

Lecture 27 – Course Review

COSE212: Programming Languages

Jihyeok Park



2025 Fall

To learn **essential concepts of programming languages**

To learn **essential concepts** of **programming languages**

- Why?

To learn **essential concepts** of **programming languages**

- Why? After this course, you will be able to:

To learn **essential concepts** of **programming languages**

- Why? After this course, you will be able to:
 - **learn new programming languages** quickly.
 - **evaluate** and pick the best language for a given task.
 - **design** your own **specialized languages** for specific tasks.

To learn **essential concepts** of **programming languages**

- Why? After this course, you will be able to:
 - **learn new programming languages** quickly.
 - **evaluate** and pick the best language for a given task.
 - **design** your own **specialized languages** for specific tasks.
- How?

To learn **essential concepts** of **programming languages**

- Why? After this course, you will be able to:
 - **learn new programming languages** quickly.
 - **evaluate** and pick the best language for a given task.
 - **design** your own **specialized languages** for specific tasks.
- How? You will learn how to:
 - **design** programming languages in a **mathematical** way.
 - **implement** 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

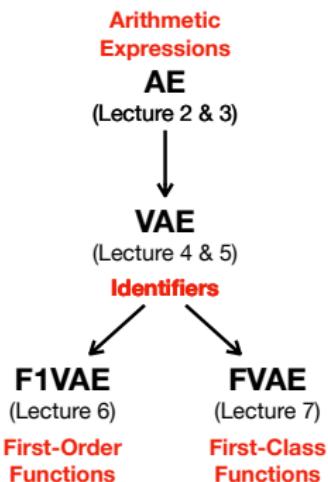
F1VAE
(Lecture 6)

