Atributovaný překlad

Typy atributů, implementace

Šárka Vavrečková

Ústav informatiky, FPF SU Opava

sarka.vavreckova@fpf.slu.cz

Poslední aktualizace: 5. prosince 2023

Typy atributů

Definice (Syntetizované a dědičné atributy)

Syntetizovaný atribut:

- *hodnota závisí na hodnotách atributů uzlů jeho vlastního podstromu*
- *k posílání údajů v derivačním stromě směrem zdola nahoru*

Dědičný atribut:

- *hodnota závisí na hodnotách atributů nadřízeného uzlu nebo uzlů na stejné úrovni*
- *k posílání údajů v derivačním stromě směrem dolů nebo vpravo*

Žádný atribut nemůže být zároveň syntetizovaný a dědičný.

$A \to BnCD \qquad \{A.val = B.val + D.val,\ C.des = (A.m + B.val) * 10 + n.lex\}$

- $A.val$ je syntetizovaný, závisí na atributech symbolů-potomků
- $C.des$ je dědičný, závisí na atributech předka a „bratrů“ vlevo

Inicializace atributů

Každý atribut musí být někde v derivačním stromě inicializován:

- dědičné atributy obvykle v kořeni stromu (obecně na horním levém začátku cesty toku hodnoty atributu derivačním stromem),
- syntetizované atributy většinou v listech stromu (tj. v terminálních pravidlech), často lexikálním analyzátorem.

Příklad

Gramatika generuje výraz ve tvaru $\langle hledej \rangle\ (n; L)\ v$

$\langle hledej \rangle$ je klíčové slovo určující příkaz pro hledání čísla v seznamu,

n je číslo, které má být hledáno,

L generuje seznam čísel,

v je výstupní terminál, do jeho atributu uložíme výsledek

Syntaktická pravidla v překladové gramatice:

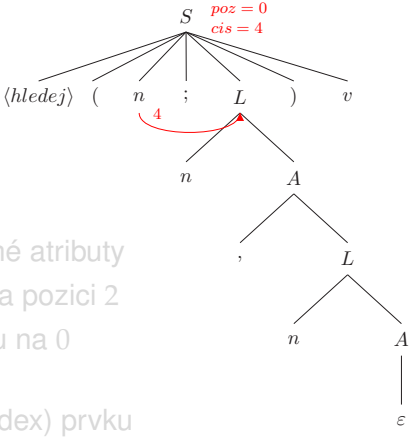
$S \rightarrow \langle hledej \rangle\ (n; L)\ v$

$L \rightarrow nA$

$A \rightarrow, L \mid \varepsilon$

$S \to \langle hledej \rangle\ (n; L)\ v$
$L \to nA$
$A \to , L \mid \varepsilon$

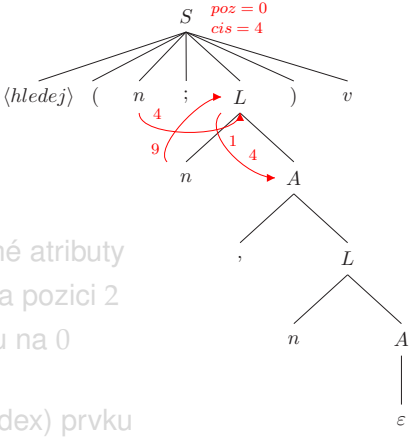
Zpracování výrazu $\langle hledej \rangle\ (4; 9, 4)$



1. hledané číslo inicializujeme na 4, pozice na 0 (dědičné atributy)
2. posíláme dolů hledanou hodnotu 4 a pozici 1 (zvýšili jsme o 1)
3. do nižšího patra přepošleme dědičné atributy
4. posíláme dolů hledanou hodnotu 4 a pozici 2
5. inicializace syntetizovaného atributu na 0 (zatím nenalezen)
6. nalezen, místo 0 pošleme pozici (index) prvku
7. přeneseme do výstupního atributu

$S \to \langle hledej \rangle\ (n; L)\ v$
$L \to nA$
$A \to , L \mid \varepsilon$

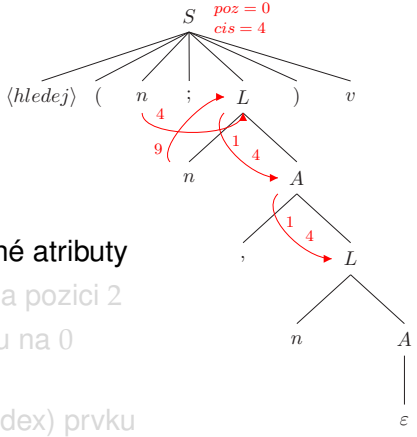
Zpracování výrazu $\langle hledej \rangle\ (4; 9, 4)$



1. hledané číslo inicializujeme na 4, pozice na 0 (dědičné atributy)
2. posíláme dolů hledanou hodnotu 4 a pozici 1 (zvýšili jsme o 1)
3. do nižšího patra přepošleme dědičné atributy
4. posíláme dolů hledanou hodnotu 4 a pozici 2
5. inicializace syntetizovaného atributu na 0 (zatím nenalezen)
6. nalezen, místo 0 pošleme pozici (index) prvku
7. přeneseme do výstupního atributu

$S \to \langle hledej \rangle\ (n; L)\ v$
$L \to nA$
$A \to , L \mid \varepsilon$

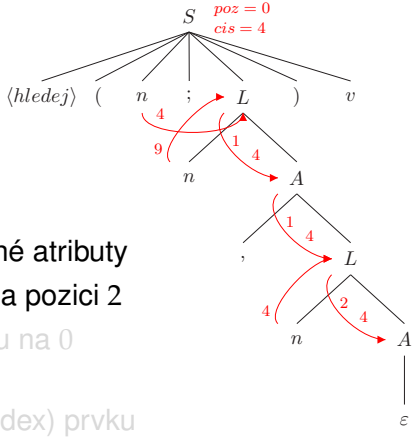
Zpracování výrazu $\langle hledej \rangle\ (4; 9, 4)$



1. hledané číslo inicializujeme na 4, pozice na 0 (dědičné atributy)
2. posíláme dolů hledanou hodnotu 4 a pozici 1 (zvýšili jsme o 1)
3. do nižšího patra přepošleme dědičné atributy
4. posíláme dolů hledanou hodnotu 4 a pozici 2
5. inicializace syntetizovaného atributu na 0 (zatím nenalezen)
6. nalezen, místo 0 pošleme pozici (index) prvku
7. přeneseme do výstupního atributu

$S \to \langle hledej \rangle\ (n; L)\ v$
$L \to nA$
$A \to , L \mid \varepsilon$

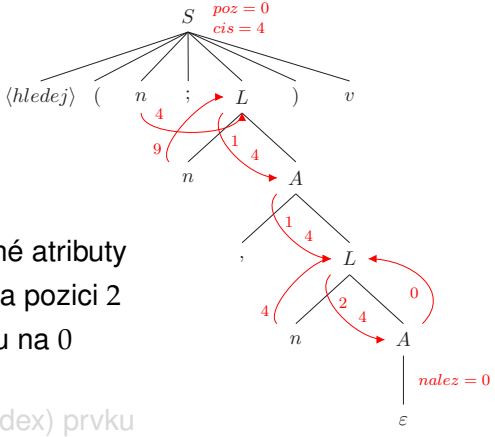
Zpracování výrazu $\langle hledej \rangle\ (4; 9, 4)$



1. hledané číslo inicializujeme na 4, pozice na 0 (dědičné atributy)
2. posíláme dolů hledanou hodnotu 4 a pozici 1 (zvýšili jsme o 1)
3. do nižšího patra přepošleme dědičné atributy
4. posíláme dolů hledanou hodnotu 4 a pozici 2
5. inicializace syntetizovaného atributu na 0 (zatím nenalezen)
6. nalezen, místo 0 pošleme pozici (index) prvku
7. přeneseme do výstupního atributu

