Lecture 27 – Course Review

COSE212: Programming Languages

Jihyeok Park

PLRG

2024 Fall

Goal of This Course

Goal of This Course

Learn **Essential Concepts** of **Programming Languages**

Goal of This Course

Learn **Essential Concepts** of **Programming Languages**

- Why?

Goal of This Course

Learn **Essential Concepts** of **Programming Languages**

- Why?
  - To **learn new programming languages** quickly.
  - To **evaluate** and pick the best language for a given task.
  - To **design** your own **specialized languages** for specific tasks.

Goal of This Course

Learn **Essential Concepts** of **Programming Languages**

- Why?
  - To **learn new programming languages** quickly.
  - To **evaluate** and pick the best language for a given task.
  - To **design** your own **specialized languages** for specific tasks.
- How?

Goal of This Course

Learn **Essential Concepts** of **Programming Languages**

- Why?
  - To **learn new programming languages** quickly.
  - To **evaluate** and pick the best language for a given task.
  - To **design** your own **specialized languages** for specific tasks.
- How?

By **Designing** Diverse **Programming Languages**

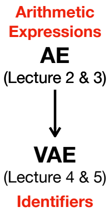
  - By **designing** programming languages in a **mathematical** way.
  - By **implementing** their **interpreters** using **Scala**.

Summary

(Part 1) Untyped Languages	(Part 2) Typed Languages
Arithmetic Expressions **AE** (Lecture 2 & 3)	

Summary

(Part 1) Untyped Languages	(Part 2) Typed Languages



Arithmetic Expressions

AE

(Lecture 2 & 3)

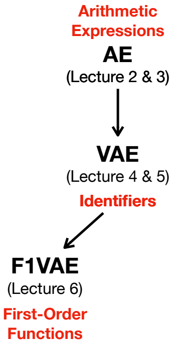
VAE

(Lecture 4 & 5)

Identifiers

Summary

(Part 1) Untyped Languages	(Part 2) Typed Languages



Arithmetic Expressions

AE

(Lecture 2 & 3)

VAE

(Lecture 4 & 5)

Identifiers

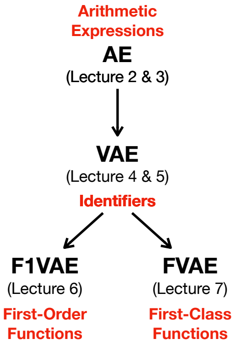
F1VAE

(Lecture 6)

First-Order Functions

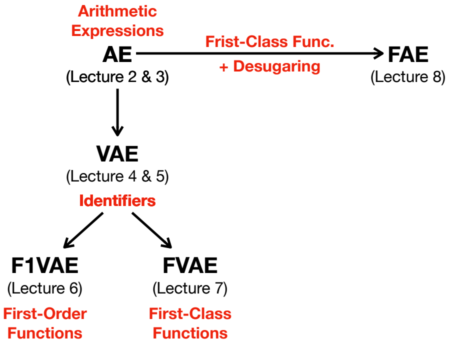
Summary

(Part 1) Untyped Languages	(Part 2) Typed Languages



Summary

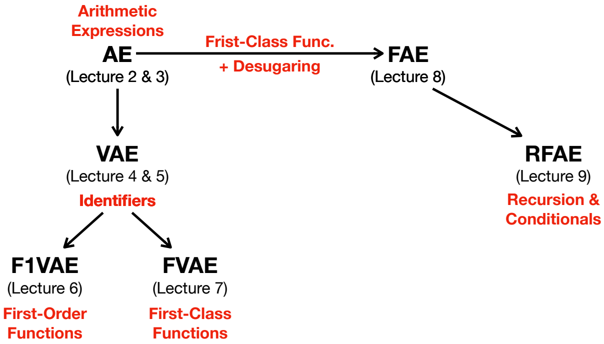
(Part 1) Untyped Languages	(Part 2) Typed Languages



