# 1 Lexical elements

Input is converted into a sequence of tokens during the lexical analysis. Tokens end if the next character is a whitespace character or if the next character cannot be added to it.

## 1.1 Whitespace characters

The characters ‘ ’ (ASCII 32), ‘\f’ (ASCII 12), ‘\v’ (ASCII 11) and ‘\t’ (ASCII 9) are whitespace charakters.

Whitespace characters delimite tokens but are otherwise ignored.

## 1.2 Tokens recoginzed

Recognized token kinds are one of:

| | |
|---|---|
| **UNKNOWN** | Illegal character (i.e. not a whitespace and can not be part of any token) was found and consumed. |
| **EOI** | End of input |
| ⟨float-literal⟩ | floating point literal matching the regular expression `[0-9]+(([.][0-9]*)([eE][+-]?[0-9]+)?)?` or `[0-9]*(([.][0-9]+)([eE][+-]?[0-9]+)?)?` Note: The first regular expression requires at least one leading digit, the second at least one trailing digit. Hence, `0.` and `.0` are legal, but `.` is not |
| + | Plus operator |
| - | Minus operator or unary minus |
| * | Multiplication operator |
| / | Division oprtator |
| ^ | Power operator |
| ( | Left parenthesis |
| ) | Right parenthesis |

## 2 Grammar (with left recursion)

| | | |
|---|---|---|
| ⟨translation-unit⟩ | ⟶ | **EOI** |
| | ⟶ | ⟨expression-sequence⟩ **EOI** |
| ⟨expression-sequence⟩ | ⟶ | ⟨expression⟩ |
| | ⟶ | ⟨expression-sequence⟩ ⟨expression⟩ |
| ⟨expression⟩ | ⟶ | **EOL** |
| | ⟶ | ⟨additive-expression⟩ **EOL** |
| ⟨additive-expression⟩ | ⟶ | ⟨multiplicative-expression⟩ |
| | ⟶ | ⟨additive-expression⟩ + ⟨multiplicative-expression⟩ |
| | ⟶ | ⟨additive-expression⟩ **-** ⟨multiplicative-expression⟩ |
| ⟨multiplicative-expression⟩ | ⟶ | ⟨power-expression⟩ |
| | ⟶ | ⟨multiplicative-expression⟩ ***** ⟨power-expression⟩ |
| | ⟶ | ⟨multiplicative-expression⟩ / ⟨power-expression⟩ |
| ⟨power-expression⟩ | ⟶ | ⟨unary-expression⟩ |
| | ⟶ | ⟨unary-expression⟩ ^ ⟨power-expression⟩ |
| ⟨unary-expression⟩ | ⟶ | ⟨primary-expression⟩ |
| | ⟶ | + ⟨unary-expression⟩ |
| | ⟶ | **-** ⟨unary-expression⟩ |
| ⟨primary-expression⟩ | ⟶ | ⟨float-literal⟩ |
| | ⟶ | **(** ⟨additive-expression⟩ **)** |

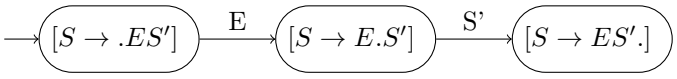
## 2.1 Expression sequence

$$\underbrace{\langle\text{expression-sequence}\rangle}_{=:S} \longrightarrow \underbrace{\langle\text{expression}\rangle}_{=:E}$$
$$\longrightarrow \langle\text{expression-sequence}\rangle\ \langle\text{expression}\rangle$$

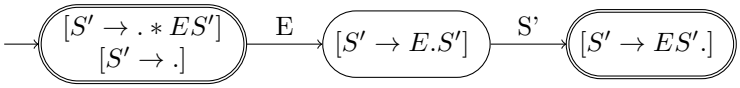
### 2.1.1 Eliminating left recursion

$$\underbrace{\langle\text{expression-sequence}\rangle}_{=:S} \longrightarrow \underbrace{\langle\text{expression}\rangle}_{=:E}\underbrace{\langle\text{expression-sequence'}\rangle}_{=:S'}$$
$$\langle\text{expression-sequence'}\rangle \longrightarrow \langle\text{expression}\rangle\ \langle\text{expression-sequence'}\rangle$$
$$\longrightarrow$$

**Item automata for S**




**Item automata for S’**




### 2.1.2 Eliminating left recursion (in EBNF Form)

Expressing the BNF notation in EBNF reveals how the item automata for $S$ and $S'$ can be combined in an implementation using a do-while loop:

$$\underbrace{\langle\text{expression-sequence}\rangle}_{=S} \longrightarrow \underbrace{\langle\text{expression}\rangle}_{=E}\underbrace{\{\ \langle\text{expression}\rangle\ \}}_{=E'}$$

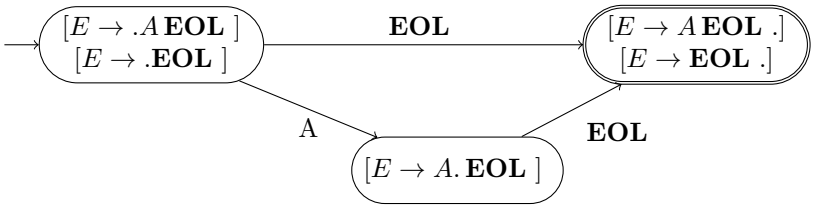
In the case the loop can terminate if the current token is Follow($E'$) = {**EOI** }.