First-Order
Functions

Summary

(Part 1)
Untyped Languages

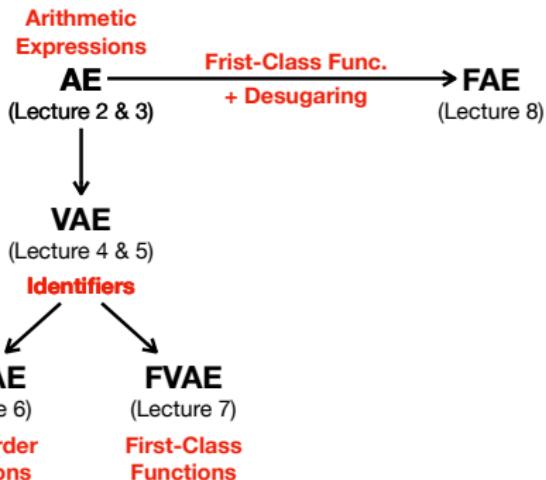
(Part 2)
Typed Languages



Summary

(Part 1)
Untyped Languages

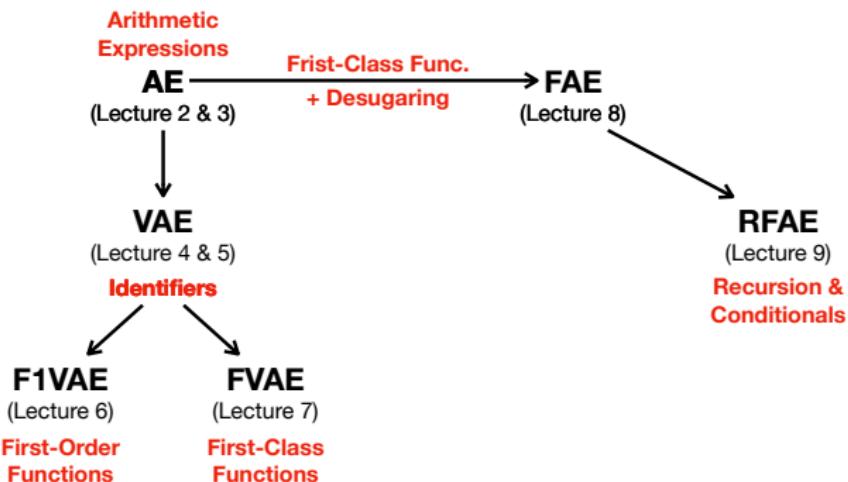
(Part 2)
Typed Languages



Summary

(Part 1) Untyped Languages