Summary

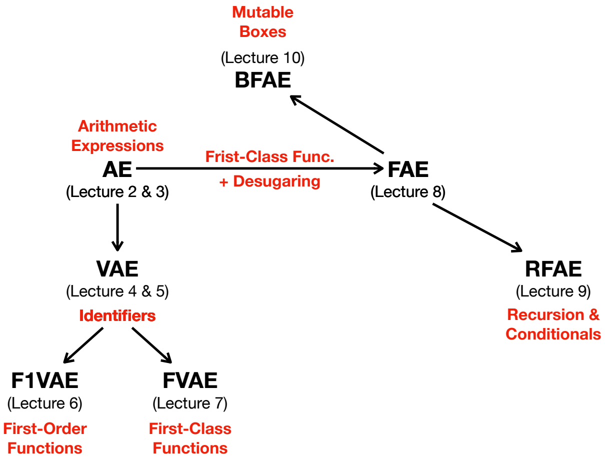
(Part 1)
Untyped Languages

(Part 2)
Typed Languages



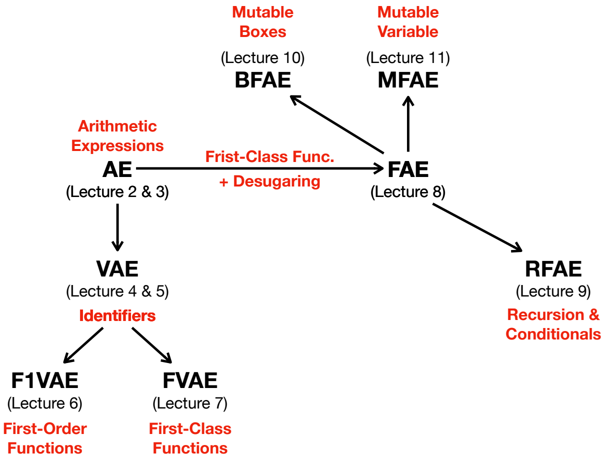
Summary

(Part 1) Untyped Languages	(Part 2) Typed Languages



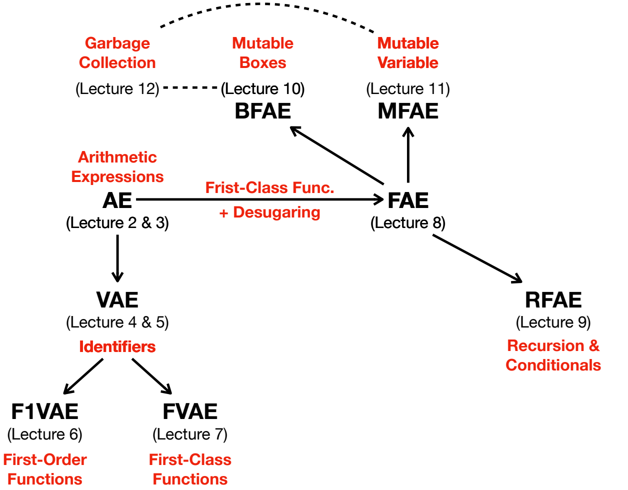
Summary

(Part 1) Untyped Languages	(Part 2) Typed Languages



Summary

(Part 1) Untyped Languages	(Part 2) Typed Languages



Garbage Collection

(Lecture 12)

Mutable Boxes

(Lecture 10)

BFAE

Mutable Variable

(Lecture 11)

MFAE

Arithmetic Expressions

AE

(Lecture 2 & 3)

Frist-Class Func.

+ Desugaring

FAE

(Lecture 8)

VAE

(Lecture 4 & 5)

Identifiers

RFAE

(Lecture 9)

Recursion & Conditionals

F1VAE

(Lecture 6)

First-Order Functions

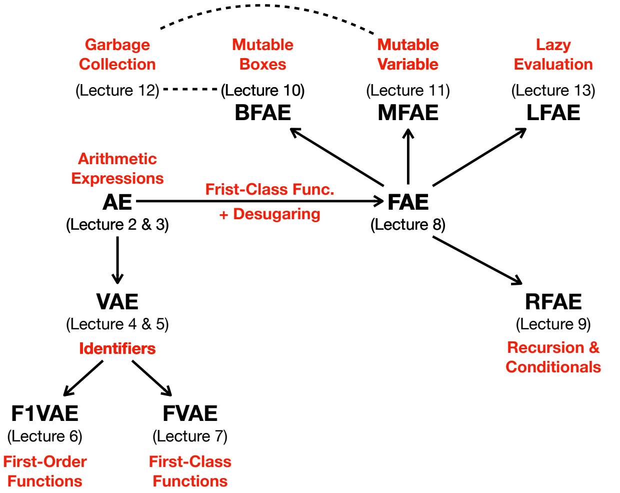
FVAE

(Lecture 7)

