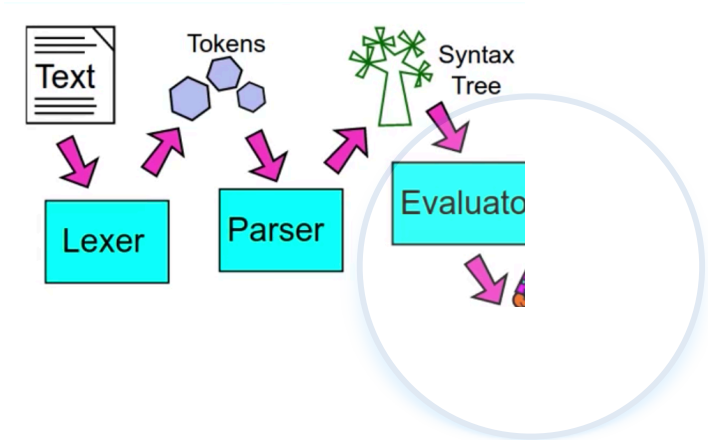
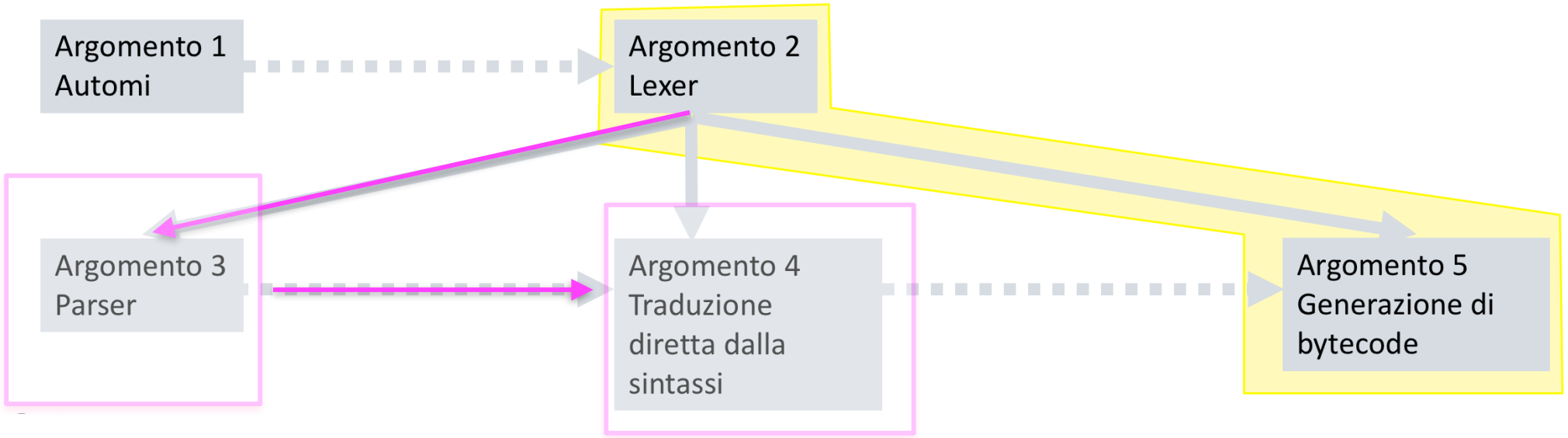