$S \to \langle hledej \rangle\ (n; L)\ v$
$L \to nA$
$A \to , L \mid \varepsilon$

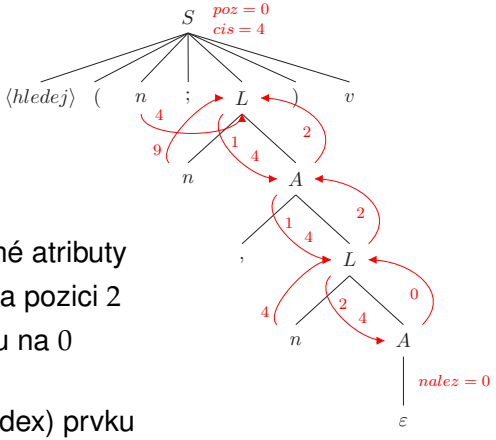
Zpracování výrazu $\langle hledej \rangle\ (4; 9, 4)$



1. hledané číslo inicializujeme na 4, pozice na 0 (dědičné atributy)
2. posíláme dolů hledanou hodnotu 4 a pozici 1 (zvýšili jsme o 1)
3. do nižšího patra přepošleme dědičné atributy
4. posíláme dolů hledanou hodnotu 4 a pozici 2
5. inicializace syntetizovaného atributu na 0 (zatím nenalezen)
6. nalezen, místo 0 pošleme pozici (index) prvku
7. přeneseme do výstupního atributu

$S \to \langle hledej \rangle\ (n; L)\ v$
$L \to nA$
$A \to , L \mid \varepsilon$

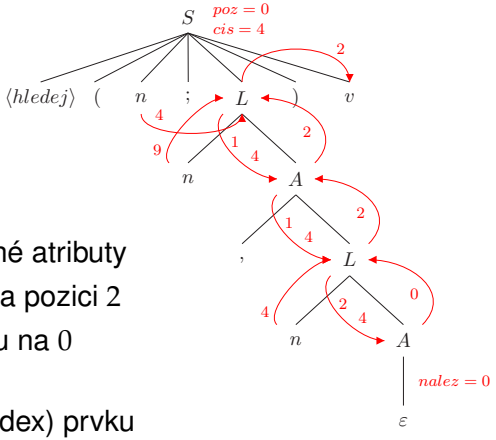
Zpracování výrazu $\langle hledej \rangle\ (4; 9, 4)$



1. hledané číslo inicializujeme na 4, pozice na 0 (dědičné atributy)
2. posíláme dolů hledanou hodnotu 4 a pozici 1 (zvýšili jsme o 1)
3. do nižšího patra přepošleme dědičné atributy
4. posíláme dolů hledanou hodnotu 4 a pozici 2
5. inicializace syntetizovaného atributu na 0 (zatím nenalezen)
6. nalezen, místo 0 pošleme pozici (index) prvku
7. přeneseme do výstupního atributu

$S \to \langle hledej \rangle\ (n; L)\ v$
$L \to nA$
$A \to , L \mid \varepsilon$

Zpracování výrazu $\langle hledej \rangle\ (4; 9, 4)$



1. hledané číslo inicializujeme na 4, pozice na 0 (dědičné atributy)
2. posíláme dolů hledanou hodnotu 4 a pozici 1 (zvýšili jsme o 1)
3. do nižšího patra přepošleme dědičné atributy
4. posíláme dolů hledanou hodnotu 4 a pozici 2
5. inicializace syntetizovaného atributu na 0 (zatím nenalezen)
6. nalezen, místo 0 pošleme pozici (index) prvku
7. přeneseme do výstupního atributu

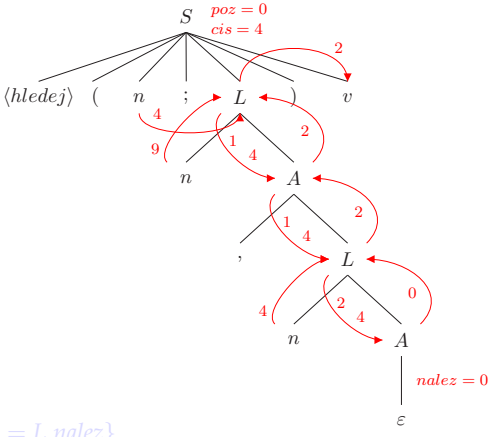
$S \to \langle hledej \rangle\ (n; L)\ v$
$L \to nA$
$A \to , L \mid \varepsilon$

Atributová gramatika

Použijeme atributy
$n[lex]$, $L[poz, cis, nalez]$,
$A[poz, cis, nalez]$, $v[vysl]$

- dědičné: *poz*, *cis*, *vysl*
- syntetizované: *lex*, *nalez*

$S \to \langle \text{hledej} \rangle\ (n;\ \{L.poz = 0, L.cis = n.lex\}$
$\quad L)\ v\ \{v.vysl = L.nalez\}$
$L \to n\ \{A.poz = L.poz + 1, A.cis = L.cis\}$
$\quad A\ \{\text{if } n.lex = A.cis \text{ then } L.nalez = A.poz$
$\qquad\qquad \text{else } L.nalez = A.nalez\}$
$A \to ,\ \{L.poz = A.poz, L.cis = A.cis\}\ L\ \{A.nalez = L.nalez\}$
$A \to \varepsilon\ \{A.nalez = 0\}$



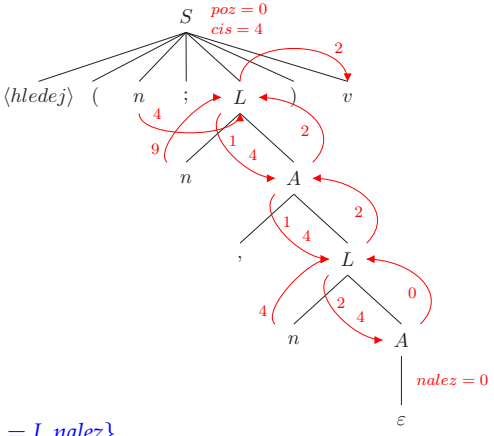
$S \to \langle hledej \rangle\ (n; L)\ v$
$L \to nA$
$A \to , L \mid \varepsilon$

Atributová gramatika

Použijeme atributy
$n[lex]$, $L[poz, cis, nalez]$,
$A[poz, cis, nalez]$, $v[vysl]$

- dědičné: *poz*, *cis*, *vysl*
- syntetizované: *lex*, *nalez*

$S \to \langle \text{hledej} \rangle\ (n;\ \{L.poz = 0, L.cis = n.lex\}$
$\quad L)\ v\ \ \{v.vysl = L.nalez\}$
$L \to n\ \ \{A.poz = L.poz + 1, A.cis = L.cis\}$
$\quad A\ \ \{\text{if } n.lex = A.cis \text{ then } L.nalez = A.poz$
$\qquad\qquad \text{else } L.nalez = A.nalez\}$
$A \to ,\ \ \{L.poz = A.poz, L.cis = A.cis\}\ \ L\ \ \{A.nalez = L.nalez\}$
$A \to \varepsilon\ \ \{A.nalez = 0\}$