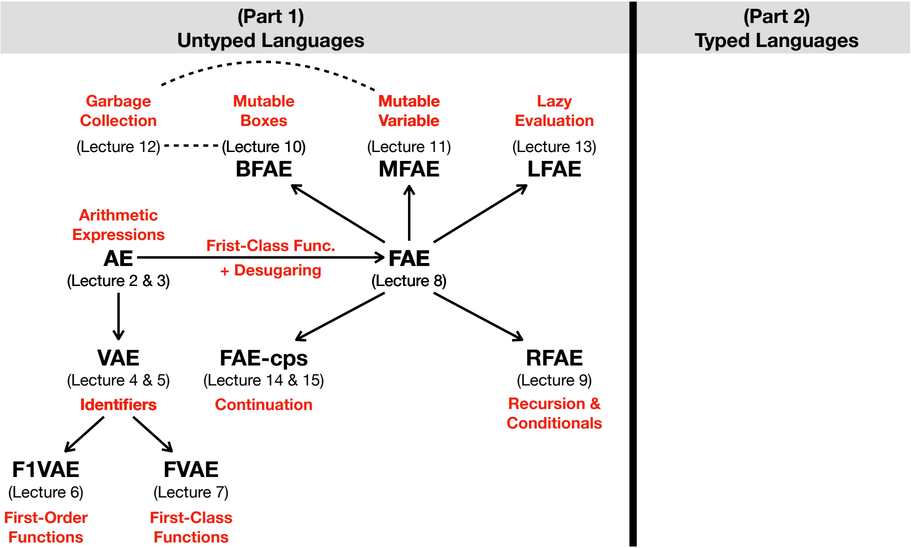
First-Class Functions

Summary

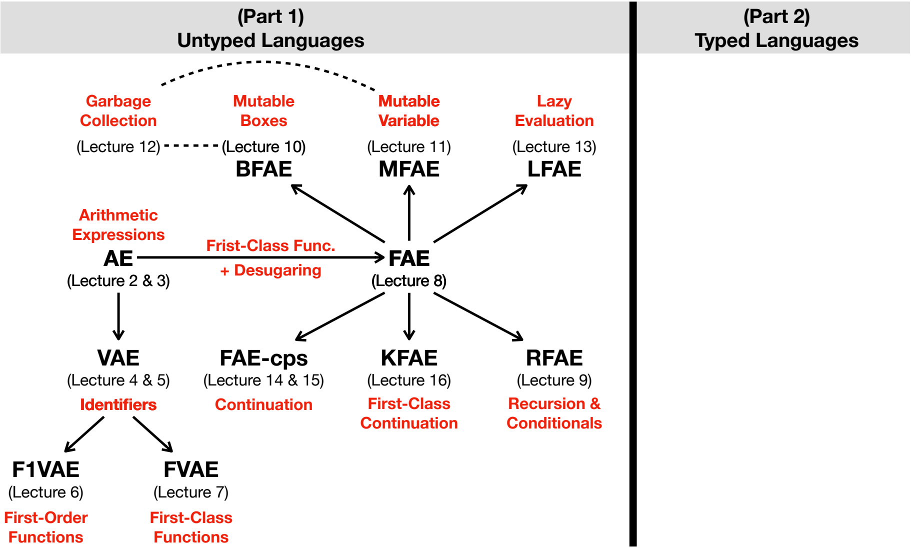
(Part 1) Untyped Languages	(Part 2) Typed Languages



Summary


(Part 1)
Untyped Languages
(Part 2)
Typed Languages
Garbage Collection
(Lecture 12)
Mutable Boxes
(Lecture 10)
BFAE
Mutable Variable
(Lecture 11)
MFAE
Lazy Evaluation
(Lecture 13)
LFAE
Arithmetic Expressions
AE
(Lecture 2 & 3)
Frist-Class Func.
+ Desugaring
FAE
(Lecture 8)
VAE
(Lecture 4 & 5)
Identifiers
FAE-cps
(Lecture 14 & 15)
Continuation
RFAE
(Lecture 9)
Recursion & Conditionals
F1VAE
(Lecture 6)
First-Order Functions
FVAE
(Lecture 7)
First-Class Functions

Summary

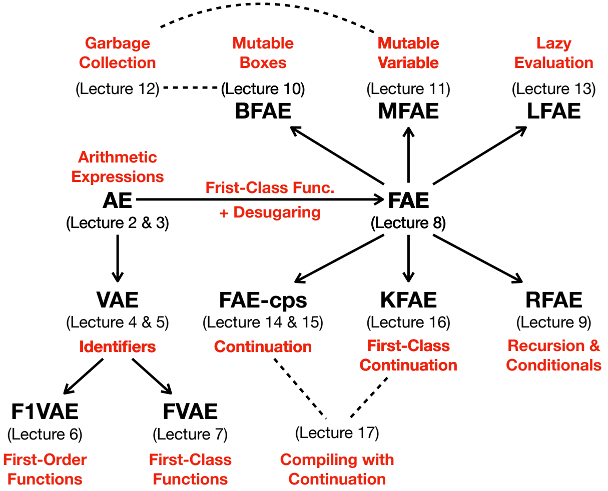


(Part 1)
Untyped Languages
(Part 2)
Typed Languages
Garbage Collection
(Lecture 12)
Mutable Boxes
(Lecture 10)
BFAE
Mutable Variable
(Lecture 11)
MFAE
Lazy Evaluation
(Lecture 13)
LFAE
Arithmetic Expressions
AE
(Lecture 2 & 3)
Frist-Class Func.
+ Desugaring
FAE
(Lecture 8)
VAE
(Lecture 4 & 5)
Identifiers
FAE-cps
(Lecture 14 & 15)
Continuation
KFAE
(Lecture 16)
First-Class Continuation
RFAE
(Lecture 9)
Recursion & Conditionals
F1VAE
(Lecture 6)
First-Order Functions
FVAE
(Lecture 7)
First-Class Functions