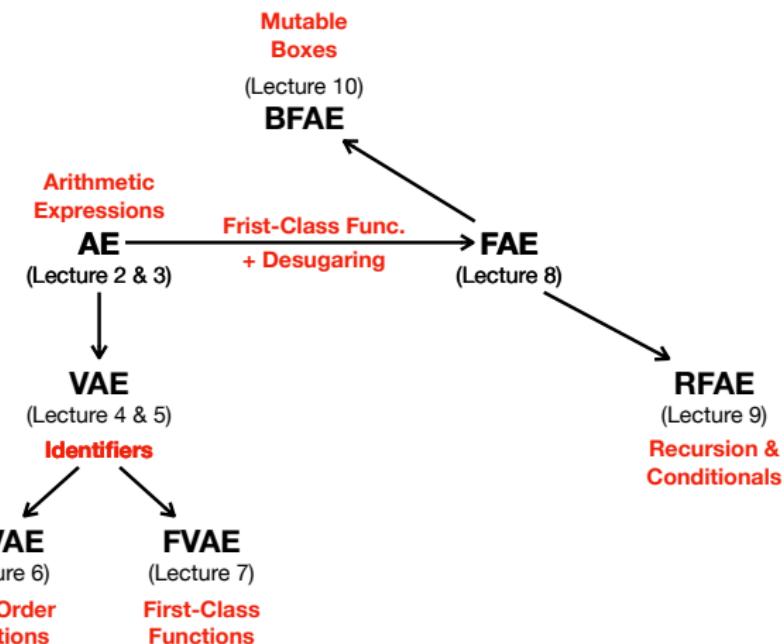
(Part 2) Typed Languages



Summary

(Part 1) Untyped Languages

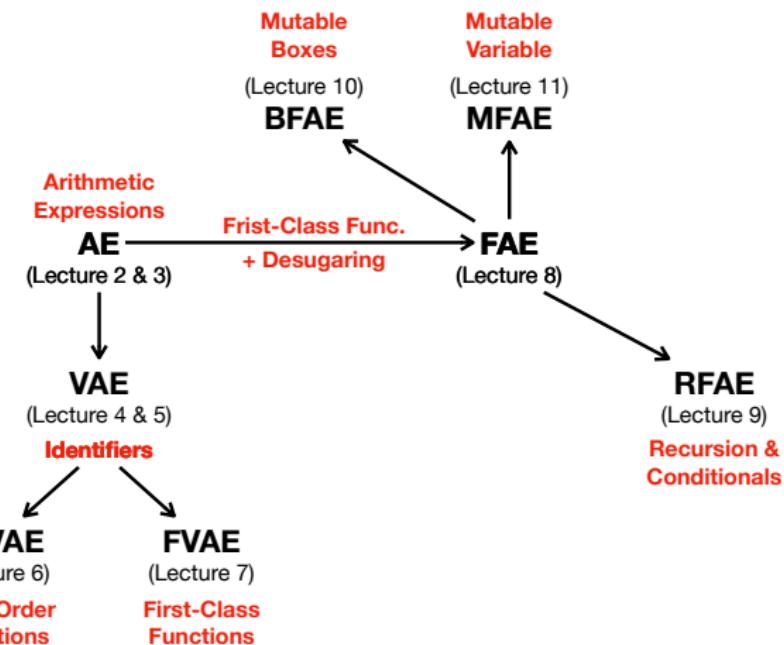
(Part 2) Typed Languages



Summary

(Part 1) Untyped Languages

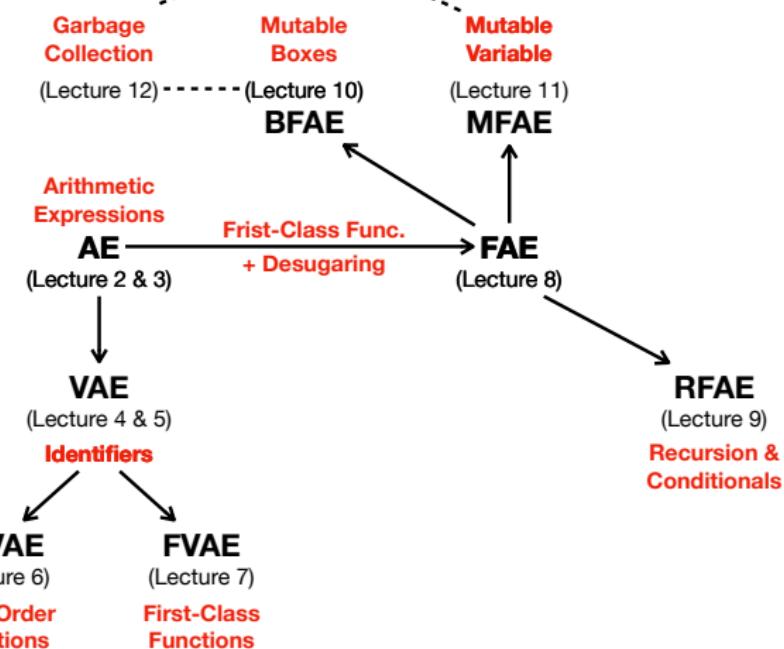
