

# Lecture 2 – Syntax and Semantics (1)

## COSE212: Programming Languages

Jihyeok Park



2023 Fall

We learn language features of **Scala**:

- **Basic Features**
  - Built-in Data Types
  - Variables
  - Functions
  - Conditionals
- **Object-Oriented Programming (OOP)**
  - Case Classes
- **Algebraic Data Types (ADTs)**
  - Pattern Matching
- **Functional Programming (FP)**
  - First-class Functions
  - Recursion
- **Immutable Collections**
  - Lists
  - Options and Pairs
  - Maps and Sets
  - For Comprehensions

## Definition (Programming Language)

A **programming language** is defined by

- **Syntax**: a grammar that defines the structure of programs
- **Semantics**: a set of rules that defines the meaning of programs

We will learn how to define the **syntax** and **semantics** of a programming language.

We define a programming language for **arithmetic expressions** (AE) as the running example.

Let's consider the arithmetic expressions (AE) supporting **addition** and **multiplication** of integers:

- $4 + 2$
- $1 * 24$
- $-42 + 4 * 10$
- $(1 + 2) * (2 + 3)$
- ...

There are **infinitely many** AEs.

How to define all the valid AEs (**syntax**)?

How to define the expected result of each AE (**semantics**)?

## 1. Syntax

- Backus-Naur Form (BNF)

- Concrete Syntax of AE

- Abstract Syntax of AE

- Concrete vs. Abstract Syntax

## 2. Semantics

- Inference Rules

- Big-Step (Natural) Semantics of AE

- Small-Step (Reduction) Semantics of AE

## 1. Syntax

- Backus-Naur Form (BNF)

- Concrete Syntax of AE

- Abstract Syntax of AE

- Concrete vs. Abstract Syntax

## 2. Semantics

- Inference Rules

- Big-Step (Natural) Semantics of AE

- Small-Step (Reduction) Semantics of AE

**Backus-Naur Form (BNF)** is a notation for **context-free grammar**:

- A **nonterminal** has a name and a set of **production rules** consisting of sequences of terminals and nonterminals.
- A **terminal** is a symbol that appears in the final output.

For example, we can define a nonterminal `<number>` with its production rules to represent all integers (allowing leading zeros) as follows:

```
<digit> ::= "0" | "1" | "2" | "3" | "4"  
          | "5" | "6" | "7" | "8" | "9"
```

```
<nat>    ::= <digit> | <digit> <nat>
```

```
<number> ::= <nat> | "-" <nat>
```

Let's define the **concrete syntax** of AE in BNF:

```
<expr> ::= <number>
          | <expr> "+" <expr>
          | <expr> "*" <expr>
          | "(" <expr> ")"
```

It determines whether a given string is a valid AE or not. For example, (1+2)\*3 is a valid AE:

<expr> $\Rightarrow$ <expr>*<expr>	$\Rightarrow$ (<expr>)*<expr>
$\Rightarrow$ (<expr>+<expr>)*<expr>	$\Rightarrow$ (<number>+<expr>)*<expr>
$\Rightarrow$ (1+<expr>)*<expr>	$\Rightarrow$ (1+<number>)*<expr>
$\Rightarrow$ (1+2)*<expr>	$\Rightarrow$ (1+2)*<number>
$\Rightarrow$ (1+2)*3	

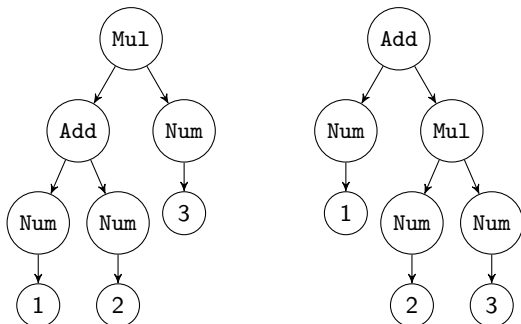


Let's define the **abstract syntax** of AE in BNF:

$$\begin{array}{lcl} e & ::= & n \quad (\text{Num}) \\ & | & e + e \quad (\text{Add}) \\ & | & e * e \quad (\text{Mul}) \end{array}$$

It captures only the essential structure of AE rather than the details.