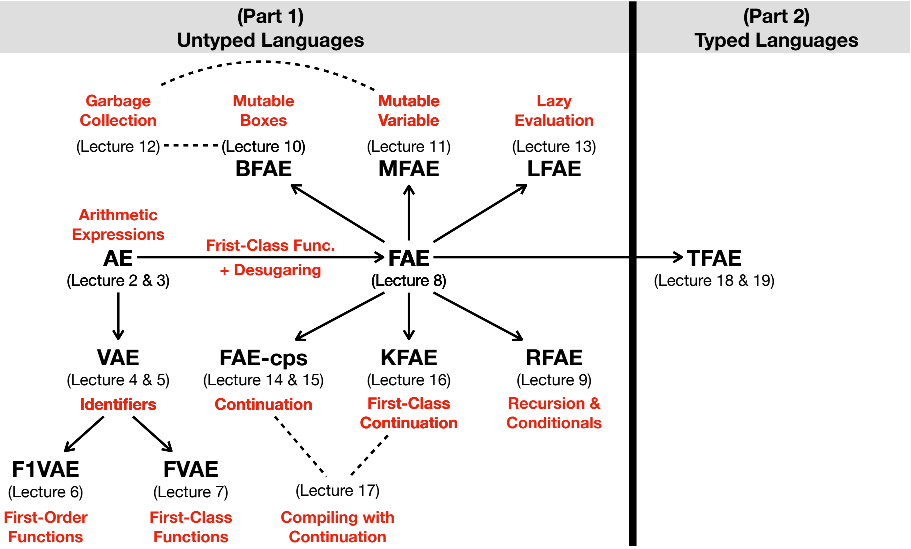
Summary

(Part 1)
Untyped Languages

(Part 2)
Typed Languages

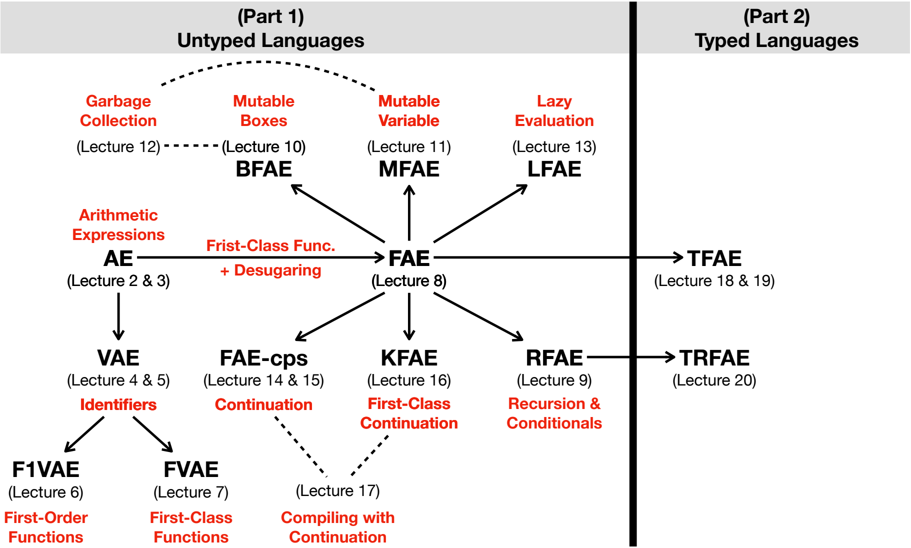


Summary

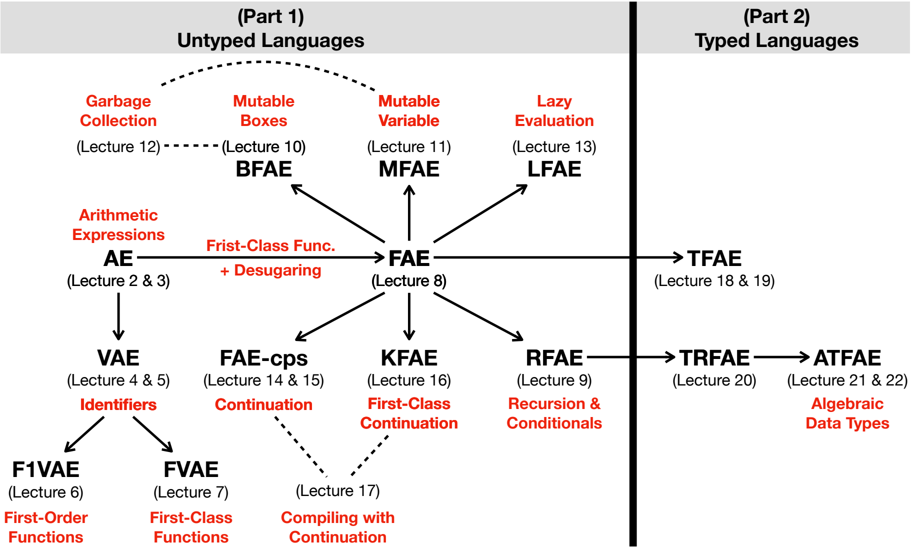


(Part 1)
Untyped Languages
(Part 2)
Typed Languages
Garbage Collection
(Lecture 12)
Mutable Boxes
(Lecture 10)
BFAE
Mutable Variable
(Lecture 11)
MFAE
Lazy Evaluation
(Lecture 13)
LFAE
Arithmetic Expressions
AE
(Lecture 2 & 3)
Frist-Class Func.
+ Desugaring
FAE
(Lecture 8)
TFAE
(Lecture 18 & 19)
VAE
(Lecture 4 & 5)
Identifiers
FAE-cps
(Lecture 14 & 15)
Continuation
KFAE
(Lecture 16)
First-Class Continuation
RFAE
(Lecture 9)
Recursion & Conditionals
F1VAE
(Lecture 6)
First-Order Functions
FVAE
(Lecture 7)
First-Class Functions
(Lecture 17)
Compiling with Continuation

