

Kapitola XI.

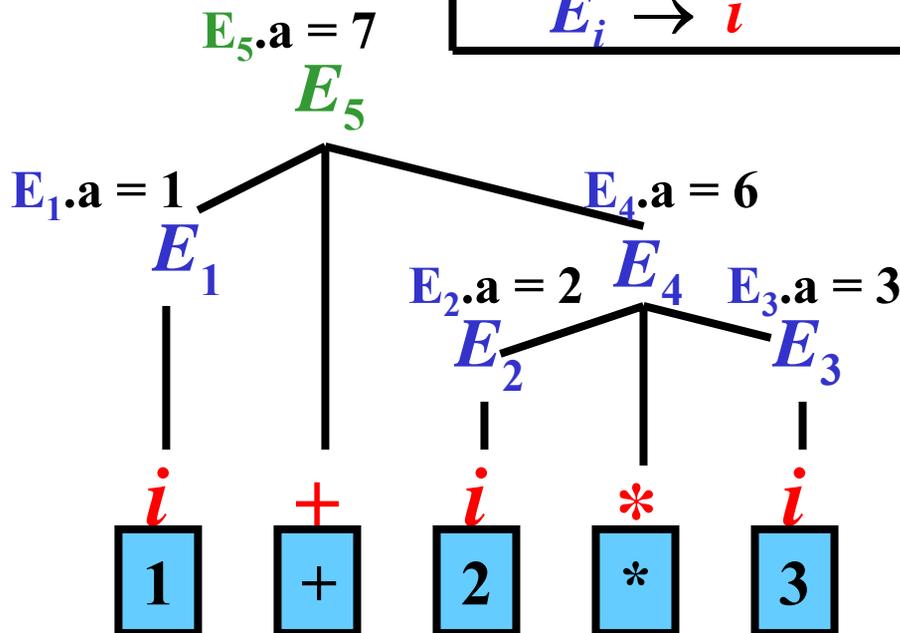
Syntaxí řízený překlad a generování vnitřního kódu

Syntaxí řízený překlad

Myšlenka: Semantické akce jsou přidruženy gramatickým pravidlům. Tyto sémantické akce mohou generovat vnitřní kód a provádět kontrolu typů

Příklad:

Pravidlo:	Sémantická akce:
$E_i \rightarrow E_j + E_k$	$\{ E_i.a := E_j.a + E_k.a \}$
$E_i \rightarrow E_j * E_k$	$\{ E_i.a := E_j.a * E_k.a \}$
$E_i \rightarrow (E_j)$	$\{ E_i.a := E_j.a \}$
$E_i \rightarrow i$	$\{ E_i.a := i.val \}$



Pravidlo:

$E_1 \rightarrow i$
 $E_2 \rightarrow i$
 $E_3 \rightarrow i$
 $E_4 \rightarrow E_2 * E_3$
 $E_5 \rightarrow E_1 + E_4$

Akce:

$E_1.a := i.val$
 $E_2.a := i.val$
 $E_3.a := i.val$
 $E_4.a := E_2.a * E_3.a$
 $E_5.a := E_1.a + E_4.a$

Třídresný kód

- Instrukce v třídresném kódu (**3AK**) má tvar:

(**o** ,  **a** ,  **b** ,  **r**)

- **o** – operátor (+, −, *, ...)
- **a** – operand 1 ( **a** = adresa **a**)
- **b** – operand 2 ( **b** = adresa **b**)
- **r** – výsledek ( **r** = adresa **r**)

Příklady:

(:= , a , , c) ... *c := a*
 (+ , a , b , c) ... *c := a + b*
 (not , a , , b) ... *b := not(a)*
 (goto , , , L1) ... *goto L1*
 (goto , a , , L1) ... *if a = true then goto L1*
 (lab , L1 , ,) ... *label L1:*

Syntaxí řízené generování 3AK

Základní přístupy:

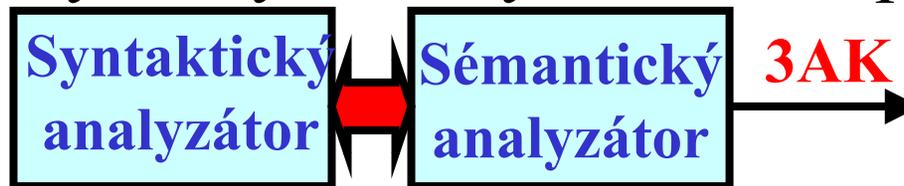
- 1) Syntaktický analyzátor vytvoří *abstraktní syntaktický strom (ASS)*, který je převeden na **3AK**.



- 2) Syntaktický analyzátor vytvoří *postfixovou reprezentaci* programu, která je převedena na **3AK**.



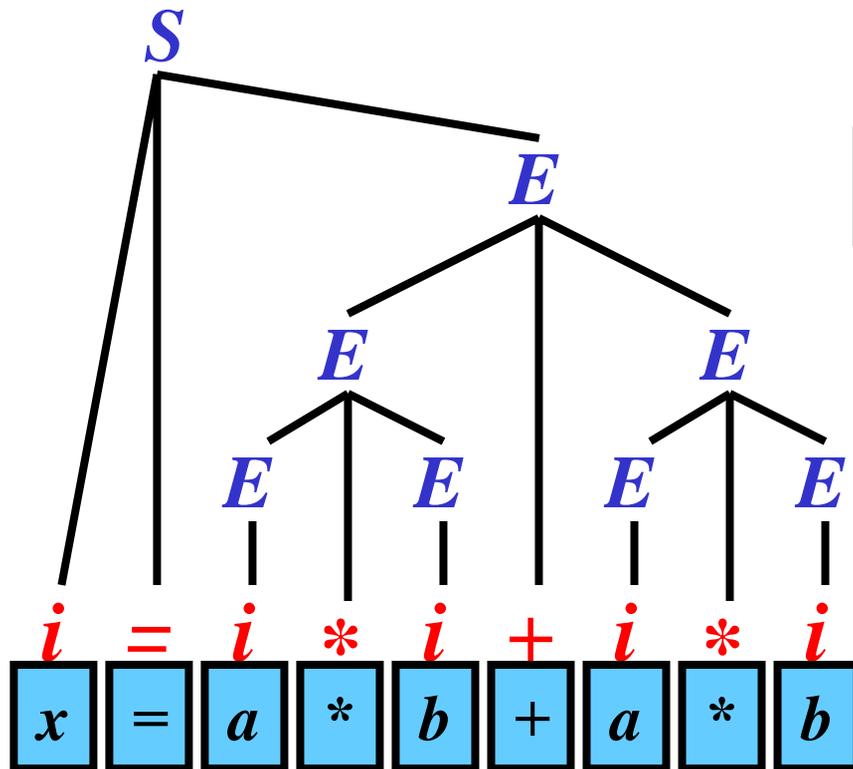
- 3) Syntaktický analyzátor vytvoří **3AK** přímo.



Z derivačního stromu k ASS: Příklad

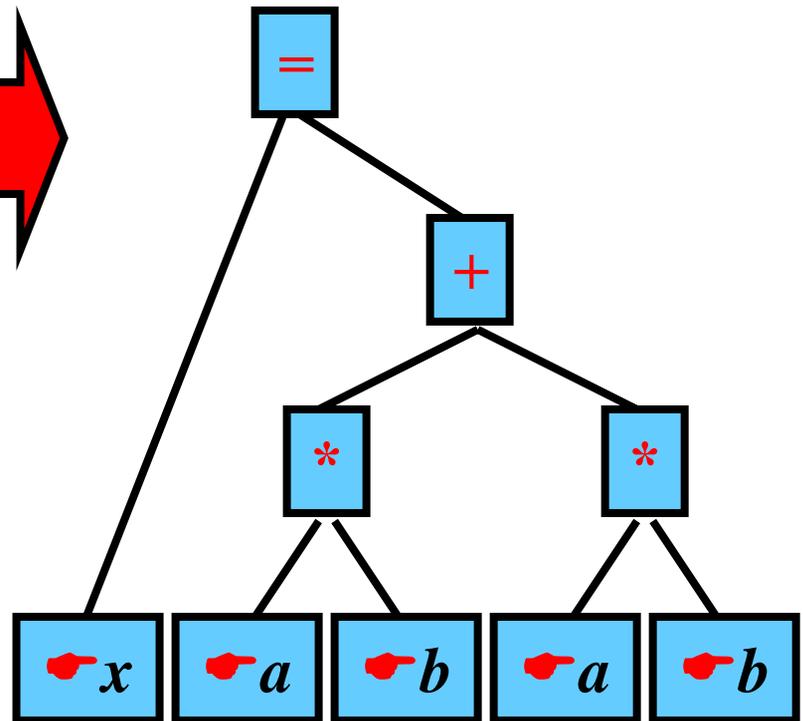
- derivační strom pro

$$x = a*b + a*b:$$



- ASS pro

$$x = a*b + a*b:$$



Generování ASS

Myšlenka: Syntaktický analyzátor simuluje vytváření derivačního stromu a současně volá sémantické akce, které vytvářejí AST.

Příklad:

Pravidlo:	Sémantická akce:
$S \rightarrow i := E_k$	{ $S.a := \text{MakeTree}('=', i.a, E_k.a)$ }
$E_i \rightarrow E_j + E_k$	{ $E_i.a := \text{MakeTree}('+', E_j.a, E_k.a)$ }
$E_i \rightarrow E_j * E_k$	{ $E_i.a := \text{MakeTree}('*', E_j.a, E_k.a)$ }
$E_i \rightarrow (E_j)$	{ $E_i.a := E_j.a$ }
$E_i \rightarrow i$	{ $E_i.a := \text{MakeLeaf}(i.a)$ }

Poznámky:

- **MakeTree(o , a , b)** vytvoří nový uzel o , naváže levého syna a , pravého syna b , a vrátí ukazatel na uzel o
- **MakeLeaf($i.a$)** vytvoří nový uzel $i.a$ ($i.a$ je adresa do tabulky symbolů) a vrátí ukazatel na tento uzel

Generování ASS: Příklad 1/2

Zásobník	Vstup	Pravidlo	Sémantická akce
\$	$i = (i + i) * i\$$		
\$i	$= (i + i) * i\$$		
\$i =	$(i + i) * i\$$		
\$i = ($i + i) * i\$$		
\$i = (i	$+ i) * i\$$	$E_1 \rightarrow i$	$E_1.a := \text{MakeLeaf}(i.a)$
\$i = (E_1	$+ i) * i\$$		
\$i = (E_1 +	$i) * i\$$		
\$i = (E_1 + i	$) * i\$$	$E_2 \rightarrow i$	$E_2.a := \text{MakeLeaf}(i.a)$
\$i = (E_1 + E_2	$) * i\$$	$E_3 \rightarrow E_1 + E_2$	$E_3.a := \text{MakeTree}('+', E_1.a, E_2.a)$
\$i = (E_3	$) * i\$$		
\$i = (E_3)	$* i\$$	$E_4 \rightarrow (E_3)$	$E_4.a := E_3.a$
\$i = E_4	$* i\$$		
\$i = E_4 *	$i\$$		
\$i = E_4 * i	$\$$	$E_5 \rightarrow i$	$E_5.a := \text{MakeLeaf}(i.a)$
\$i = E_4 * E_5	$\$$	$E_6 \rightarrow E_4 * E_5$	$E_6.a := \text{MakeTree}('*', E_4.a, E_5.a)$
\$i = E_6	$\$$	$S \rightarrow i = E_6$	$S.a := \text{MakeTree}('=', i.a, E_6.a)$
\$\$	$\$$		

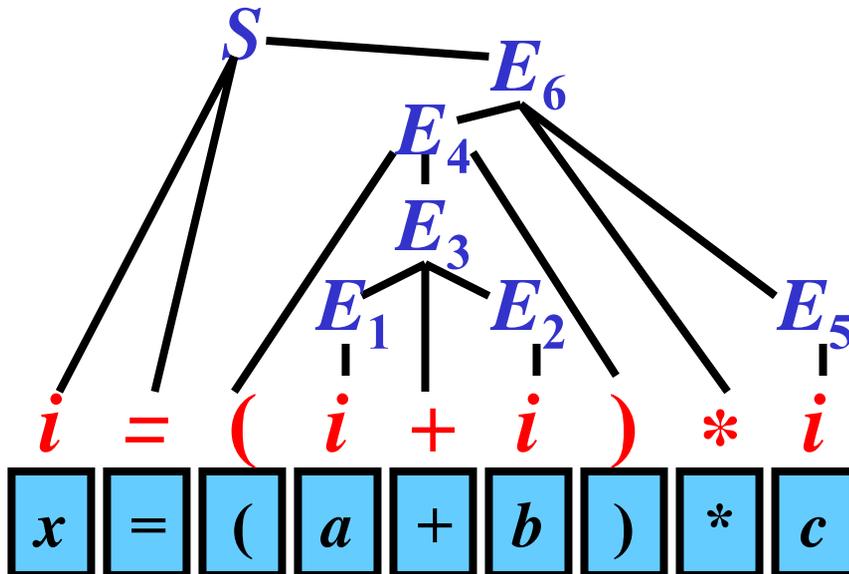
Syntaktická analýza zdola nahoru

Sémantické akce

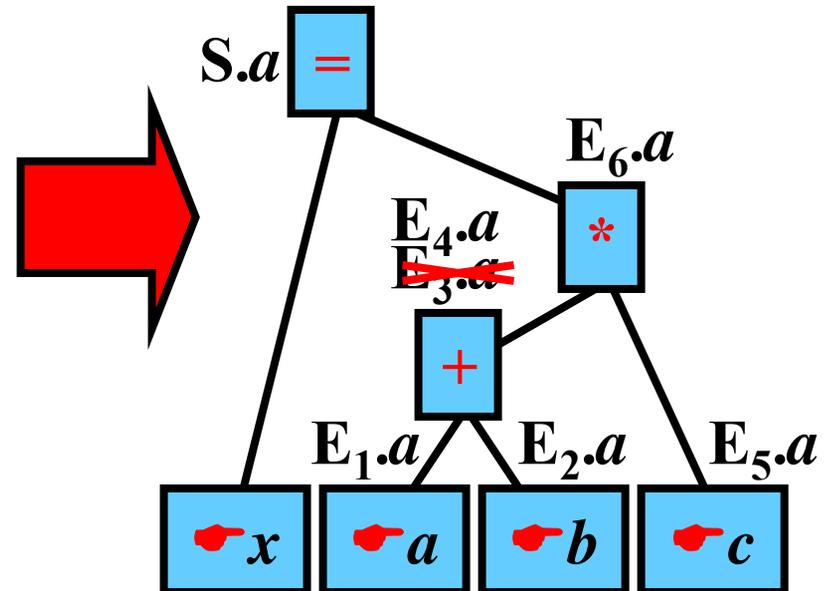
Generování ASS: Příklad 1/2

Pravidlo:	Sémantická akce:
$E_1 \rightarrow i$	$E_1.a := \text{MakeLeaf}(i.a)$
$E_2 \rightarrow i$	$E_2.a := \text{MakeLeaf}(i.a)$
$E_3 \rightarrow E_1 + E_2$	$E_3.a := \text{MakeTree}('+', E_1.a, E_2.a)$
$E_4 \rightarrow (E_3)$	$E_4.a := E_3.a$
$E_5 \rightarrow i$	$E_5.a := \text{MakeLeaf}(i.a)$
$E_6 \rightarrow E_4 * E_5$	$E_6.a := \text{MakeTree}('*', E_4.a, E_5.a)$
$S \rightarrow i = E_6$	$S.a := \text{MakeTree}('=', i.a, E_6.a)$

Simulace Derivačního stromu:



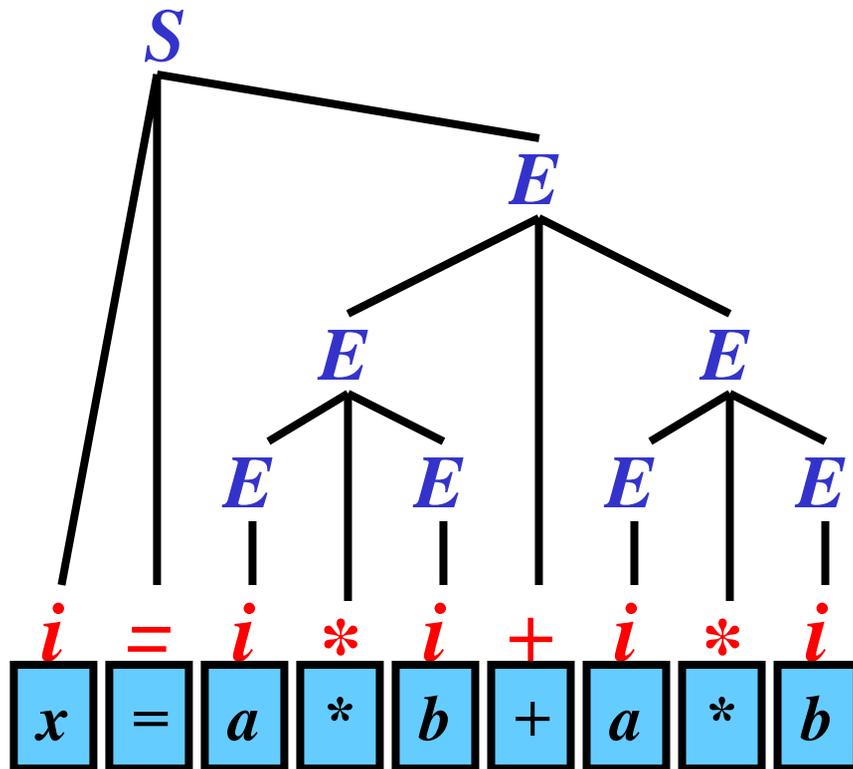
Abstraktní syntaktický strom:



DAG: Příklad

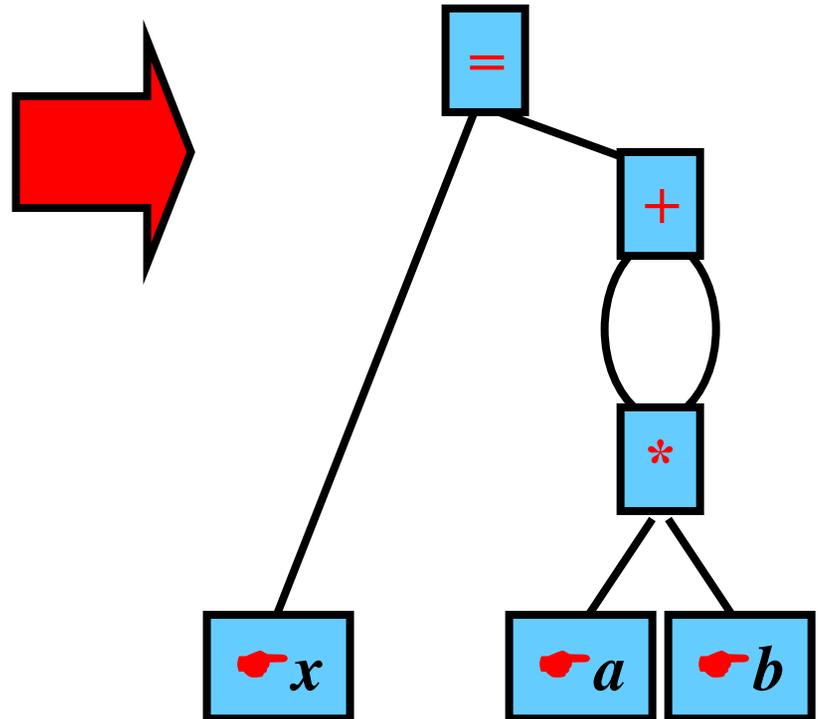
- Parse tree pro

$$x = a*b + a*b:$$



- DAG pro

$$x = a*b + a*b:$$



Pozn.: DAG nemá nadbytečné uzly.