The **abstract syntax trees (ASTs)** of  $(1+2)*3$  and  $1+2*3$  are as follows:



While **concrete syntax** is the **surface-level** representation of programs, **abstract syntax** is the **essential** representation of programs.

There might be **multiple** concrete syntax for the **same** abstract syntax:

```
<expr> ::= <number>
         | <expr> "+" <expr>
         | <expr> "*" <expr>
         | "(" <expr> ")"
```

```
<expr> ::= <number>
         | "(" "+" <expr> <expr> ")"
         | "(" "*" <expr> <expr> ")"
```

```
<expr> ::= <number>
         | "ADD" "[" <expr> ";" <expr> "]"
         | "MUL" "[" <expr> ";" <expr> "]"
```

```
e ::= n      (Num)
    | e + e  (Add)
    | e * e  (Mul)
```

While **concrete syntax** is the **surface-level** representation of programs, **abstract syntax** is the **essential** representation of programs.

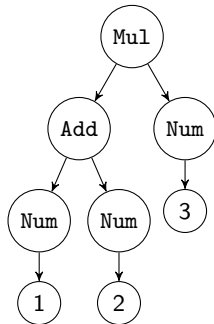
There might be **multiple** concrete syntax for the **same** abstract syntax:

$(1 + 2) * 3$

$(* (+ 1 2) 3)$

$\Rightarrow$

MUL[ADD[1; 2]; 3]



## 1. Syntax

- Backus-Naur Form (BNF)
- Concrete Syntax of AE
- Abstract Syntax of AE
- Concrete vs. Abstract Syntax

## 2. Semantics

- Inference Rules
- Big-Step (Natural) Semantics of AE
- Small-Step (Reduction) Semantics of AE

There exist many different kinds of semantics:

- **Axiomatic semantics** defines the meaning of a program by specifying the properties that hold after its execution.
- **Denotational semantics** defines the meaning of a program by mapping it to a mathematical object that represents its meaning.
- **Operational semantics** defines the meaning of a program by specifying how it executes on a machine.
- ...

In this course, we will focus on **operational semantics**, and there are two different representative styles:

- **Big-Step (Natural) Semantics** defines the meaning of a program by specifying how it executes on a machine in one big step.
- **Small-Step (Reduction) Semantics** defines the meaning of a program by specifying how it executes on a machine step-by-step.

Operational semantics is defined by **inference rules**.

An **inference rule** consists of multiple **premises** and one **conclusion**:

$$\frac{premise_1 \quad premise_2 \quad \cdots \quad premise_n}{conclusion}$$

meaning that “if all the premises are true, then the conclusion is true”:

$$premise_1 \wedge premise_2 \wedge \cdots \wedge premise_n \implies conclusion$$

For example,

$$\frac{A \implies B \quad B \implies C}{A \implies C}$$

means that “if  $A$  implies  $B$ , and  $B$  implies  $C$ , then  $A$  implies  $C$ ”.

$$\boxed{\vdash e \Rightarrow n}$$

It means that “the expression  $e$  evaluates to the number  $n$ ”.

Let's define the **big-step (natural) semantics** of AE:

$$\begin{array}{lcl} e ::= n & \text{(Num)} & \\ \quad | e + e & \text{(Add)} & \\ \quad | e * e & \text{(Mul)} & \end{array} \quad \Longrightarrow \quad \begin{array}{l} \text{Num} \frac{}{\vdash n \Rightarrow n} \\ \\ \text{ADD} \frac{\vdash e_1 \Rightarrow n_1 \quad \vdash e_2 \Rightarrow n_2}{\vdash e_1 + e_2 \Rightarrow n_1 + n_2} \\ \\ \text{MUL} \frac{\vdash e_1 \Rightarrow n_1 \quad \vdash e_2 \Rightarrow n_2}{\vdash e_1 * e_2 \Rightarrow n_1 \times n_2} \end{array}$$

$$\begin{array}{l} \text{NUM} \frac{}{\vdash n \Rightarrow n} \quad \text{ADD} \frac{\vdash e_1 \Rightarrow n_1 \quad \vdash e_2 \Rightarrow n_2}{\vdash e_1 + e_2 \Rightarrow n_1 + n_2} \quad \text{MUL} \frac{\vdash e_1 \Rightarrow n_1 \quad \vdash e_2 \Rightarrow n_2}{\vdash e_1 * e_2 \Rightarrow n_1 \times n_2} \end{array}$$