Deterministický překlad výrazů při interpretaci

Požadavky na gramatiku

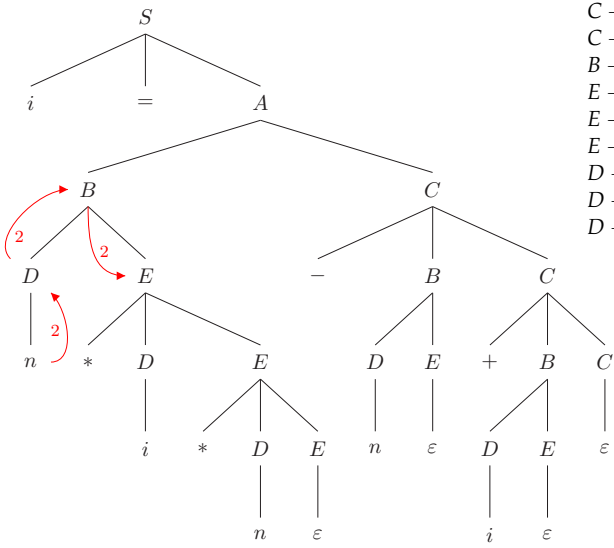
- gramatika typu $LL(1)$, silná $LR(1)$ nebo jiná, kterou lze použít pro deterministický překlad
- často není třeba použít výstupní terminály
- zachováváme prioritu operátorů

Příklad – $LL(1)$ gramatika

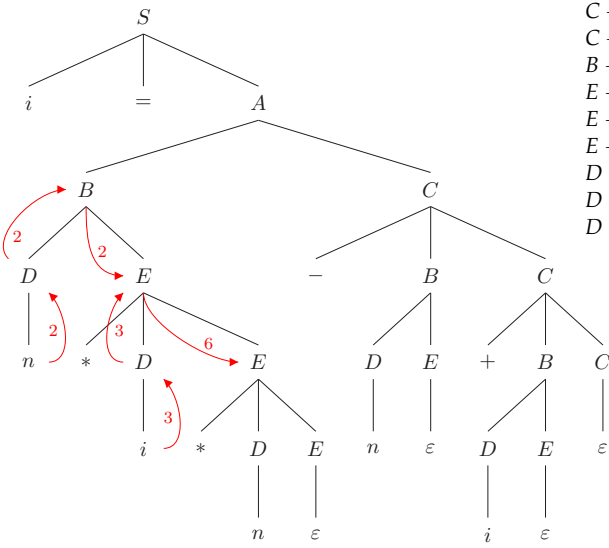
$S \to i = A$
$A \to BC$
$C \to +BC$
$C \to -BC$
$C \to \varepsilon$
$B \to DE$
$E \to *DE$
$E \to /DE$
$E \to \varepsilon$
$D \to n$
$D \to i$
$D \to (A)$

Příklad – $LL(1)$ gramatika

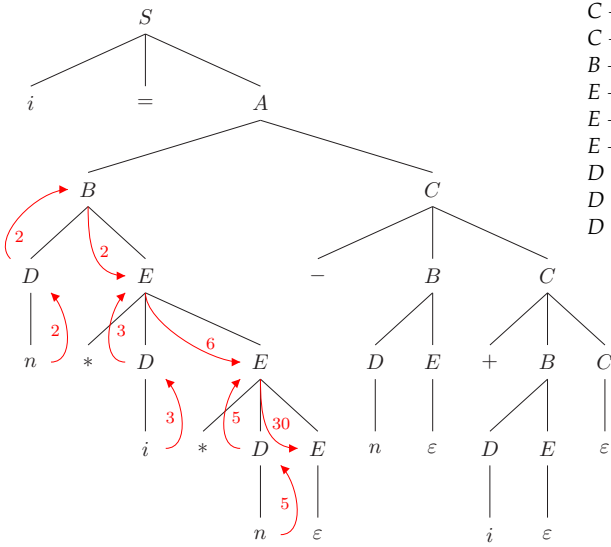
$S \to i = A \ \{\text{Uloz}(i.nazev, A.val)\}$
$A \to B \ \{C.m = B.val\} \ C \ \{A.val = C.val\}$
$C \to +B \ \{C_1.m = C_0.m + B.val\} \ C \ \{C_0.val = C_1.val\}$
$C \to -B \ \{C_1.m = C_0.m - B.val\} \ C \ \{C_0.val = C_1.val\}$
$C \to \varepsilon \ \{C.val = C.m\}$
$B \to D \ \{E.m = D.val\} \ E \ \{B.val = E.val\}$
$E \to *D \ \{E_1.m = E_0.m * D.val\} \ E \ \{E_0.val = E_1.val\}$
$E \to /D \ \{E_1.m = E_0.m/D.val\} \ E \ \{E_0.val = E_1.val\}$
$E \to \varepsilon \ \{E.val = E.m\}$
$D \to n \ \{D.val = n.lex\}$
$D \to i \ \{D.val = \text{ZjistiHodnotu}(i.nazev)\}$
$D \to (A) \ \{D.val = A.val\}$



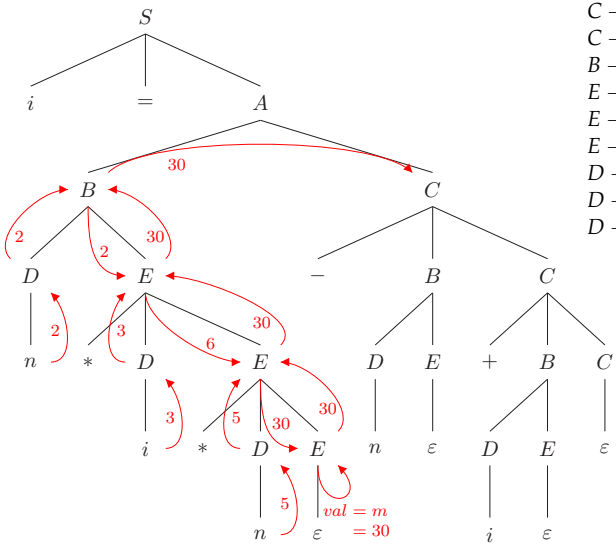
$S \to i = A$ {Uloz($i.nazev, A.val$)}
$A \to B$ $\{C.m = B.val\}$ C $\{A.val = C.val\}$
$C \to +B$ $\{C_1.m = C_0.m + B.val\}$ C $\{C_0.val = C_1.val\}$
$C \to -B$ $\{C_1.m = C_0.m - B.val\}$ C $\{C_0.val = C_1.val\}$
$C \to \varepsilon$ $\{C.val = C.m\}$
$B \to D$ $\{E.m = D.val\}$ E $\{B.val = E.val\}$
$E \to *D$ $\{E_1.m = E_0.m * D.val\}$ E $\{E_0.val = E_1.val\}$
$E \to /D$ $\{E_1.m = E_0.m/D.val\}$ E $\{E_0.val = E_1.val\}$
$E \to \varepsilon$ $\{E.val = E.m\}$
$D \to n$ $\{D.val = n.lex\}$
$D \to i$ $\{D.val =$ ZjistiHodnotu($i.nazev$)$\}$
$D \to (A)$ $\{D.val = A.val\}$