Postfixová Notace

Myšlenka: Každý operátor se vyskytuje až za operandy

Příklad:

Infixová notace	Postfixová notace
$a + b$	$a b +$
$a = b$	$a b =$
if C then S_1 else S_2	$C S_1 S_2$ if-then-else

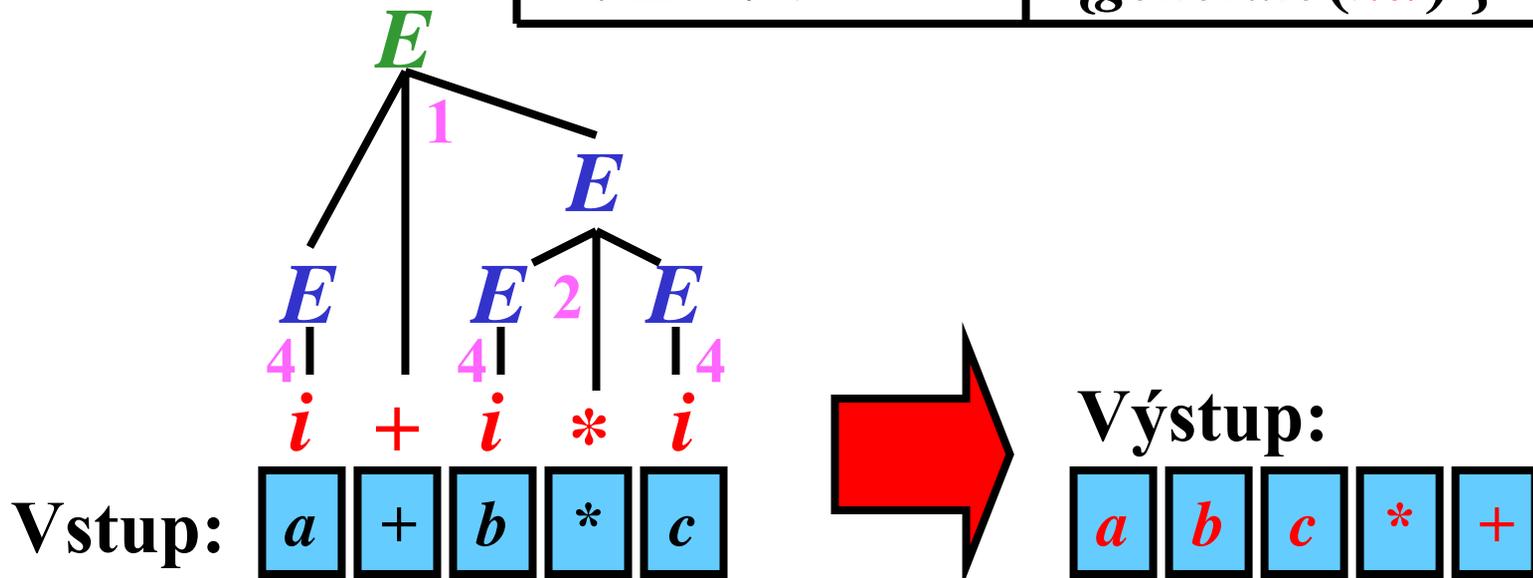
Pozn.: Postfixovou notaci můžeme také získat průchodem postorder ASS.

Z infixu do postfixu pomocí SA zdola nahoru

Myšlenka: Sémantická akce vytvářejí postfixovou verzi zdrojového programu

Příklad:

Pravidlo:	Sémantická akce:
1: $E \rightarrow E+E$	{generate('+')}
2: $E \rightarrow E * E$	{generate('*')}
3: $E \rightarrow (E)$	{ - }
4: $E \rightarrow i$	{generate(<i>i.a</i>) }



Překladové gramatiky

Myšlenka: Překladové gramatiky překládají vstupní řetězec na výstupní řetězec

1) Překlad pomocí dvou gramatik:



2) Překlad řízený jednou gramatikou

SA založený na gramatikách se dvěma pravými stranami

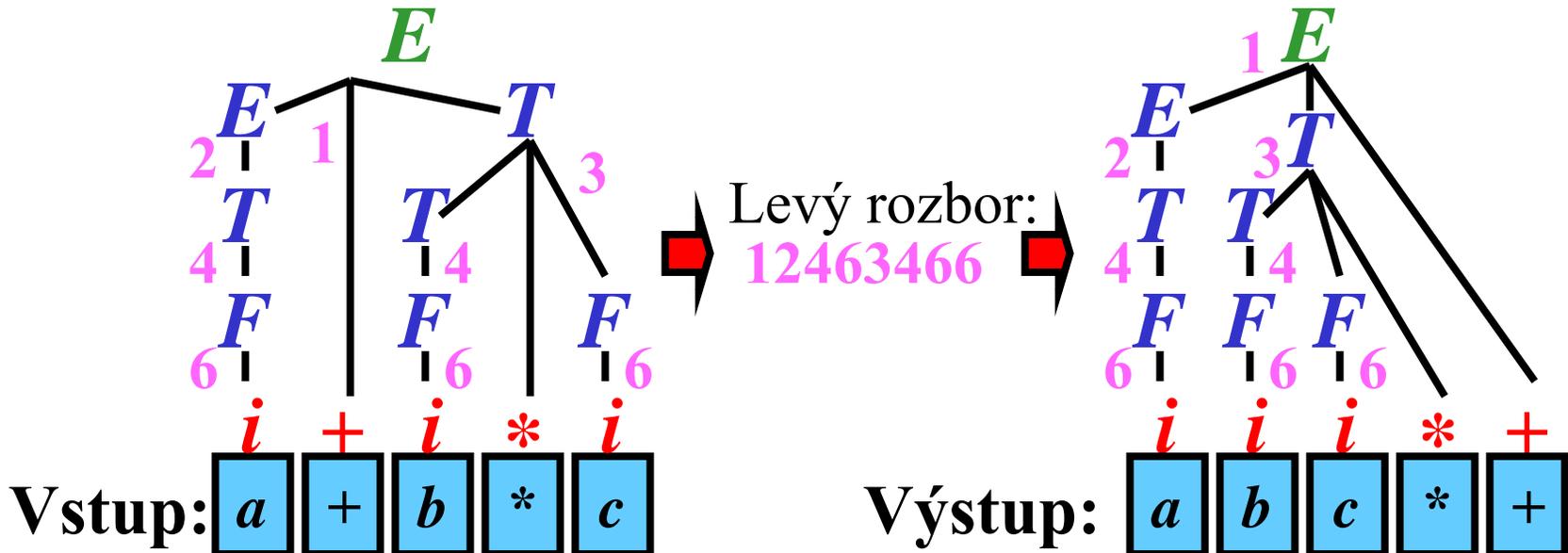


Pozn.: V průběhu syntaktické analýzy vstupního řetězce je současně vytvářen výstupní řetězec

Překlad pomocí dvou grammatik

Překlad z infixu
do postfixu:

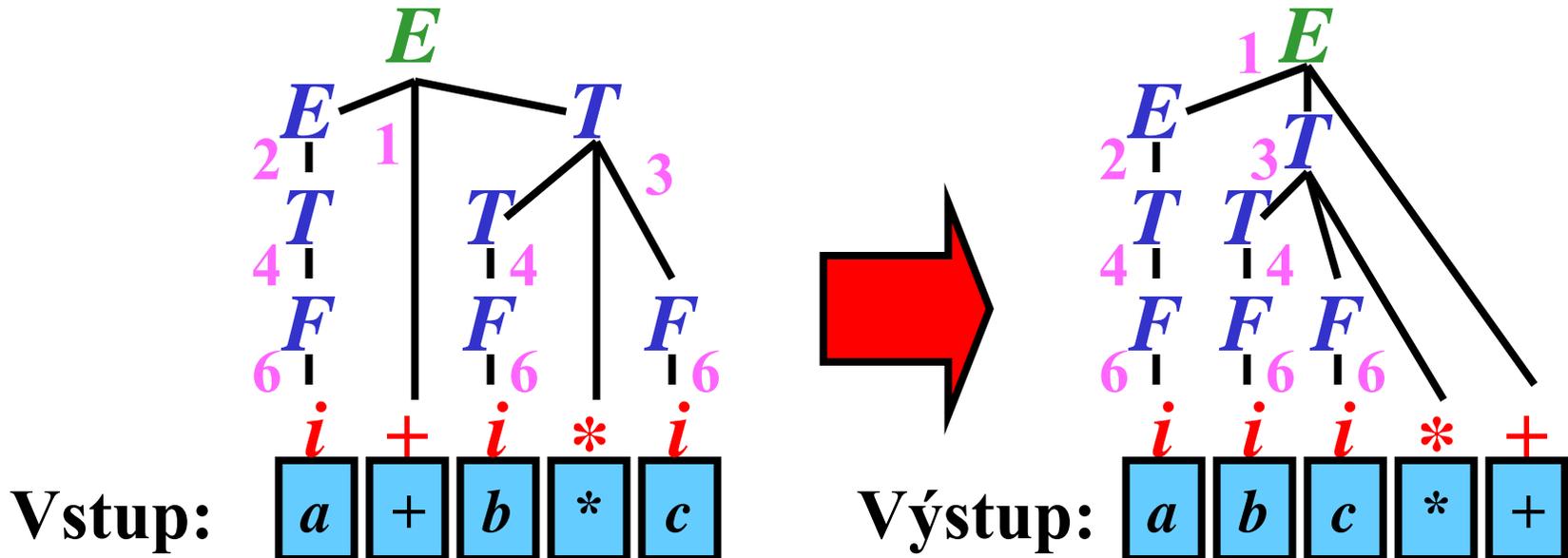
Pravidla G_1	Pravidla G_2
1: $E \rightarrow E+T$	1: $E \rightarrow ET+$
2: $E \rightarrow T$	2: $E \rightarrow T$
3: $T \rightarrow T*F$	3: $T \rightarrow TF*$
4: $T \rightarrow F$	4: $T \rightarrow F$
5: $F \rightarrow (E)$	5: $F \rightarrow E$
6: $F \rightarrow i$	6: $F \rightarrow i$



Překlad pomocí jedné gramatiky

Infix to postfix translation:

Pravidlo	Překl. element
1: $E \rightarrow E+T$	$ET+$
2: $E \rightarrow T$	T
3: $T \rightarrow T*F$	$TF*$
4: $T \rightarrow F$	F
5: $F \rightarrow (E)$	E
6: $F \rightarrow i$	i

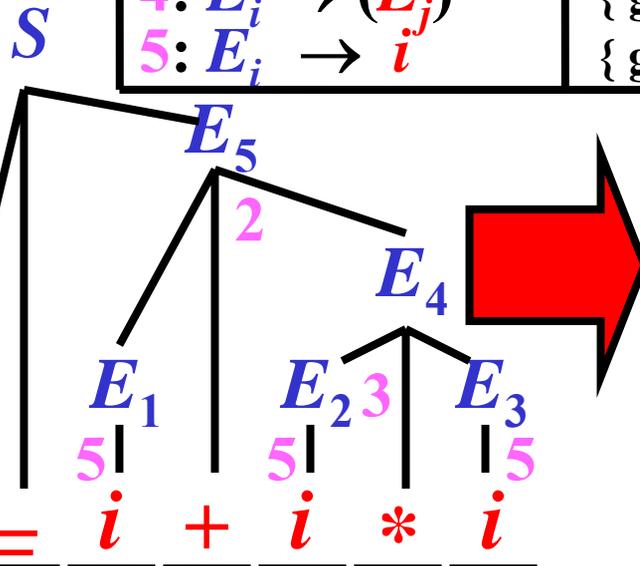


Přímé generování 3AK

Myšlenka: SA pracující metodou zdola nahoru generuje 3AK přímo

Příklad:

Pravidlo:	Sémantická akce:
1: $S \rightarrow i = E_k$	{ generate('=', $E_k.loc$, , $i.loc$) }
2: $E_i \rightarrow E_j + E_k$	{ generate('+', $E_j.loc$, $E_k.loc$, $E_i.loc$) }
3: $E_i \rightarrow E_j * E_k$	{ generate('*', $E_j.loc$, $E_k.loc$, $E_i.loc$) }
4: $E_i \rightarrow (E_j)$	{ generate('=', $E_j.loc$, , $E_i.loc$) }
5: $E_i \rightarrow i$	{ generate('=', $i.loc$, , $E_i.loc$) }



Výstup:

```
(='=',  $\leftarrow a$ , ,  $E_1.loc$ )
(='=',  $\leftarrow b$ , ,  $E_2.loc$ )
(='=',  $\leftarrow c$ , ,  $E_3.loc$ )
('*',  $E_2.loc$ ,  $E_3.loc$ ,  $E_4.loc$ )
(+,  $E_1.loc$ ,  $E_4.loc$ ,  $E_5.loc$ )
(='=',  $E_5.loc$ , ,  $\leftarrow x$ )
```

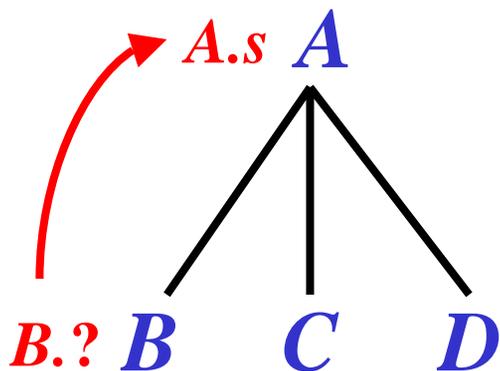
Vstup:

$x = a + b * c$

Překlad shora dolů: Úvod

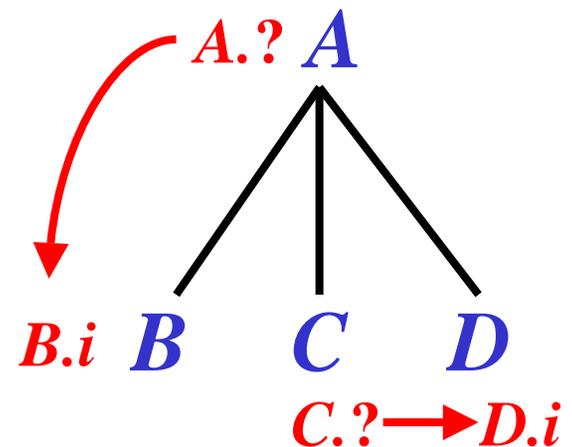
- LL-gramatiky s atributy
- Dva zásobníky:
 - **pro synt. analýzu** × **pro sémant. analýzu**
- Dva typy atributů:

• **syntetizované:**
(z dítěte na rodiče)



• **dědičné:**

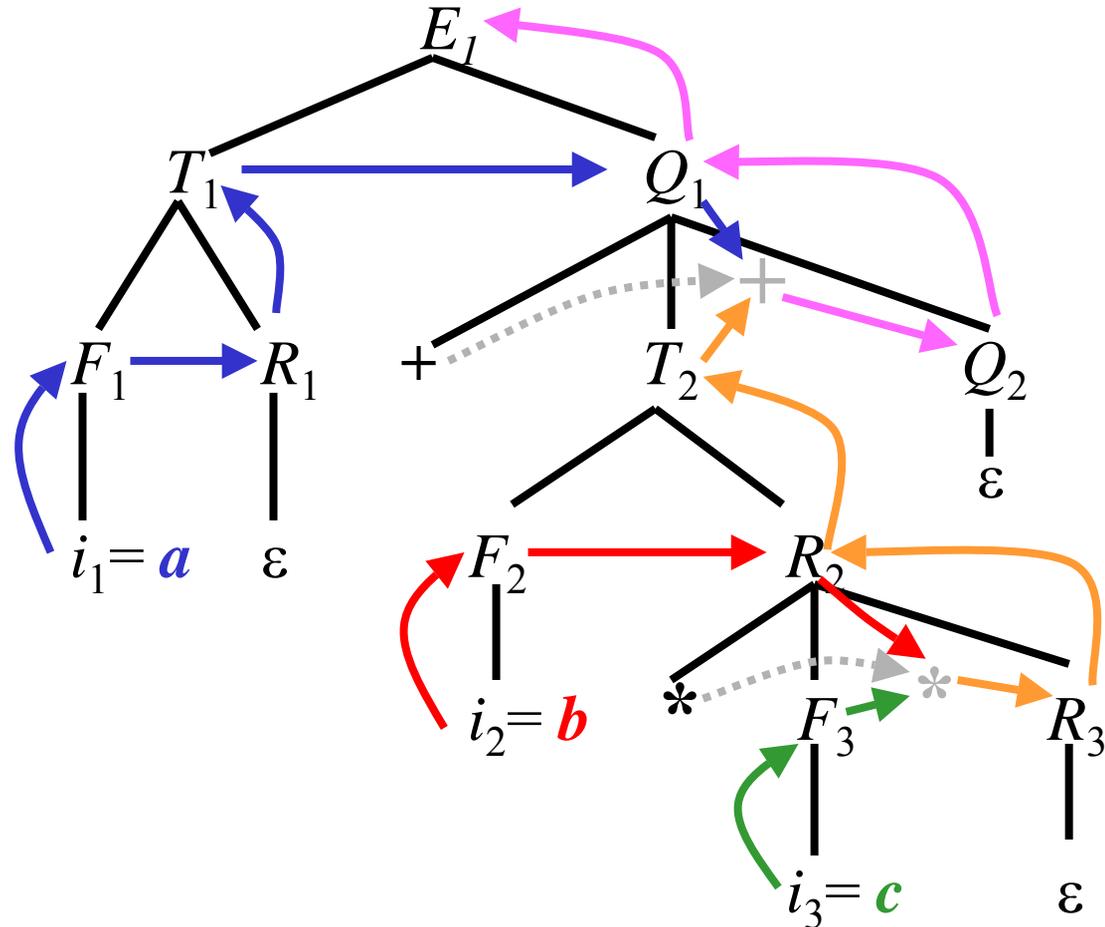
(z rodiče na děti nebo mezi sourozenci)



Překlad shora dolů: Aritmetické výrazy

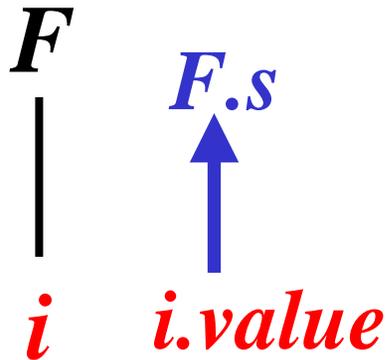
Gramatika: **Derivační strom pro $a + b * c$:**

$E \rightarrow TQ$
 $Q \rightarrow +TQ$
 $Q \rightarrow \varepsilon$
 $T \rightarrow FR$
 $R \rightarrow *FR$
 $R \rightarrow \varepsilon$
 $F \rightarrow (E)$
 $F \rightarrow i$

