Laboratorio di Linguaggi Formali e Traduttori
LFT lab T2, a.a. 2021/2022

Traduzione diretta dalla sintassi

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:

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

Traduzione diretta dalla sintassi

Grammatica + azioni semantiche

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

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

Traduzione diretta dalla sintassi

Grammatica + azioni semantiche

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

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

Traduzione diretta dalla sintassi

Grammatica + azioni semantiche

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

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

Richiamo teoria

- Integrazione di uno SDT in un parser ricorsivo discendente (dal slide 7 del file «sdt.pdf» con titolo «5.2 - Schemi di traduzione».

- 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 $A \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$ 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 $\{ print(expr.val) \}$
$\langle expr \rangle$	::=	$\langle term \rangle \{ exprp.i = term.val \} \langle exprp \rangle \{ expr.val = exprp.val \}$
$\langle exprp \rangle$	::=	$+ \langle term \rangle \{ exprp_1.i = exprp.i + term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \}$
	|	$- \langle term \rangle \{ exprp_1.i = exprp.i - term.val \} \langle exprp_1 \rangle \{ exprp.val = exprp_1.val \}$
	|	$\varepsilon \{ exprp.val = exprp.i \}$
$\langle term \rangle$	::=	$\langle fact \rangle \{ termp.i = fact.val \} \langle termp \rangle \{ term.val = termp.val \}$
$\langle termp \rangle$	::=	$* \langle fact \rangle \{ termp_1.i = termp.i * fact.val \} \langle termp_1 \rangle \{ termp.val = termp_1.val \}$
	|	$/ \langle fact \rangle \{ termp_1.i = termp.i / fact.val \} \langle termp_1 \rangle \{ termp.val = termp_1.val \}$
	|	$\varepsilon \{ termp.val = termp.i \}$
$\langle fact \rangle$	::=	($\langle expr \rangle$) $\{ fact.val = expr.val \}$ | 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.

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

```java
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.

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

```
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 $A \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$ da usare per riscrivere A.

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

- 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.

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

```java
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.

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

```java
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.

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

```java
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 ...
    }
}
```

Codice per le azioni semantiche

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

```
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 esplicito gli elementi della produzione e le sue azioni semantiche.

```
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.

Codice per le azioni semantiche

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

```
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 esplicito gli elementi della produzione e le sue azioni semantiche.

```
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/eseguiti.
- Consiglio: per accedere ad un valore associato con un terminale, bisogna assegnare il valore ad un campo *prima* di fare avanzare il lexer al token successivo.
  - Motivazione: fare avanzare il lexer al token successivo ha come conseguenza la perdità di tutte le informazioni che riguarda il token attuale, compresi eventuali valori associati con il token.
  - Esempio: $\texttt{NUM}\ \{\ fact.val = \texttt{NUM}.value\ \}$ 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.