4. Traduzione diretta dalla sintassi

Valutatore di espressioni aritmetiche semplici con sintassi da esercizio. 3.1

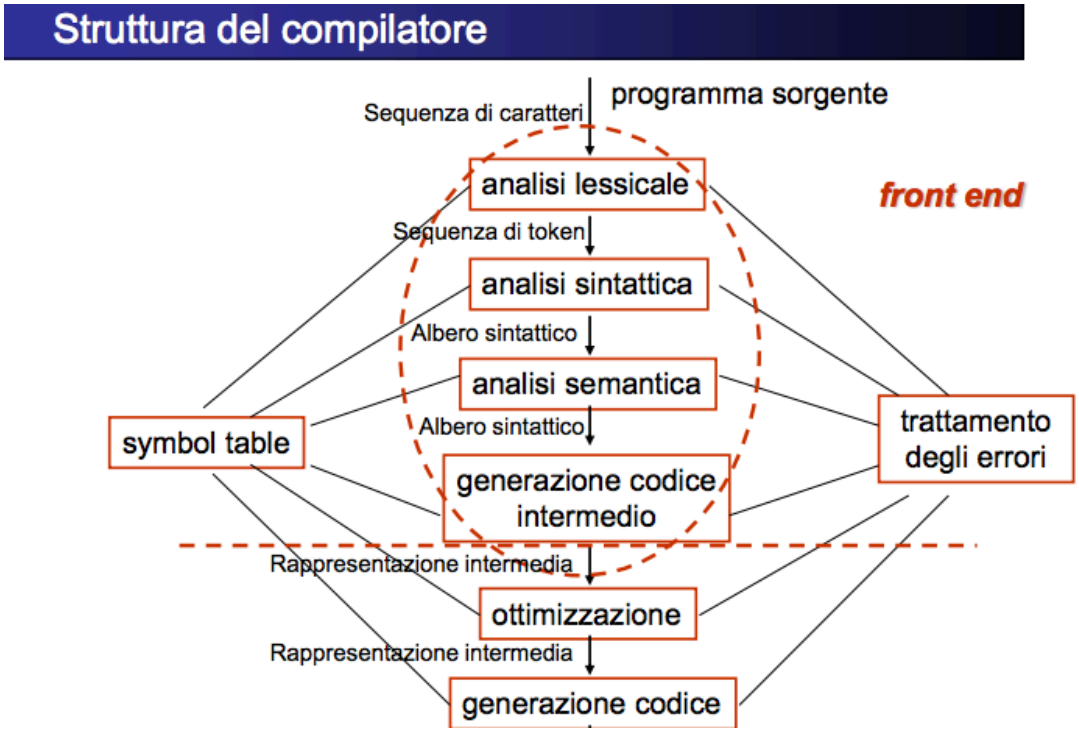
Progetto di laboratorio LFT LAB

- Il progetto di laboratorio consiste in una serie di esercitazioni assistite mirate allo sviluppo di un semplice traduttore.
- Dove siamo:
  - Terzo step: Traduzione diretta dalla sintassi



Traduzione diretta dalla sintassi

- Dove si posiziona in una pipeline di traduzione?
- L'analizzatore sintattico (o parser) implementato fornirà l'input agli step successivi all'analisi sintattica -> traduzione guidata dalla sintassi.



(...continua...)

Traduzione diretta dalla sintassi

- Obiettivo: dare un *significato* all'input.
  - Ad esempio:
    - la traduzione in un altro linguaggio di un programma (argomento delle lezioni che rimangono dopo la lezione di oggi)
    - la valutazione di un'espressione aritmetica (lezione di. oggi), ….
- Definizioni dirette (o guidate) dalla sintassi (SDD):
  - Grammatica

  + attributi (associati alle variabili della grammatica)

  + regole semantiche (come calcolare il valore degli attributi; associati con le produzioni della grammatica).
- Schema di traduzione diretto dalla sintassi (SDT):
  - L'ordine di valutazione degli attributi è esplicito (ciascun *SDD L-attribuite* può essere convertito in uno SDT).
  - Azioni semantiche:
    - Frammenti di codice inseriti nelle produzioni.
    - Possono contenere, oltre ad azioni per calcolare il valore degli attributi, anche codice arbitrario.
  - Adattato ad essere integrato in un parser ricorsivo discendente.

Esercizio 4.1

- Implementazione in Java di un valutatore di espressioni aritmetiche semplici
- Modificare l'analizzatore sintattico dell'esercizio 3.1 (parser per espressioni aritmetiche con notazione infissa) in modo da valutare le espressioni aritmetiche semplici, facendo riferimento allo schema di traduzione diretto dalla sintassi seguente

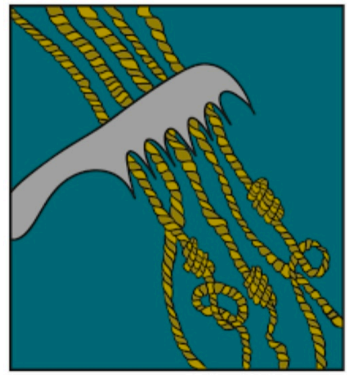
Esercizio 4.1

- Schema di traduzione diretto dalla sintassi
- Azioni semantiche descritte in verde nel corpo delle produzioni

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

il terminale NUM ha l'attributo value , che è il valore numerico del terminale, fornito dal lexer

Esercizio 4.1



- Schema di traduzione diretto dalla sintassi
- Azioni semantiche descritte in verde nel corpo delle produzioni

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

Il terminale NUM ha l'attributo value , che è il valore numerico del terminale, fornito dal lexer (analisi lessicale)

Attenzione: qui molti nodi eventuali relativi a come avete implementato la classe NumberTok verranno al pettine

Richiamo teoria

- Traduzione guidata dalla sintassi: SDD (syntax directed definitions) e grafo delle dipendenze
- Schemi di traduzione (SDT) e traduzione top-down di SDD L-attribuite.
- Differenza fra attributi sintetizzati e attributi ereditati
  - è importante nel contesto di questo esercizio per sapere se un valore deve essere passato come parametro ai metodi (ad esempio, in exprp e termp)

Richiamo teoria

Schemi di traduzione: SDT

Gli **schemi di traduzione** (**SDT**) sono un'utile notazione per specificare la traduzione durante la parsificazione.

Uno schema di traduzione è una definizione guidata dalla sintassi in cui le azioni semantiche, racchiuse tra parentesi graffe, sono inserite nei membri destri delle produzioni, in posizione tale che il valore di un attributo sia disponibile quando un'azione fa ad esso riferimento.

Gli schemi di traduzione impongono un ordine di valutazione da sinistra a destra e permettono che nelle azioni semantiche siano contenuti frammenti di programma e in generale side-effect che non influiscano sulla valutazione degli attributi.