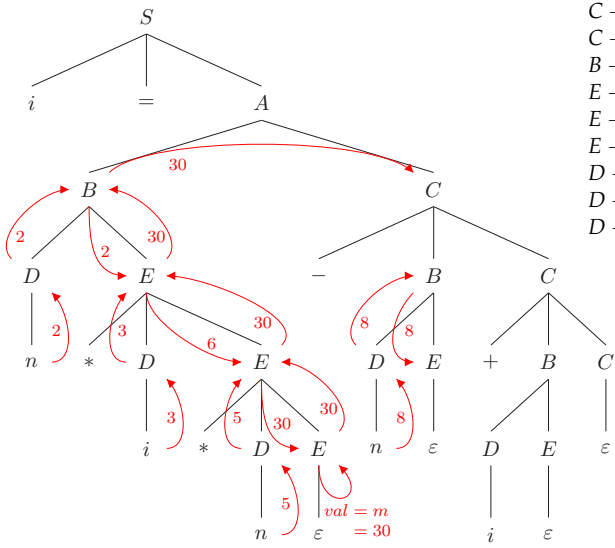
$S \to i = A$ {Uloz($i.nazev, A.val$)}
$A \to B$ $\{C.m = B.val\}$ C $\{A.val = C.val\}$
$C \to +B$ $\{C_1.m = C_0.m + B.val\}$ C $\{C_0.val = C_1.val\}$
$C \to -B$ $\{C_1.m = C_0.m - B.val\}$ C $\{C_0.val = C_1.val\}$
$C \to \varepsilon$ $\{C.val = C.m\}$
$B \to D$ $\{E.m = D.val\}$ E $\{B.val = E.val\}$
$E \to *D$ $\{E_1.m = E_0.m * D.val\}$ E $\{E_0.val = E_1.val\}$
$E \to /D$ $\{E_1.m = E_0.m/D.val\}$ E $\{E_0.val = E_1.val\}$
$E \to \varepsilon$ $\{E.val = E.m\}$
$D \to n$ $\{D.val = n.lex\}$
$D \to i$ $\{D.val =$ ZjistiHodnotu($i.nazev$)$\}$
$D \to (A)$ $\{D.val = A.val\}$



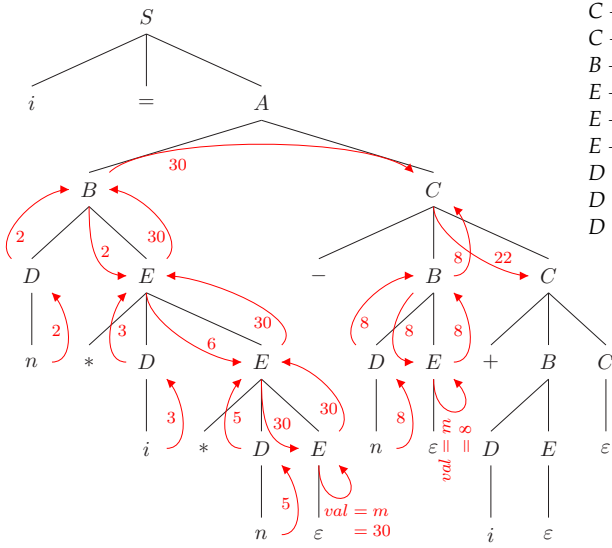
$S \to i = A$ {Uloz($i.nazev, A.val$)}
$A \to B$ $\{C.m = B.val\}$ C $\{A.val = C.val\}$
$C \to +B$ $\{C_1.m = C_0.m + B.val\}$ C $\{C_0.val = C_1.val\}$
$C \to -B$ $\{C_1.m = C_0.m - B.val\}$ C $\{C_0.val = C_1.val\}$
$C \to \varepsilon$ $\{C.val = C.m\}$
$B \to D$ $\{E.m = D.val\}$ E $\{B.val = E.val\}$
$E \to *D$ $\{E_1.m = E_0.m * D.val\}$ E $\{E_0.val = E_1.val\}$
$E \to /D$ $\{E_1.m = E_0.m/D.val\}$ E $\{E_0.val = E_1.val\}$
$E \to \varepsilon$ $\{E.val = E.m\}$
$D \to n$ $\{D.val = n.lex\}$
$D \to i$ $\{D.val =$ ZjistiHodnotu($i.nazev$)$\}$
$D \to (A)$ $\{D.val = A.val\}$



$S \to i = A$ {Uloz($i.nazev, A.val$)}
$A \to B$ $\{C.m = B.val\}$ C $\{A.val = C.val\}$
$C \to +B$ $\{C_1.m = C_0.m + B.val\}$ C $\{C_0.val = C_1.val\}$
$C \to -B$ $\{C_1.m = C_0.m - B.val\}$ C $\{C_0.val = C_1.val\}$
$C \to \varepsilon$ $\{C.val = C.m\}$
$B \to D$ $\{E.m = D.val\}$ E $\{B.val = E.val\}$
$E \to *D$ $\{E_1.m = E_0.m * D.val\}$ E $\{E_0.val = E_1.val\}$
$E \to /D$ $\{E_1.m = E_0.m/D.val\}$ E $\{E_0.val = E_1.val\}$
$E \to \varepsilon$ $\{E.val = E.m\}$
$D \to n$ $\{D.val = n.lex\}$
$D \to i$ {$D.val =$ ZjistiHodnotu($i.nazev$)}
$D \to (A)$ $\{D.val = A.val\}$

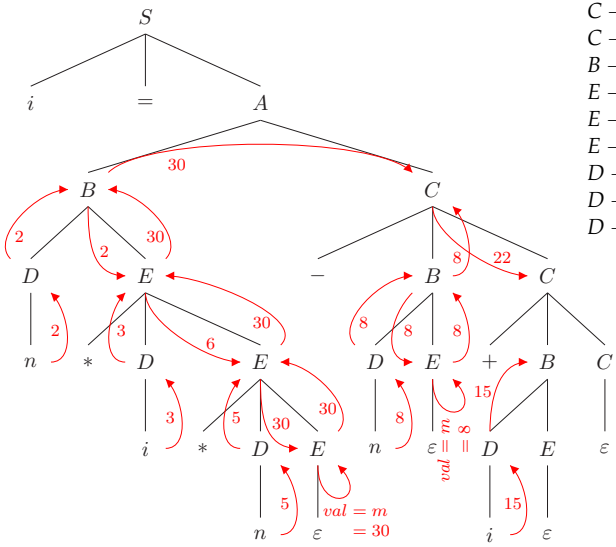


$S \to i = A$ {Uloz($i.nazev, A.val$)}
$A \to B$ {$C.m = B.val$} C {$A.val = C.val$}
$C \to +B$ {$C_1.m = C_0.m + B.val$} C {$C_0.val = C_1.val$}
$C \to -B$ {$C_1.m = C_0.m - B.val$} C {$C_0.val = C_1.val$}
$C \to \varepsilon$ {$C.val = C.m$}
$B \to D$ {$E.m = D.val$} E {$B.val = E.val$}
$E \to *D$ {$E_1.m = E_0.m * D.val$} E {$E_0.val = E_1.val$}
$E \to /D$ {$E_1.m = E_0.m/D.val$} E {$E_0.val = E_1.val$}
$E \to \varepsilon$ {$E.val = E.m$}
$D \to n$ {$D.val = n.lex$}
$D \to i$ {$D.val =$ ZjistiHodnotu($i.nazev$)}
$D \to (A)$ {$D.val = A.val$}

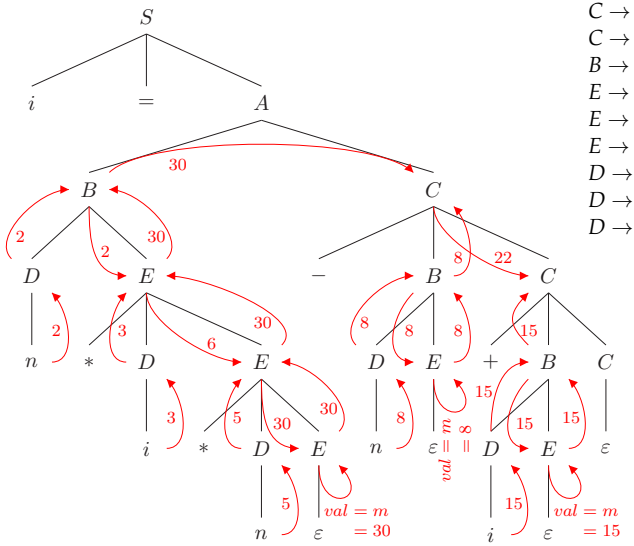


$S \to i = A$ {Uloz($i.nazev, A.val$)}
$A \to B$ {$C.m = B.val$} C {$A.val = C.val$}
$C \to +B$ {$C_1.m = C_0.m + B.val$} C {$C_0.val = C_1.val$}
$C \to -B$ {$C_1.m = C_0.m - B.val$} C {$C_0.val = C_1.val$}
$C \to \varepsilon$ {$C.val = C.m$}
$B \to D$ {$E.m = D.val$} E {$B.val = E.val$}
$E \to *D$ {$E_1.m = E_0.m * D.val$} E {$E_0.val = E_1.val$}
$E \to /D$ {$E_1.m = E_0.m/D.val$} E {$E_0.val = E_1.val$}
$E \to \varepsilon$ {$E.val = E.m$}
$D \to n$ {$D.val = n.lex$}
$D \to i$ {$D.val =$ ZjistiHodnotu($i.nazev$)}
$D \to (A)$ {$D.val = A.val$}