(Part 2) Typed Languages



Summary

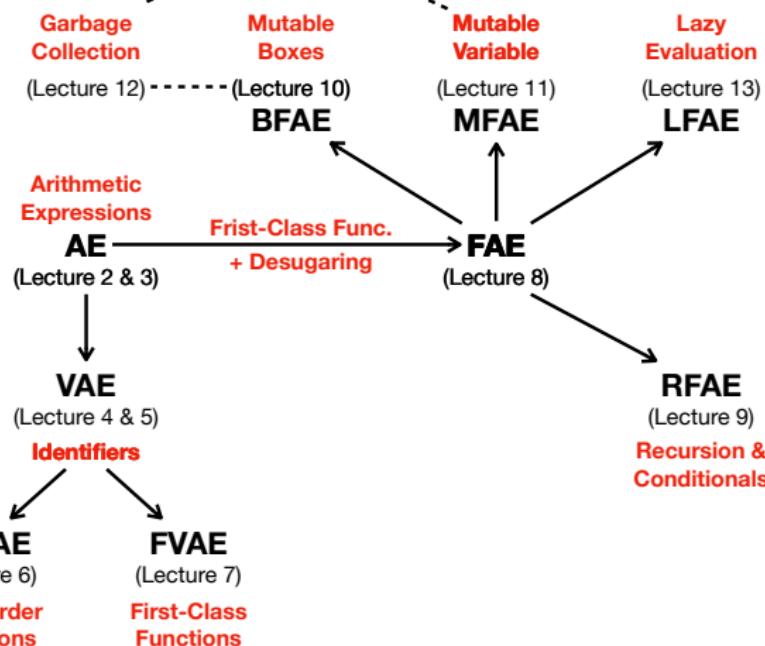
(Part 1) Untyped Languages

(Part 2) Typed Languages



Summary

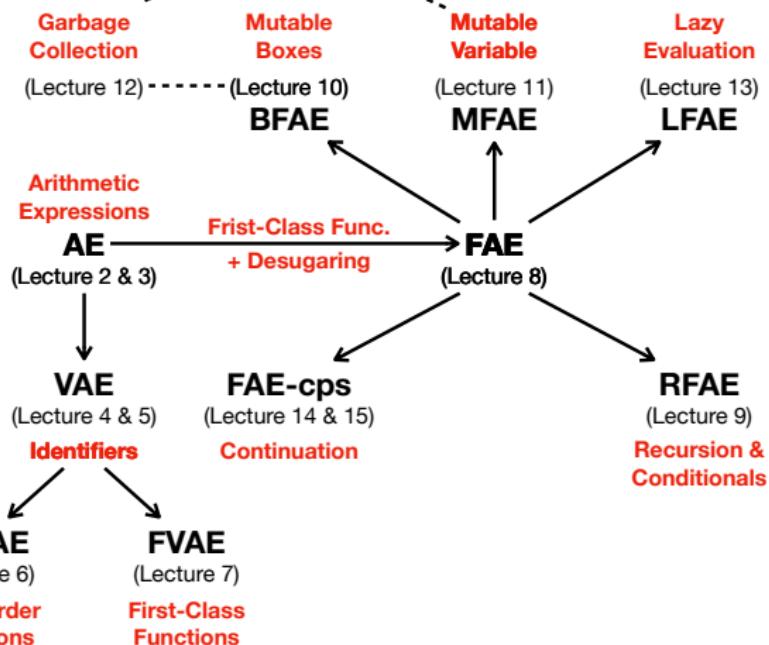
(Part 1) Untyped Languages



(Part 2) Typed Languages

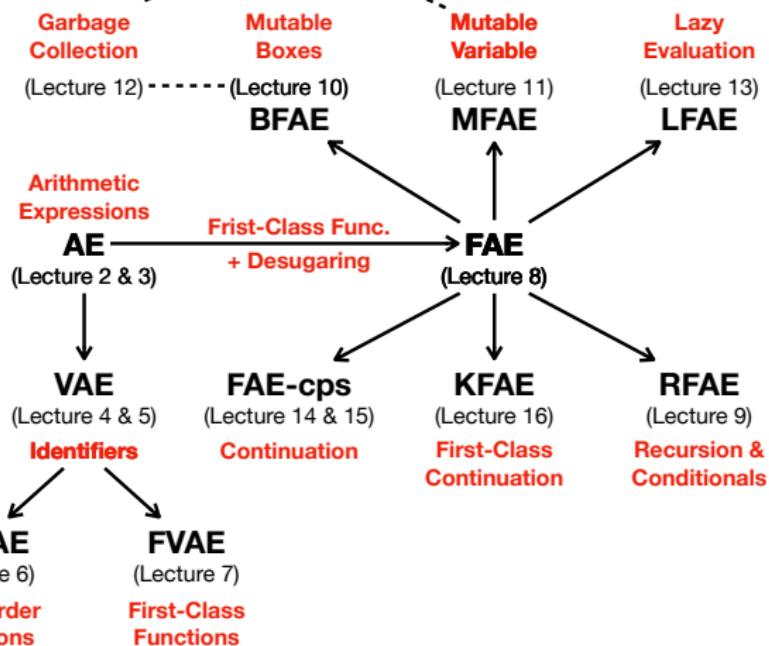