Summary

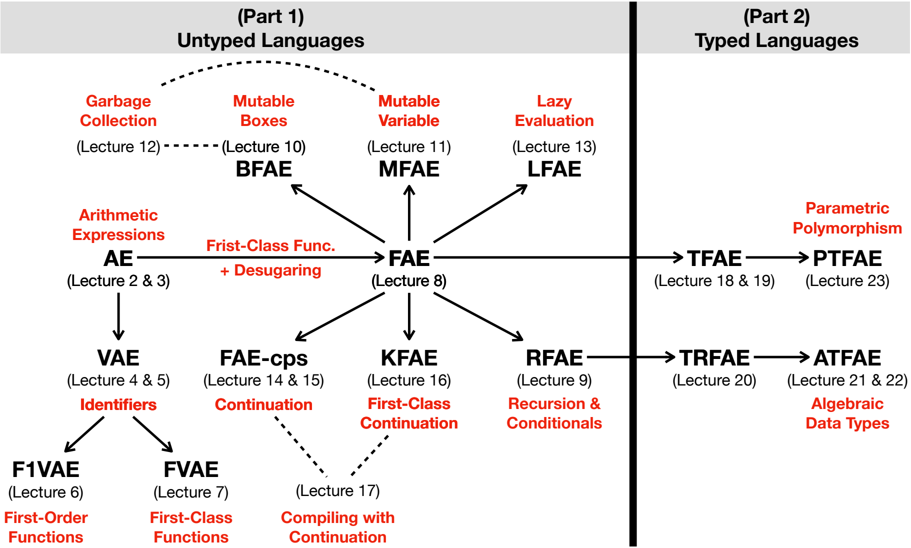

(Part 1)
Untyped Languages
(Part 2)
Typed Languages
Garbage Collection
(Lecture 12)
Mutable Boxes
(Lecture 10)
BFAE
Mutable Variable
(Lecture 11)
MFAE
Lazy Evaluation
(Lecture 13)
LFAE
Arithmetic Expressions
AE
(Lecture 2 & 3)
Frist-Class Func.
+ Desugaring
FAE
(Lecture 8)
TFAE
(Lecture 18 & 19)
VAE
(Lecture 4 & 5)
Identifiers
FAE-cps
(Lecture 14 & 15)
Continuation
KFAE
(Lecture 16)
First-Class Continuation
RFAE
(Lecture 9)
Recursion & Conditionals
TRFAE
(Lecture 20)
F1VAE
(Lecture 6)
First-Order Functions
FVAE
(Lecture 7)
First-Class Functions
(Lecture 17)
Compiling with Continuation

Summary



(Part 1)
Untyped Languages
(Part 2)
Typed Languages
Garbage Collection
(Lecture 12)
Mutable Boxes
(Lecture 10)
BFAE
Mutable Variable
(Lecture 11)
MFAE
Lazy Evaluation
(Lecture 13)
LFAE
Arithmetic Expressions
AE
(Lecture 2 & 3)
Frist-Class Func.
+ Desugaring
FAE
(Lecture 8)
TFAE
(Lecture 18 & 19)
VAE
(Lecture 4 & 5)
Identifiers
FAE-cps
(Lecture 14 & 15)
Continuation
KFAE
(Lecture 16)
First-Class Continuation
RFAE
(Lecture 9)
Recursion & Conditionals
TRFAE
(Lecture 20)
ATFAE
(Lecture 21 & 22)
Algebraic Data Types
F1VAE
(Lecture 6)
First-Order Functions
FVAE
(Lecture 7)
First-Class Functions
(Lecture 17)
Compiling with Continuation

Summary



(Part 1)
Untyped Languages

(Part 2)
Typed Languages

Garbage Collection
(Lecture 12)

Mutable Boxes
(Lecture 10)
BFAE

Mutable Variable
(Lecture 11)
MFAE

Lazy Evaluation
(Lecture 13)
LFAE

Arithmetic Expressions
AE
(Lecture 2 & 3)

Frist-Class Func.
+ Desugaring

FAE
(Lecture 8)

TFAE
(Lecture 18 & 19)

Parametric Polymorphism
PTFAE
(Lecture 23)

VAE
(Lecture 4 & 5)
Identifiers

FAE-cps
(Lecture 14 & 15)
Continuation

KFAE
(Lecture 16)
First-Class Continuation

RFAE
(Lecture 9)
Recursion & Conditionals

TRFAE
(Lecture 20)

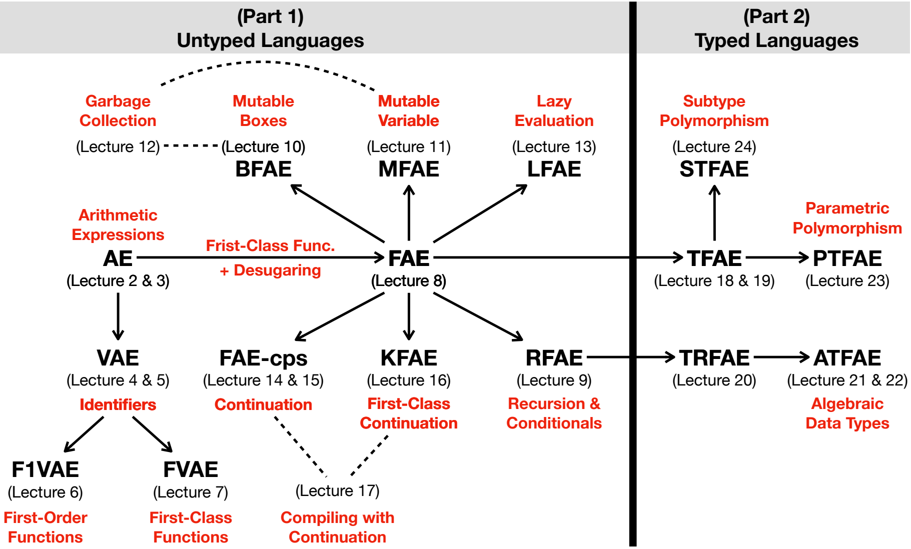
ATFAE
(Lecture 21 & 22)
Algebraic Data Types

F1VAE
(Lecture 6)
First-Order Functions

FVAE
(Lecture 7)
First-Class Functions

(Lecture 17)
Compiling with Continuation

Summary



(Part 1)
Untyped Languages

(Part 2)
Typed Languages

Garbage Collection
(Lecture 12)

