

Laboratorio di Linguaggi Formali e Traduttori

LFT lab T2, a.a. 2020/2021

Traduzione diretta dalla sintassi

Traduzione diretta dalla sintassi

- Obiettivo: dare un *significato* al input.
 - Ad esempio, la traduzione di un programma in un altro linguaggio (le tre lezioni che seguono la lezione di oggi), la valutazione di un'espressione aritmetica (questa lezione),
- *Definizioni diretta dalla sintassi (SDD)*:
 - Grammatica
 - + attributi (associati alle variabili della grammatica)
 - + regole semantiche (come calcolare il valore degli attributi; associate con le produzioni della grammatica).
- *Schema di traduzione (SDT)*:
 - Azioni semantiche:
 - Frammenti di codice inseriti nelle produzioni.
 - Possono contenere, oltre ad azioni per calcolare il valore di attributi, anche codice arbitrario.
 - Ciascun SDD L-attribuite può essere convertito in uno SDT (l'ordine di valutazione degli attributi è esplicito).
 - Adattato ad essere integrato in un parser ricorsivo discendente.

Traduzione diretta dalla sintassi

- Esercizio 4.1: modificare l'analizzatore sintattico dell'esercizio 3.1 in modo da valutare le espressioni aritmetiche semplici, facendo riferimento allo SDT seguente:

$$\begin{aligned}\langle start \rangle &::= \langle expr \rangle \text{ EOF } \{ print(expr.val) \} \\ \langle expr \rangle &::= \langle term \rangle \{ exprp.i = term.val \} \langle exprp \rangle \{ expr.val = exprp.val \} \\ \langle exprp \rangle &::= \begin{array}{l} + \langle term \rangle \{ exprp_1.i = exprp.i + term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \} \\ | \\ - \langle term \rangle \{ exprp_1.i = exprp.i - term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \} \\ | \\ \varepsilon \{ exprp.val = exprp.i \} \end{array} \\ \langle term \rangle &::= \langle fact \rangle \{ termp.i = fact.val \} \langle termp \rangle \{ term.val = termp.val \} \\ \langle termp \rangle &::= \begin{array}{l} * \langle fact \rangle \{ termp_1.i = termp.i * fact.val \} \langle termp_1 \rangle \{ termp.val = termp_1.val \} \\ | \\ / \langle fact \rangle \{ termp_1.i = termp.i / fact.val \} \langle termp_1 \rangle \{ termp.val = termp_1.val \} \\ | \\ \varepsilon \{ termp.val = termp.i \} \end{array} \\ \langle fact \rangle &::= (\langle expr \rangle) \{ fact.val = expr.val \} \mid \text{ NUM } \{ fact.val = \text{NUM.value} \} \end{aligned}$$

Traduzione diretta dalla sintassi

Grammatica + azioni semantiche

Variabili e produzioni (esempio $\langle start \rangle ::= \langle expr \rangle \text{ EOF } \{ \}$)

$\langle start \rangle$	$::=$	$\langle expr \rangle \text{ EOF } \{ print(expr.val) \}$
$\langle expr \rangle$	$::=$	$\langle term \rangle \{ exprp.i = term.val \} \langle exprp \rangle \{ expr.val = exprp.val \}$
$\langle exprp \rangle$	$::=$	$+ \langle term \rangle \{ exprp_1.i = exprp.i + term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \}$ $ $ $- \langle term \rangle \{ exprp_1.i = exprp.i - term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \}$ $ $ $\varepsilon \{ exprp.val = exprp.i \}$
$\langle term \rangle$	$::=$	$\langle fact \rangle \{ termp.i = fact.val \} \langle termp \rangle \{ term.val = termp.val \}$
$\langle termp \rangle$	$::=$	$* \langle fact \rangle \{ termp_1.i = termp.i * fact.val \} \langle termp_1 \rangle \{ termp.val = termp_1.val \}$ $ $ $/ \langle fact \rangle \{ termp_1.i = termp.i / fact.val \} \langle termp_1 \rangle \{ termp.val = termp_1.val \}$ $ $ $\varepsilon \{ termp.val = termp.i \}$
$\langle fact \rangle$	$::=$	$(\langle expr \rangle) \{ fact.val = expr.val \} \mid \text{NUM} \{ fact.val = \text{NUM.value} \}$