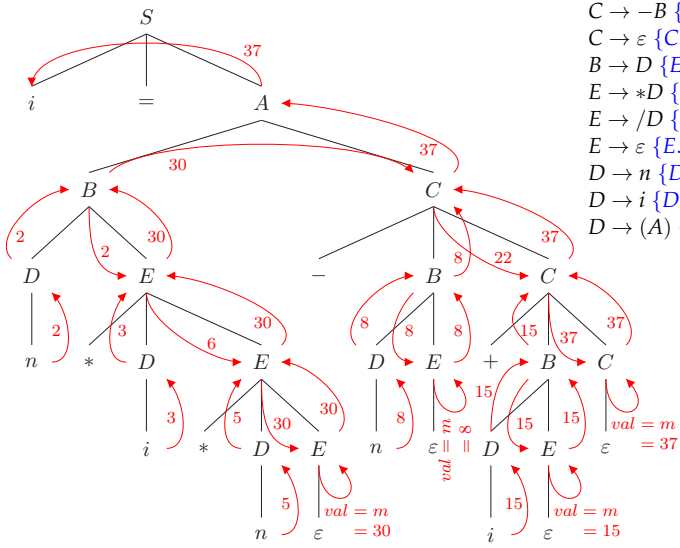
$S \to i = A$ {Uloz($i.nazev, A.val$)}
$A \to B$ $\{C.m = B.val\}$ C $\{A.val = C.val\}$
$C \to +B$ $\{C_1.m = C_0.m + B.val\}$ C $\{C_0.val = C_1.val\}$
$C \to -B$ $\{C_1.m = C_0.m - B.val\}$ C $\{C_0.val = C_1.val\}$
$C \to \varepsilon$ $\{C.val = C.m\}$
$B \to D$ $\{E.m = D.val\}$ E $\{B.val = E.val\}$
$E \to *D$ $\{E_1.m = E_0.m * D.val\}$ E $\{E_0.val = E_1.val\}$
$E \to /D$ $\{E_1.m = E_0.m/D.val\}$ E $\{E_0.val = E_1.val\}$
$E \to \varepsilon$ $\{E.val = E.m\}$
$D \to n$ $\{D.val = n.lex\}$
$D \to i$ $\{D.val =$ ZjistiHodnotu($i.nazev$)$\}$
$D \to (A)$ $\{D.val = A.val\}$



$S \to i = A$ {Uloz($i.nazev, A.val$)}
$A \to B$ {$C.m = B.val$} C {$A.val = C.val$}
$C \to +B$ {$C_1.m = C_0.m + B.val$} C {$C_0.val = C_1.val$}
$C \to -B$ {$C_1.m = C_0.m - B.val$} C {$C_0.val = C_1.val$}
$C \to \varepsilon$ {$C.val = C.m$}
$B \to D$ {$E.m = D.val$} E {$B.val = E.val$}
$E \to *D$ {$E_1.m = E_0.m * D.val$} E {$E_0.val = E_1.val$}
$E \to /D$ {$E_1.m = E_0.m / D.val$} E {$E_0.val = E_1.val$}
$E \to \varepsilon$ {$E.val = E.m$}
$D \to n$ {$D.val = n.lex$}
$D \to i$ {$D.val =$ ZjistiHodnotu($i.nazev$)}
$D \to (A)$ {$D.val = A.val$}



$S \to i = A$ {Uloz($i.nazev, A.val$)}
$A \to B$ $\{C.m = B.val\}$ C $\{A.val = C.val\}$
$C \to +B$ $\{C_1.m = C_0.m + B.val\}$ C $\{C_0.val = C_1.val\}$
$C \to -B$ $\{C_1.m = C_0.m - B.val\}$ C $\{C_0.val = C_1.val\}$
$C \to \varepsilon$ $\{C.val = C.m\}$
$B \to D$ $\{E.m = D.val\}$ E $\{B.val = E.val\}$
$E \to *D$ $\{E_1.m = E_0.m * D.val\}$ E $\{E_0.val = E_1.val\}$
$E \to /D$ $\{E_1.m = E_0.m/D.val\}$ E $\{E_0.val = E_1.val\}$
$E \to \varepsilon$ $\{E.val = E.m\}$
$D \to n$ $\{D.val = n.lex\}$
$D \to i$ {$D.val =$ ZjistiHodnotu($i.nazev$)}
$D \to (A)$ $\{D.val = A.val\}$



Implementace $LL(1)$ překladu

Oproti syntaxi rozšíříme:

- naprogramujeme všechny použité sémantické funkce
- do funkcí naprogramovaných pro syntaxi přidáme sémantiku – volání sémantických funkcí a vyhodnocování sémantických pravidel
- určíme způsob předávání atributů mezi symboly a pravidly

Použijeme metodu rekurzivního sestupu.

Terminály

- vstupní – zpracovávají se stejně jako u syntaxe (funkce `pop_in`)
- výstupní – naprogramujeme funkci `pop_out` ošetřující výstupní terminály

Atributy přenášíme

Různé typy atributů posíláme uvnitř pravidla lokálními proměnnými; mezi pravidly (předek–potomek) budeme zpracovávat takto:

- *syntetizované atributy neterminálů* posíláme rekurzí směrem nahoru (*parametry volané odkazem*),
- *dědičné atributy neterminálů* posíláme rekurzí směrem dolů (*parametry volané hodnotou*),
- *syntetizované atributy vstupních terminálů* ukládáme do globální proměnné (jsou součástí proměnné typu `TSymbol`), obvykle je zde ukládá lexikální analyzátor,
- *dědičné atributy výstupních terminálů* buď přímo vypisujeme, nebo ukládáme do vhodné globální proměnné.

Atributy přenášíme

Různé typy atributů posíláme uvnitř pravidla lokálními proměnnými; mezi pravidly (předek–potomek) budeme zpracovávat takto:

- *syntetizované atributy neterminálů* posíláme rekurzí směrem nahoru (*parametry volané odkazem*),
- *dědičné atributy neterminálů* posíláme rekurzí směrem dolů (*parametry volané hodnotou*),
- *syntetizované atributy vstupních terminálů* ukládáme do globální proměnné (jsou součástí proměnné typu `TSymbol`), obvykle je zde ukládá lexikální analyzátor,
- *dědičné atributy výstupních terminálů* buď přímo vypisujeme, nebo ukládáme do vhodné globální proměnné.

Atributy přenášíme

Různé typy atributů posíláme uvnitř pravidla lokálními proměnnými; mezi pravidly (předek–potomek) budeme zpracovávat takto:

- *syntetizované atributy neterminálů* posíláme rekurzí směrem nahoru (*parametry volané odkazem*),
- *dědičné atributy neterminálů* posíláme rekurzí směrem dolů (*parametry volané hodnotou*),
- *syntetizované atributy vstupních terminálů* ukládáme do globální proměnné (jsou součástí proměnné typu `TSymbol`), obvykle je zde ukládá lexikální analyzátor,
- *dědičné atributy výstupních terminálů* buď přímo vypisujeme, nebo ukládáme do vhodné globální proměnné.