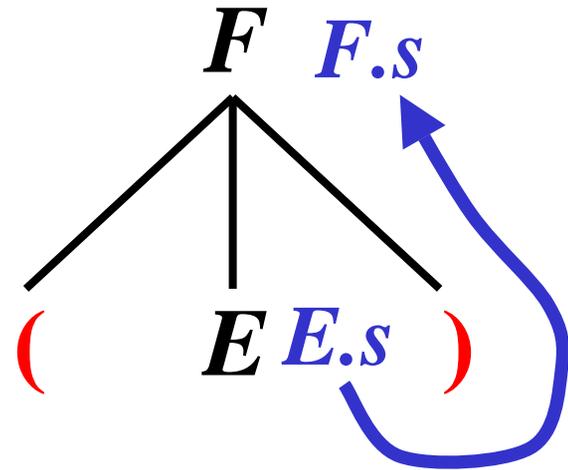


Výrazy: Proměnné a závorky

Proměná:



Závorky:

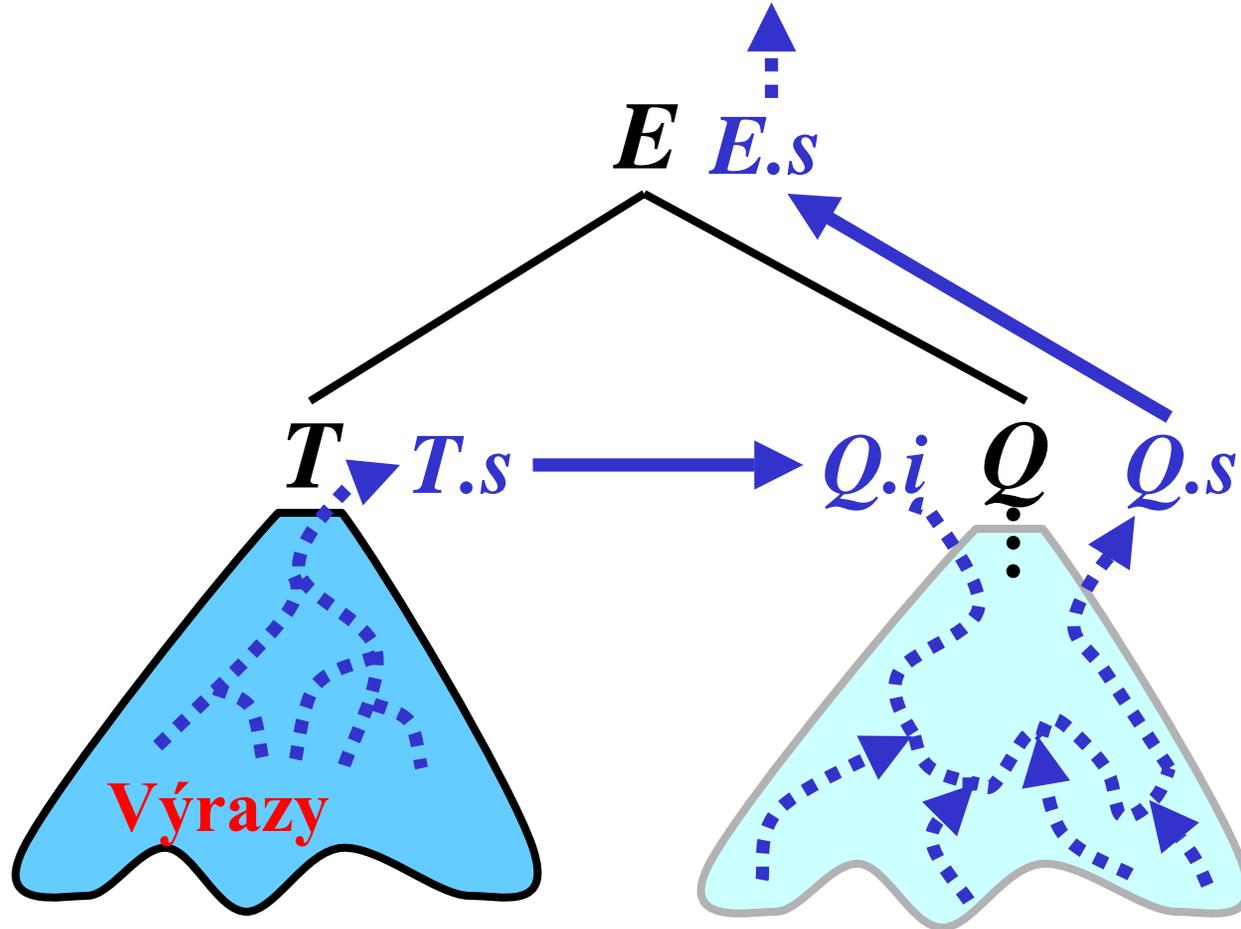


$$F \rightarrow i \{F.s := i.value\}$$

$$E \rightarrow (F \{F.s := E.s\})$$

Výrazy: Sčítání 1/4

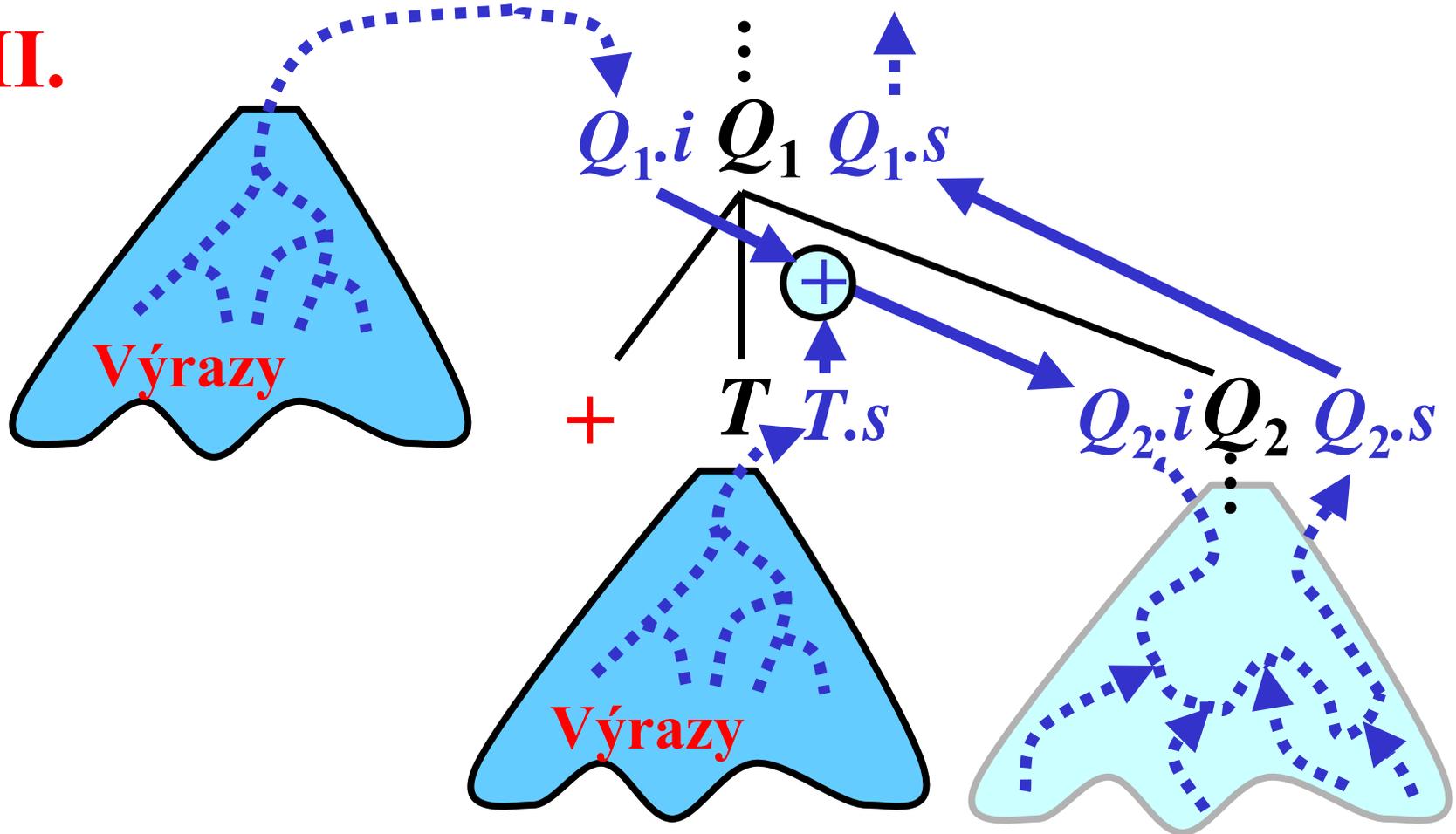
I.



$$E \rightarrow T \{ Q.i := T.s \} Q \{ E.s := Q.s \}$$

Výrazy: Sčítání 2/4

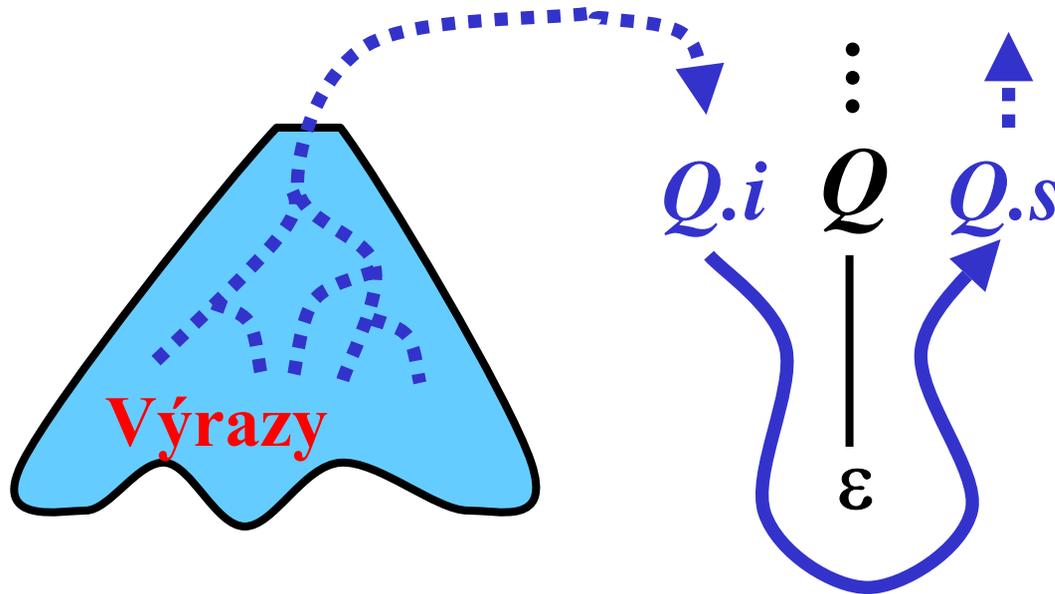
II.



$$Q_1 \rightarrow +T \{ Q_{2.i} := Q_{1.i} + T.s \} Q_2 \{ Q_{1.s} := Q_{2.s} \}$$

Výrazy: Sčítání 3/4

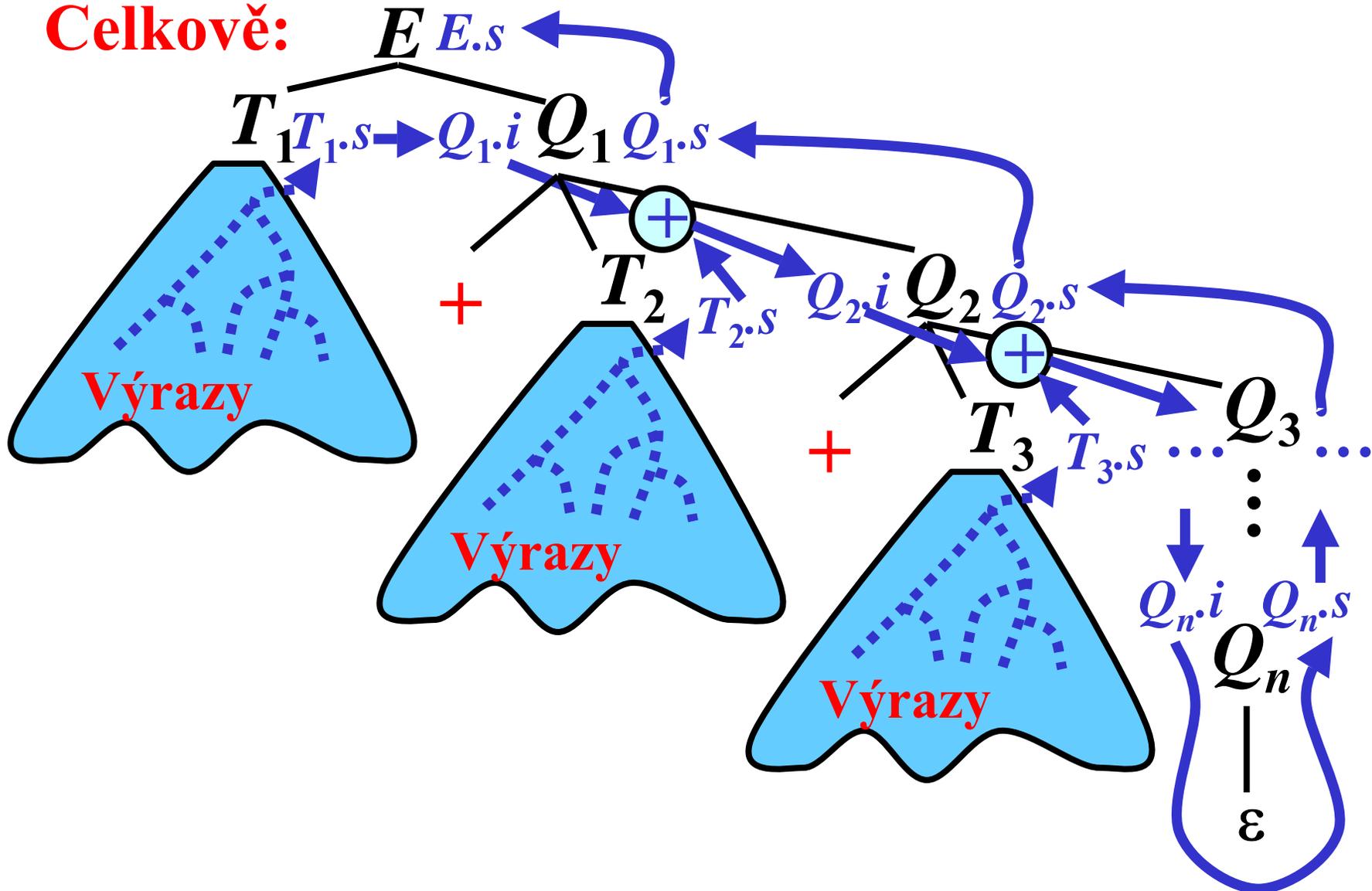
III.



$$Q \rightarrow \varepsilon \quad \{Q.s := Q.i\}$$

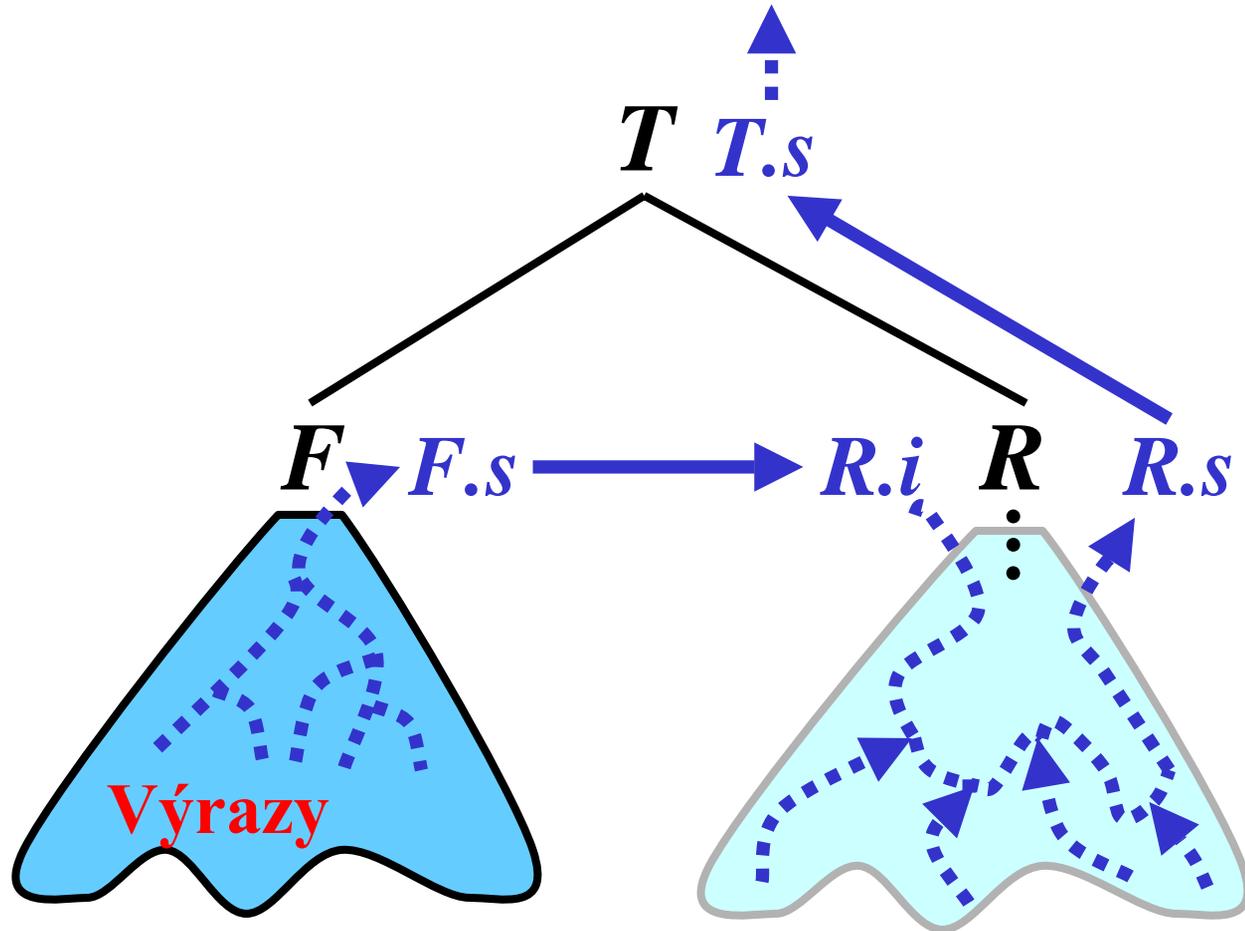
Výrazy: Sčítání 4/4

Celkově:



Výrazy: Násobení 1/4

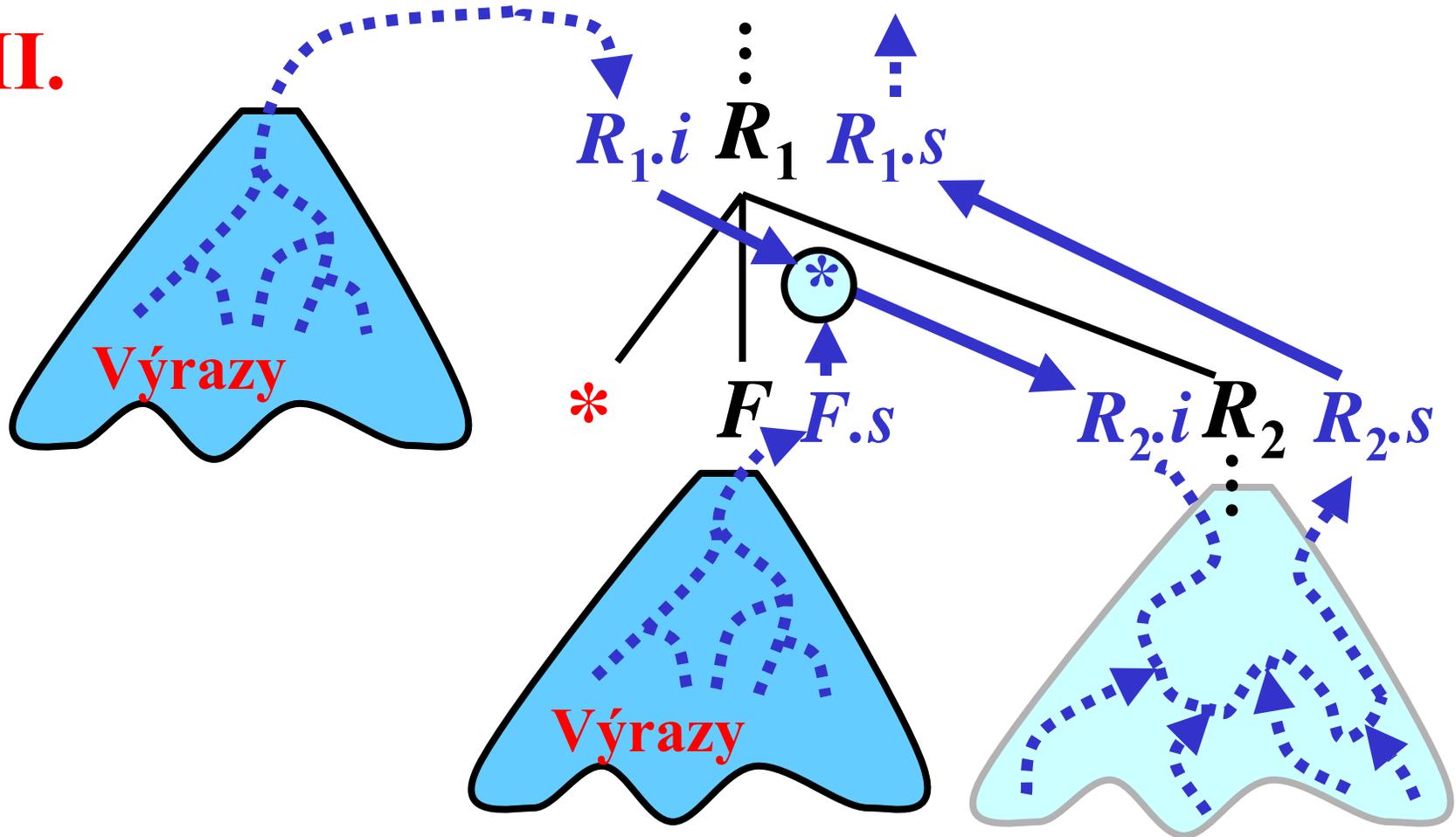
I.



$$T \rightarrow F \{ R.i := F.s \} R \{ T.s := R.s \}$$

Výrazy: Násobení 2/4

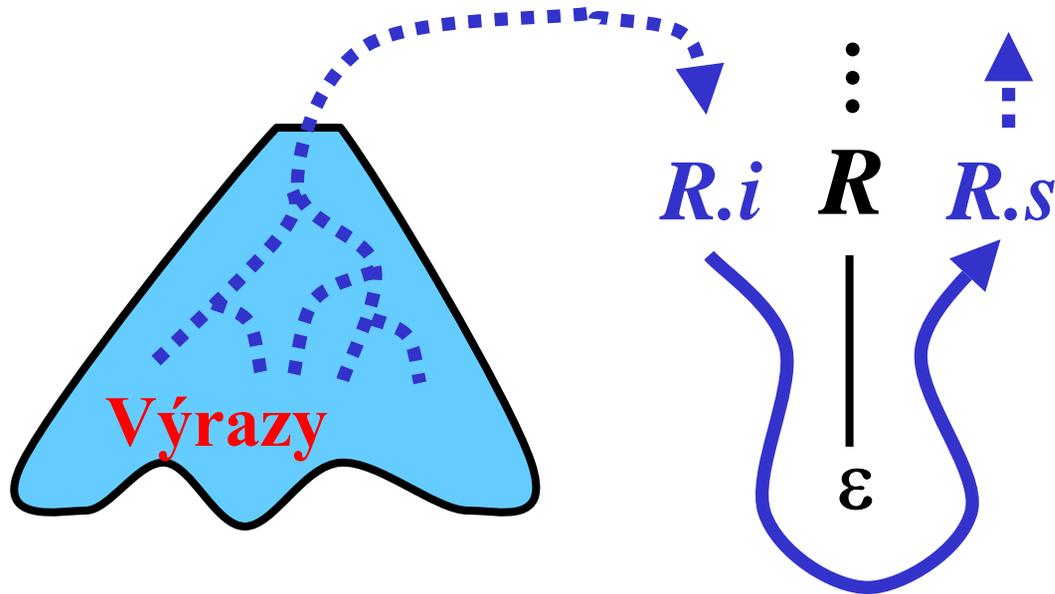
II.



$$R_1 \rightarrow *F \{ R_{2.i} := R_{1.i} * F.s \} Q_2 \{ R_{1.s} := R_{2.s} \}$$

Výrazy: Násobení 3/4

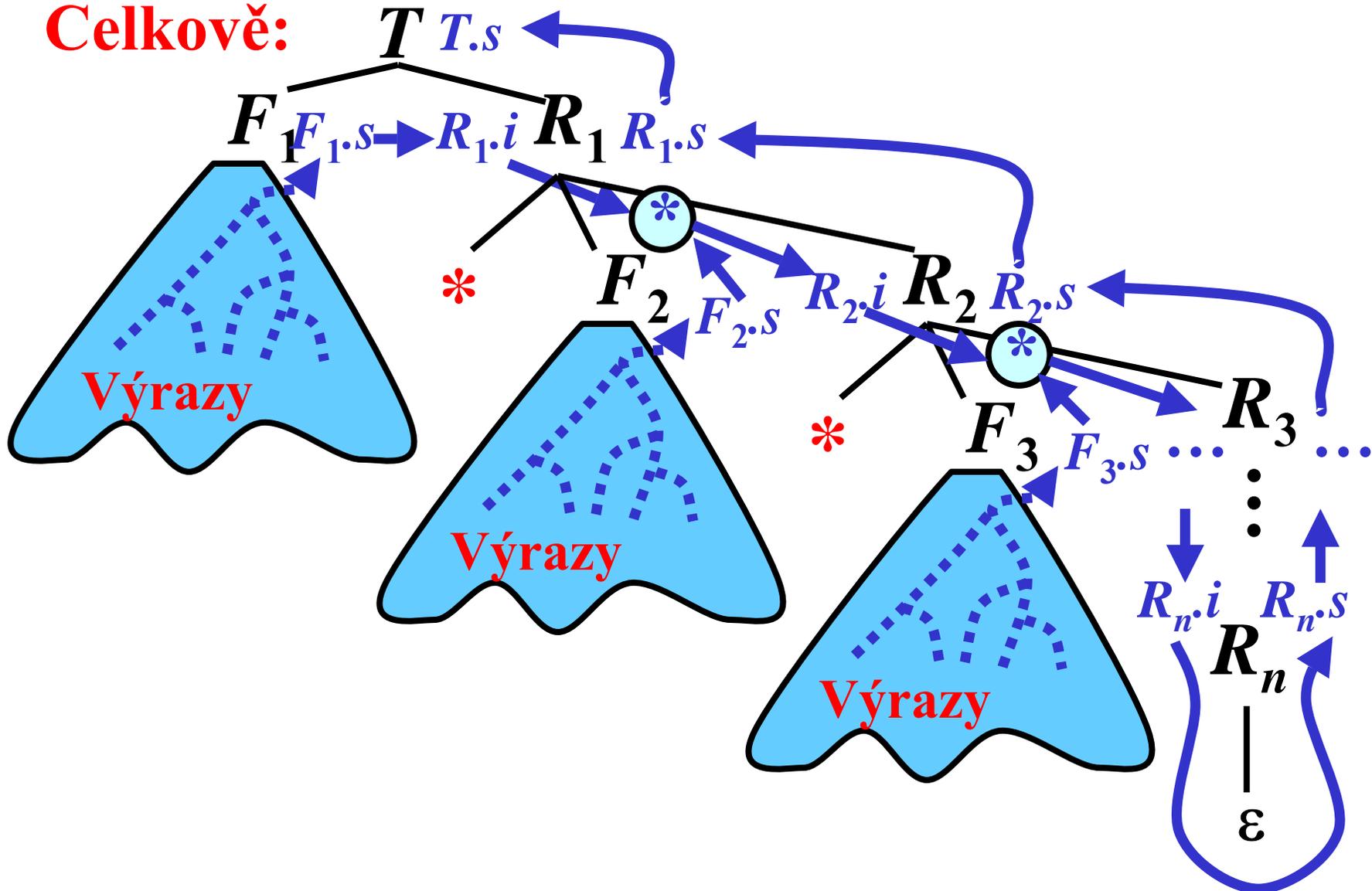
III.



$$R \rightarrow \varepsilon \quad \{R.s := R.i\}$$

Výrazy: Násobení 4/4

Celkově:



Gramatika pro výrazy: Celkově

1. $E \rightarrow T \{Q.i := T.s\} Q \{E.s := Q.s\}$
2. $Q_1 \rightarrow +T \{Q_2.i := Q_1.i + T.s\} Q_2 \{Q_1.s := Q_2.s\}$
3. $Q \rightarrow \varepsilon \{Q.s := Q.i\}$
4. $T \rightarrow F \{R.i := F.s\} R \{T.s := R.s\}$
5. $R_1 \rightarrow *F \{R_2.i := R_1.i * F.s\} R_2 \{R_1.s := R_2.s\}$
6. $R \rightarrow \varepsilon \{R.s := R.i\}$
7. $F \rightarrow (E \{F.s := E.s\})$
8. $F \rightarrow i \{F.s := i.value\}$

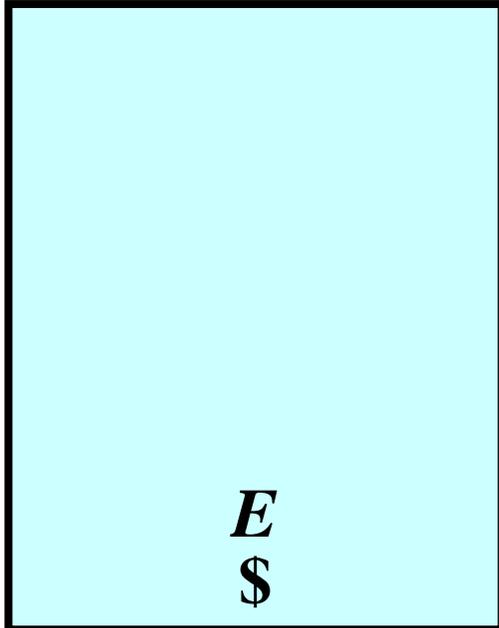
Vyhodnocení výrazů: Příklad 1/16

Příklad pro $a + b$, kde $a.value = 10$, $b.value = 20$

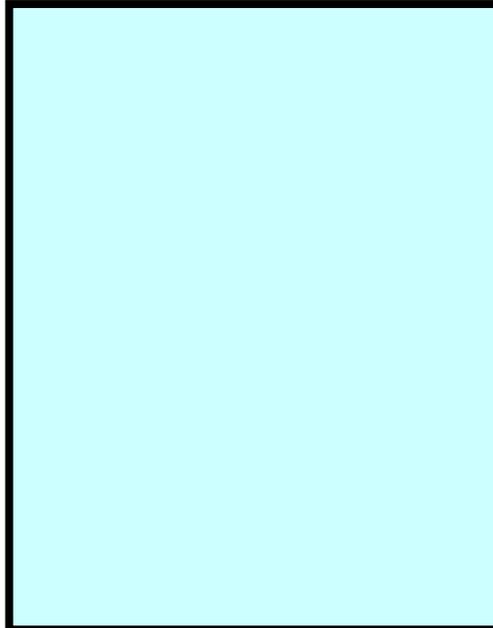
Vstup: $i_1 + i_2 \$$

Prav.: $E \rightarrow T_1 \{Q_1.i := T_1.s\} Q_1 \{E.s := Q_1.s\}$

Zásobník synt. an.:

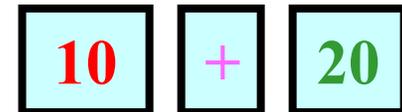


Zásobník sém. an.:



Ilustrace:

E



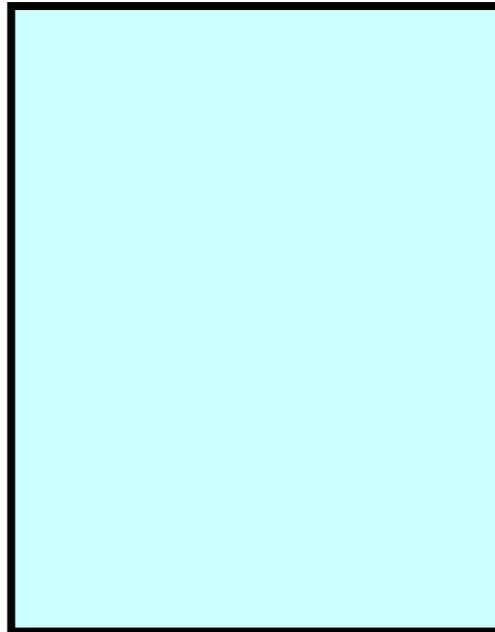
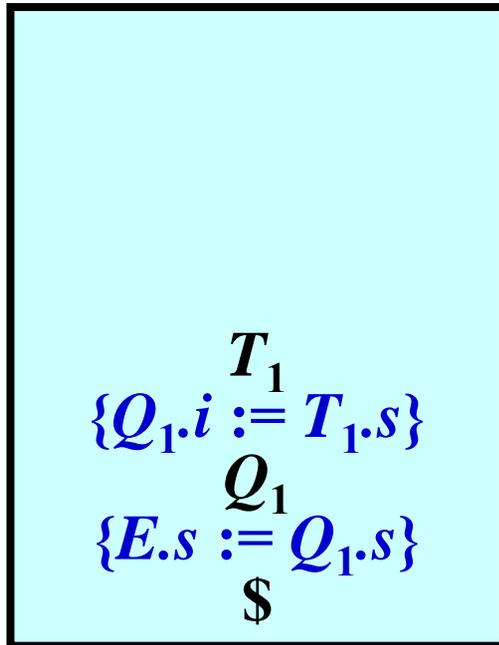
Vyhodnocení výrazů: Příklad 2/16

Příklad pro $a + b$, kde $a.value = 10$, $b.value = 20$

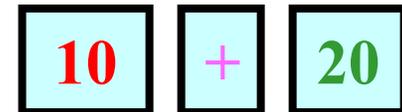
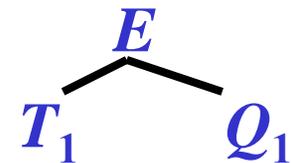
Vstup: $i_1 + i_2 \$$

Prav.: $T_1 \rightarrow F_1 \{R_1.i := F_1.s\} R_1 \{T_1.s := R_1.s\}$

Zásobník synt. an.: Zásobník sém. an.:



Ilustrace:



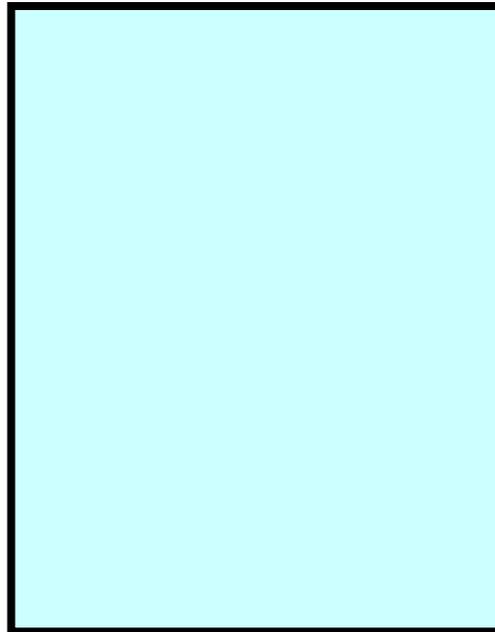
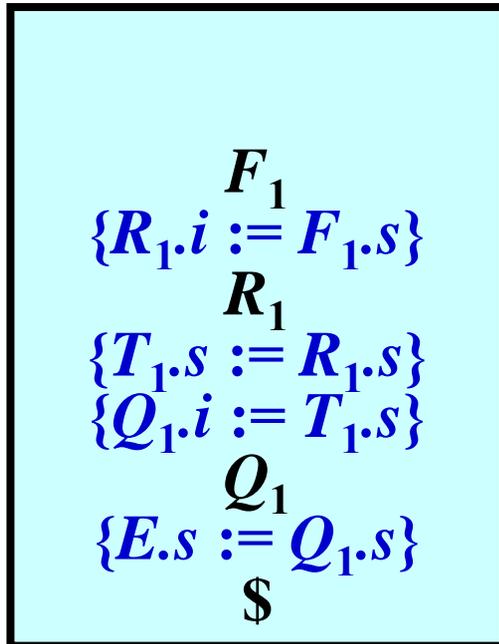
Vyhodnocení výrazů: Příklad 3/16

Příklad pro $a + b$, kde $a.value = 10$, $b.value = 20$

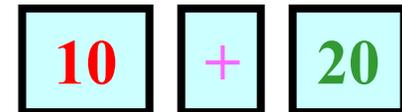
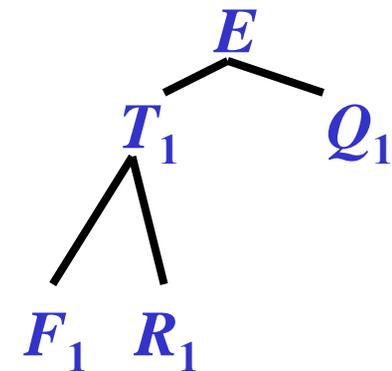
Vstup: $i_1 + i_2 \$$

Prav.: $F_1 \rightarrow i_1 \{F_{1.s} := i.value\}$

Zásobník synt. an.: Zásobník sém. an.:



Ilustrace:



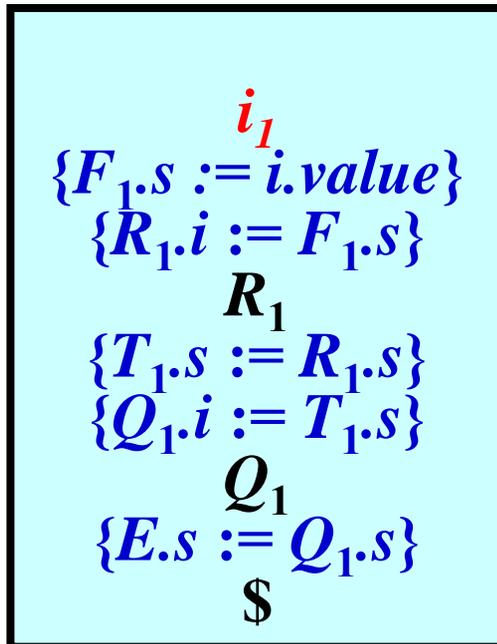
Vyhodnocení výrazů: Příklad 4/16

Příklad pro $a + b$, kde $a.value = 10$, $b.value = 20$

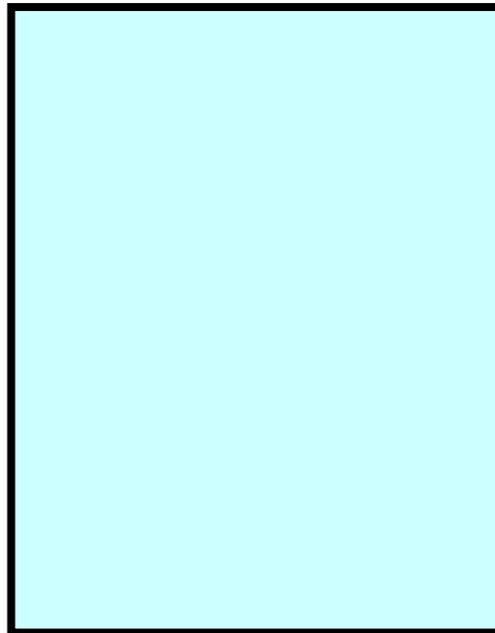
Vstup: $i_1 + i_2 \$$

Prav.:

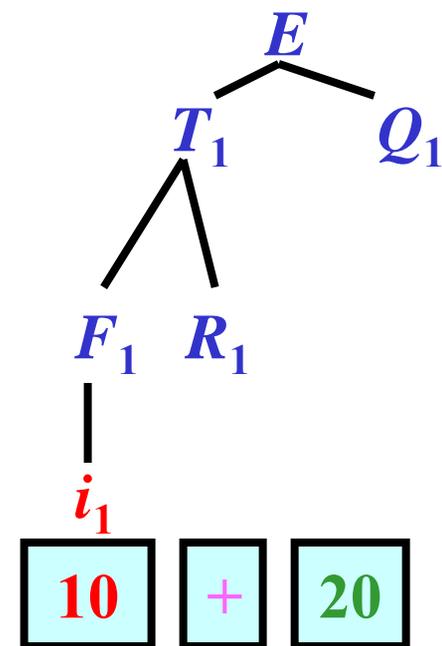
Zásobník synt. an.:



Zásobník sém. an.:



Ilustrace:



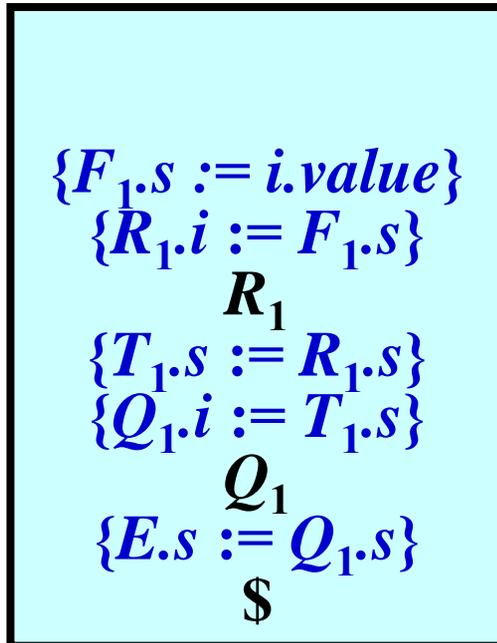
Vyhodnocení výrazů: Příklad 5/16

Příklad pro $a + b$, kde $a.value = 10$, $b.value = 20$

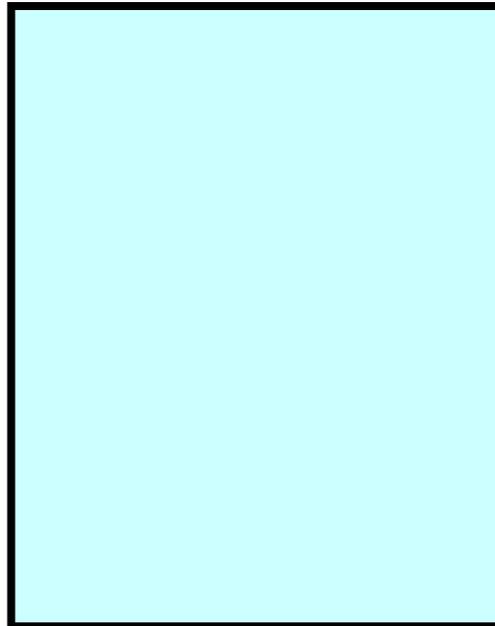
Vstup: $+ i_2 \$$

Prav.:

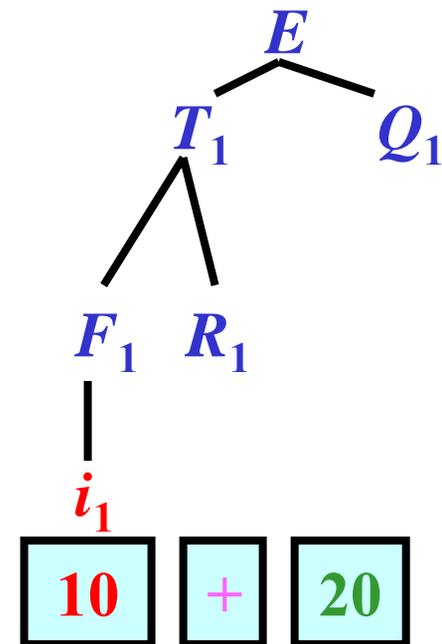
Zásobník synt. an.:



Zásobník sém. an.:



Ilustrace:



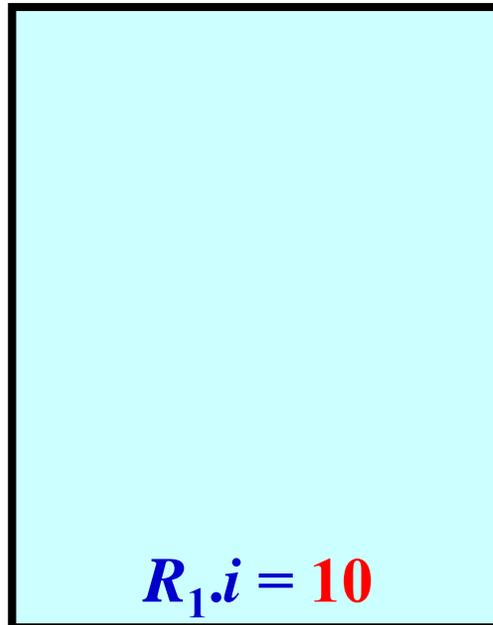
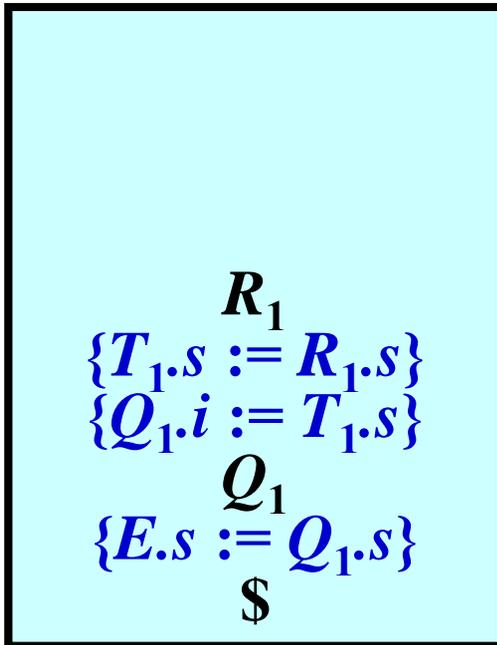
Vyhodnocení výrazů: Příklad 6/16

Příklad pro $a + b$, kde $a.value = 10$, $b.value = 20$

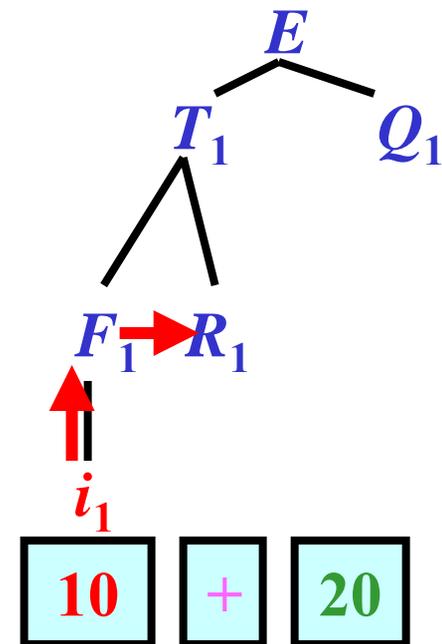
Vstup: $+ i_2 \$$

Prav.: $R_1 \rightarrow \varepsilon \{R_1.s := R_1.i\}$

Zásobník synt. an.: Zásobník sém. an.:



Ilustrace:



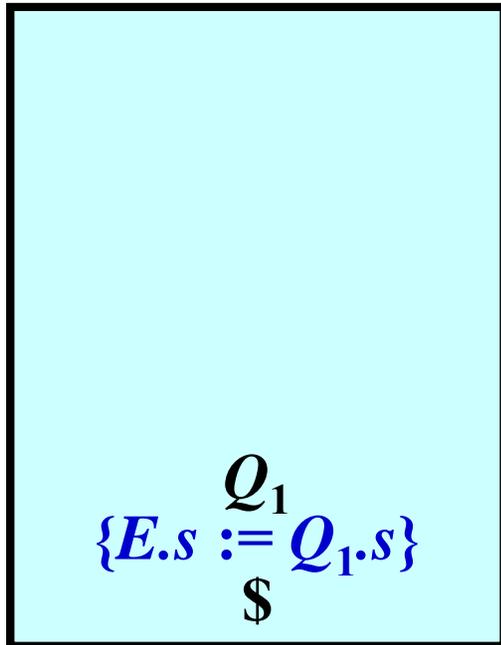
Vyhodnocení výrazů: Příklad 7/16

Příklad pro $a + b$, kde $a.value = 10$, $b.value = 20$

Vstup: $+ i_2 \$$

Prav.: $Q_1 \rightarrow +T_2 \{Q_2.i := Q_1.i + T_2.s\} Q_2 \{Q_1.s := Q_2.s\}$

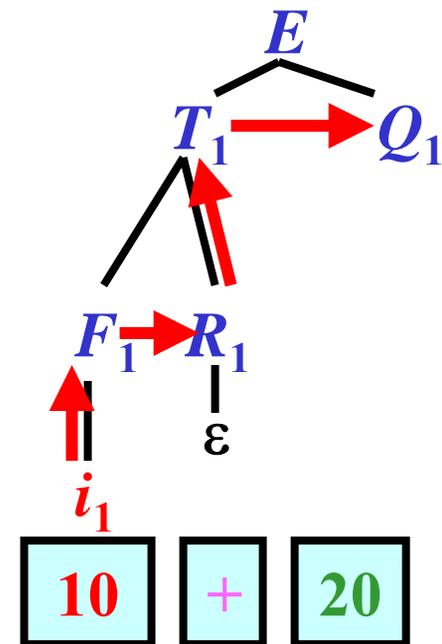
Zásobník synt. an.:



Zásobník sém. an.:



Ilustrace:



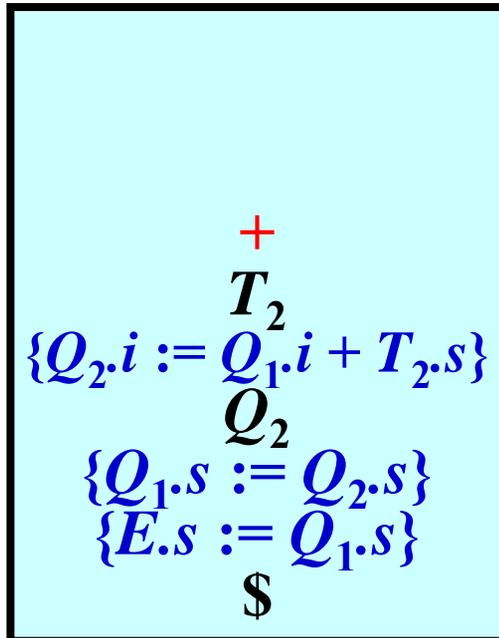
Vyhodnocení výrazů: Příklad 8/16

Příklad pro $a + b$, kde $a.value = 10$, $b.value = 20$

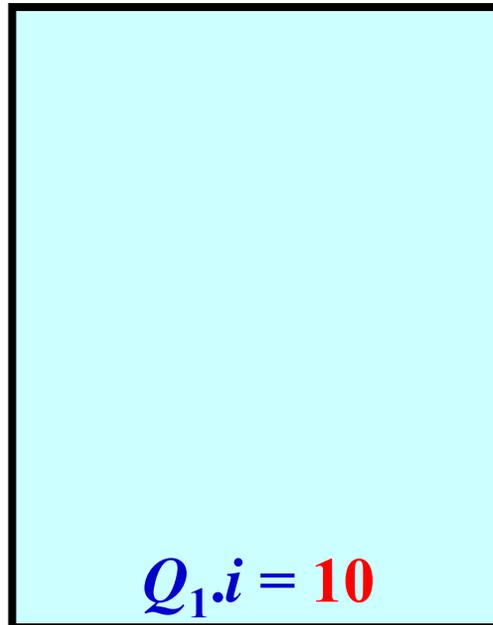
Vstup: $+ i_2 \$$

Prav.:

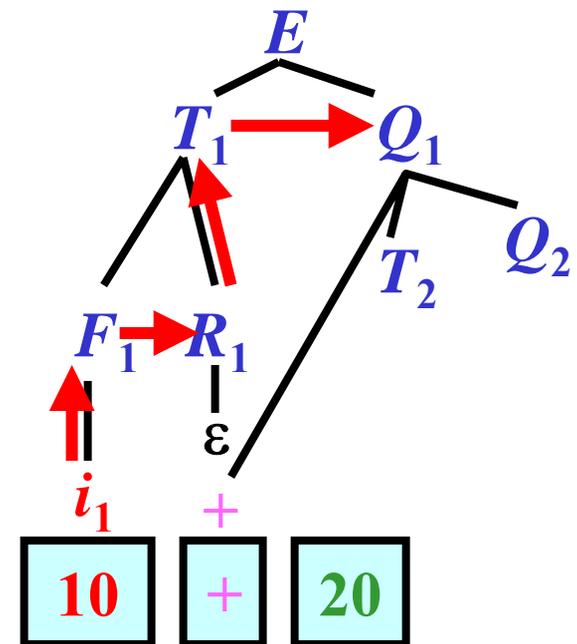
Zásobník synt. an.:



Zásobník sém. an.:



Ilustrace:



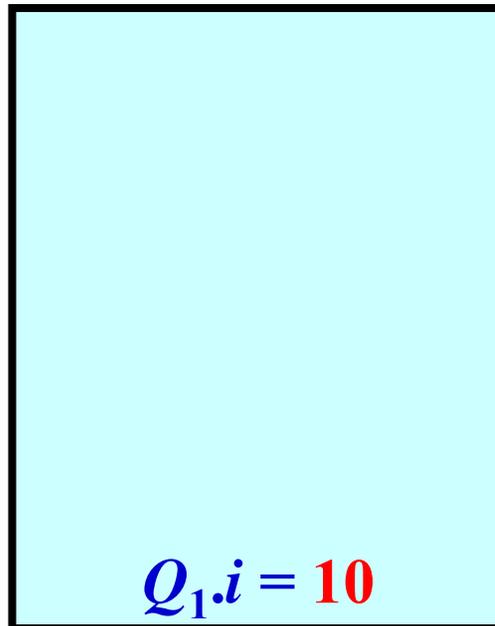
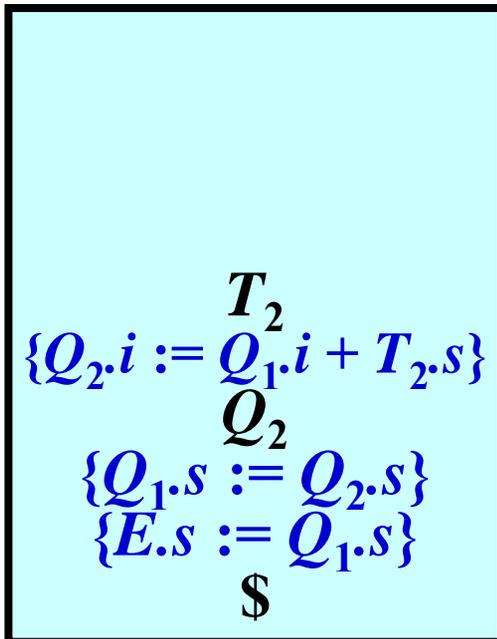
Vyhodnocení výrazů: Příklad 9/16

Příklad pro $a + b$, kde $a.value = 10$, $b.value = 20$

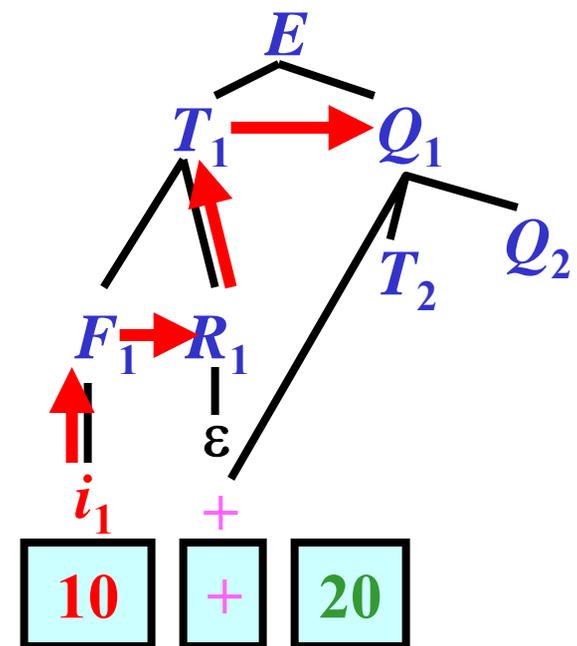
Vstup: i_2 \$

Prav.: $T_2 \rightarrow F_2 \{R_2.i := F_2.s\} R_2 \{T_2.s := R_2.s\}$

Zásobník synt. an.: Zásobník sém. an.:



Ilustrace:



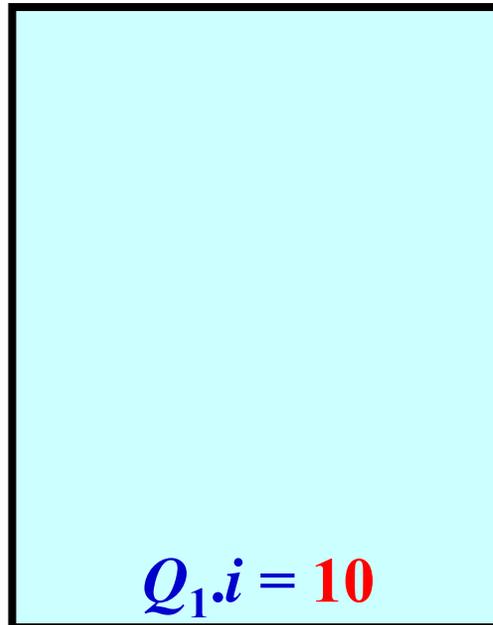
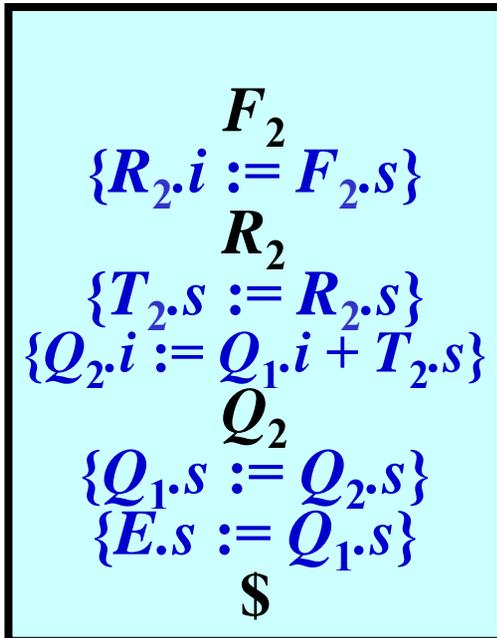
Vyhodnocení výrazů: Příklad 10/16

Příklad pro $a + b$, kde $a.value = 10$, $b.value = 20$

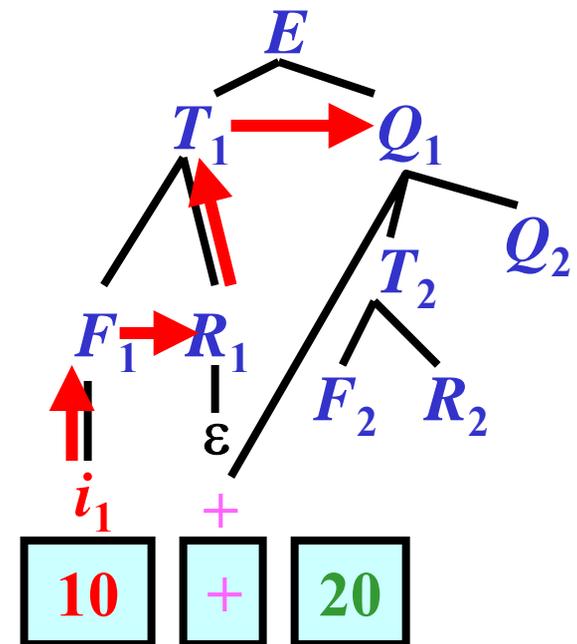
Vstup: $i_2 \$$

Prav.: $F_2 \rightarrow i_2 \{F_2.s := i.value\}$

Zásobník synt. an.: Zásobník sém. an.:



Ilustrace:



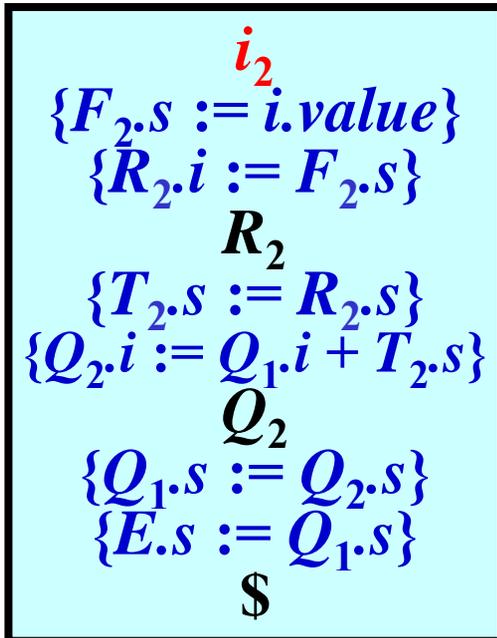
Vyhodnocení výrazů: Příklad 11/16

Příklad pro $a + b$, kde $a.value = 10$, $b.value = 20$

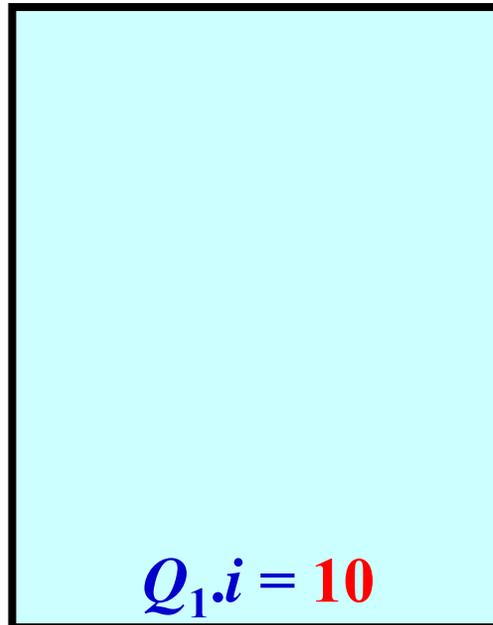
Vstup: $i_2 \$$

Prav.:

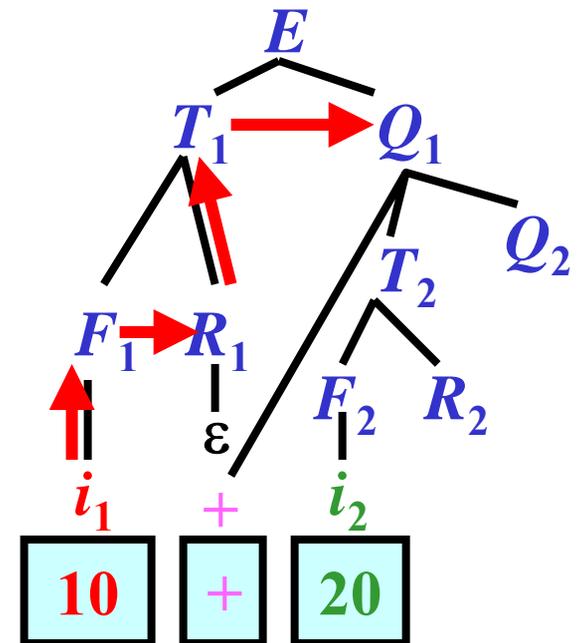
Zásobník synt. an.:



Zásobník sém. an.:



Ilustrace:



Vyhodnocení výrazů: Příklad 12/16

Příklad pro $a + b$, kde $a.value = 10$, $b.value = 20$

Vstup: \$

Prav.:

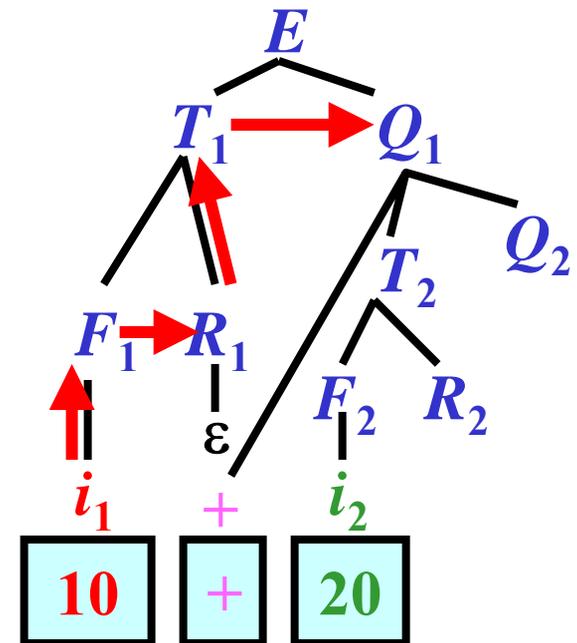
Zásobník synt. an.:

$$\begin{aligned} &\{F_{2.s} := i.value\} \\ &\{R_{2.i} := F_{2.s}\} \\ &\quad R_2 \\ &\{T_{2.s} := R_{2.s}\} \\ &\{Q_{2.i} := Q_{1.i} + T_{2.s}\} \\ &\quad Q_2 \\ &\{Q_{1.s} := Q_{2.s}\} \\ &\{E.s := Q_{1.s}\} \\ &\quad \$ \end{aligned}$$

Zásobník sém. an.:

$$Q_1.i = 10$$

Ilustrace:



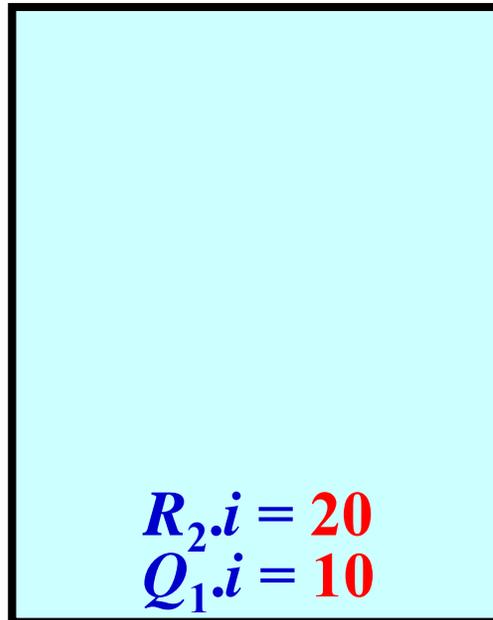
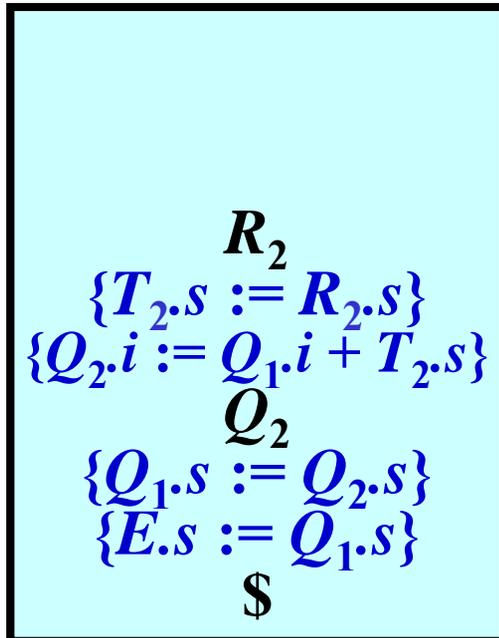
Vyhodnocení výrazů: Příklad 13/16

Příklad pro $a + b$, kde $a.value = 10$, $b.value = 20$

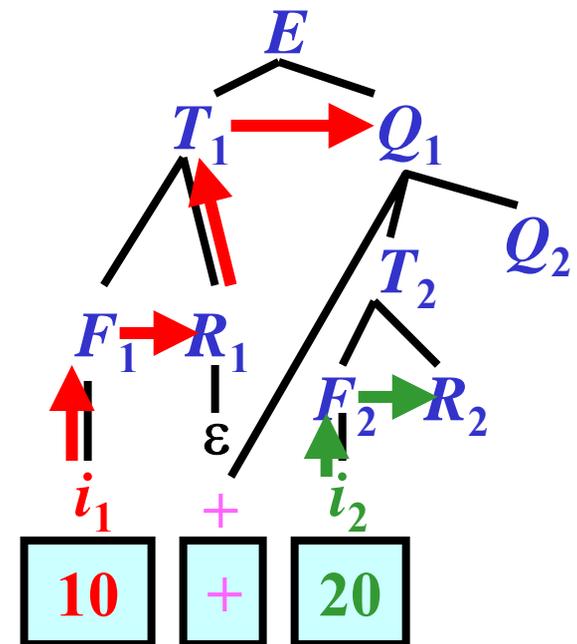
Vstup: $\$$

Prav.: $R_2 \rightarrow \varepsilon \{R_2.s := R_2.i\}$

Zásobník synt. an.: Zásobník sém. an.:



Ilustrace:



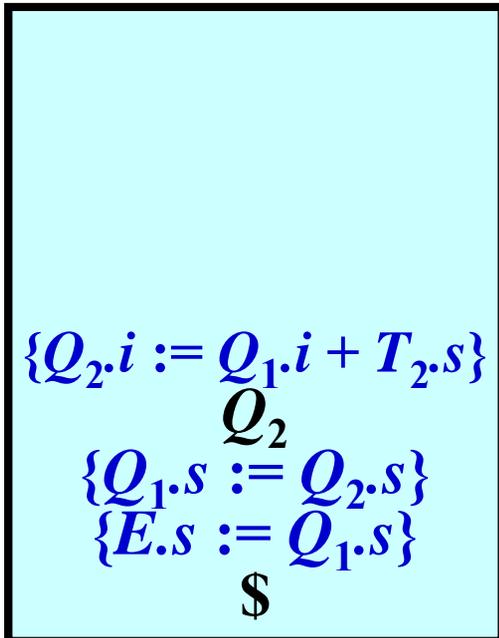
Vyhodnocení výrazů: Příklad 14/16

Příklad pro $a + b$, kde $a.value = 10$, $b.value = 20$

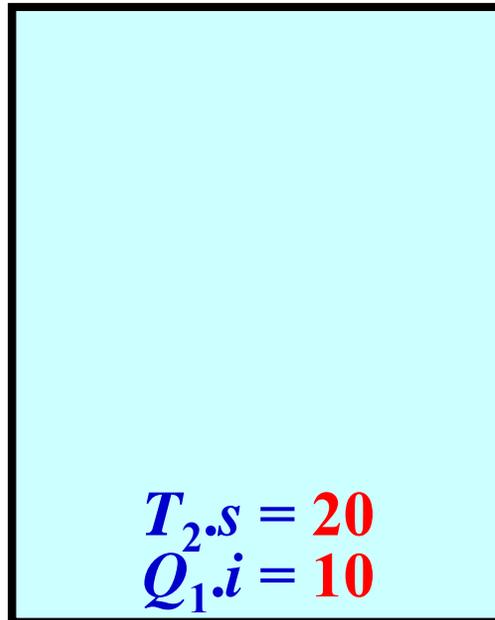
Vstup: $\$$

Prav.:

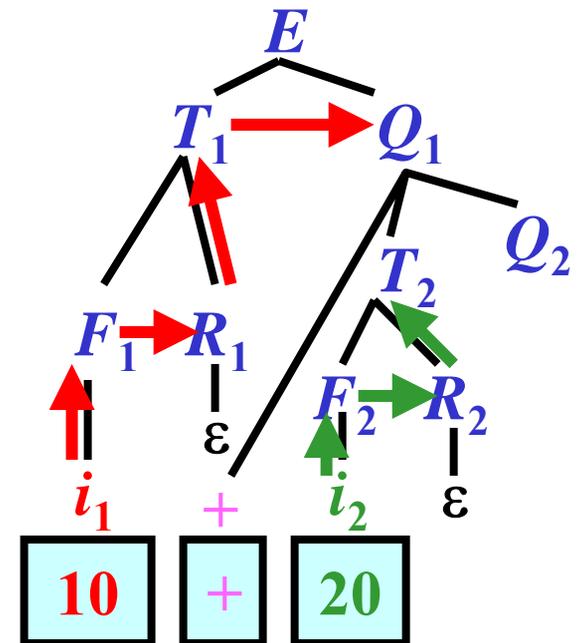
Zásobník synt. an.:



Zásobník sém. an.:



Ilustrace:



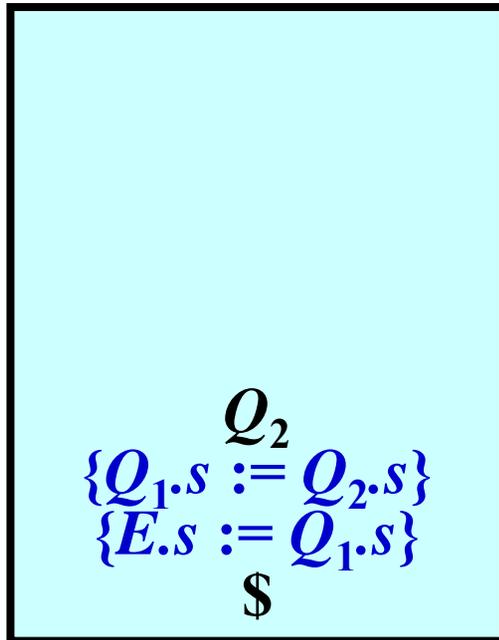
Vyhodnocení výrazů: Příklad 15/16

Příklad pro $a + b$, kde $a.value = 10$, $b.value = 20$

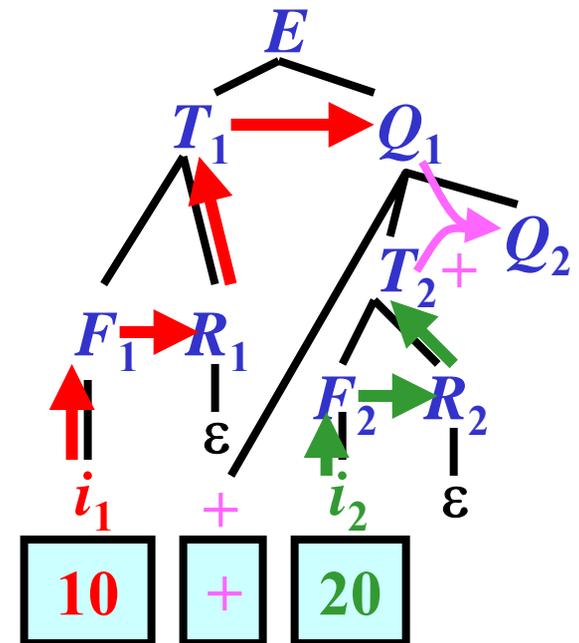
Vstup: $\$$

Prav.: $Q_2 \rightarrow \varepsilon \{Q_2.s := Q_2.i\}$

Zásobník synt. an.: Zásobník sém. an.:



Ilustrace:



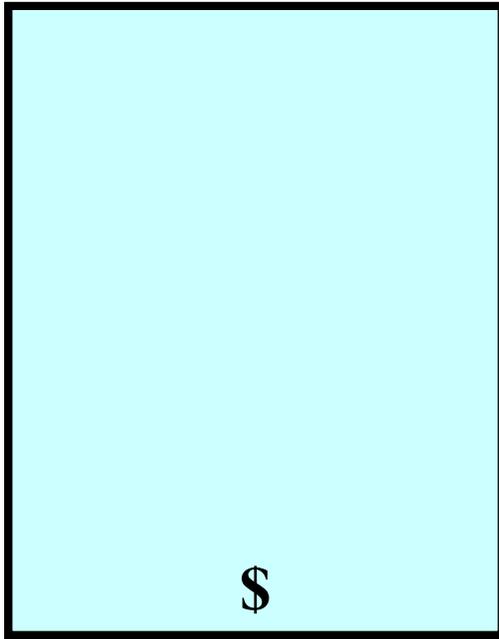
Vyhodnocení výrazů: Příklad 16/16

Příklad pro $a + b$, kde $a.value = 10$, $b.value = 20$

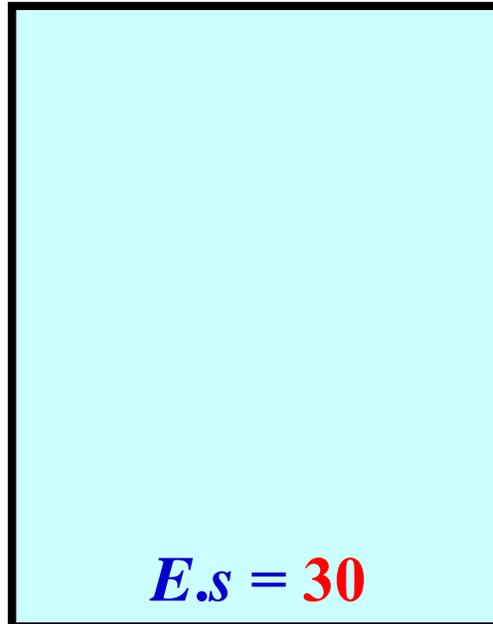
Vstup: \$

Prav.:

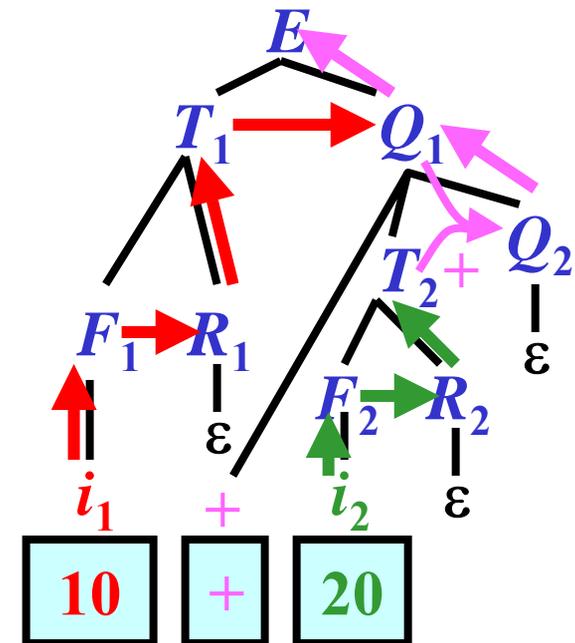
Zásobník synt. an.:



Zásobník sém. an.:



Ilustrace:



Sémantická analýza: Kontrola typů

1) Pravidlo: E
 $\quad \quad \quad |$
 $\quad \quad \quad id$

Akce:
 $E.type := id.type$

2) Pravidlo: E
 $\quad \quad \quad \swarrow \quad \downarrow \quad \searrow$
 $\quad \quad \quad E_1 \quad op \quad E_2$

Operace op je
 definována nad typy:

$t_1 \quad op \quad t_2 \rightarrow t_3$

Akce:
if ($E_1.type = t_1$ nebo
 $E_1.type$ je převeditelný na t_1)
and
 ($E_2.type = t_2$ nebo
 $E_2.type$ je převeditelný na t_2)
then
 $E.type := t_3$
else
Semantic Error.

Kontrola typů: Příklad 2/3

Pravidlo: $E_i \rightarrow E_j + T_k$ {

```

    if  $E_j.type = T_k.type$  then begin
         $E_i.type := E_j.type$ 
        generate(+,  $E_j.loc$ ,  $T_k.loc$ ,  $E_i.loc$ )
    end
    else begin
        generate(new.loc,  $h$ , , )
        if  $E_j.type = int$  then begin
            generate(int-to-real,  $E_j.loc$ , ,  $h$ )
            generate(+,  $h$ ,  $T_k.loc$ ,  $E_i.loc$ )
        end
        else begin
            generate(int-to-real,  $T_k.loc$ , ,  $h$ )
            generate(+,  $E_j.loc$ ,  $h$ ,  $E_i.loc$ )
        end
         $E_i.type := real$ 
    end
}

```

Kontrola typů: Příklad 3/3

Pravidlo: $T_i \rightarrow T_j * F_k$ {

```

    if  $T_j.type = F_k.type$  then begin
         $T_i.type := T_j.type$ 
        generate(*,  $T_j.loc$ ,  $F_k.loc$ ,  $T_i.loc$ )
    end
    else begin
        generate(new.loc,  $h$ , , )
        if  $T_j.type = int$  then begin
            generate(int-to-real,  $T_j.loc$ , ,  $h$ )
            generate(*,  $h$ ,  $F_k.loc$ ,  $T_i.loc$ )
        end
        else begin
            generate(int-to-real,  $F_k.loc$ , ,  $h$ )
            generate(*,  $T_j.loc$ ,  $h$ ,  $T_i.loc$ )
        end
         $T_i.type := real$ 
    end
}

```

Zkratové vyhodnocování

Myšlenka:

- $a = \textit{true}$ implikuje $a \textit{ or } (\dots ? \dots) = \textit{true}$
- $a = \textit{false}$ implikuje $a \textit{ and } (\dots ? \dots) = \textit{false}$

Pozn.: $(\dots ? \dots)$ není vyhodnoceno.

1) $(a \textit{ and } b) = p$:

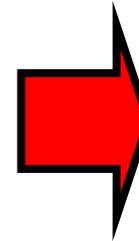
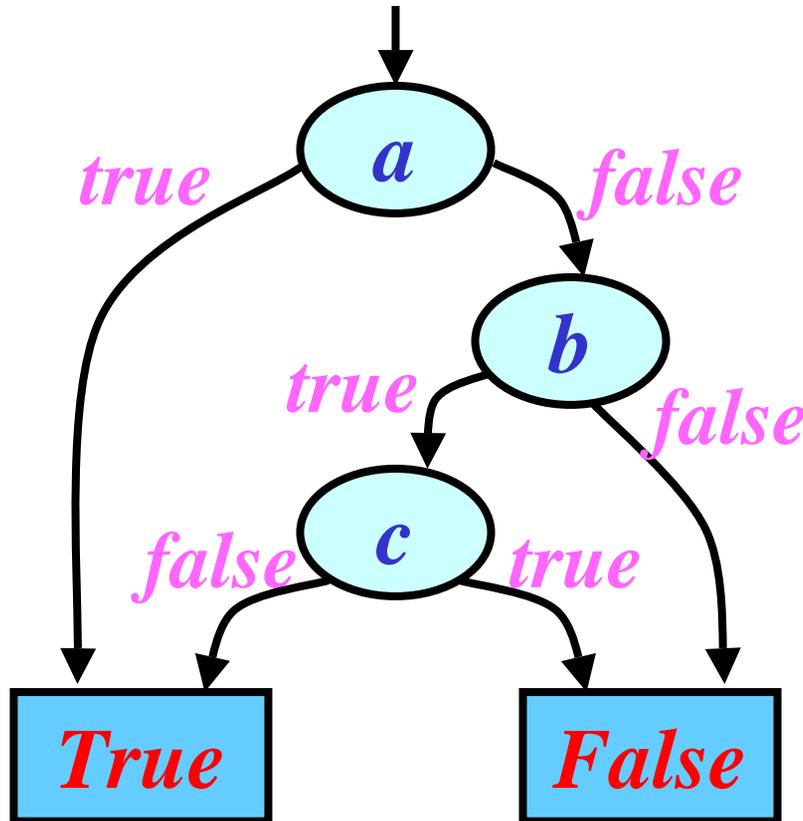
if $a = \textit{false}$ then $p = \textit{false}$
 else $p = b$

2) $(a \textit{ or } b) = p$:

if $a = \textit{true}$ then $p = \textit{true}$
 else $p = b$

Zkratové vyhodnocování: Graf. reprezentace

Příklad: *a* or (*b* and (not *c*)):



```
if a goto True
goto X
```

```
X:
if b goto Y
goto False
```

```
Y:
if c goto False
goto True
```

```
True: ...
```

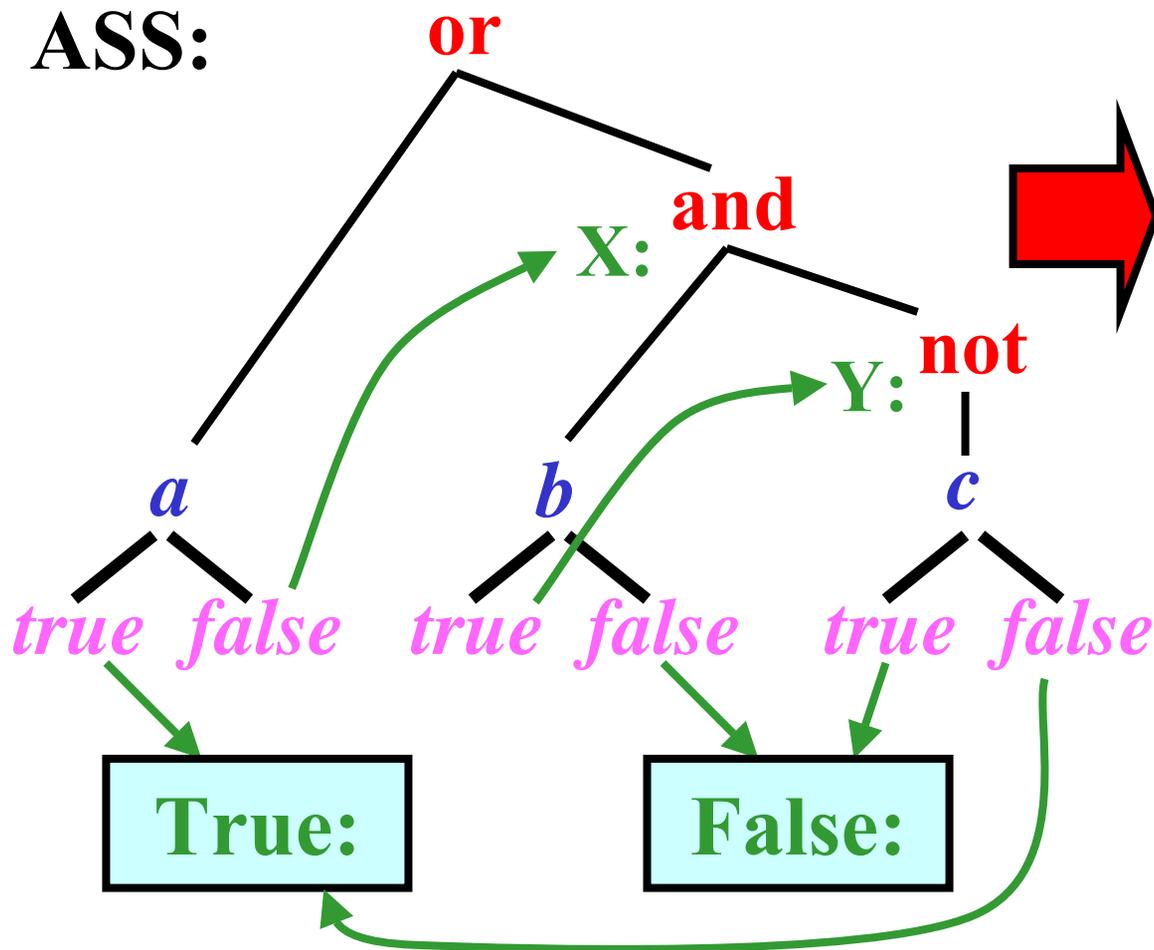
```
False: ...
```

- Simulace grafické reprezentace 3AK kódem se skoky.

Zkratové vyhodnocování pomocí ASS

Příklad: a or (b and (not c)):

ASS:



```
if  $a$  goto True
goto X
```

```
X:
if  $b$  goto Y
goto False
```

```
Y:
if  $c$  goto False
goto True
```

```
True: ...
```

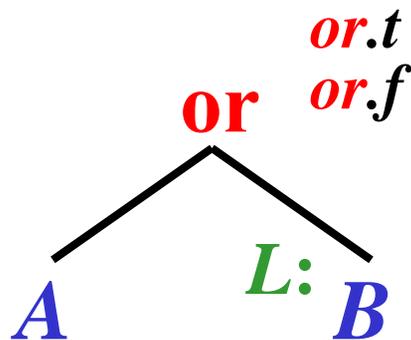
```
False: ...
```

Zkr. vyh. pomocí ASS: Implementace

- Každý ASS uzel X má přiřazeny dva atributy: $X.t$, $X.f$

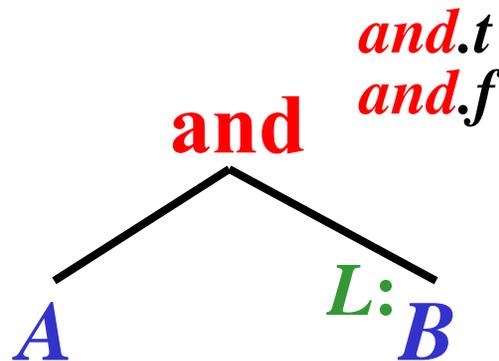
Elementární ASS:

1)



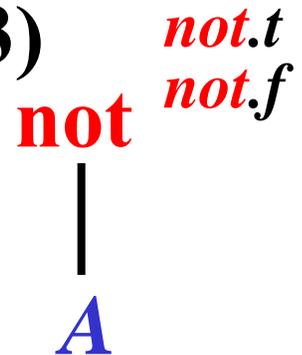
$A.t := or.t$ $B.t := or.t$
 $A.f := L$ $B.f := or.f$

2)



$A.t := L$ $B.t := and.t$
 $A.f := and.f$ $B.f := and.f$

3)



$A.t := not.f$
 $A.f := not.t$

- Pozn.: L = nové návěští
- Pozn.: L = nové návěští

- **Initializace:** Necht' R je kořen ASS. Potom:

$R.t := \mathbf{True}$, $R.f := \mathbf{False}$ (\mathbf{True} & \mathbf{False} jsou návěští)

- **Šíření hodnot atributů:** Atributy jsou šířeny z kořene do listů použitím pravidel 1), 2) a 3).

Zkr. vyh. pomocí ASS: Příklad

Příklad: a or (b and (not c)):

ASS:

or $or.t := T$
 $or.f := F$

Note:

- $T = True$
- $F = False$

X: and $and.t := or.t (= T)$
 $and.f := or.f (= F)$

Y: not $not.t := and.t (= T)$
 $not.f := and.f (= F)$

a

$a.t := or.t (= T)$

$a.f := X$

b

$b.t := Y$

$b.f := and.f (= F)$

c

$c.t := not.f (= F)$

$c.f := not.t (= T)$

if a goto T
 goto X

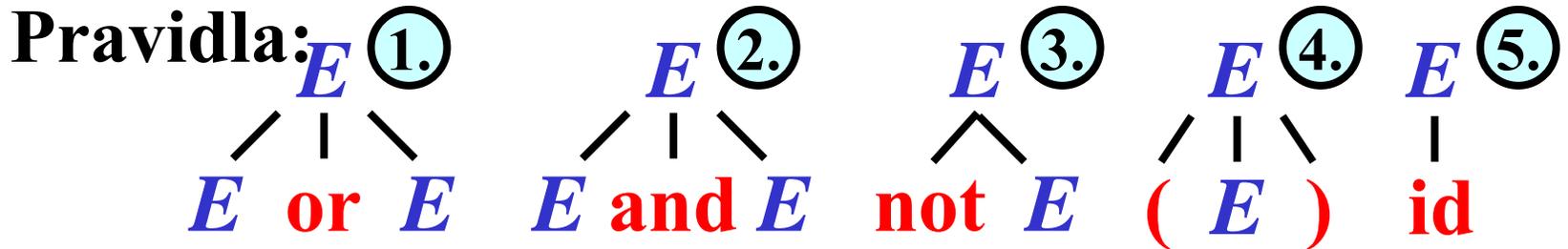
X :
 if b goto Y
 goto F

Y :
 if c goto F
 goto T

T : ...
 F : ...

Zkr. vyh.: Přímé generování kódu 1/5

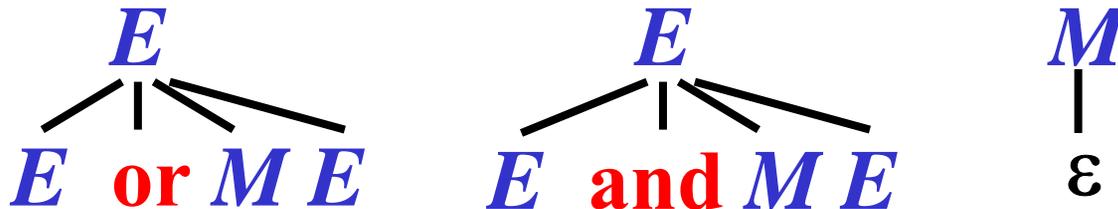
- Gramatika pro boolovské výrazy:



Poznámka: Ošetřit nejednoznačnost gramatiky!