Traduzione diretta dalla sintassi

Grammatica + azioni semantiche

Associate con le produzioni (scritte in verde; esempio $\{ \text{expr.val} = \text{exprp.val} \}$)

$\langle \text{start} \rangle$	$::=$	$\langle \text{expr} \rangle \text{ EOF } \{ \text{print}(\text{expr.val}) \}$
$\langle \text{expr} \rangle$	$::=$	$\langle \text{term} \rangle \{ \text{exprp.i} = \text{term.val} \} \langle \text{exprp} \rangle \{ \text{expr.val} = \text{exprp.val} \}$
$\langle \text{exprp} \rangle$	$::=$	$\begin{array}{l} + \langle \text{term} \rangle \{ \text{exprp}_1.\text{i} = \text{exprp.i} + \text{term.val} \} \langle \text{exprp}_1 \rangle \{ \text{exprp.val} = \text{exprp}_1.\text{val} \} \\ \\ - \langle \text{term} \rangle \{ \text{exprp}_1.\text{i} = \text{exprp.i} - \text{term.val} \} \langle \text{exprp}_1 \rangle \{ \text{exprp.val} = \text{exprp}_1.\text{val} \} \\ \\ \varepsilon \{ \text{exprp.val} = \text{exprp.i} \} \end{array}$
$\langle \text{term} \rangle$	$::=$	$\langle \text{fact} \rangle \{ \text{termp.i} = \text{fact.val} \} \langle \text{termp} \rangle \{ \text{term.val} = \text{termp.val} \}$
$\langle \text{termp} \rangle$	$::=$	$\begin{array}{l} * \langle \text{fact} \rangle \{ \text{termp}_1.\text{i} = \text{termp.i} * \text{fact.val} \} \langle \text{termp}_1 \rangle \{ \text{termp.val} = \text{termp}_1.\text{val} \} \\ \\ / \langle \text{fact} \rangle \{ \text{termp}_1.\text{i} = \text{termp.i} / \text{fact.val} \} \langle \text{termp}_1 \rangle \{ \text{termp.val} = \text{termp}_1.\text{val} \} \\ \\ \varepsilon \{ \text{termp.val} = \text{termp.i} \} \end{array}$
$\langle \text{fact} \rangle$	$::=$	$(\langle \text{expr} \rangle) \{ \text{fact.val} = \text{expr.val} \} \mid \text{NUM} \{ \text{fact.val} = \text{NUM.value} \}$

Traduzione diretta dalla sintassi

Grammatica + azioni semantiche

Valori di eventuali attributi dei terminali: fornito dal lexer (esempio $\{ fact.val = NUM.value \}$)

```

$$\begin{aligned} \langle start \rangle &::= \langle expr \rangle \text{ EOF } \{ print(expr.val) \} \\ \langle expr \rangle &::= \langle term \rangle \{ exprp.i = term.val \} \langle exprp \rangle \{ expr.val = exprp.val \} \\ \langle exprp \rangle &::= \begin{array}{l} + \langle term \rangle \{ exprp_1.i = exprp.i + term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \} \\ | \\ - \langle term \rangle \{ exprp_1.i = exprp.i - term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \} \\ | \\ \varepsilon \{ exprp.val = exprp.i \} \end{array} \\ \langle term \rangle &::= \langle fact \rangle \{ termp.i = fact.val \} \langle termp \rangle \{ term.val = termp.val \} \\ \langle termp \rangle &::= \begin{array}{l} * \langle fact \rangle \{ termp_1.i = termp.i * fact.val \} \langle termp_1 \rangle \{ termp.val = termp_1.val \} \\ | \\ / \langle fact \rangle \{ termp_1.i = termp.i / fact.val \} \langle termp_1 \rangle \{ termp.val = termp_1.val \} \\ | \\ \varepsilon \{ termp.val = termp.i \} \end{array} \\ \langle fact \rangle &::= ( \langle expr \rangle ) \{ fact.val = expr.val \} \mid \text{ NUM } \{ fact.val = NUM.value \} \end{aligned}$$

