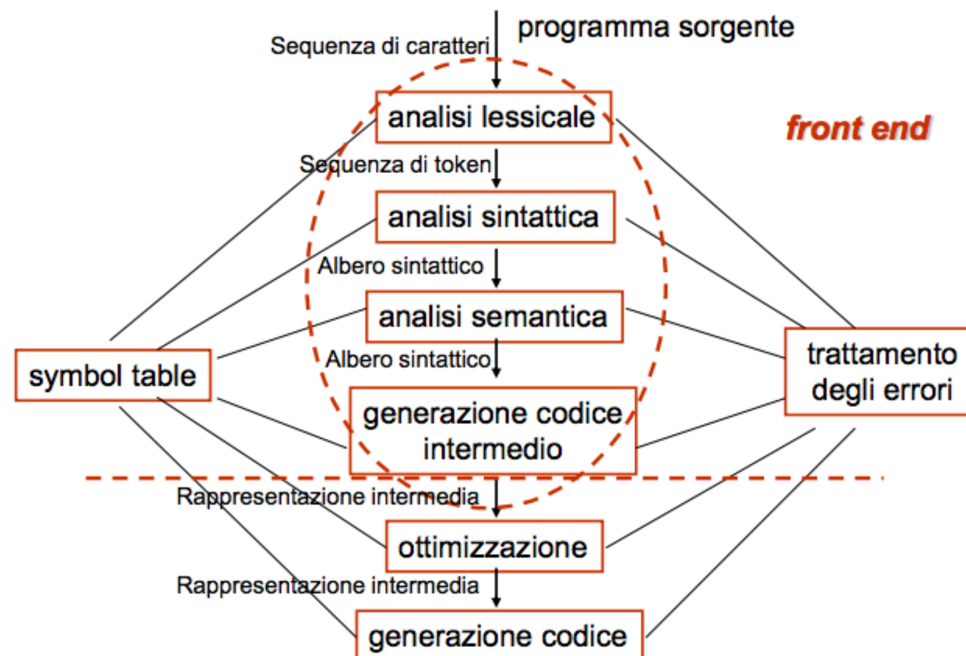
3. Analisi sintattica

Implementazione in Java di un analizzatore sintattico a discesa ricorsiva per espressioni aritmetiche semplici

Analizzatore sintattico

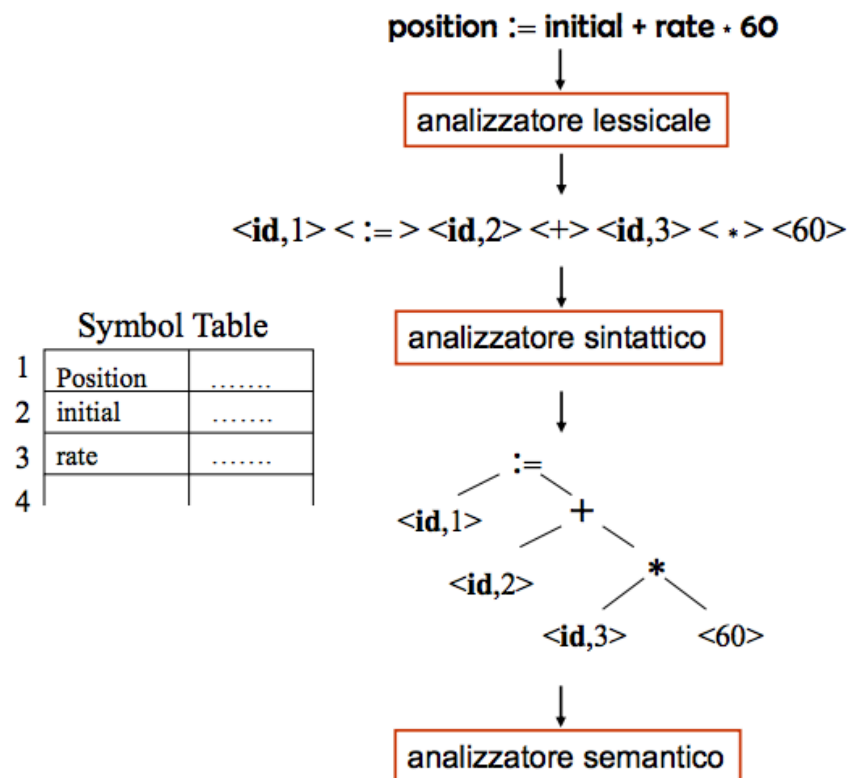
- Dove si posiziona in una pipeline di traduzione?
- L'analizzatore lessicale implementato fornirà l'input al programma di analisi sintattica e di traduzione (step successivi all'analisi sintattica).



(...continua...)

Analizzatore sintattico

- Dove si posiziona in una pipeline di traduzione?
- L’analizzatore lessicale implementato fornirà l’input al programma di analisi sintattica e di traduzione (step successivi all’analisi sintattica).



