T'



Traduzione di a+b*c in a b c * +

E' → + T {*emit*('+')} E'

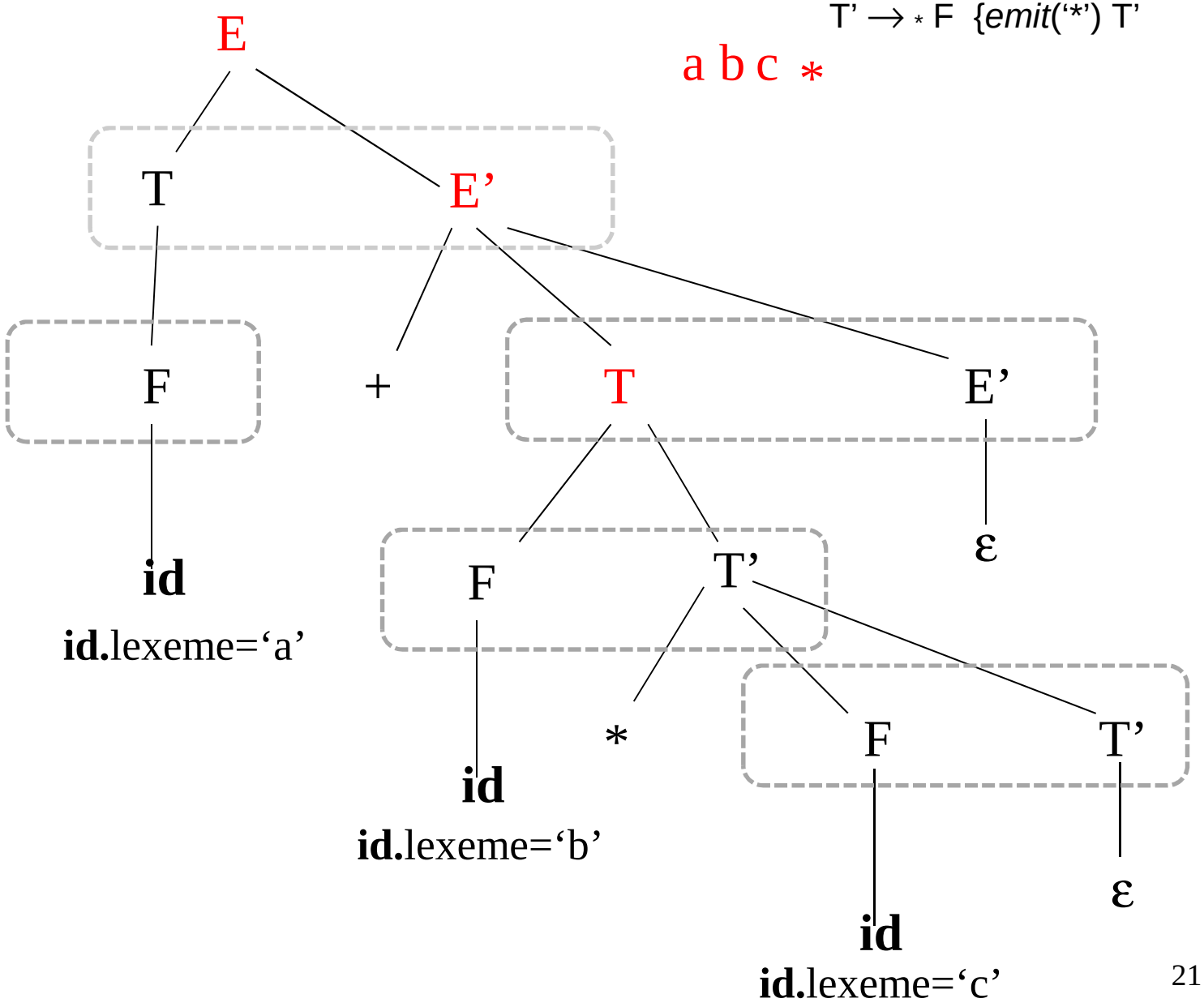
T' → * F {*emit*('*') T'



Traduzione di a+b*c in a b c * +

E' → + T {*emit*('+')} E'