```

Richiamo teoria

- Integrazione di uno SDT in un parser ricorsivo discendente (dal slide 7 del file «26_sdt» con titolo «5.2 - Schemi di traduzione»/videoregistrazione «26a Schemi di traduzione (SDT); traduzione on-the-fly» sulla pagina LFT (teoria)).

- Il parser ha una procedura per ogni variabile della grammatica che riconosce le stringhe generate da A nella grammatica.
- La procedura A ha tanti argomenti quanti sono gli attributi ereditati di A e restituisce tanti valori quanti sono gli attributi sintetizzati di A .
- La procedura A usa il simbolo corrente e gli insiemi guida, per scegliere la produzione $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$ da usare per riscrivere A .
- Per ogni simbolo o azione semantica X nel corpo della produzione scelta:
 - Se X è un simbolo terminale, il metodo controlla che il simbolo corrente sia proprio X . In tal caso, fa avanzare il lexer al simbolo successivo. In caso contrario, il metodo segnala un errore di sintassi.
 - Se X è una variabile, il metodo invoca la procedura X passando a X come argomenti i suoi attributi ereditati e raccogliendo in variabili locali gli attributi sintetizzati restituiti da X .
 - Se X è una azione semantica, il metodo la esegue.

Dallo SDT al codice

- Il parser ha una procedura per ogni variabile della grammatica che riconosce le stringhe generate da A nella grammatica.

$\langle start \rangle$	$::= \langle expr \rangle EOF \{ print(expr.val) \}$
$\langle expr \rangle$	$::= \langle term \rangle \{ exprp.i = term.val \} \langle exprp \rangle \{ expr.val = exprp.val \}$
$\langle exprp \rangle$	$::= + \langle term \rangle \{ exprp_1.i = exprp.i + term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \}$ $- \langle term \rangle \{ exprp_1.i = exprp.i - term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \}$ $\varepsilon \{ exprp.val = exprp.i \}$
$\langle term \rangle$	$::= \langle fact \rangle \{ termp.i = fact.val \} \langle termp \rangle \{ term.val = termp.val \}$
$\langle termp \rangle$	$::= * \langle fact \rangle \{ termp_1.i = termp.i * fact.val \} \langle termp_1 \rangle \{ termp.val = termp_1.val \}$ $/ \langle fact \rangle \{ termp_1.i = termp.i / fact.val \} \langle termp_1 \rangle \{ termp.val = termp_1.val \}$ $\varepsilon \{ termp.val = termp.i \}$
$\langle fact \rangle$	$::= (\langle expr \rangle) \{ fact.val = expr.val \} \mid NUM \{ fact.val = NUM.value \}$

```
public void start() {
    int expr_val;
    // ... completare ...
    expr_val = expr();
    match(Tag.EOF);
    System.out.println(expr_val);
    // ... completare ...
}

private int expr() {
    int term_val, exprp_val;
    // ... completare ...
    term_val = term();
    exprp_val = exprp(term_val);
    // ... completare ...
    return exprp_val;
}

private int exprp(int exprp_i) {
    int term_val, exprp_val;
    switch (look.tag) {
        case '+':
            match('+');
            term_val = term();
            exprp_val = exprp(exprp_i + term_val);
            break;
        // ... completare ...
    }
}

private int term() {
    // ... completare ...
}

private int termp(int termp_i) {
    // ... completare ...
}

private int fact() {
    // ... completare ...
}
```