Mutable Boxes
(Lecture 10)
BFAE

Mutable Variable
(Lecture 11)
MFAE

Lazy Evaluation
(Lecture 13)
LFAE

Subtype Polymorphism
(Lecture 24)
STFAE

Arithmetic Expressions
AE
(Lecture 2 & 3)

Frist-Class Func.
+ Desugaring

FAE
(Lecture 8)

TFAE
(Lecture 18 & 19)

Parametric Polymorphism
PTFAE
(Lecture 23)

VAE
(Lecture 4 & 5)
Identifiers

FAE-cps
(Lecture 14 & 15)
Continuation

KFAE
(Lecture 16)
First-Class Continuation

RFAE
(Lecture 9)
Recursion & Conditionals

TRFAE
(Lecture 20)

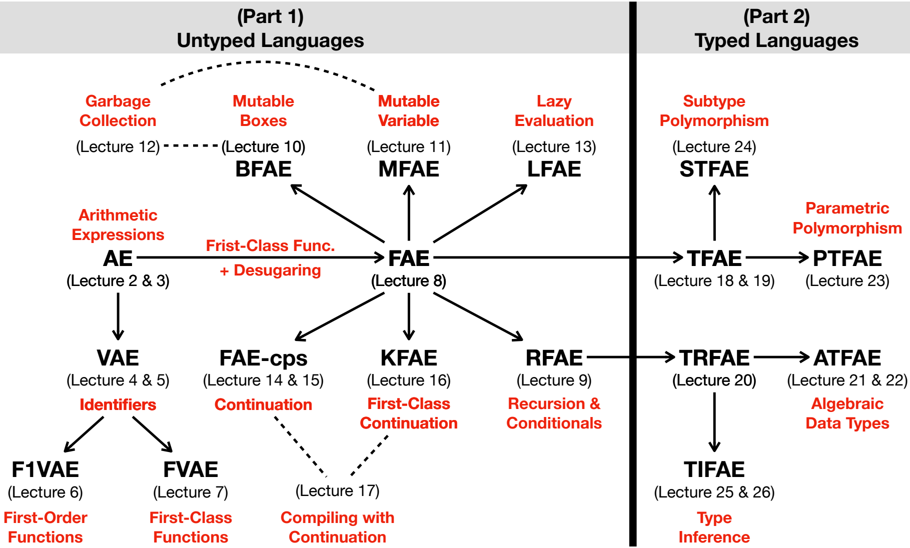
ATFAE
(Lecture 21 & 22)
Algebraic Data Types

F1VAE
(Lecture 6)
First-Order Functions

FVAE
(Lecture 7)
First-Class Functions

(Lecture 17)
Compiling with Continuation

Summary



(Part 1)
Untyped Languages

(Part 2)
Typed Languages

Garbage Collection (Lecture 12)

Mutable Boxes (Lecture 10) BFAE

Mutable Variable (Lecture 11) MFAE

Lazy Evaluation (Lecture 13) LFAE

Subtype Polymorphism (Lecture 24) STFAE

Arithmetic Expressions AE (Lecture 2 & 3)

Frist-Class Func. + Desugaring

FAE (Lecture 8)

TFAE (Lecture 18 & 19)

Parametric Polymorphism PTFAE (Lecture 23)

VAE (Lecture 4 & 5) Identifiers

FAE-cps (Lecture 14 & 15) Continuation

KFAE (Lecture 16) First-Class Continuation

RFAE (Lecture 9) Recursion & Conditionals

TRFAE (Lecture 20)

ATFAE (Lecture 21 & 22) Algebraic Data Types

F1VAE (Lecture 6) First-Order Functions

FVAE (Lecture 7) First-Class Functions

(Lecture 17) Compiling with Continuation

TIFAE (Lecture 25 & 26) Type Inference

Applications of PL Foundations

A deeper and broader understanding of programming languages can help you in:

- Software Engineering
- Software Testing
- Software Verification
- Software Analysis
- Software Security
- . . .

Application 1 – Software Analysis

We can develop a **static analyzer** for diverse purposes (e.g., optimization, understanding, bug detection, etc.) using the PL foundations.[1]

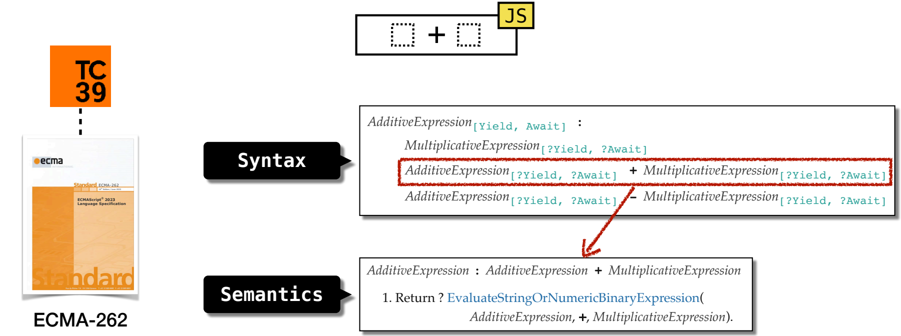
```
let x = /* 1 or 2 */;
let y = /* any str */;
let o = new Observable(subscriber => {
  subscriber.next(1);
  subscriber.next(2);
  subscriber.next(3);
});
o.subscribe(k => x *= k); // x: 6 or 12
o.subscribe(k => y += k); // y: any str + "123"
```

An example of static analysis for a JavaScript program.