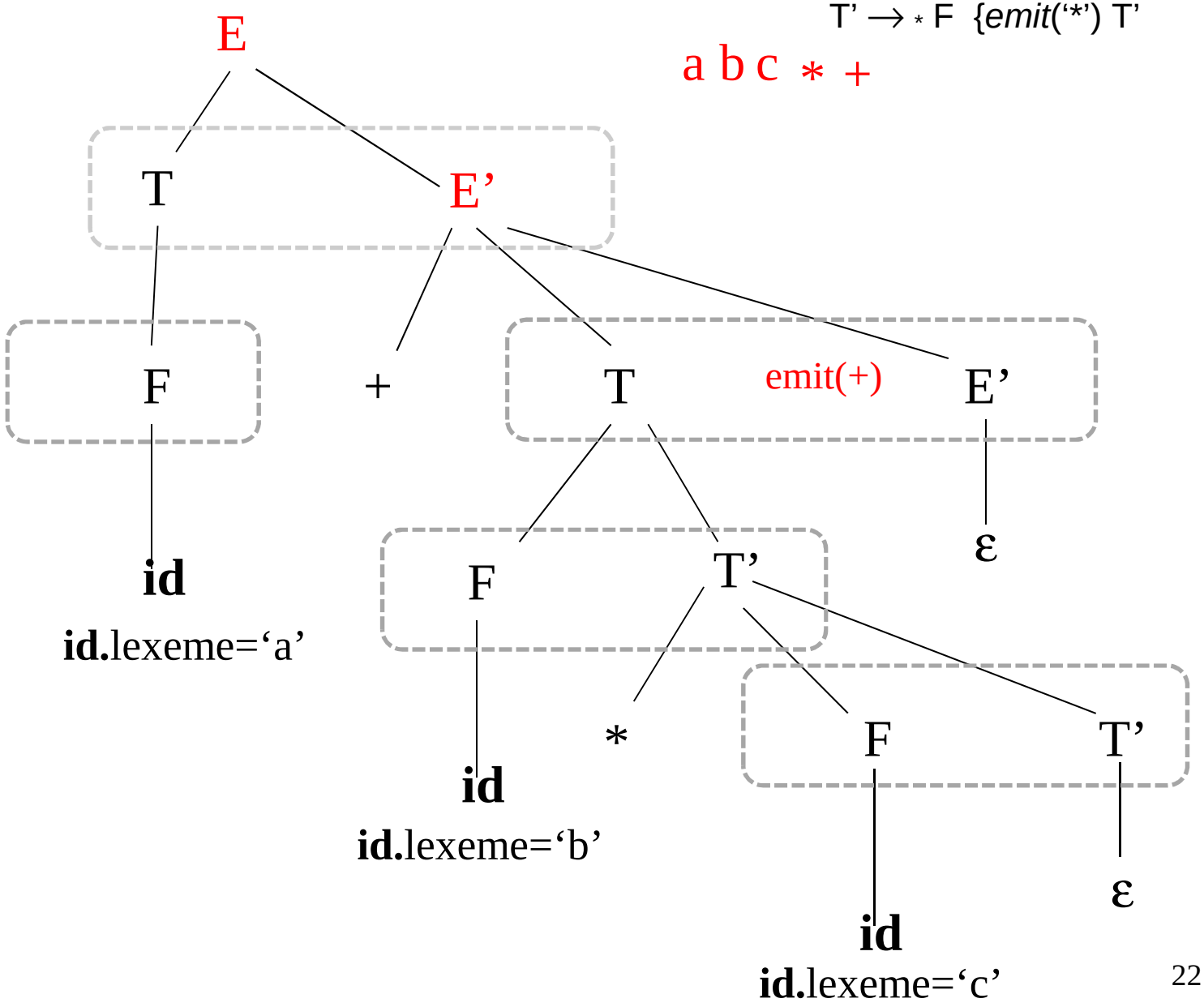
T' → * F {*emit*('*') T'



Traduzione di a+b*c in a b c * +

E' → + T {*emit*('+')} E'