Atributy přenášíme

Různé typy atributů posíláme uvnitř pravidla lokálními proměnnými; mezi pravidly (předek–potomek) budeme zpracovávat takto:

- *syntetizované atributy neterminálů* posíláme rekurzí směrem nahoru (*parametry volané odkazem*),
- *dědičné atributy neterminálů* posíláme rekurzí směrem dolů (*parametry volané hodnotou*),
- *syntetizované atributy vstupních terminálů* ukládáme do globální proměnné (jsou součástí proměnné typu `TSymbol`), obvykle je zde ukládá lexikální analyzátor,
- *dědičné atributy výstupních terminálů* buď přímo vypisujeme, nebo ukládáme do vhodné globální proměnné.

$S \to Av \ \{v.vysl = A.val, \text{VypisText}(v.vysl)\}$
$A \to B \ \{C.m = B.val\} \ C \ \{A.val = C.val\}$
$C \to +B \ \{C_1.m = C_0.m + B.val\} \ C \ \{C_0.val = C_1.val\}$
$C \to -B \ \{C_1.m = C_0.m - B.val\} \ C \ \{C_0.val = C_1.val\}$
$C \to \varepsilon \ \{C.val = C.m\}$
$B \to D \ \{E.m = D.val\} \ E \ \{B.val = E.val\}$
$E \to *D \ \{E_1.m = E_0.m * D.val\} \ E \ \{E_0.val = E_1.val\}$
$E \to /D \ \{E_1.m = E_0.m/D.val\} \ E \ \{E_0.val = E_1.val\}$
$E \to \varepsilon \ \{E.val = E.m\}$
$D \to n \ \{D.val = n.lex\}$
$D \to i \ \{D.val = \text{ZjistiHodnotu}(i.nazev)\}$
$D \to (A) \ \{D.val = A.val\}$

Datové typy a proměnné

```
enum TTypSymbolu { S_NOTHING, S_ENDOFFILE, S_LPAR, S_RPAR,
  S_ID, S_NUM, S_IS, S_PLUS, S_MINUS, S_MUL, S_DIV };
```

```
struct TSymbol {
  TTypSymbolu typ;
    int atribcislo;            // celé číslo (S_NUM)
  TObjekt *atribstr;          // název proměnné (S_ID)
};
```

```
TSymbol symbol;
...
```

Ošetření chyb a pomocné funkce pro výstup

```
void error(string hlaska) {
  konec = true;
  printf("Chyba při syntaktické analýze na řádku %d, sloupci %d: %s",
    vstup.cisloRad, vstup.pozice, hlaska);
}
```

Pomocné funkce pro výpis

- `VypisTyp(TTypSymbolu typ)` – převede datový typ na řetězec
- `VypisHodn(TSymbol sym)` – převede symbol včetně atributu na řetězec

Vstupní a výstupní terminály

```
int pop_in(TSymbol terminal) {
  if (symbol.typ == terminal.typ)
    Lex();
  else error("chybný symbol na vstupu -" +VypisTyp(symbol.typ));
}
```

```
int pop_out(TSymbol terminal) {
  if (symbol.typ == terminal.typ) {     // případně switch
     ...        // například výpis atributu na výstup
  } // chyba není pravděpodobná, ale taky můžeme ošetřit
}
```

Pravidla

$S \rightarrow Av \ \ \{v.vysl = A.val, \text{VypisText}(v.vysl)\}$

```
void S(){
  TSymbol pom_sym;
  if (symbol.typ==S_NUM || symbol.typ==S_ID || symbol.typ==S_LZAV) {
    A(pom_sym);
    pom_sym.typ = S_VYSTUP;    // výstupní terminál 'v'
    pop_out(pom_sym);
  }
  else
    error("symbol "+VypisHodn(symbol)+" není očekávaného typu "+
    VypisTyp(S_NUM)+"+ "+VypisTyp(S_ID)+" nebo "+VypisTyp(S_LZAV));
}
```

Pravidla

$A \rightarrow B \ \{C.m = B.val\} \ C \ \{A.val = C.val\}$

```
void A(int &val) {
  int pomval;
  if (symbol.typ==S_NUM || symbol.typ==S_ID || symbol.typ==S_LZAV) {
    B(pomval);
    C(pomval,val);
  }
  else
    error("symbol "+VypisHodn(symbol)+" není očekávaného typu "+
    VypisTyp(S_NUM)+"+ "+VypisTyp(S_ID)+" nebo "+VypisTyp(S_LZAV));
}
```

Pravidla

$C \to +B \ \{C_1.m = C_0.m + B.val\} \ C \ \{C_0.val = C_1.val\}$
$C \to -B \ \{C_1.m = C_0.m - B.val\} \ C \ \{C_0.val = C_1.val\}$
$C \to \varepsilon \ \{C.val = C.m\}$

```
void C(int m, int &val) {
int pomval;
  switch (symbol.typ) {
    case S_PLUS:
      pop_in(S_PLUS);
      B(pomval);
      C(m + pomval,val); break;
    case S_MINUS:
      pop_in(S_MINUS);
      B(pomval);
      C(m - pomval,val); break;
    case S_RZAV: case S_ENDOFFILE: val = m; break;
    default: error("symbol "+VypisHodn(symbol)+" není očekávaného typu "+
      VypisTyp(S_PLUS)+"+ "+VypisTyp(S_MINUS)+"+ "+VypisTyp(S_RZAV)+
      " nebo konec vstupu");
  }
```

Pravidla

$B \rightarrow D \ \{E.m = D.val\} \ E \ \{B.val = E.val\}$

```
void B(int &val) {
  int pomval;
  if (symbol.typ==S_NUM || symbol.typ==S_ID || symbol.typ==S_LZAV) {
    D(pomval);
    E(pomval,val);
  }
  else error(...);
}
```

Pravidla

$E \to *D \ \{E_1.m = E_0.m * D.val\} \ E \ \{E_0.val = E_1.val\}$
$E \to /D \ \{E_1.m = E_0.m/D.val\} \ E \ \{E_0.val = E_1.val\}$
$E \to \varepsilon \ \{E.val = E.m\}$

```
void E(int m, int &val) {
  int pomval;
  switch (symbol.typ) {
    case S_MUL:
      pop_in(S_MUL);
      D(pomval);
      E(m * pomval,val); break;
    case S_DIV:
      pop_in(S_DIV);
      D(pomval);
      if (pomval == 0) error("dělení nulou"); else E(m/pomval,val);
      break;
    case S_PLUS: case S_MINUS: case S_RZAV: case S_ENDOFFILE:
      val = m; break;
    default: error(...);
  }
}
```

Pravidla

$D \to n \ \{D.val = n.lex\}$
$D \to i \ \{D.val = \text{ZjistiHodnotu}(i.nazev)\}$
$D \to (A) \ \{D.val = A.val\}$

```
void D(int &val) {
  switch (symbol.typ) {
    case S_NUM:
      val = symbol.atribcis;  // musíme zachytit předem, pak bude
      pop_in(S_NUM); break;   // přepsáno lexikálním analyzátorem
    case S_ID:
      val = symbol.atribstr->hodnota.i;   // z tabulky symbolů
      pop_in(S_ID); break;
    case S_LZAV:
      pop_in(LZAV);
      A(val);
      pop_in(RZAV); break;
    default: error(...);
  }
}
```

