Lecture 2 – Syntax and Semantics (1)

COSE212: Programming Languages

Jihyeok Park

PLRG

2023 Fall

Recall

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

Programming Languages

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.

Arithmetic Expressions

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.

Arithmetic Expressions

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**)?

Arithmetic Expressions

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**)?

Contents

Contents

Backus-Naur Form (BNF)

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.

Backus-Naur Form (BNF)

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>
```

Concrete Syntax of AE

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

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

Concrete Syntax of AE

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> ⇒ <expr>*<expr>                ⇒ (<expr>)*<expr>
       ⇒ (<expr>+<expr>)*<expr>       ⇒ (<number>+<expr>)*<expr>
       ⇒ (1+<expr>)*<expr>            ⇒ (1+<number>)*<expr>
       ⇒ (1+2)*<expr>                 ⇒ (1+2)*<number>
       ⇒ (1+2)*3
```

Abstract Syntax of AE

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

$$
\begin{array}{rcll}
e & ::= & n & (\texttt{Num}) \\
  & \mid & e + e & (\texttt{Add}) \\
  & \mid & e * e & (\texttt{Mul})
\end{array}
$$

Abstract Syntax of AE

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

$$
\begin{array}{rcll}
e & ::= & n & (\texttt{Num}) \\
 & \mid & e + e & (\texttt{Add}) \\
 & \mid & e * e & (\texttt{Mul})
\end{array}
$$

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

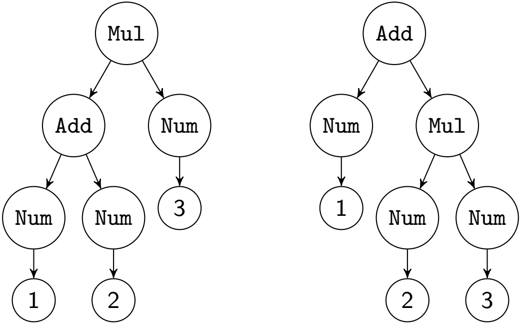
Abstract Syntax of AE

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

$$
\begin{array}{rrll}
e & ::= & n & (\texttt{Num}) \\
 & \mid & e + e & (\texttt{Add}) \\
 & \mid & e * e & (\texttt{Mul})
\end{array}
$$

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:



Concrete vs. Abstract Syntax

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

Concrete vs. Abstract Syntax

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> "]"
```

$$
\begin{array}{rrll}
e & ::= & n & (\texttt{Num}) \\
  & \mid & e + e & (\texttt{Add}) \\
  & \mid & e * e & (\texttt{Mul})
\end{array}
$$

Concrete vs. Abstract Syntax

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

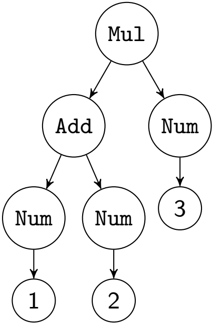
There might be **multiple** concrete syntax for the **same** abstract syntax:

```
(1 + 2) * 3

(* (+ 1 2) 3)