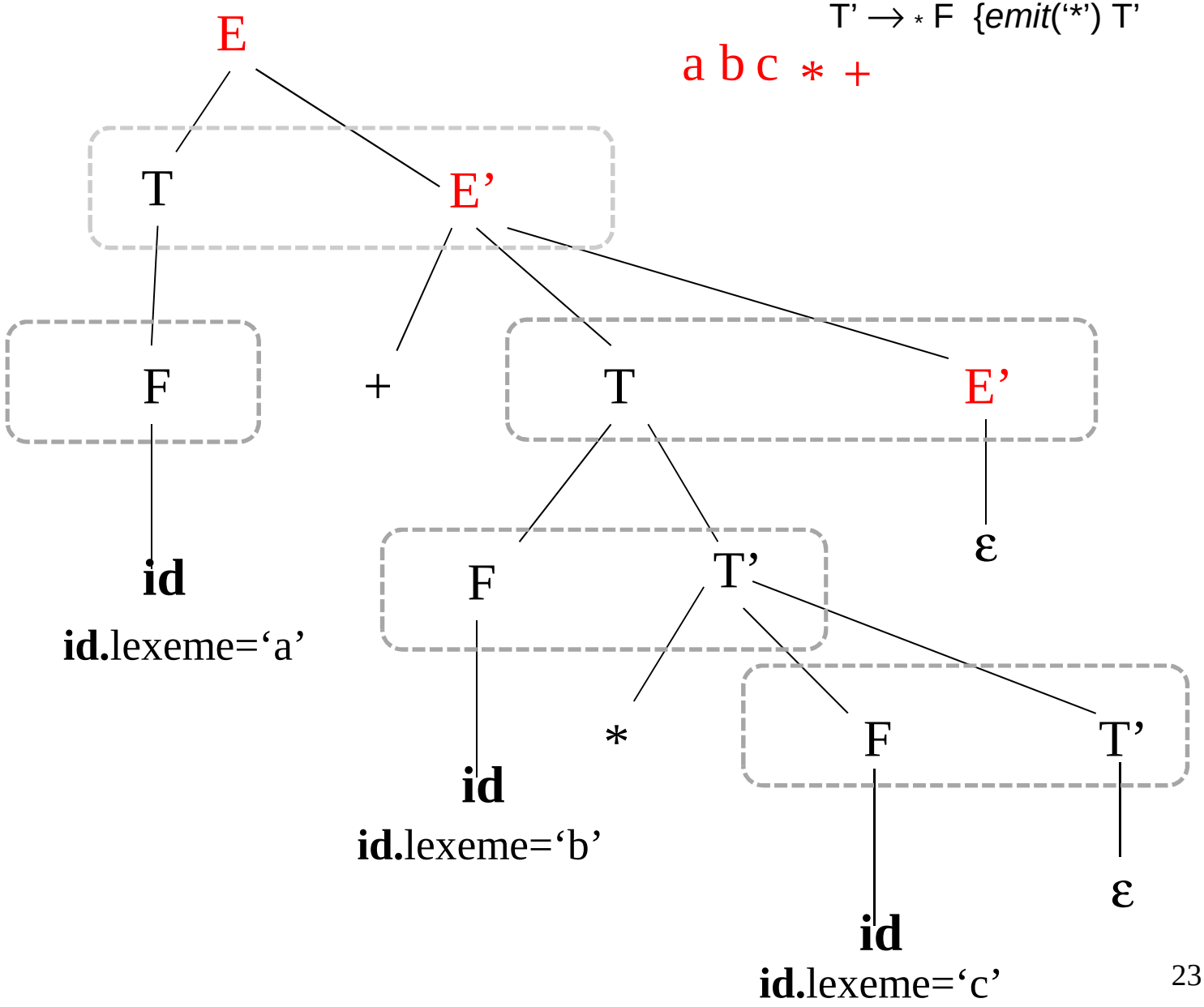
T' → * F {*emit*('*') T'



Traduzione di a+b*c in a b c * +

E' → + T {*emit*('+')} E'