Esercizio 3.1

- Si scriva un analizzatore sintattico a discesa ricorsiva che parsifichi espressioni aritmetiche molto semplici, composte soltanto da
  - numeri non negativi (ovvero sequenze di cifre decimali)
  - operatori di somma e sottrazione: + –
  - operatori di moltiplicazione e divisione: * /
  - simboli di parentesi: ( )

Esercizio 3.1

- In particolare, l’analizzatore deve riconoscere le espressioni generate dalla grammatica:

	$\langle start \rangle$	::=	$\langle expr \rangle$ EOF	→ $
E ←	$\langle expr \rangle$	::=	$\langle term \rangle \langle exprp \rangle$	
E’ ←	$\langle exprp \rangle$	::=	$+ \langle term \rangle \langle exprp \rangle$	
		|	$- \langle term \rangle \langle exprp \rangle$	
		|	ε	
T ←	$\langle term \rangle$	::=	$\langle fact \rangle \langle termp \rangle$	
T’ ←	$\langle termp \rangle$	::=	$\star \langle fact \rangle \langle termp \rangle$	
		|	$/ \langle fact \rangle \langle termp \rangle$	
		|	ε	
F ←	$\langle fact \rangle$	::=	($\langle expr \rangle$) | NUM	

Esercizio 3.1

- Il programma deve fare uso dell'analizzatore lessicale sviluppato in precedenza.
- Si noti che l'insieme di token corrispondente alla grammatica di questa sezione è un sottoinsieme dell'insieme di token corrispondente alle regole lessicali specificate nella Sezione 2.

Esercizio 3.1

- Nei casi in cui l'input ***non*** corrisponde alla grammatica, l'output del programma deve consistere di un messaggio di errore (come illustrato nelle lezioni di teoria) indicando la procedura in esecuzione quando l'errore è stato individuato.

Richiamo teoria

Grammatiche LL(1) e parsificazione deterministica look ahead

Analizzatore ricorsivo: esempio

```
main  zero-uno( )
      { cc ← PROSS
        S( )
        if (cc = '$') "stringa accettata"
        else ERRORE(…)
      }
```

	0	1	$
S	S→0A		
A	A→S1	A→1	

```
function  S( )
    {  if (cc = 0)
            cc ← PROSS
            A( )
       else ERRORE(…)
    }
```

```
function  A( )
      { if (cc = 0)
              S( )
              if (cc = 1) cc ← PROSS
              else ERRORE (…)
        else if (cc = 1)
              cc ← PROSS
        else ERRORE (…)
      }
```

Richiamo teoria

Grammatiche LL(1) e parsificazione deterministica look ahead

Analizzatore a discesa ricorsiva

Ad ogni variabile A con produzioni

$A \to \alpha_1$

$A \to \alpha_2$

...

$A \to \alpha_k$

si associa una funzione:

```
function A( )
    {
      if (cc ∈ Gui(A → α1)) body (α1)
      else if (cc ∈ Gui(A → α2)) body (α2)
       ....
      else if (cc ∈ Gui(A → αk)) body (αk)
      else ERRORE (...)
    }
```

Richiamo teoria

Esempio: espressioni aritmetiche

Produzioni	Insiemi guida	
1. $E \rightarrow T\ E'$	$F(TE') = F(T) = F(F) = \{ (, a\}$	
2. $E' \rightarrow +\ T\ E'$	$\{ + \}$	$\{ + \} \cap \{ - \} \cap \{), \$\} = \Phi$
3. $E' \rightarrow -\ T\ E'$	$\{ - \}$	
4. $E' \rightarrow \varepsilon$	$FW(E') = \{), \$\}$	
5. $T \rightarrow F\ T'$	$F(F) = \{(, a\}$	
6. $T' \rightarrow *\ F\ T'$	$\{ * \}$	$\{ * \} \cap \{ / \} \cap \{ +, -,), \$\} = \Phi$
7. $T' \rightarrow /\ F\ T'$	$\{ / \}$	
8. $T' \rightarrow \varepsilon$	$FW(T') = \{ +, -,), \$\}$	
9. $F \rightarrow (E)$	$\{ (\}$	$\{ (\} \cap \{ a \} = \Phi$
10. $F \rightarrow a$	$\{ a \}$	

$FW(E') = FW(E) = \{), \$\}$

$FW(T') = FW(T) = (F(E') - \{\varepsilon\}) \cup FW(E) = \{+, -,), \$\}$

Richiamo teoria

Esempio

Tabella

	(	)	+	-	*	/	a	$
E	E → T E'						E → T E'	
E'		E' → ε	E'→+T E'	E'→ -T E'				E' → ε
T	T → F T'						T → F T'	
T'		T' → ε	T' → ε	T' → ε	T'→ *FT'	T'→ / FT'		T' → ε
F	F → (E)						F → a	

La grammatica è LL(1) in quanto in ogni elemento della tabella compare al massimo una produzione.

Codice di partenza

```
import java.io.*;

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

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


    void move() {
        look = lex.lexical_scan(pbr);
        System.out.println("token = " + look);
    }

    void error(String s) {
        throw new Error("near line " + lex.line + ": " + s);
    }

    void match(int t) {
        if (look.tag == t) {
            if (look.tag != Tag.EOF) move();
        } else error("syntax error");
    }

    public void start() {
        // ... completare ...
        expr();
```