Hlavní funkce

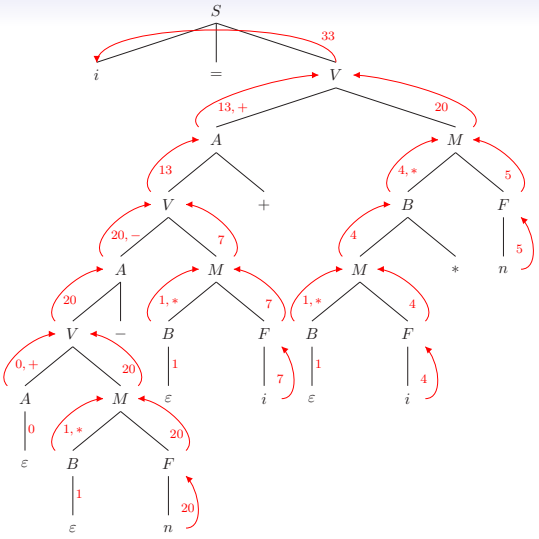
```
void Preklad() {
  init();      // inicializace překladu včetně lexikálního analyzátoru
  Lex();       // přednačteme jeden symbol
  S();         // spustíme rekurzivní volání
  done();      // ukončení překladu, úklid paměti apod.
}
```

Příklad – silná $LR(1)$ gramatika

$S \to i = V$
$V \to AM$
$A \to V+$
$A \to V-$
$A \to \varepsilon$
$M \to BF$
$B \to M*$
$B \to M/$
$B \to \varepsilon$
$F \to n$
$F \to i$
$F \to (V)$

Příklad – silná $LR(1)$ gramatika

$S \to i = V \ \{\text{Uloz}(i.nazev, V.val)\}$
$V \to AM \ \{\text{if } A.op = \text{S\_PLUS} \text{ then } V.val = A.val + M.val$
$\text{else } V.val = A.val - M.val\}$
$A \to V + \ \{A.val = V.val, A.op = \text{S\_PLUS}\}$
$A \to V - \ \{A.val = V.val, A.op = \text{S\_MINUS}\}$
$A \to \varepsilon \ \{A.val = 0, A.op = \text{S\_PLUS}\}$
$M \to BF \ \{\text{if } B.op = \text{S\_MUL} \text{ then } M.val = B.val * F.val$
$\text{else } M.val = B.val / F.val\}$
$B \to M * \ \{B.val = M.val, B.op = \text{S\_MUL}\}$
$B \to M / \ \{B.val = M.val, B.op = \text{S\_DIV}\}$
$B \to \varepsilon \ \{B.val = 1, B.op = \text{S\_MUL}\}$
$F \to n \ \{F.val = n.lex\}$
$F \to i \ \{F.val = \text{ZjistiHodnotu}(i.nazev)\}$
$F \to (V) \ \{F.val = V.val\}$



$S \to i = V$ $\{\text{Uloz}(i.nazev, V.val)\}$
$V \to AM$ $\{\text{if } A.op = \text{S\_PLUS} \text{ then } V.val = A.val + M.val$
$\text{else } V.val = A.val - M.val\}$
$A \to V +$ $\{A.val = V.val, A.op = \text{S\_PLUS}\}$
$A \to V -$ $\{A.val = V.val, A.op = \text{S\_MINUS}\}$
$A \to \varepsilon$ $\{A.val = 0, A.op = \text{S\_PLUS}\}$
$M \to BF$ $\{\text{if } B.op = \text{S\_MUL} \text{ then } M.val = B.val * F.val$
$\text{else } M.val = B.val / F.val\}$
$B \to M *$ $\{B.val = M.val, B.op = \text{S\_MUL}\}$
$B \to M /$ $\{B.val = M.val, B.op = \text{S\_DIV}\}$
$B \to \varepsilon$ $\{B.val = 1, B.op = \text{S\_MUL}\}$
$F \to n$ $\{F.val = n.lex\}$
$F \to i$ $\{F.val = \text{ZjistiHodnotu}(i.nazev)\}$
$F \to (V)$ $\{F.val = V.val\}$

Postup

- použijeme metodu přepisu rozkladové tabulky se zásobníkem
- do zásobníku ukládáme celé symboly včetně atributů
- atributy jsou jen syntetizované

$S' \to \#S$ ⓪

$S \to i = V \ \{Uloz(i.nazev, V.val)\}$ ①

$V \to AM \ \{\text{if } A.op = \text{S\_PLUS} \text{ then } V.val = A.val + M.val \text{ else } V.val = A.val - M.val\}$ ②

$A \to V + \ \{A.val = V.val, A.op = \text{S\_PLUS}\}$ ③

$A \to V - \ \{A.val = V.val, A.op = \text{S\_MINUS}\}$ ④

$A \to \varepsilon \ \{A.val = 0, A.op = \text{S\_PLUS}\}$ ⑤

$M \to BF \ \{\text{if } B.op = \text{S\_MUL} \text{ then } M.val = B.val * F.val \text{ else } M.val = B.val/F.val\}$ ⑥

$B \to M * \ \{B.val = M.val, B.op = \text{S\_MUL}\}$ ⑦

$B \to M / \ \{B.val = M.val, B.op = \text{S\_DIV}\}$ ⑧

$B \to \varepsilon \ \{B.val = 1, B.op = \text{S\_MUL}\}$ ⑨

$F \to n \ \{F.val = n.lex\}$ ⑩

$F \to i \ \{F.val = \text{ZjistiHodnotu}(i.nazev)\}$ ⑪

$F \to (V) \ \{F.val = V.val\}$ ⑫

Datové typy

```
enum TTypSymbolu { S_NOTHING, S_ENDOFFILE, S_LPAR, S_RPAR,
   S_ID, S_NUM, S_IS, S_PLUS, S_MINUS, S_MUL, S_DIV,      // terminály
   S_NS, S_NSC, S_NV, S_NA, S_NM, S_NB, S_NF, S_HASH }; // neterminály
```

```
struct TSymbol {
    TTypSymbolu typ;
    int atribcislo;
    TObjekt *atribstr;
};
```

```
struct TSymbolZasob {
    TTypSymbolu typ;
    TTypSymbolu atribop;// pro S_PLUS, ..., S_DIV
    int atribcislo;     // atributy val a lex
    TObjekt *atribstr;  // atribut nazev
};
```

```
bool konec;                  // indikátor ukončení výpočtu
TSymbol symbol;              // aktuální symbol ze vstupu
TSymbolZasob vrchol_zas;     // symbol na vrcholu zásobníku
TZasobnik zasobnik;          // prvky jsou typu TSymbolZasob
  ...              // další používané datové typy a proměnné
```

Pravidla

```
void reduce(int cislo_prav) {
  TSymbolZasob SymbolZas;
  int val;
  switch (cislo_prav) {
    case 0: ...
    case 1: ...
    atd.
  }
}
```

doplníme podle pravidel

Pravidla

$S' \to \#S$ ⓪

```
case 0:
  Vyjmi_ze_zasobniku(vrchol_zas);                    // S
  Vyjmi_ze_zasobniku(vrchol_zas);                    // #
  SymbolZas.typ = S_NSC;
  Pridej_do_zasobniku(SymbolZas);
  break;
```

Pravidla

$S \rightarrow i = V \ \{Uloz(i.nazev, V.val)\}$ ①

```
case 1:
  Vyjmi_ze_zasobniku(vrchol_zas);                    // V
  val = vrchol_zas.atribcislo;
  Vyjmi_ze_zasobniku(vrchol_zas);                    // =
  Vyjmi_ze_zasobniku(vrchol_zas);                    // i
  // uložíme vypočtenou hodnotu do tabulky symbolů:
  vrchol_zas.atribstr->(hodnota.i) = val;
  SymbolZas.typ = S_NS;
  Pridej_do_zasobniku(SymbolZas);
  break;
```