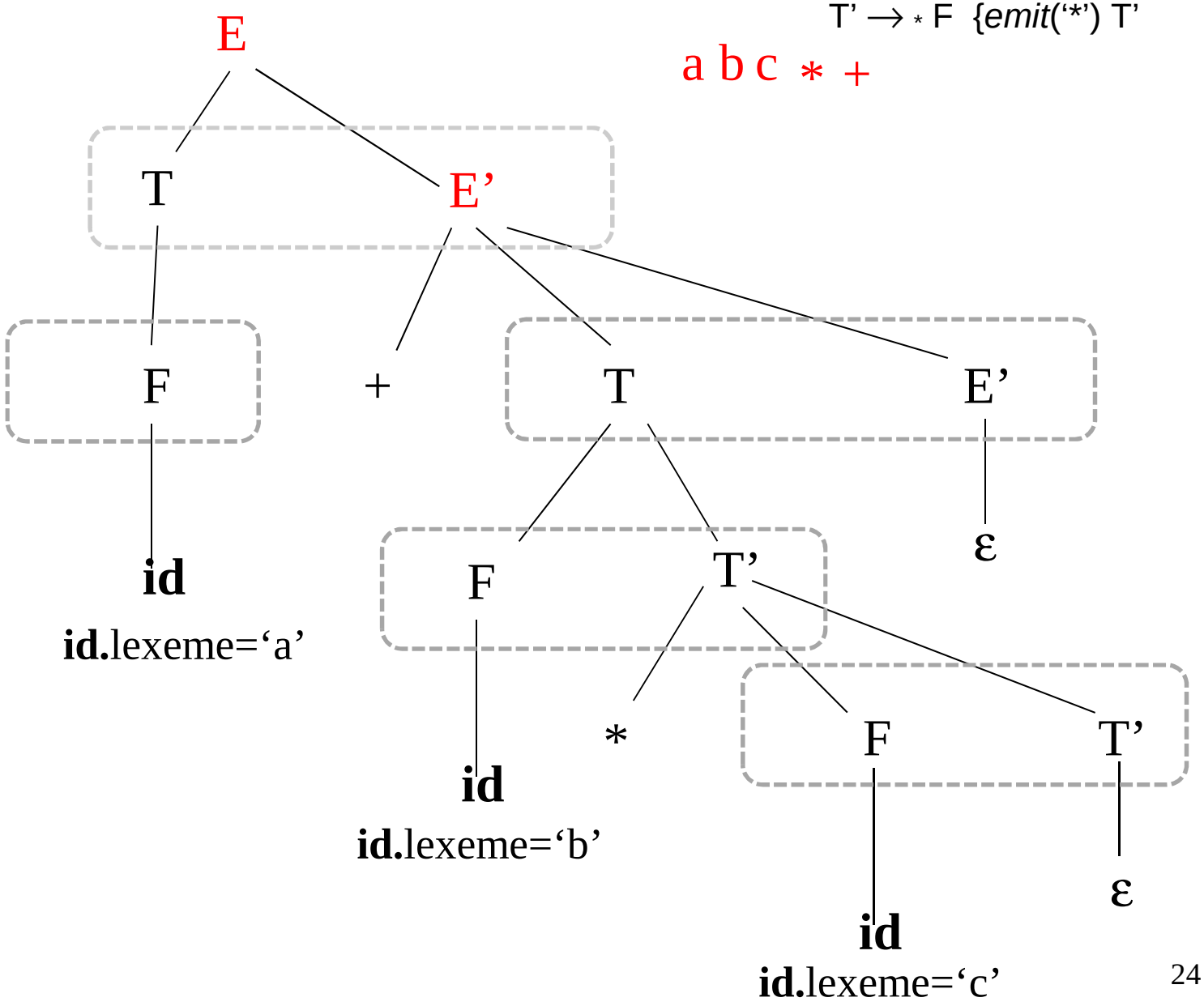
T' → * F {*emit*('*') T'



Traduzione di a+b*c in a b c * +

E' → + T {*emit*('+')} E'