MUL[ADD[1; 2]; 3]
```

$\Rightarrow$



Contents

Semantics

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.
- . . .

Semantics

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.

Inference Rules

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

Inference Rules

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

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

$$\frac{\mathit{premise}_1 \qquad \mathit{premise}_2 \qquad \cdots \qquad \mathit{premise}_n}{\mathit{conclusion}}$$

Inference Rules

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

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

$$\frac{\textit{premise}_1 \qquad \textit{premise}_2 \qquad \cdots \qquad \textit{premise}_n}{\textit{conclusion}}$$

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

$$\textit{premise}_1 \land \textit{premise}_2 \land \cdots \land \textit{premise}_n \implies \textit{conclusion}$$

Inference Rules

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

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

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

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,

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

means that "if A implies B, and B implies C, then A implies C".

Big-Step (Natural) Semantics of AE

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

It means that "the expression e evaluates to the number n".

Big-Step (Natural) Semantics of AE

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

It means that "the expression e evaluates to the number n".

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

$$\begin{array}{rcll} e & ::= & n & (\texttt{Num}) \\ & \mid & e \texttt{ + } e & (\texttt{Add}) \\ & \mid & e \texttt{ * } e & (\texttt{Mul}) \end{array} \quad \Longrightarrow$$

$$\textsc{Num}\ \frac{}{\vdash n \Rightarrow n}$$

$$\textsc{Add}\ \frac{\vdash e_1 \Rightarrow n_1 \qquad \vdash e_2 \Rightarrow n_2}{\vdash e_1 \texttt{ + } e_2 \Rightarrow n_1 + n_2}$$

$$\textsc{Mul}\ \frac{\vdash e_1 \Rightarrow n_1 \qquad \vdash e_2 \Rightarrow n_2}{\vdash e_1 \texttt{ * } e_2 \Rightarrow n_1 \times n_2}$$

Big-Step (Natural) Semantics of AE

$$\textsc{Num}\ \frac{}{\vdash n \Rightarrow n} \qquad \textsc{Add}\ \frac{\vdash e_1 \Rightarrow n_1 \qquad \vdash e_2 \Rightarrow n_2}{\vdash e_1\ \texttt{+}\ e_2 \Rightarrow n_1 + n_2} \qquad \textsc{Mul}\ \frac{\vdash e_1 \Rightarrow n_1 \qquad \vdash e_2 \Rightarrow n_2}{\vdash e_1\ \texttt{*}\ e_2 \Rightarrow n_1 \times n_2}$$

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

Big-Step (Natural) Semantics of AE

$$\textsc{Num}\ \frac{}{\vdash n \Rightarrow n} \qquad \textsc{Add}\ \frac{\vdash e_1 \Rightarrow n_1 \qquad \vdash e_2 \Rightarrow n_2}{\vdash e_1 \texttt{ + } e_2 \Rightarrow n_1 + n_2} \qquad \textsc{Mul}\ \frac{\vdash e_1 \Rightarrow n_1 \qquad \vdash e_2 \Rightarrow n_2}{\vdash e_1 \texttt{ * } e_2 \Rightarrow n_1 \times n_2}$$

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

$$\textsc{Mul}\ \dfrac{\textsc{Add}\ \dfrac{\textsc{Num}\ \dfrac{}{\vdash \texttt{1} \Rightarrow 1} \quad \textsc{Num}\ \dfrac{}{\vdash \texttt{2} \Rightarrow 2}}{\vdash \texttt{1 + 2} \Rightarrow 3} \quad \textsc{Num}\ \dfrac{}{\vdash \texttt{3} \Rightarrow 3}}{\vdash \texttt{(1 + 2) * 3} \Rightarrow 9}$$

Big-Step (Natural) Semantics of AE

$$\textsc{Num}\ \frac{}{\vdash n \Rightarrow n} \qquad \textsc{Add}\ \frac{\vdash e_1 \Rightarrow n_1 \qquad \vdash e_2 \Rightarrow n_2}{\vdash e_1 \texttt{ + } e_2 \Rightarrow n_1 + n_2} \qquad \textsc{Mul}\ \frac{\vdash e_1 \Rightarrow n_1 \qquad \vdash e_2 \Rightarrow n_2}{\vdash e_1 \texttt{ * } e_2 \Rightarrow n_1 \times n_2}$$