Dallo SDT al codice

- La procedura **A** ha tanti argomenti quanti sono gli attributi ereditati di **A** e restituisce tanti valori quanti sono gli attributi sintetizzati di **A**.

```
 $\langle start \rangle ::= \langle expr \rangle EOF \{ print(expr.val) \}$   
 $\langle expr \rangle ::= \langle term \rangle \{ exprp.i = term.val \} \langle exprp \rangle \{ expr.val = exprp.val \}$   
 $\langle exprp \rangle ::=$   
     $+ \langle term \rangle \{ exprp_1.i = exprp.i + term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \}$   
     $- \langle term \rangle \{ exprp_1.i = exprp.i - term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \}$   
     $\varepsilon \{ exprp.val = exprp.i \}$   
 $\langle term \rangle ::= \langle fact \rangle \{ termp.i = fact.val \} \langle termp \rangle \{ term.val = termp.val \}$   
 $\langle termp \rangle ::=$   
     $* \langle fact \rangle \{ termp_1.i = termp.i * fact.val \} \langle termp_1 \rangle \{ termp.val = termp_1.val \}$   
     $/ \langle fact \rangle \{ termp_1.i = termp.i / fact.val \} \langle termp_1 \rangle \{ termp.val = termp_1.val \}$   
     $\varepsilon \{ termp.val = termp.i \}$   
 $\langle fact \rangle ::= ( \langle expr \rangle ) \{ fact.val = expr.val \} \mid NUM \{ fact.val = NUM.value \}$ 
```

```
private int expr() {  
    int term_val, exprp_val;  
    // ... completare ...  
    term_val = term();  
    exprp_val = exprp(term_val);  
    // ... completare ...  
    return exprp_val;  
}  
  
private int exprp(int exprp_i) {  
    int term_val, exprp_val;  
    switch (look.tag) {  
        case '+':  
            match('+');  
            term_val = term();  
            exprp_val = exprp(exprp_i + term_val);  
            break;  
        // ... completare ...  
    }  
}
```

Dallo SDT al codice

- La procedura A ha tanti argomenti quanti sono gli attributi ereditati di A e restituisce tanti valori quanti sono gli attributi sintetizzati di A .

```
 $\langle start \rangle ::= \langle expr \rangle EOF \{ print(expr.val) \}$   
 $\langle expr \rangle ::= \langle term \rangle \{ exprp.i = term.val \} \langle exprp \rangle \{ expr.val = exprp.val \}$   
 $\langle exprp \rangle ::=$   
     $+ \langle term \rangle \{ exprp_1.i = exprp.i + term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \}$   
     $- \langle term \rangle \{ exprp_1.i = exprp.i - term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \}$   
     $\varepsilon \{ exprp.val = exprp.i \}$   
 $\langle term \rangle ::= \langle fact \rangle \{ termp.i = fact.val \} \langle termp \rangle \{ term.val = termp.val \}$   
 $\langle termp \rangle ::=$   
     $* \langle fact \rangle \{ termp_1.i = termp.i * fact.val \} \langle termp_1 \rangle \{ termp.val = termp_1.val \}$   
     $/ \langle fact \rangle \{ termp_1.i = termp.i / fact.val \} \langle termp_1 \rangle \{ termp.val = termp_1.val \}$   
     $\varepsilon \{ termp.val = termp.i \}$   
 $\langle fact \rangle ::= ( \langle expr \rangle ) \{ fact.val = expr.val \} \mid NUM \{ fact.val = NUM.value \}$ 
```

```
private int expr() {  
    int term_val, exprp_val;  
    // ... completare ...  
    term_val = term();  
    exprp_val = exprp(term_val);  
    // ... completare ...  
    return exprp_val;  
}  
  
private int exprp(int exprp_i) {  
    int term_val, exprp_val;  
    switch (look.tag) {  
        case '+':  
            match('+');  
            term_val = term();  
            exprp_val = exprp(exprp_i + term_val);  
            break;  
        // ... completare ...  
    }  
}
```