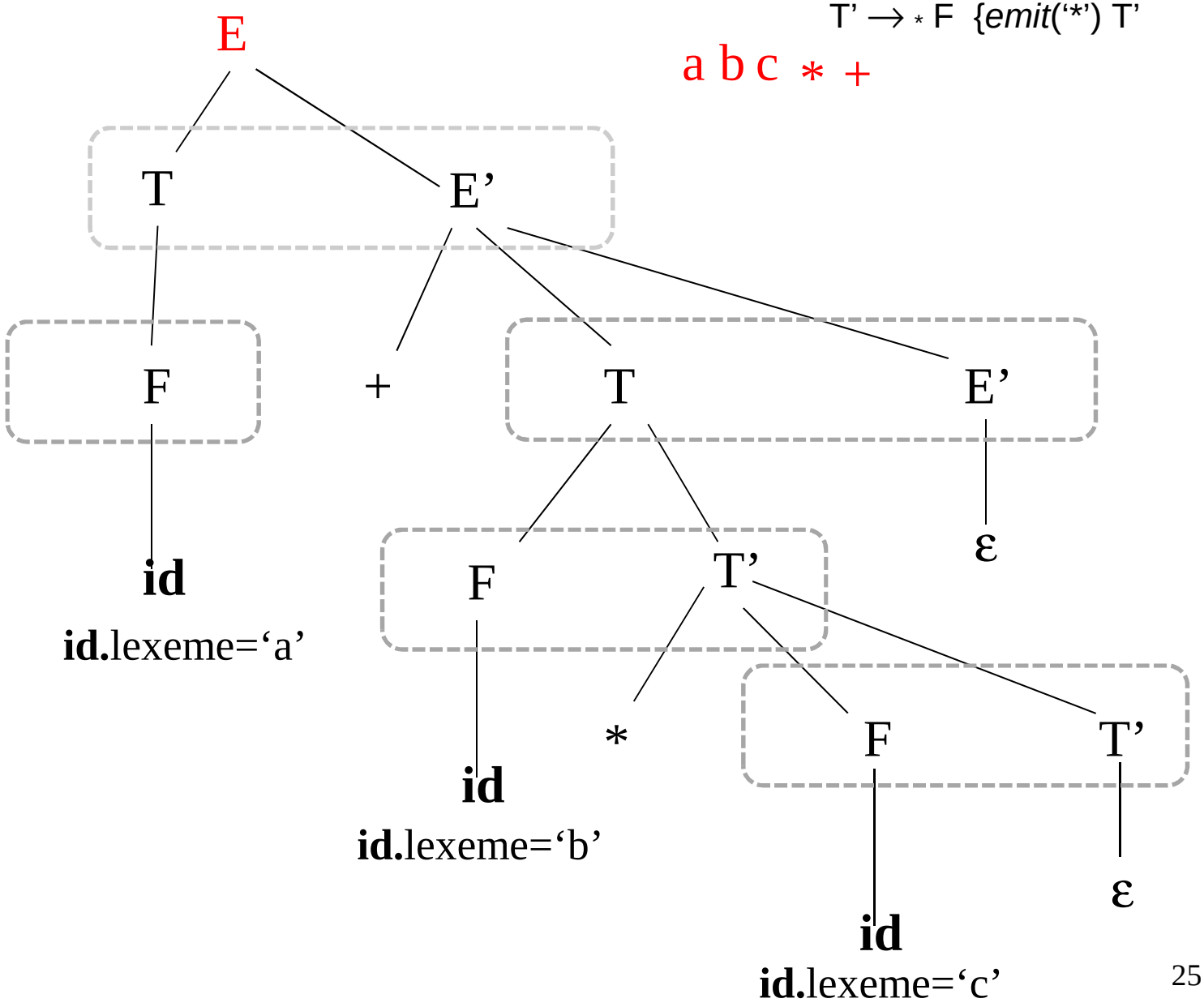
T' → * F {*emit*('*') T'