Let's prove  $\vdash (1 + 2) * 3 \Rightarrow 9$  by drawing a **derivation tree**:

$$\begin{array}{c} \text{NUM} \frac{}{\vdash 1 \Rightarrow 1} \quad \text{NUM} \frac{}{\vdash 2 \Rightarrow 2} \\ \text{ADD} \frac{}{\vdash 1 + 2 \Rightarrow 3} \quad \text{NUM} \frac{}{\vdash 3 \Rightarrow 3} \\ \text{MUL} \frac{}{\vdash (1 + 2) * 3 \Rightarrow 9} \end{array}$$

Let's prove  $\vdash 1 + 2 * 3 \Rightarrow 7$  by drawing a **derivation tree**:

---


$$\vdash 1 + 2 * 3 \Rightarrow$$



$$\boxed{e \rightarrow e}$$

It means that “the one-step evaluation result of  $e$  is  $e$ ”.

Let's define the **small-step (reduction) semantics** of AE:

$$\begin{array}{lcl}
 e ::= & n & (\text{Num}) \\
 & | \quad e + e & (\text{Add}) \\
 & | \quad e * e & (\text{Mul})
 \end{array}
 \quad \Longrightarrow \quad
 \begin{array}{c}
 \frac{e_1 \rightarrow e'_1}{e_1 + e_2 \rightarrow e'_1 + e_2} \quad \frac{e_1 \rightarrow e'_1}{e_1 * e_2 \rightarrow e'_1 * e_2} \\
 \\
 \frac{e_2 \rightarrow e'_2}{n_1 + e_2 \rightarrow n_1 + e'_2} \quad \frac{e_2 \rightarrow e'_2}{n_1 * e_2 \rightarrow n_1 * e'_2} \\
 \\
 \frac{}{n_1 + n_2 \rightarrow n_1 + n_2} \quad \frac{}{n_1 * n_2 \rightarrow n_1 \times n_2}
 \end{array}$$

$$\frac{e_1 \rightarrow e'_1}{e_1 + e_2 \rightarrow e'_1 + e_2}$$

$$\frac{e_2 \rightarrow e'_2}{n_1 + e_2 \rightarrow n_1 + e'_2}$$

$$\frac{}{n_1 + n_2 \rightarrow n_1 + n_2}$$

$$\frac{e_1 \rightarrow e'_1}{e_1 * e_2 \rightarrow e'_1 * e_2}$$

$$\frac{e_2 \rightarrow e'_2}{n_1 * e_2 \rightarrow n_1 * e'_2}$$

$$\frac{}{n_1 * n_2 \rightarrow n_1 \times n_2}$$

Let's prove  $(1 + 2) * 3 \rightarrow^* 9$  by showing a **reduction sequence**:

(Note that  $\rightarrow^*$  denotes the reflexive-transitive closure of  $\rightarrow$ .)

$$(1 + 2) * 3 \quad \rightarrow \quad 3 * 3 \quad \rightarrow \quad 9$$

Let's prove  $(1 + 2) * 3 \rightarrow^* 9$  by showing a **reduction sequence**:

$$1 + 2 * 3 \quad \rightarrow$$

## 1. Syntax

- Backus-Naur Form (BNF)
- Concrete Syntax of AE
- Abstract Syntax of AE
- Concrete vs. Abstract Syntax

## 2. Semantics

- Inference Rules
- Big-Step (Natural) Semantics of AE
- Small-Step (Reduction) Semantics of AE

- Syntax and Semantics (2)

Jihyeok Park

[jihyeok\\_park@korea.ac.kr](mailto:jihyeok_park@korea.ac.kr)

<https://plrg.korea.ac.kr>