- Modifikace gramatiky:

1) Zaměnit pravidla ① & ② na:



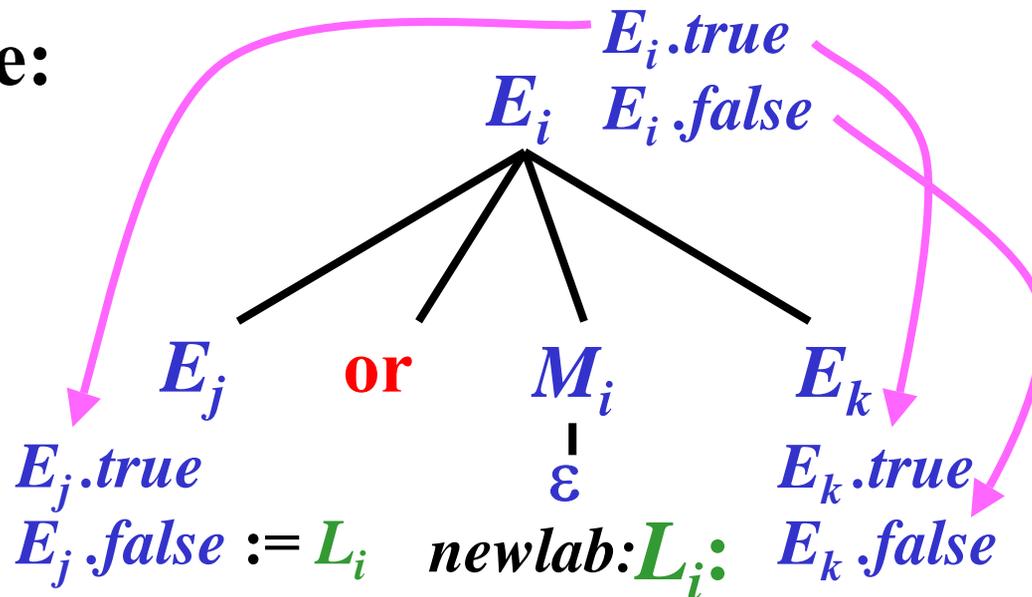
2) Přiřadit každému pravidlu následující sémantické akce:

Zkr. vyh.: Přímé generování kódu 2/5

$M_i \rightarrow \varepsilon$ {generate " $M_i.lab:$ "} // Generování
nového návěští

$E_i \rightarrow E_j$ **or** $M_i E_k$ { $M_i.lab := GenerateNewLab$;
 $E_j.true := E_i.true$; $E_j.false := M_i.lab$
 $E_k.true := E_i.true$; $E_k.false := E_i.false$ }

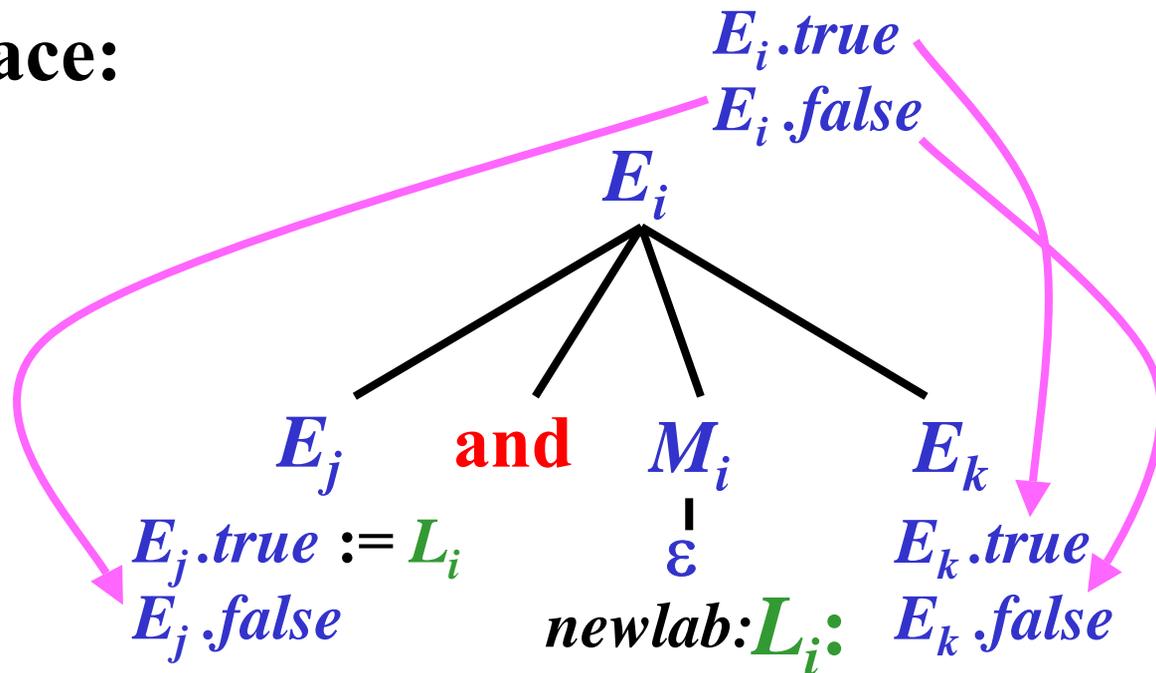
Ilustrace:



Zkr. vyh.: Přímé generování kódu 3/5

$$E_i \rightarrow E_j \text{ and } M_i E_k \{ M_i.lab := GenerateNewLab; \\ E_j.true := M_i.lab; E_j.false := E_i.false \\ E_k.true := E_i.true; E_k.false := E_i.false \}$$

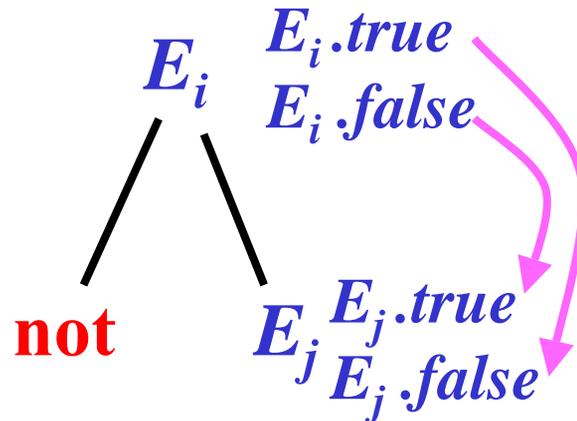
Ilustrace:



Zkr. vyh.: Přímé generování kódu 4/5

$$E_i \rightarrow \mathbf{not} E_j \quad \{ E_j.true := E_i.false; \\ E_j.false := E_i.true \}$$

Ilustrace:

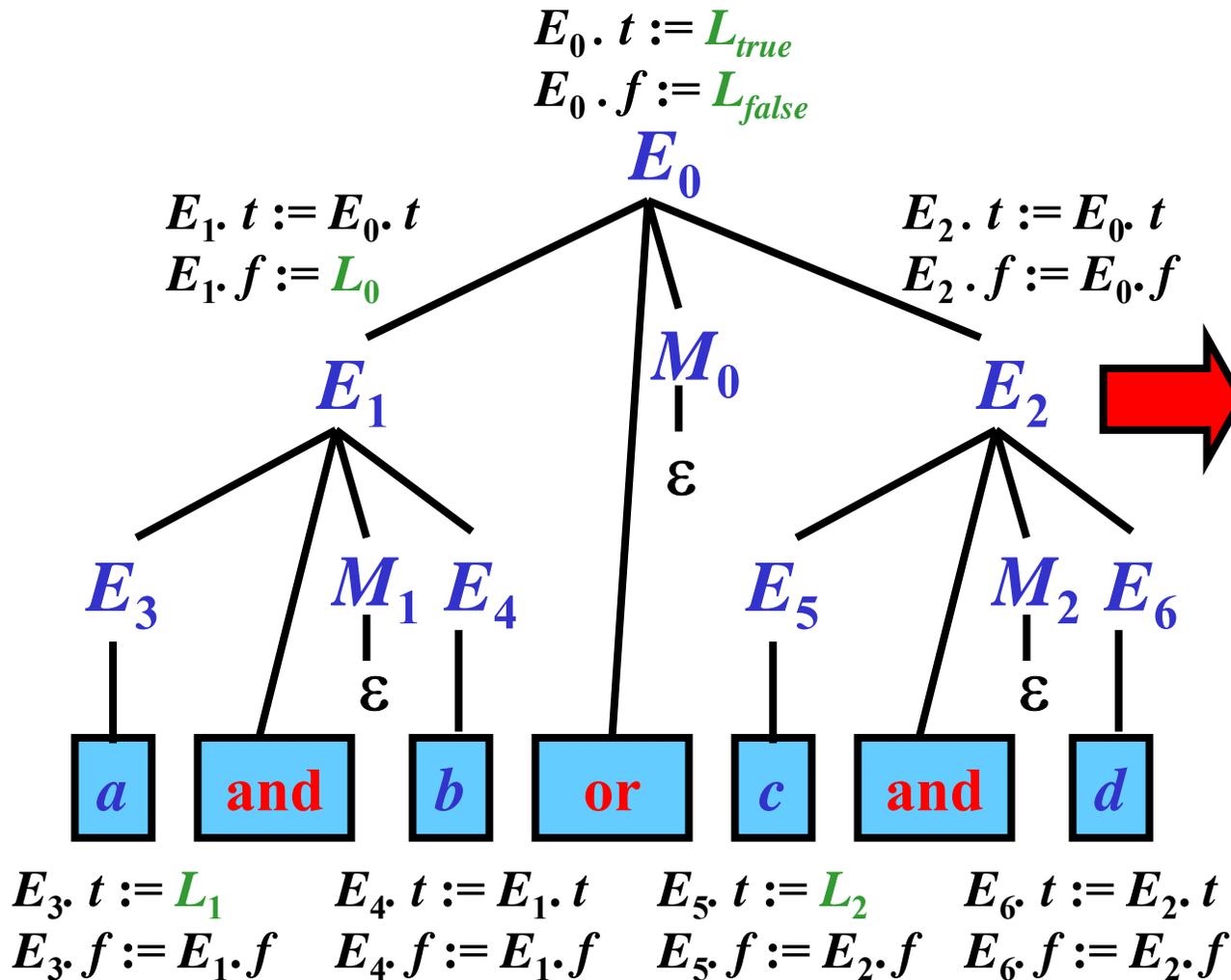


$$E_i \rightarrow (E_j) \quad \{ E_j.true := E_i.true; \\ E_j.false := E_i.false \}$$

$$E_i \rightarrow \mathbf{id}_j \quad \{ \text{generate “if } \mathbf{id}_j.val \text{ goto } E_i.true\text{”}; \\ \text{generate “goto } E_i.false\text{”} \}$$

Zkr. vyh.: Přímé generování kódu 5/5

Příklad: *a* and *b* or *c* and *d*:



```

if a goto  $L_1$ 
goto  $L_0$ 
 $L_1$ :
if b goto  $L_{true}$ 
goto  $L_0$ 
 $L_0$ :
if c goto  $L_2$ 
goto  $L_{false}$ 
 $L_2$ :
if d goto  $L_{true}$ 
goto  $L_{false}$ 
 $L_{true}$ : ...
 $L_{false}$ : ...
  
```

Větvení: If-Then

Pravidlo: **<if-then>**

if **<cond>** **then** **<stat₁>**

Sémantická akce:

```

{ // vyhodnocení cond
  // do proměnné c.val
  (not , c.val, , c.val)
  (goto, c.val, , L1 )
  // kód stat1
}
(lab , L1 , , )

```

Větvení: If-Then-Else

Pravidlo: **<if-then-else>**

if **<cond>** **then** **<stat₁>** **else** **<stat₂>**

Sémantická akce:

```

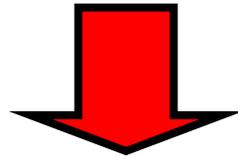
{ // vyhodnocení cond
  // do proměnné c.val
  (not , c.val, , c.val)
  (goto, c.val, , L1 )
  // kód stat1 }
  (goto, , , L2 )
  (lab , L1 , , )
  // kód stat2 }
  (lab , L2 , , )

```

While cyklus

Pravidlo: **<while-loop>**

while **<cond>** **do** **<stat>**



Sémantická akce:

```
(lab , L1 , , )
{ // vyhodnocení cond
  // do proměnné c.val
  (not , c.val , , c.val)
  (goto , c.val , , L2 )
  // kód stat }
(goto , , , L1 )
(lab , L2 , , )
```

Repeat cyklus

Pravidlo: **<repeat-loop>**

repeat **<stat>** **until** **<cond>**

Sémantická akce:

```
(lab , L1 , , )
{ // kód stat

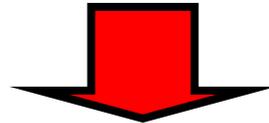
  // vyhodnocení cond }
  // do proměnné c.val }
(not , c.val , , c.val)
(goto , c.val , , L1 )
```

YACC: Základní myšlenka

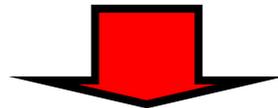
- Automatická konstrukce SA z BKG
 - Yacc jako překladač × Yacc jako jazyk
 - *Yacc* = *Yet another compiler compiler*
-

Ilustrace:

Bezkontextová gramatika G

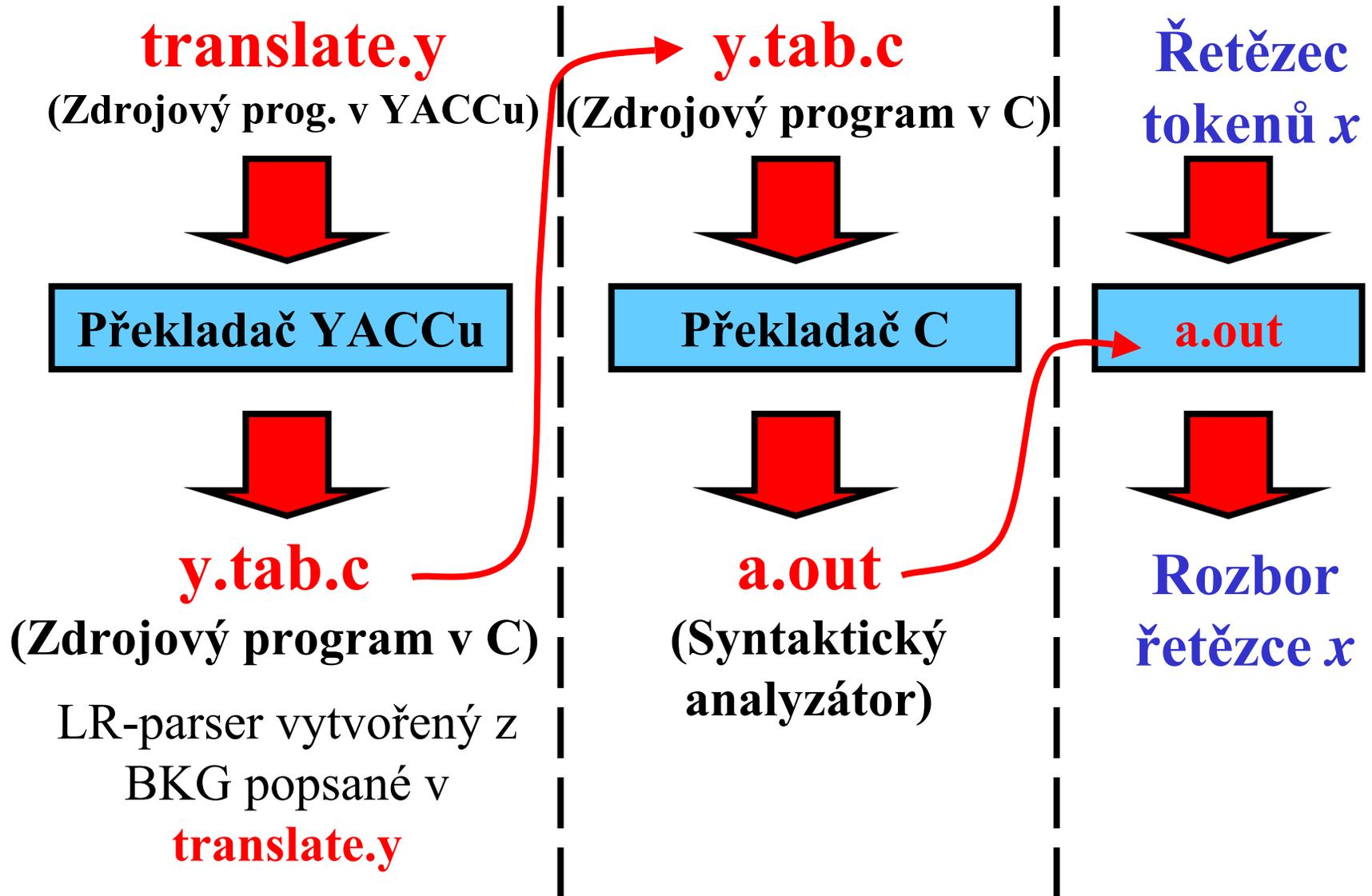


YACC



Syntaktický analyzátor pro G

YACC: Fáze kompilace



Struktura zdrojového programu v YACCu

/* Sekce I: Deklarace */

d_1, d_2, \dots, d_i

%% /* Konec sekce I*/

/* Sekce II: Překladová pravidla */

r_1, r_2, \dots, r_j

%% /* Konec sekce II*/

/* Sekce III: Pomocné procedury */

p_1, p_2, \dots, p_k

Popis gramatiky v YACCu

- **Neterminály:** názvy (= řetězce)
- **Příklad:** `prog`, `stat`, `expr`, ...

- **Terminály:** Znaky v uvozovkách nebo deklarované tokeny
- **Příklad:** `'+'`, `'*'`, `'('`, `')'`, `ID`, `INTEGER`

- **Pravidla:** Množina A-Pravidel: $\{ A \rightarrow x_1, A \rightarrow x_2, \dots, A \rightarrow x_n \}$
je zapsána:

A	:	x1
		x2
		...
		xn
- **Příklad:**

expr	:	expr	'+'	expr
		ID		

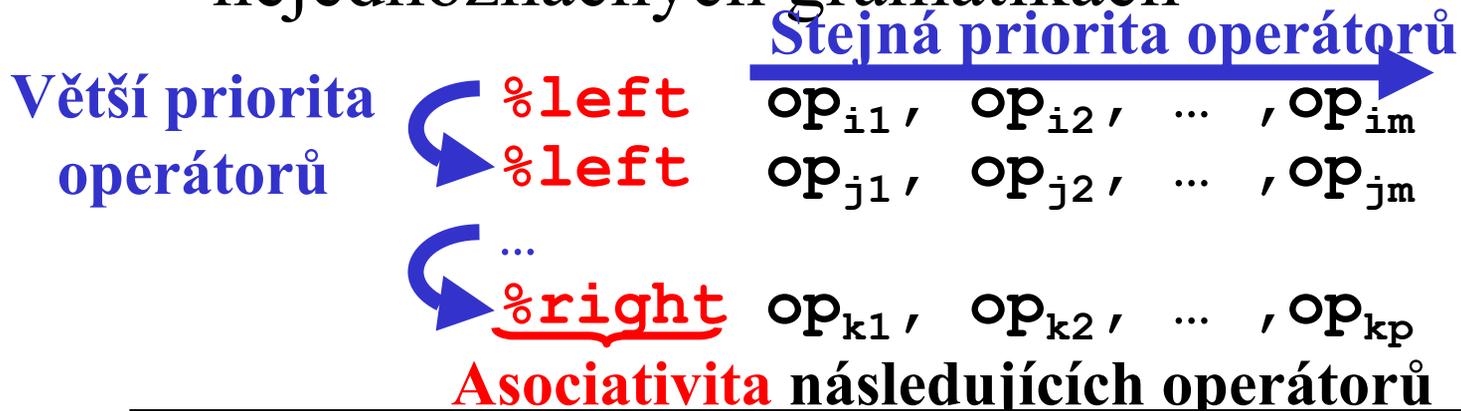
- **Počáteční neterminál:** Levá strana prvního pravidla

Sekce I: Deklarace

1) Deklarace tokenů

```
%token TYP_TOKENU
```

2) Specifikace asociativit & precedencí v nejednoznačných gramatikách



Příklad:

```
%token INTEGER
%token ID
%left '+'
%left '*'
```

Sekce II: Překládová pravidla

- Překládová pravidla jsou ve tvaru:

Pravidlo **Semanticka_Akce**

- **Semanticka_Akce** je podprogram, který je zavolán, pokud právě **Pravidlo** je použito.

Speciální symboly pro pravidla r :

\$\$ = atribut symbolu na levé straně pravidla r

$\$i$ = atribut i -tého symbolu na pravé straně pravidla r

Příklad:

```

expr : expr '+' expr { $$ = $1 + $3 }
      | expr '*' expr { $$ = $1 * $3 }
      | '(' expr ')' { $$ = $2 }
      | INTEGER
      | ID
  
```

Sekce III: Pomocné procedury

- Pomocné procedury jsou volány v sémantických akcí pravidel

Pozn.: Pokud YACC nespolupracuje s LEXem, musí být v této sekci implementována funkce `yylex()` plnící činnost lexikálního analyzátoru.

Příklad:

```
int yylex() {  
    /* Get the next token */  
    &yylval = attribute;  
    return TYPE_OF_TOKEN;  
}
```

Zdrojový program v YACCu

```
%token INTEGER
```

```
%token ID
```

```
%left '+'
```

```
%left '*'
```

```
%%
```

```
expr : expr '+' expr {$$ = $1 + $3}
     | expr '*' expr {$$ = $1 * $3}
     | '(' expr ')' {$$ = $2}
     | INTEGER
     | ID
```

```
%%
```

```
int yylex () { ... }
```