Traduzione di a+b*c in a b c * +

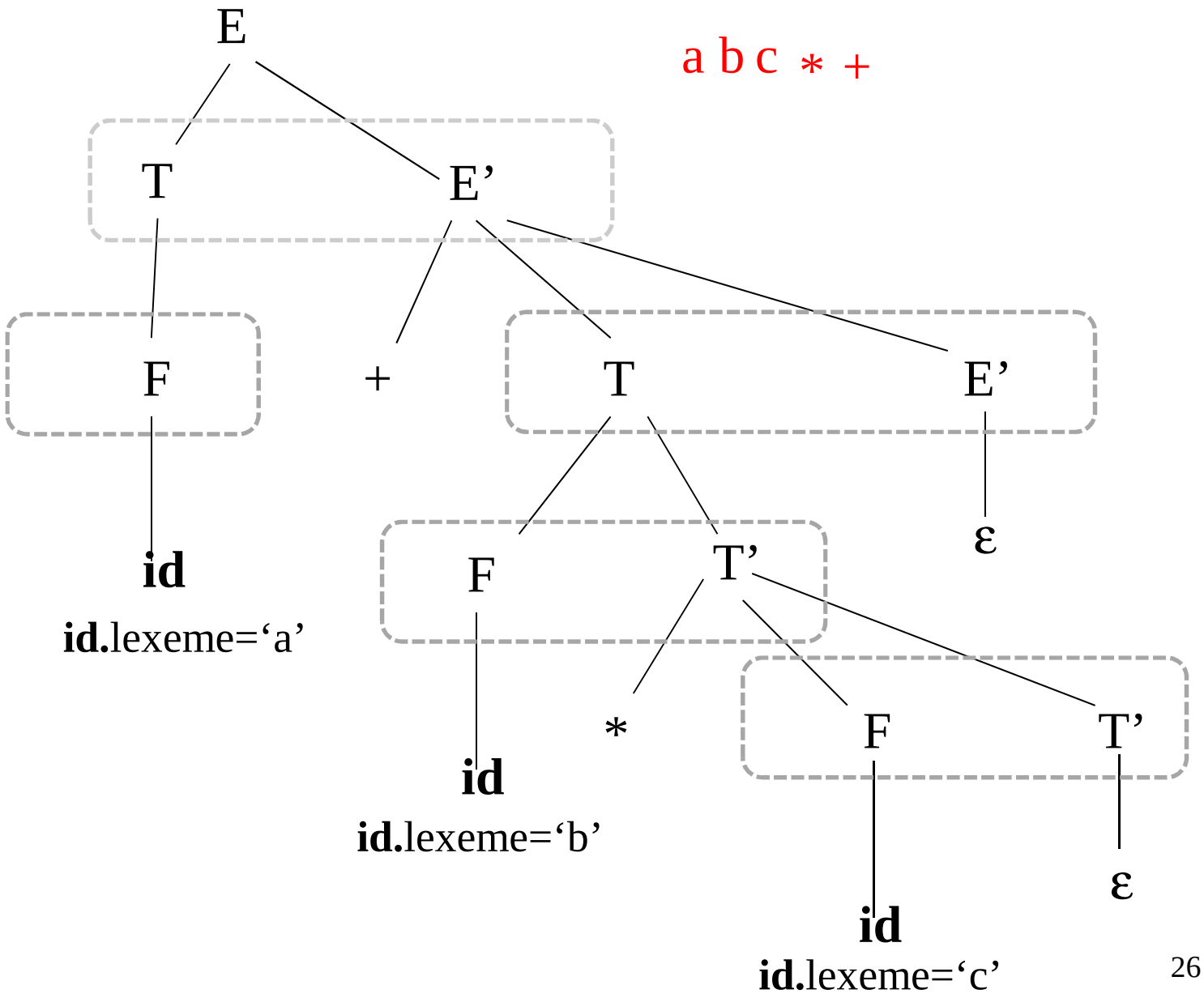
E' → + T {*emit*('+')} E'

T' → * F {*emit*('*') T'



Traduzione di a+b*c in a b c * +

FINE



Generazione di output ‘’on-the fly”

Un caso di particolare interesse si verifica quando in una traduzione un attributo *sintetizzato* definisce la generazione di un output in modo incrementale. E’ per esempio il caso dell’attributi *list* dell’esempio della lista delle differenze e, tipicamente, della generazione del codice nei compilatori-traduttori.

Prendiamo il caso della lista delle differenze in cui l’attributo list è calcolato in:

$L \rightarrow N$; $\{L_1.elem = L.elem\}$
L_1 $\{L.list = \underline{cons}\ (N.val - L.elem, L_1.list)\}$
$L \rightarrow \varepsilon$ $\{L.list = \underline{null}\}$

invece di gestire il campo list come attributo si può assumere una funzione *addlist* che aggiunge un elemento ad un file di output ed una funzione *create_empty_list* che inizializza il file con la lista vuota.

Lista delle diffferenze:output ‘’on-the fly”

Nella lista delle differenze, la valutazione dell’attributo .list è sostituita dalla generazione incrementale di una lista contenuta in una variablile globale LIST.