Richiamo teoria

- Integrazione di uno SDT in un parser ricorsivo discendente (file «Schemi di traduzione e valutazione top-down» sulla pagina di LFT teoria).

Traduzione_discesa_ricorsiva

```
main( )
begin
    cc ← PROSS
    risultato ← S( )
    if (cc = '$') stampa ("stringa corretta, la sua traduzione è:" risultato)
    else ERRORE(…)
end

A(e1, . . . en)                                   A → α1 | α2 | . . . | αk
begin
    if (cc ∈ Gui(A → α1)) body'(α1)
        elseif (cc ∈ Gui(A → α2)) body'(α2)
        ….
        elseif (cc ∈ Gui(A → αk)) body'(αk)
        else ERRORE(…)
        return (<s1, . . . , sm>)
end
```

Traduzione diretta dalla sintassi

Grammatica + attributi + azioni semantiche

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

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

Traduzione diretta dalla sintassi

Grammatica + attributi + azioni semantiche

(esempio $exprp.i$)

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

Traduzione diretta dalla sintassi

Grammatica + attributi + 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 + attributi + azioni semantiche

Valori di eventuali attributi dei terminali: fornito dal lexer

(esempio $\{ fact.val = \texttt{NUM}.value \}$)

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

Esercizio 4.1

- Estendere il parser che avete costruito in 3.1. e trasformarlo in traduttore
- Create una nuova classe: (scaricare codice con spunti)
- Aggiungere il codice necessario nei vari metodi già realizzati
- Per valutare le espressioni aritmetiche

```
import java.io.*;

public class Valutatore {
    private Lexer lex;
    private BufferedReader pbr;
    private Token look;

    public Valutatore(Lexer l, BufferedReader br) {
        lex = l;
     pbr = br;
        move();
    }

    void move() {
        // come in Esercizio 3.1
    }

    void error(String s) {
        // come in Esercizio 3.1
    }

    void match(int t) {
        // come in Esercizio 3.1
    }

    public void start() {
        int expr_val;

        // ... completare ...

        expr_val = expr();
```

Dallo SDT al codice

Ad ogni non terminale si associa una funzione che ha come parametri in input i valori degli attributi ereditati della variabile (le informazioni che devono essere note quando si esegue la funzione) e restituisce i valori dei suoi attributi sintetizzati (i valori che la funzione calcola).

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

```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 ...
    }
}

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

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

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

Dallo SDT al codice

Ad ogni non terminale si associa una funzione che ha come parametri in input i valori degli attributi ereditati della variabile (le informazioni che devono essere note quando si esegue la funzione) e restituisce i valori dei suoi attributi sintetizzati (i valori che la funzione calcola).

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

```
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

Ad ogni non terminale si associa una funzione che ha come parametri in input i valori degli attributi ereditati della variabile (le informazioni che devono essere note quando si esegue la funzione) e restituisce i valori dei suoi attributi sintetizzati (i valori che la funzione calcola).

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

```
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 ...
    }
}
```

Richiamo teoria

Definizioni guidate dalla sintassi: SDD

Per i **simboli non terminali** consideriamo due tipi di attributi:

- ***sintetizzati***: un attributo sintetizzato per una variabile A in un nodo *n* dell'albero di parsificazione è definito da una regola semantica associata alla produzione in *n* e il suo valore è calcolato solo in termini dei valori degli attributi nei nodi figli di *n* e in *n* stesso.
  (A è il simbolo a sinistra nella produzione, cioè la testa).

- ***ereditati***: un attributo ereditato per una variabile A in un nodo *n* dell'albero di parsificazione è definito da una regola semantica associata alla produzione nel nodo padre di *n* e il suo valore è calcolato solo in termini dei valori degli attributi del padre di *n*, di *n* stesso e dei suoi fratelli.
  (A è un simbolo nel corpo della produzione, cioè al membro destro).

Conservare in variabili locali i valori necessari per il calcolo degli attributi ereditati relativi ai non terminali
Es. caso della somma/sottrazione:
la difficoltà è tenere traccia dei valori da sommare, sottrarre ecc.

Esempio della somma

- Alcuni valori sono passati come parametri ai metodi, e alcuni valori vengono restituiti da altri metodi
- Esempio della somma nel codice con spunti come modello:
  - uso del parametro corrisponde attributo ereditato
  - uso del valore restituito da una chiamata di un metodo corrisponde a attributo sintetizzato

```
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;
```

Trasformazione di comandi

```
private int expr() {
 int term_val, exprp_val;

 // ... completare ...

 term_val = term();
 exprp_i = term_val;
 exprp_val = exprp(exprp_i);
 expr_val = exprp_val;

 // ... completare ...
 return expr_val;
 }
```

```
private int expr() {
int term_val, exprp_val;

// ... completare ...

term_val = term();
exprp_val = exprp(term_val);

// ... completare ...
return exprp_val;
}
```

- In entrambe le versioni, il valore restituito è lo stesso (inoltre, exprp_val ha lo stesso valore)
- Il frammento di codice scaricabile dalla pagina Moodle contiene codice “ridotto” in questo modo