

Lecture 4 – Identifiers (1)

COSE212: Programming Languages

Jihyeok Park



2023 Fall

- **ADT** for **Abstract Syntax** of AE

```
enum Expr:  
  case Num(number: BigInt)  
  case Add(left: Expr, right: Expr)  
  case Mul(left: Expr, right: Expr)
```

- **Parser** for **Concrete Syntax** of AE

```
lazy val expr: P[Expr] = ...
```

- **Interpreter** for **Semantics** of AE

```
def interp(expr: Expr): Value = ...
```

- **ADT** for **Abstract Syntax** of AE

```
enum Expr:  
  case Num(number: BigInt)  
  case Add(left: Expr, right: Expr)  
  case Mul(left: Expr, right: Expr)
```

- **Parser** for **Concrete Syntax** of AE

```
lazy val expr: P[Expr] = ...
```

- **Interpreter** for **Semantics** of AE

```
def interp(expr: Expr): Value = ...
```

- In this lecture, we will learn **identifiers**.

1. Identifiers

- Bound Identifiers

- Free Identifiers

- Shadowing

2. VAE – AE with Variables

- Concrete Syntax

- Abstract Syntax

- Examples

1. Identifiers

- Bound Identifiers

- Free Identifiers

- Shadowing

2. VAE – AE with Variables

- Concrete Syntax

- Abstract Syntax

- Examples

An **identifier** is a **name** for a certain element in a program.

In Scala, there are diverse kinds of identifiers:

```
// variable names
val x: Int = 42

// function and parameter names
def add(a: Int, b: Int): Int = a + b

// class and field names
case class Person(name: String, age: Int)

...
```

```
val x: Int = 3
def add(a: Int, b: Int): Int = {
  val x: Int = a + b
  val y: Int = x + 1
  x + y
}
add(y, z)
```

A **bound identifier** is an identifier that is **defined** in a program.

- A **binding occurrence** of an identifier is the occurrence in its definition.
- A **scope** of an identifier is a code region where the identifier is usable.
- A **bound occurrence** of an identifier is an occurrence in its scope.

```
val x: Int = 3
def add(a: Int, b: Int): Int = {
  val x: Int = a + b
  val y: Int = x + 1
  x + y
}
add(y, z)
```

A **bound identifier** is an identifier that is **defined** in a program.

- A **binding occurrence** of an identifier is the occurrence in its definition.
- A **scope** of an identifier is a code region where the identifier is usable.
- A **bound occurrence** of an identifier is an occurrence in its scope.


```
val x: Int = 3
def add(a: Int, b: Int): Int = {
  val x: Int = a + b
  val y: Int = x + 1
  x + y
}
add(y, z)
```

A **bound identifier** is an identifier that is **defined** in a program.

- A **binding occurrence** of an identifier is the occurrence in its definition.
- A **scope** of an identifier is a code region where the identifier is usable.
- A **bound occurrence** of an identifier is an occurrence in its scope.

```
val x: Int = 3
def add(a: Int, b: Int): Int = {
  val x: Int = a + b
  val y: Int = x + 1
  x + y
}
add(y, z)
```

A **free identifier** is an identifier that is **not defined** in a program.

```
val x: Int = 3
def add(a: Int, b: Int): Int = {
  val x: Int = a + b
  val y: Int = x + 1
  x + y
}
add(y, z)
```

Shadowing means that the innermost binding occurrence shadows the outer binding occurrences of the same name.

- A **shadowing identifier** is an identifier that shadows another identifier.
- A **shadowed identifier** is an identifier that is shadowed by another identifier.

1. Identifiers

Bound Identifiers

Free Identifiers

Shadowing

2. VAE – AE with Variables

Concrete Syntax

Abstract Syntax

Examples

Now, we want to extend AE into VAE with **variables**:

```
/* VAE */  
val x = 1 + 2; // x = 1 + 2 = 3  
val y = x + 3; // y = x + 3 = 3 + 3 = 6  
y + 4          // 6 + 4 = 10
```

First, we need to define the **syntax** of VAE.

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

```
<expr> ::= <number>  
         | <expr> "+" <expr>  
         | <expr> "*" <expr>  
         | "(" <expr> ")"  
         | "{" <expr> "}"  
         | "val" <id> "=" <expr> ";" <expr>  
         | <id>
```

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

```
<expr> ::= <number>
        | <expr> "+" <expr>
        | <expr> "*" <expr>
        | "(" <expr> ")"
        | "{" <expr> "}"
        | "val" <id> "=" <expr> ";" <expr>
        | <id>
```

For example, the following string is a valid VAE program:

```
/* VAE */
val x = 1 + 2;
val y = x + 3;
y + 4
```

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

```
<expr> ::= <number>
        | <expr> "+" <expr>
        | <expr> "*" <expr>
        | "(" <expr> ")"
        | "{" <expr> "}"
        | "val" <id> "=" <expr> ";" <expr>
        | <id>
```

For example, the following string is a valid VAE program:

```
/* VAE */
val x = 1 + 2;
val y = x + 3;
y + 4
```

means

```
/* VAE */
val x = 1 + 2; {
    val y = x + 3; {
        y + 4
    }
}
```


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

$e ::=$	n	(Num)
	$e + e$	(Add)
	$e \times e$	(Mul)
	$\text{val } x = e; e$	(Val)
	x	(Id)

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

e	$::=$	n	(Num)
		$e + e$	(Add)
		$e \times e$	(Mul)
		$\text{val } x = e; e$	(Val)
		x	(Id)

Let's define Scala **ADTs** for VAE.

```
enum Expr:  
  case Num(number: BigInt)  
  case Add(left: Expr, right: Expr)  
  case Mul(left: Expr, right: Expr)  
  case Val(name: String, init: Expr, body: Expr)  
  case Id(name: String)
```

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

$e ::= n$	(Num)
$e + e$	(Add)
$e \times e$	(Mul)
$\text{val } x = e; e$	(Val)
x	(Id)

Let's define Scala **ADTs** for VAE.

```
enum Expr:  
  case Num(number: BigInt)  
  case Add(left: Expr, right: Expr)  
  case Mul(left: Expr, right: Expr)  
  case Val(name: String, init: Expr, body: Expr)  
  case Id(name: String)
```

For example, a VAE expression `val x = 1; x + 2` is represented as:

```
Val("x", Num(1), Add(Id("x"), Num(2)))
```

For each VAE program, please draw:

- an **arrow** from each **bound occurrence** to its **binding occurrence**.
- a **dotted arrow** from each **shadowing variable** to its **shadowed one**.
- an **X** mark on each **free identifier**.

```
/* VAE */  
val x = 1; x
```

```
/* VAE */  
val x = x + 1;  
val y = x * 2;  
val x = y + x;  
x * z
```

```
/* VAE */  
val x = 1;  
val y = {  
    val x = 2 * x;  
    { val y = x; y } + { val y = 3; y }  
};  
x + y
```

1. Identifiers

- Bound Identifiers

- Free Identifiers

- Shadowing

2. VAE – AE with Variables

- Concrete Syntax

- Abstract Syntax

- Examples

- Identifiers (2)

Jihyeok Park

`jihyeok_park@korea.ac.kr`

`https://plrg.korea.ac.kr`