Lo schema diventa:

$C \rightarrow N\#$ {L.*elem* = N.*val*}
L
$L \rightarrow N;$ {L_1.*elem* = L.*elem*}
L_1 {*addlist*(N.*val* - L.*elem*, *LIST*)}
$L \rightarrow \varepsilon$ {LIST← *create_empty_list*}
$N \rightarrow$ **num** {N.*val* = **num**.*val*}

Valutazione top-down di definizioni L-attribuite VI

Esempio: Lista delle differenze III

$C \rightarrow N\ \#\ \ \{L.elem = N.val\}$
$\qquad\quad L$

```
function C
    var C_list, N_val, L_elem, L_list
    begin if (cc = num)
                N_val ← N()
                if (cc = '#')
                        cc ← PROSS
                        L_elem ← N_val
                        L (L_elem)
                        else ERRORE(…)
            else ERRORE(…)
            return
     end
```

La traduzione si troverà nella variabile globale LIST

Valutazione top-down di definizioni L-attribuite VII

Esempio: Lista delle differenze IV

$L \rightarrow N$;	$\{L_1.elem = L.elem\}$
L_1	$\{addlist(N.val - L.elem, LIST)\}$
$L \rightarrow \varepsilon$	$\{LIST \leftarrow empty\_list\}$

```
function L (L_elem)
    var L_list, N_val, L1_elem, L1_list
    begin if (cc = num)
                N_val ← N()
                if (cc = ';')
                        cc ← PROSS
                        L1_elem ← L_elem
                        L (L1_elem)
                        addlist(N_val – L_elem, LIST)
                else ERRORE(…)
          else if (cc = '$')    LIST ← empty_list()
          else ERRORE(…)
          return
    end
```

Valutazione top-down di definizioni L-attribuite VIII

Esempio: Lista delle differenze V

4 # 6 ; 3 ; **$**

↑

```
function C
var .....
begin if (cc = num)
        N_val ← N()
        if (cc = num) N_val ← num.val
                cc ← PROSS ; return N_val
        if (cc = '#') cc ← PROSS
              L_elem ← N_val
              L (L_elem)
               if (cc = num)   N_val← N()
                               if (cc = num) N_val ← num.val
                                       cc ← PROSS ; return N_val
                       if (cc = ';') cc ← PROSS
                             L1_elem1 ← L_elem
                             L(L1_elem)
                               if (cc = num)  N_val ← N()
                                              if (cc = num) N_val ← num.val
                                                      cc ← PROSS ; return N_val
                                      if (cc = ';') cc ← PROSS
                                            L1_elem ← L_elem
                                            L (L1_elem2)
                                                if (cc = '$') list ←empty_list
                                                        return
                                                addlist(N_val – L1_elem, list)
                                      return
                               addlist (N_val – L_elem, list)
                       return
        return
end
```

{N_val = 4}

{L_elem = 4}

{L(4)} =>

{N_val = 6}

{L1_elem = 4}

{L(4)} =>

{N_val = 3}

{L1_elem= 4}

{L(4)} =>

{list = <>}

{list = < -1 >}

{list = < 2,-1 >}

{list = < 2,-1 >}

5. Data la grammatica con il seguente insieme di produzioni:

$L \rightarrow S\ T$ $\quad$ $T \rightarrow S\ T$

$T \rightarrow \varepsilon$ $\quad$ $S \rightarrow id\ :=\ E;$

$E \rightarrow id$ $\quad$ $E \rightarrow num\ G$

$G \rightarrow +\ num\ G$ $\quad$ $G \rightarrow \varepsilon$

- trovare gli insiemi guida delle produzioni;
- scrivere l'analizzatore a discesa ricorsiva;
- attribuire la grammatica in modo da associare ad ogni stringa in input il programma in cui le espressioni numeriche sono state sostituite dal loro valore, gli identificatori sostituiti dai lessemi associati e le istruzion sono separate da una virgola.

Ad esempio la frase: id := num + num + num; id := id; deve essere tradotta come:

A ← 18, B ← A, nell'ipotesi che id1.*name* = A, id2.*name* = B, id3.*name* = A e num1.*val* = 10, num2.*val* = 6, num3.*val* = 2.