Pravidla

$V \rightarrow AM$ $\{$if $A.op =$ S_PLUS then $V.val = A.val + M.val$ else $V.val = A.val - M.val\}$ ②

```
case 2:
  Vyjmi_ze_zasobniku(vrchol_zas);                    // M
  val = vrchol_zas.atribcislo;
  Vyjmi_ze_zasobniku(vrchol_zas);                    // A
  SymbolZas.typ = S_NV;
  if (vrchol_zas.atribop == S_PLUS)
    SymbolZas.atribcislo =  vrchol_zas.atribcislo + val;
  else if (vrchol_zas.atribop == S_MINUS)
    SymbolZas.atribcislo =  vrchol_zas.atribcislo - val;
  else error("chyba v syntaxi aritmetického výrazu, nenalezen žádný
    ze symbolů "+ VypisTyp(S_PLUS)+" nebo "+VypisTyp(S_MINUS));
  Pridej_do_zasobniku(SymbolZas);
  break;
```

Pravidla

$A \rightarrow V +\ \ \{A.val = V.val, A.op = \text{S\_PLUS}\}$ ③

```
case 3:
  Vyjmi_ze_zasobniku(vrchol_zas);                    // +
  Vyjmi_ze_zasobniku(vrchol_zas);                    // V
  SymbolZas.typ = S_NA;
  SymbolZas.atribop = S_PLUS;
  SymbolZas.atribcislo = vrchol_zas.atribcislo;
  Pridej_do_zasobniku(SymbolZas);
  break;
```

Pravidla

$A \to V -\quad \{A.val = V.val, A.op = \text{S\_MINUS}\}$ ④

```
case 4:
  Vyjmi_ze_zasobniku(vrchol_zas);                    // -
  Vyjmi_ze_zasobniku(vrchol_zas);                    // V
  SymbolZas.typ = S_NA;
  SymbolZas.atribop = S_MINUS;
  SymbolZas.atribcislo = vrchol_zas.atribcislo;
  Pridej_do_zasobniku(SymbolZas);
  break;
```

Pravidla

$A \to \varepsilon \ \{A.val = 0, A.op = \text{S\_PLUS}\}$ ⑤

```
case 5:
  SymbolZas.typ = S_NA;
  SymbolZas.atribop = S_PLUS;
  SymbolZas.atribcislo = 0;
  Pridej_do_zasobniku(SymbolZas);
  break;
... // pro symboly M a B to bude podobné jako pro V a A,
    // ale nesmíme zapomenout ošetřit dělení nulou
```

Pravidla

$F \rightarrow n \ \{F.val = n.lex\}$ ⑩

```
case 10:
  Vyjmi_ze_zasobniku(vrchol_zas);                    // n
  SymbolZas.typ = S_NF;
  SymbolZas.atribcislo = vrchol_zas.atribcislo;
  Pridej_do_zasobniku(SymbolZas);
  break;
```

Pravidla

$F \rightarrow i \quad \{F.val = \text{ZjistiHodnotu}(i.nazev)\}$ (11)

```
case 11:
  Vyjmi_ze_zasobniku(vrchol_zas);                    // i
  SymbolZas.typ = S_NF;
  SymbolZas.atribcislo = vrchol_zas.atribstr->hodnota.i;
  Pridej_do_zasobniku(SymbolZas);
  break;
```

Pravidla

$F \rightarrow (V) \ \{F.val = V.val\}$ ⑫

```
case 12:
  Vyjmi_ze_zasobniku(vrchol_zas);                  // (
  Vyjmi_ze_zasobniku(vrchol_zas);                  // V
  SymbolZas.typ = S_NF;
  SymbolZas.atribcislo = vrchol_zas.atribcislo;
  Vyjmi_ze_zasobniku(vrchol_zas);                  // )
  Pridej_do_zasobniku(SymbolZas);
  break;
```

Rozhodování mezi pravidly podle tabulky

	n	i	$=$	$+$	$-$	$*$	$/$	$($	$)$	$\$$
S'										*acc*
S										r0
V				*push*	*push*				*push*	r1
A	r9	r9						r9		
M				r2	r2	*push*	*push*		r2	r2
B	*push*	*push*						*push*		
F				r6	r6	r6	r6		r6	r6
n				r10	r10	r10	r10		r10	r10
i			*push*	r11	r11	r11	r11		r11	r11
$=$	r5	r5						r5		
$+$	r3	r3						r3		
$-$	r4	r4						r4		
$*$	r7	r7						r7		
$/$	r8	r8						r8		
$($	r5	r5						r5		
$)$				r12	r12	r12	r12		r12	r12
$\#$		*push*								

Rozhodování mezi pravidly podle tabulky

	n	i	$=$	$+$	$-$	$*$	$/$	$($	$)$	\$
S'										*acc*
S										r0

```
void Akce() {
  switch (vrchol_zas) {
    case S_NSC: if (symbol.typ == S_ENDOFFILE) accept();
      else error("očekáván konec souboru");
      break;
    case S_NS: if (symbol.typ == S_ENDOFFILE) reduce(0);
      else error("očekáván konec souboru");
      break;
```

Rozhodování mezi pravidly podle tabulky

	n	i	$=$	$+$	$-$	$*$	$/$	$($	$)$	$
V				*push*	*push*				*push*	r1

```
case S_NV: switch (symbol.typ) {
    case S_PLUS: case S_MINUS: case S_RZAV:
      push();
      break;
    case S_ENDOFFILE:
      reduce(1);
      break;
    default:
      error("symbol "+VypisHodn(symbol)+" není očekávaného typu "+
      VypisTyp(S_PLUS)+", "+VypisTyp(S_MINUS),"+ "+
      VypisTyp(S_RZAV)+" nebo konec souboru");
  }
  break;
  ...          // atd. pro všechny neterminály
```

Pravidla

	n	i	$=$	$+$	$-$	$*$	$/$	$($	$)$	\$
n				r10	r10	r10	r10		r10	r10
i			*push*	r11	r11	r11	r11		r11	r11

```
case S_NUM:
  if (symbol.typ==S_PLUS || symbol.typ==S_MINUS || symbol.typ==S_MUL
    || symbol.typ==S_DIV || symbol.typ==S_RZAV || symbol.typ==S_ENDOFFILE)
    reduce(10);
  else error(...);
  break;
case S_ID: switch (symbol.typ) {
    case S_PLUS: case S_MINUS: case S_MUL: case S_DIV:
    case S_RZAV: case S_ENDOFFILE:
      reduce(11); break;
    case S_ROVNASE: push(); break;
    default: error(...);
  }
  ... // atd. pro všechny terminály
```

Ošetření chyb a akceptování

```
void error(string hlaska) {
  konec = true;
  printf("Chyba při syntaktické analýze na řádku %d, sloupci %d: %s",
    vstup.cisloRad, vstup.pozice, hlaska);
}

void accept() {
  konec = true;
}
```

Práce se vstupem

```
int push() {
  SymbolZas.typ = symbol.typ
  SymbolZas.atribcislo = symbol.atribcislo;
  SymbolZas.atribstr=symbol.atribstr;
  Pridej_do_zasobniku(symbol.typ);
  lex();          // lexikální analyzátor načte další symbol
}
```

Inicializace a ukončení

```
void Init() {
  TSymbolZas SymbolZas;
  ...                              // inicializace vstupu a výstupu
  konec = false;
  Vytvor_zasobnik();
  SymbolZas.typ = S_HASH;
  Pridej_do_zasobniku(SymbolZas); // symbol konce zásobníku
  lex();                           // načte symbol ze vstupu do symbol
}
```

```
void Done() {
  Zlikviduj_zasobnik();          // uvolní paměť zabranou zásobníkem
  ...                            // uzavření vstupu a výstupu
}
```

```
void Preklad() {
  Init();
  while(!konec) Akce();
  Done();
}
```