## 2.2 Expression

$$\underbrace{\langle \text{expression} \rangle}_{=:E} \longrightarrow \mathbf{EOL}$$

$$\longrightarrow \underbrace{\langle \text{additive-expression} \rangle}_{=:A} \mathbf{EOL}$$

**Item automata for E**



## 2.3 Additive Expression

$$
\begin{array}{lcl}
\langle \text{additive-expression} \rangle & \longrightarrow & \langle \text{multiplicative-expression} \rangle \\
& \longrightarrow & \langle \text{additive-expression} \rangle \; + \; \langle \text{multiplicative-expression} \rangle \\
& \longrightarrow & \langle \text{additive-expression} \rangle \; \texttt{-} \; \langle \text{multiplicative-expression} \rangle
\end{array}
$$

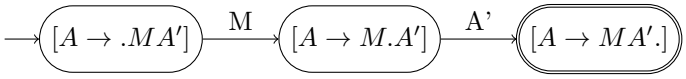
### 2.3.1 Eliminating left recursion

Eliminating left recursion leads to

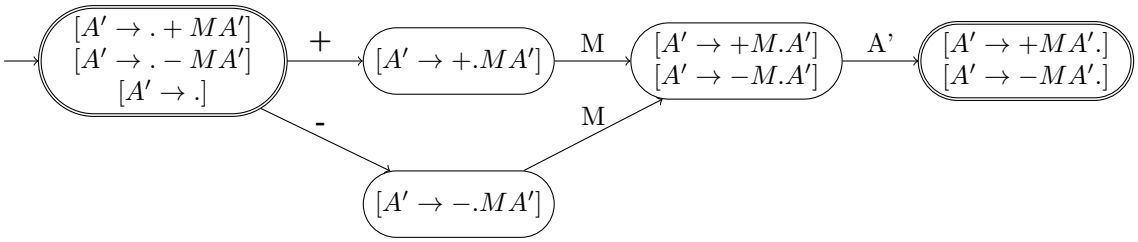
$$
\begin{array}{lcl}
\underbrace{\langle \text{additive-expression} \rangle}_{=:A} & \longrightarrow & \underbrace{\langle \text{multiplicative-expression} \rangle}_{=:M} \underbrace{\langle \text{additive-expression'} \rangle}_{=:A'} \\
\langle \text{additive-expression'} \rangle & \longrightarrow & + \; \langle \text{multiplicative-expression} \rangle \; \langle \text{additive-expression'} \rangle \\
& \longrightarrow & \texttt{-} \; \langle \text{multiplicative-expression} \rangle \; \langle \text{additive-expression'} \rangle \\
& \longrightarrow &
\end{array}
$$

The left associativity of the operators + and - needs to be handled manually by the parser.

**Item automata for A**




**Item automata for A’**




### 2.3.2 Eliminating left recursion (in EBNF Form)

Expressing the BNF notation in EBNF reveals how the item automata for $A$ and $A'$ can be combined in an implementation using a while loop checking if the current token is in First$(A') = \{+ \, , \texttt{-} \}$:

$$
\begin{array}{lcl}
\underbrace{\langle \text{additive-expression} \rangle}_{=A} & \longrightarrow & \underbrace{\langle \text{multiplicative-expression} \rangle}_{=M} \underbrace{\{ \; \langle \text{additive-op} \rangle \; \langle \text{multiplicative-expression} \rangle \; \}}_{=A'} \\
\langle \text{additive-op} \rangle & \longrightarrow & + \\
& \longrightarrow & \texttt{-}
\end{array}
$$

## 2.4 Multiplicative Expression

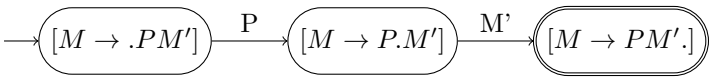
$$
\begin{array}{lcl}
\langle \text{multiplicative-expression} \rangle & \longrightarrow & \langle \text{power-expression} \rangle \\
& \longrightarrow & \langle \text{multiplicative-expression} \rangle \ \textbf{*} \ \langle \text{power-expression} \rangle \\
& \longrightarrow & \langle \text{multiplicative-expression} \rangle \ / \ \langle \text{power-expression} \rangle
\end{array}
$$