Dallo SDT al codice

- La procedura A usa il simbolo corrente e gli insiemi guida, per scegliere la produzione $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$ da usare per riscrivere A .

```
 $\langle start \rangle ::= \langle expr \rangle \text{ EOF } \{ print(expr.val) \}$   
 $\langle expr \rangle ::= \langle term \rangle \{ exprp.i = term.val \} \langle exprp \rangle \{ expr.val = exprp.val \}$   
 $\langle exprp \rangle ::=$   
     $+ \langle term \rangle \{ exprp_1.i = exprp.i + term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \}$   
     $- \langle term \rangle \{ exprp_1.i = exprp.i - term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \}$   
     $\varepsilon \{ exprp.val = exprp.i \}$   
 $\langle term \rangle ::= \langle fact \rangle \{ termp.i = fact.val \} \langle termp \rangle \{ term.val = termp.val \}$   
 $\langle termp \rangle ::=$   
     $* \langle fact \rangle \{ termp_1.i = termp.i * fact.val \} \langle termp_1 \rangle \{ termp.val = termp_1.val \}$   
     $/ \langle fact \rangle \{ termp_1.i = termp.i / fact.val \} \langle termp_1 \rangle \{ termp.val = termp_1.val \}$   
     $\varepsilon \{ termp.val = termp.i \}$   
 $\langle fact \rangle ::= ( \langle expr \rangle ) \{ fact.val = expr.val \} \mid \text{ NUM } \{ fact.val = \text{ NUM.value } \}$ 
```

- L'utilizzo del token corrente e gli insiemi guida per scegliere una produzione è esattamente come per il parsing ricorsivo discendente (Sezione 3 del documento degli esercizi).

- Per ogni simbolo o azione semantica X nel corpo della produzione scelta:

- Se X è un simbolo terminale, il metodo controlla che il simbolo corrente sia proprio X . In tal caso, fa avanzare il lexer al simbolo successivo. In caso contrario, il metodo segnala un errore di sintassi.
- Se X è una variabile, il metodo invoca la procedura X passando a X come argomenti i suoi attributi ereditati e raccogliendo in variabili locali gli attributi sintetizzati restituiti da X .
- Se X è una azione semantica, il metodo la esegue.

$\langle start \rangle ::= \langle expr \rangle EOF \{ print(expr.val) \}$

$\langle expr \rangle ::= \langle term \rangle \{ exprp.i = term.val \} \langle exprp \rangle \{ expr.val = exprp.val \}$

$\langle exprp \rangle ::= \boxed{+} \langle term \rangle \{ exprp_1.i = exprp.i + term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \}$
 $\quad \quad \quad | \quad - \langle term \rangle \{ exprp_1.i = exprp.i - term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \}$
 $\quad \quad \quad | \quad \varepsilon \{ exprp.val = exprp.i \}$

$\langle term \rangle ::= \langle fact \rangle \{ term.p.i = fact.val \} \langle term.p \rangle \{ term.val = term.p.val \}$

$\langle term.p \rangle ::= * \langle fact \rangle \{ term.p_1.i = term.p.i * fact.val \} \langle term.p_1 \rangle \{ term.p.val = term.p_1.val \}$
 $\quad \quad \quad | \quad / \langle fact \rangle \{ term.p_1.i = term.p.i / fact.val \} \langle term.p_1 \rangle \{ term.p.val = term.p_1.val \}$
 $\quad \quad \quad | \quad \varepsilon \{ term.p.val = term.p.i \}$

$\langle fact \rangle ::= (\langle expr \rangle) \{ fact.val = expr.val \} \mid NUM \{ fact.val = NUM.value \}$

```
public void start() {
    int expr_val;
    // ... completare ...
    expr_val = expr();
    match(Tag.EOF);
    System.out.println(expr_val);
    // ... completare ...
}
```

```
private int expr() {
    int term_val, exprp_val;
    // ... completare ...
    term_val = term();
    exprp_val = exprp(term_val);
    // ... completare ...
    return exprp_val;
}
```

```
private int exprp(int exprp_i) {
    int term_val, exprp_val;
    switch (look.tag) {
        case '+':
            match('+');
            term_val = term();
            exprp_val = exprp(exprp_i + term_val);
            break;
        // ... completare ...
    }
}
```