[1] [FSE'22] **Jihyeok Park**, Seungmin An, and Sukyoung Ryu, "Automatically Deriving JavaScript Static Analyzers from Specifications using Meta-level Static Analysis"

Application 2 – Mechanized Specification

To understand syntax and semantics of JavaScript language, we need to read the official language specification, called ECMA-262.[2]



AdditiveExpression[Yield, Await] :
  MultiplicativeExpression[?Yield, ?Await]
  AdditiveExpression[?Yield, ?Await] + *MultiplicativeExpression*[?Yield, ?Await]
  AdditiveExpression[?Yield, ?Await] - *MultiplicativeExpression*[?Yield, ?Await]

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

1. Return ? EvaluateStringOrNumericBinaryExpression(*AdditiveExpression*, +, *MultiplicativeExpression*).

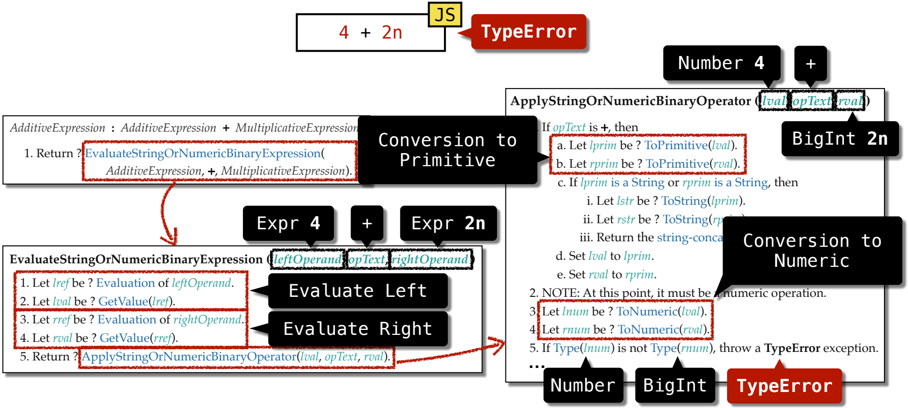
ECMA-262
(JavaScript Spec.)

However, it consists of **800+ pages** with pseudocode-style algorithms as language semantics. It is laborious to understand and maintain the spec.

[2] https://tc39.es/ecma262/

Application 2 – Mechanized Specification

For example, we need to read the following steps to understand why the JavaScript program `4 + 2n` throws a run-time `TypeError`:



JS

```
4 + 2n
```

TypeError

AdditiveExpression **:** *AdditiveExpression* **+** *MultiplicativeExpression*

1. Return ? EvaluateStringOrNumericBinaryExpression(*AdditiveExpression*, **+**, *MultiplicativeExpression*).

Conversion to Primitive

Expr 4

+

Expr 2n

EvaluateStringOrNumericBinaryExpression (*leftOperand*, *opText*, *rightOperand*)

1. Let *lref* be ? Evaluation of *leftOperand*.
2. Let *lval* be ? GetValue(*lref*).
3. Let *rref* be ? Evaluation of *rightOperand*.
4. Let *rval* be ? GetValue(*rref*).
5. Return ? ApplyStringOrNumericBinaryOperator(*lval*, *opText*, *rval*).

Evaluate Left

Evaluate Right

Number 4

+

BigInt 2n

ApplyStringOrNumericBinaryOperator (*lval*, *opText*, *rval*)

```
1. If opText is +, then
   a. Let lprim be ? ToPrimitive(lval).
   b. Let rprim be ? ToPrimitive(rval).
   c. If lprim is a String or rprim is a String, then
      i. Let lstr be ? ToString(lprim).
      ii. Let rstr be ? ToString(rp
      iii. Return the string-conca
   d. Set lval to lprim.
   e. Set rval to rprim.
2. NOTE: At this point, it must be a numeric operation.
3. Let lnum be ? ToNumeric(lval).
4. Let rnum be ? ToNumeric(rval).
5. If Type(lnum) is not Type(rnum), throw a TypeError exception.
...
```

Conversion to Numeric

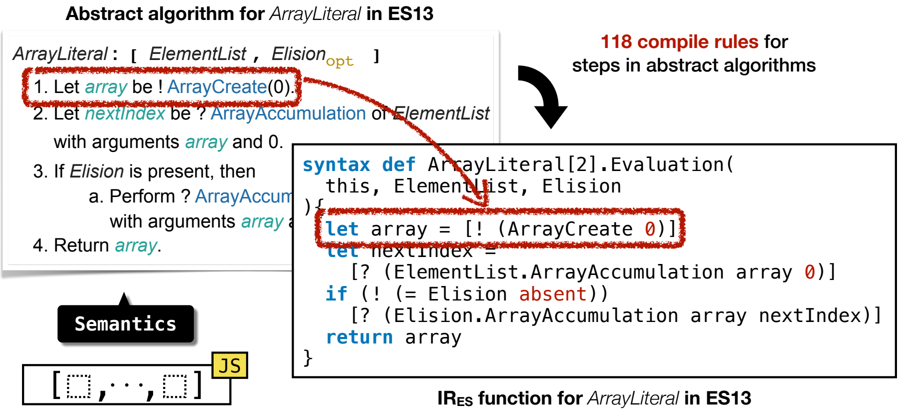
Number

BigInt

TypeError

Application 2 – Mechanized Specification

To alleviate the problem, we **design** a programming language to represent algorithms in the spec and automatically **extract** the semantics from the language specification.[3]