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

$$\textsc{Mul}\ \dfrac{\textsc{Add}\ \dfrac{\textsc{Num}\ \dfrac{}{\vdash \texttt{1} \Rightarrow 1} \quad \textsc{Num}\ \dfrac{}{\vdash \texttt{2} \Rightarrow 2}}{\vdash \texttt{1 + 2} \Rightarrow 3} \quad \textsc{Num}\ \dfrac{}{\vdash \texttt{3} \Rightarrow 3}}{\vdash \texttt{(1 + 2) * 3} \Rightarrow 9}$$

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

$$\frac{}{\vdash \texttt{1 + 2 * 3} \Rightarrow}$$

Small-Step (Reduction) Semantics of AE

PLRG

$$\boxed{e \to e}$$

It means that "the one-step evaluation result of e is e".

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

It means that "the one-step evaluation result of e is e".

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

$$
\begin{array}{rcll}
e & ::= & n & (\texttt{Num}) \\
  & \mid & e + e & (\texttt{Add}) \\
  & \mid & e * e & (\texttt{Mul})
\end{array}
\quad \Longrightarrow \quad
\begin{array}{cc}
\dfrac{e_1 \rightarrow e_1'}{e_1 + e_2 \rightarrow e_1' + e_2} & \dfrac{e_1 \rightarrow e_1'}{e_1 * e_2 \rightarrow e_1' * e_2} \\[2em]
\dfrac{e_2 \rightarrow e_2'}{n_1 + e_2 \rightarrow n_1 + e_2'} & \dfrac{e_2 \rightarrow e_2'}{n_1 * e_2 \rightarrow n_1 * e_2'} \\[2em]
\dfrac{}{n_1 + n_2 \rightarrow n_1 + n_2} & \dfrac{}{n_1 * n_2 \rightarrow n_1 \times n_2}
\end{array}
$$

Small-Step (Reduction) Semantics of AE

$$\frac{e_1 \to e_1'}{e_1 \texttt{ + } e_2 \to e_1' \texttt{ + } e_2} \qquad \frac{e_2 \to e_2'}{n_1 \texttt{ + } e_2 \to n_1 \texttt{ + } e_2'} \qquad \frac{}{n_1 \texttt{ + } n_2 \to n_1 + n_2}$$

$$\frac{e_1 \to e_1'}{e_1 \texttt{ * } e_2 \to e_1' \texttt{ * } e_2} \qquad \frac{e_2 \to e_2'}{n_1 \texttt{ * } e_2 \to n_1 \texttt{ * } e_2'} \qquad \frac{}{n_1 \texttt{ * } n_2 \to n_1 \times n_2}$$

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

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

Small-Step (Reduction) Semantics of AE

PLRG

$$\frac{e_1 \to e_1'}{e_1 \texttt{ + } e_2 \to e_1' \texttt{ + } e_2} \qquad \frac{e_2 \to e_2'}{n_1 \texttt{ + } e_2 \to n_1 \texttt{ + } e_2'} \qquad \frac{}{n_1 \texttt{ + } n_2 \to n_1 + n_2}$$

$$\frac{e_1 \to e_1'}{e_1 \texttt{ * } e_2 \to e_1' \texttt{ * } e_2} \qquad \frac{e_2 \to e_2'}{n_1 \texttt{ * } e_2 \to n_1 \texttt{ * } e_2'} \qquad \frac{}{n_1 \texttt{ * } n_2 \to n_1 \times n_2}$$

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

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

$$\texttt{(1 + 2) * 3} \quad \to \quad \texttt{3 * 3} \quad \to \quad \texttt{9}$$

Small-Step (Reduction) Semantics of AE

$$\frac{e_1 \to e_1'}{e_1 \texttt{ + } e_2 \to e_1' \texttt{ + } e_2} \qquad \frac{e_2 \to e_2'}{n_1 \texttt{ + } e_2 \to n_1 \texttt{ + } e_2'} \qquad \frac{}{n_1 \texttt{ + } n_2 \to n_1 + n_2}$$

$$\frac{e_1 \to e_1'}{e_1 \texttt{ * } e_2 \to e_1' \texttt{ * } e_2} \qquad \frac{e_2 \to e_2'}{n_1 \texttt{ * } e_2 \to n_1 \texttt{ * } e_2'} \qquad \frac{}{n_1 \texttt{ * } n_2 \to n_1 \times n_2}$$

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

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

$$\texttt{(1 + 2) * 3} \quad \to \quad \texttt{3 * 3} \quad \to \quad 9$$

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

$$\texttt{1 + 2 * 3} \quad \to$$

Summary

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

Next Lecture

- Syntax and Semantics (2)

Jihyeok Park
jihyeok_park@korea.ac.kr
https://plrg.korea.ac.kr