- Per ogni simbolo o azione semantica X nel corpo della produzione scelta:
 - Se X è un simbolo terminale, il metodo controlla che il simbolo corrente sia proprio X . In tal caso, fa avanzare il lexer al simbolo successivo. In caso contrario, il metodo segnala un errore di sintassi.
 - Se X è una variabile, il metodo invoca la procedura X passando a X come argomenti i suoi attributi ereditati e raccogliendo in variabili locali gli attributi sintetizzati restituiti da X .
 - Se X è una azione semantica, il metodo la esegue.

$\langle start \rangle ::= \langle expr \rangle EOF \{ print(expr.val) \}$
 $\langle expr \rangle ::= \langle term \rangle \{ exprp.i = term.val \} \langle exprp \rangle \{ expr.val = exprp.val \}$
 $\langle exprp \rangle ::=$
 $+ \langle term \rangle \{ exprp_1.i = exprp.i + term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \}$
 $- \langle term \rangle \{ exprp_1.i = exprp.i - term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \}$
 $\varepsilon \{ exprp.val = exprp.i \}$
 $\langle term \rangle ::= \langle fact \rangle \{ termp.i = fact.val \} \langle termp \rangle \{ term.val = termp.val \}$
 $\langle termp \rangle ::=$
 $* \langle fact \rangle \{ termp_1.i = termp.i * fact.val \} \langle termp_1 \rangle \{ termp.val = termp_1.val \}$
 $/ \langle fact \rangle \{ termp_1.i = termp.i / fact.val \} \langle termp_1 \rangle \{ termp.val = termp_1.val \}$
 $\varepsilon \{ termp.val = termp.i \}$
 $\langle fact \rangle ::= (\langle expr \rangle) \{ fact.val = expr.val \} \mid NUM \{ fact.val = NUM.value \}$

```

public void start() {
    int expr_val;
    // ... completare ...
    expr_val = expr();
    match(Tag.EOF);
    System.out.println(expr_val);
    // ... completare ...
}

private int expr() {
    int term_val, exprp_val;
    // ... completare ...
    term_val = term();
    exprp_val = exprp(term_val);
    // ... completare ...
    return exprp_val;
}

private int exprp(int exprp_i) {
    int term_val, exprp_val;
    switch (look.tag) {
        case '+':
            match('+');
            term_val = term();
            exprp_val = exprp(exprp_i + term_val);
            break;
        // ... completare ...
    }
}

```


- Per ogni simbolo o azione semantica X nel corpo della produzione scelta:
 - Se X è un simbolo terminale, il metodo controlla che il simbolo corrente sia proprio X . In tal caso, fa avanzare il lexer al simbolo successivo. In caso contrario, il metodo segnala un errore di sintassi.
 - Se X è una variabile, il metodo invoca la procedura X passando a X come argomenti i suoi attributi ereditati e raccogliendo in variabili locali gli attributi sintetizzati restituiti da X .
 - Se X è una azione semantica, il metodo la esegue.

$\langle start \rangle ::= \langle expr \rangle EOF \{ print(expr.val) \}$

$\langle expr \rangle ::= \langle term \rangle \{ exprp.i = term.val \} \langle exprp \rangle \{ expr.val = exprp.val \}$

$\langle exprp \rangle ::=$
 $+$ $\langle term \rangle \{ exprp_1.i = exprp.i + term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \}$
 $-$ $\langle term \rangle \{ exprp_1.i = exprp.i - term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \}$
 $\varepsilon \{ exprp.val = exprp.i \}$

$\langle term \rangle ::= \langle fact \rangle \{ term.p.i = fact.val \} \langle term.p \rangle \{ term.val = term.p.val \}$

$\langle term.p \rangle ::=$
 $*$ $\langle fact \rangle \{ term.p_1.i = term.p.i * fact.val \} \langle term.p_1 \rangle \{ term.p.val = term.p_1.val \}$
 $/$ $\langle fact \rangle \{ term.p_1.i = term.p.i / fact.val \} \langle term.p_1 \rangle \{ term.p.val = term.p_1.val \}$
 $\varepsilon \{ term.p.val = term.p.i \}$