Abstract algorithm for *ArrayLiteral* in ES13

ArrayLiteral : [*ElementList* , *Elision*opt]

```
1. Let array be ! ArrayCreate(0).
2. Let nextIndex be ? ArrayAccumulation of ElementList
   with arguments array and 0.
3. If Elision is present, then
    a. Perform ? ArrayAccum
       with arguments array a
4. Return array.
```

118 compile rules for steps in abstract algorithms

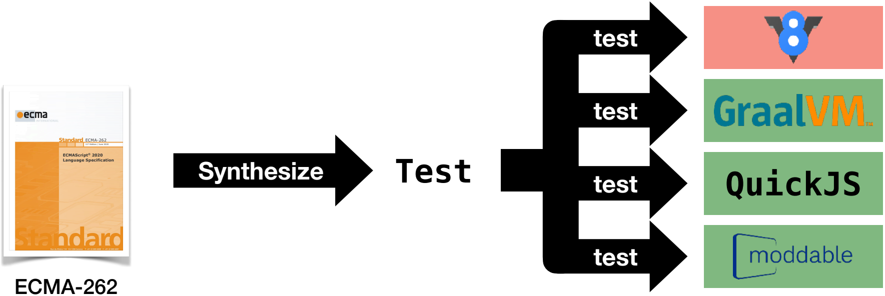
```
syntax def ArrayLiteral[2].Evaluation(
  this, ElementList, Elision
){
  let array = [! (ArrayCreate 0)]
  let nextIndex =
    [? (ElementList.ArrayAccumulation array 0)]
  if (! (= Elision absent))
    [? (Elision.ArrayAccumulation array nextIndex)]
  return array
}
```

IR_{ES} function for *ArrayLiteral* in ES13

[3] [ASE'21] **Jihyeok Park**, Seungmin An, Wonho Shin, Yusung Sim, and Sukyoung Ryu, "JSTAR: JavaScript Specification Type Analyzer using Refinement"

Application 3 – Program Synthesis

Another application is to **synthesize** programs from specifications. For example, we can synthesize JavaScript programs to detect real-world bugs in JavaScript engines.[4]



Synthesize

Test

test

test

test

test

ECMA-262
(JavaScript Spec.)

JavaScript Engines

[4] [PLDI'21] **Jihyeok Park**, Dongjun Youn, Kanguk Lee, and Sukyoung Ryu, "Feature-Sensitive Coverage for Conformance Testing of Programming Language Implementations"

Application 3 – Program Synthesis

For example, we found a bug in the SpiderMonkey JavaScript engine (v107.0b4) used in Firefox by synthesizing the following JavaScript program from the JavaScript language specification.[5]

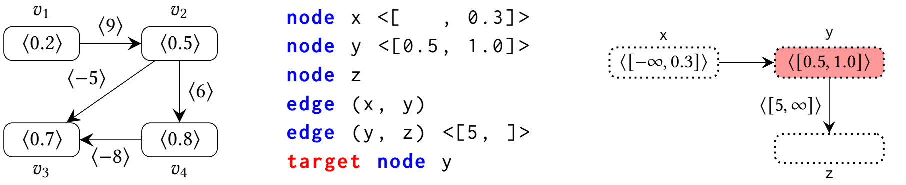
```
// [EXIT] normal
var x = (async function ([]) {})();
// Assertions
...
$assert.sameValue(Object.getPrototypeOf(globalThis["x"]), Promise.
    prototype);
$assert.sameValue(Object.isExtensible(globalThis["x"]), true);
$assert.notCallable(globalThis["x"]);
$assert.notConstructable(globalThis["x"]);
...
```

While it should be terminated normally, SpiderMonkey engine throws a run-time `TypeError` when executing the it.

[5] https://bugzilla.mozilla.org/show_bug.cgi?id=1799288

Application 4 – Explainable AI

We can also use PL foundations to provide a better **explainability** for AI systems. For example, we can design a graph description language to explain the graph learning model.[6]



```
node x <[   , 0.3]>
node y <[0.5, 1.0]>
node z
edge (x, y)
edge (y, z) <[5, ]>
target node y
```

(a) A featured graph G_2

(b) A GDL program P_4

(c) A graphical representation of P_4

Fig. 3. A running example of GDL

Using the above graph description language, we can automatically generate explanations for the classification results of the graph learning model.

[6] [PLDI'24] Minseok Jeon, **Jihyeok Park**, and Hakjoo Oh, "PL4XGL: A Programming Language Approach to Explainable Graph Learning"

Final Exam

- **Date:** 18:30 – 21:00 (150 min.), December 18 (Wed.).
- **Location:** 205, Woojung Hall of Informatics (우정정보관)
- **Coverage:** Lectures 14 – 26
- **Format:** closed book and closed notes
  - Fill-in-the-blank questions about the PL concepts.
  - Write the evaluation results of given expressions.
  - Draw derivation trees of given expressions.
  - Define the syntax or semantics of extended language features.
  - Define typing rules for the given language features.
  - etc.
- Note that there is **no class** on **December 16 (Mon.)**.

- I hope you enjoyed the class!

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