### 2.4.1 Eliminating left recursion (in BNF Form)

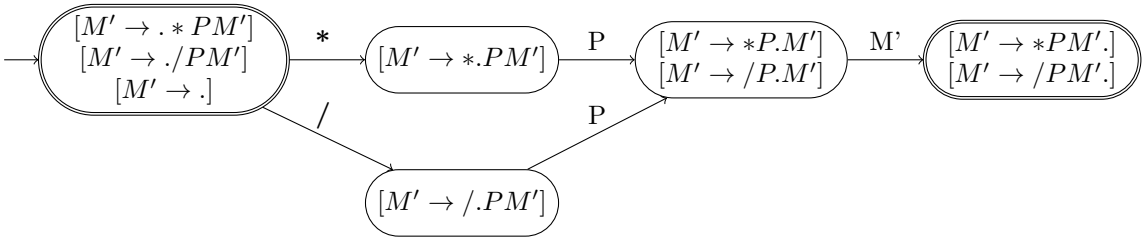
Eliminating left recursion leads to

$$
\begin{array}{lcl}
\underbrace{\langle \text{multiplicative-expression} \rangle}_{=:M} & \longrightarrow & \underbrace{\langle \text{power-expression} \rangle}_{=:P} \underbrace{\langle \text{multiplicative-expression'} \rangle}_{=:M'} \\
\langle \text{multiplicative-expression'} \rangle & \longrightarrow & \textbf{*} \ \langle \text{power-expression} \rangle \ \langle \text{multiplicative-expression'} \rangle \\
& \longrightarrow & / \ \langle \text{power-expression} \rangle \ \langle \text{multiplicative-expression'} \rangle \\
& \longrightarrow &
\end{array}
$$

**Item automata for M**




**Item automata for M'**




### 2.4.2 Eliminating left recursion (in EBNF Form)

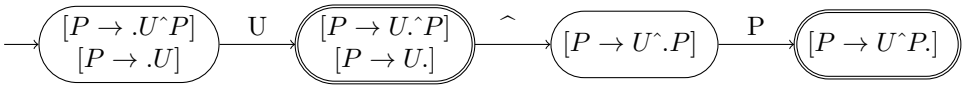
Expressing the BNF notation in EBNF reveals how the item automata for $M$ and $M'$ can be combined in an implementation using a while loop checking if the current token is in First$(M') = \{$***** , / $\}$:

$$
\begin{array}{lcl}
\underbrace{\langle \text{multiplicative-expression} \rangle}_{=M} & \longrightarrow & \underbrace{\langle \text{power-expression} \rangle}_{=P} \{ \underbrace{\langle \text{multiplicative-op} \rangle \ \langle \text{power-expression} \rangle}_{=M'} \} \\
\langle \text{multiplicative-op} \rangle & \longrightarrow & \textbf{*} \\
& \longrightarrow & /
\end{array}
$$

## 2.5 Power Expression

$$\begin{aligned} \underbrace{\langle \text{power-expression} \rangle}_{=:P} &\longrightarrow \underbrace{\langle \text{unary-expression} \rangle}_{=:U} \\ &\longrightarrow \langle \text{unary-expression} \rangle \mathbin{\hat{}} \langle \text{power-expression} \rangle \end{aligned}$$

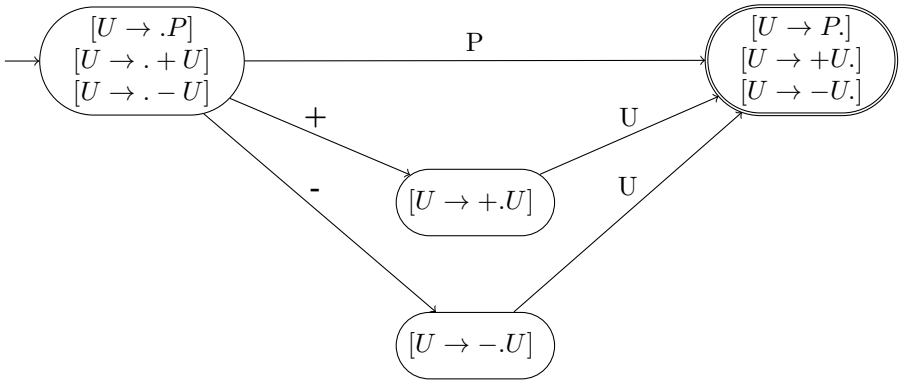
**Item automata for P**



## 2.6 Unary Expression

$$
\begin{aligned}
\underbrace{\langle \text{unary-expression} \rangle}_{=:U} &\longrightarrow \underbrace{\langle \text{primary-expression} \rangle}_{=:P} \\
&\longrightarrow \; + \; \langle \text{unary-expression} \rangle \\
&\longrightarrow \; \texttt{-} \; \langle \text{unary-expression} \rangle
\end{aligned}
$$

**Item automata for U**



## 2.7 Primary Expression

$$\underbrace{\langle\text{primary-expression}\rangle}_{=:P} \longrightarrow \underbrace{\langle\text{float-literal}\rangle}_{=:F}$$
$$\longrightarrow \text{(}\ \underbrace{\langle\text{additive-expression}\rangle}_{=:A}\ \text{)}$$

**Item automata for P**