$\langle fact \rangle ::= (\langle expr \rangle) \{ fact.val = expr.val \} \mid NUM \{ fact.val = NUM.value \}$

```
public void start() {
    int expr_val;
    // ... completare ...
    expr_val = expr();
    match(Tag.EOF);
    System.out.println(expr_val);
    // ... completare ...
}
```

```
private int expr() {
    int term_val, exprp_val;
    // ... completare ...
    term_val = term();
    exprp_val = exprp(term_val);
    // ... completare ...
    return exprp_val;
}
```

```
private int exprp(int exprp_i) {
    int term_val, exprp_val;
    switch (look.tag) {
        case '+':
            match('+');
            term_val = term();
            exprp_val = exprp(exprp_i + term_val);
            break;
        // ... completare ...
    }
}
```

Riduzione del codice

$$\langle expr \rangle ::= \langle term \rangle \{ exprp.i = term.val \} \langle exprp \rangle \{ expr.val = exprp.val \}$$

```
private int expr() {  
    int term_val, exprp_i, exprp_val, expr_val;  
    // ... completare ...  
    term_val = term();  
    exprp_i = term_val;  
    exprp_val = exprp(exprp_i);  
    expr_val = exprp_val;  
    // ... completare ...  
    return expr_val;  
}
```

Rappresenta in modo chiaro gli elementi della produzione e le sue azioni semantiche (una riga di codice per elemento/azione semantica).

```
private int expr() {  
    int term_val, exprp_val;  
    // ... completare ...  
    term_val = term();  
    exprp_val = exprp(term_val);  
    // ... completare ...  
    return exprp_val;  
}
```

Versione più sintetica: elimina l'utilizzo dei campi `exprp_i` e `expr_val`, mantenendo lo stesso valore di ritorno.

Riduzione del codice

$\langle exprp \rangle ::= + \langle term \rangle \{ exprp_1.i = exprp.i + term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \}$

```
private int exprp(int exprp_i) {
    int term_val, exprp1_i, exprp1_val, exprp_val;
    switch (look.tag) {
        case '+':
            match('+');
            term_val = term();
            exprp1_i = exprp_i + term_val;
            exprp1_val = exprp(exprp1_i);
            exprp_val = exprp1_val;
            break;
        // ... completare ...
    }
}
```

Rappresenta in modo chiaro gli elementi della produzione e le sue azioni semantiche (una riga di codice per elemento/azione semantica).

```
private int exprp(int exprp_i) {
    int term_val, exprp_val;
    switch (look.tag) {
        case '+':
            match('+');
            term_val = term();
            exprp_val = exprp(exprp_i + term_val);
            break;
        // ... completare ...
    }
}
```

Versione più sintetica: elimina l'utilizzo dei campi `exprp1_i` e `exprp1_val`, mantenendo lo stesso valore di ritorno.

Ordine delle istruzioni

- Approccio generale per le azioni semantiche: eseguire dopo gli elementi alla loro sinistra sono stati riconosciuti.
- Consiglio: per accedere ad un valore associato con un terminale, bisogna assegnare il valore ad una variabile *prima* di fare avanzare il lexer al token successivo.
 - Motivazione: fare avanzare il lexer al token successivo ha come conseguenza la perdita di tutte le informazioni che riguarda il token attuale, compresi eventuali valori associati con il token.
 - Esempio: `NUM { fact.val = NUM.value }` nel corpo della produzione
 - Per eseguire l'azione semantica, abbiamo bisogno di `NUM.value`: però se facciamo avanzare il lexer al token successivo al token numerico «matched» con `NUM`, non possiamo più accedere a `NUM.value`.
 - Soluzione: prima di fare avanzare il lexer al token successivo, assegnare `NUM.value` al variabile che corrisponde a *fact.val* nel codice.