Summary

(Part 1) Untyped Languages



(Part 2) Typed Languages

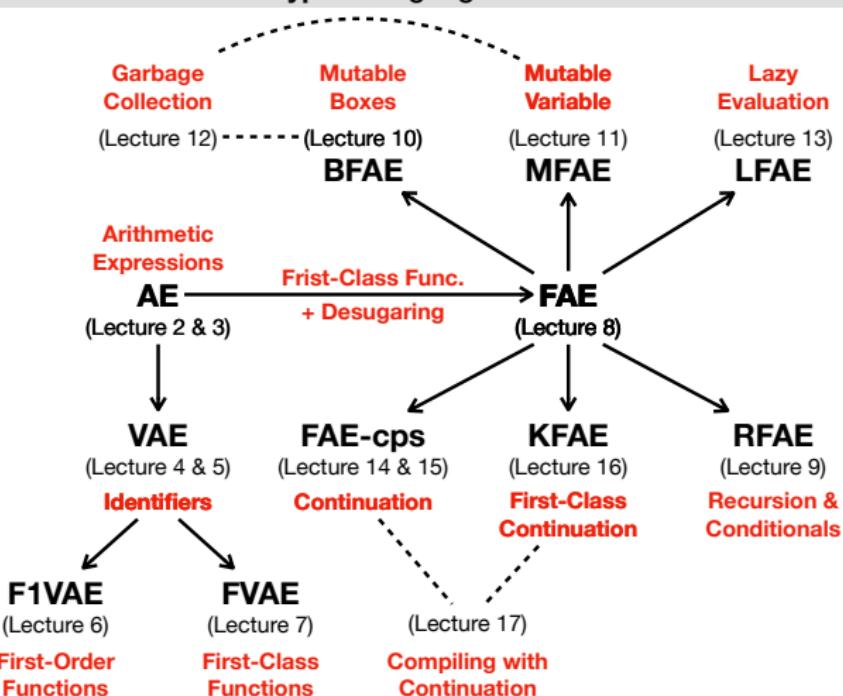
(Part 1)
Untyped Languages



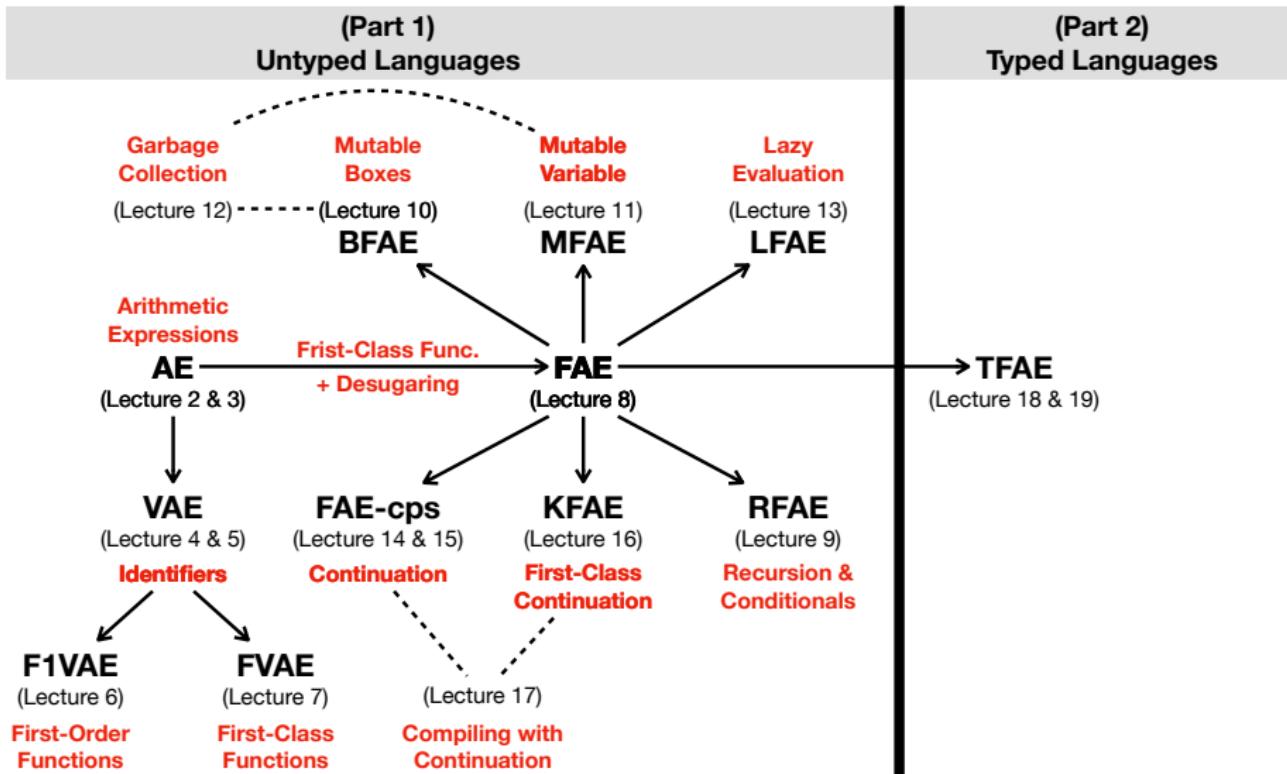
(Part 2)
Typed Languages

(Part 1)
Untyped Languages

(Part 2)
Typed Languages

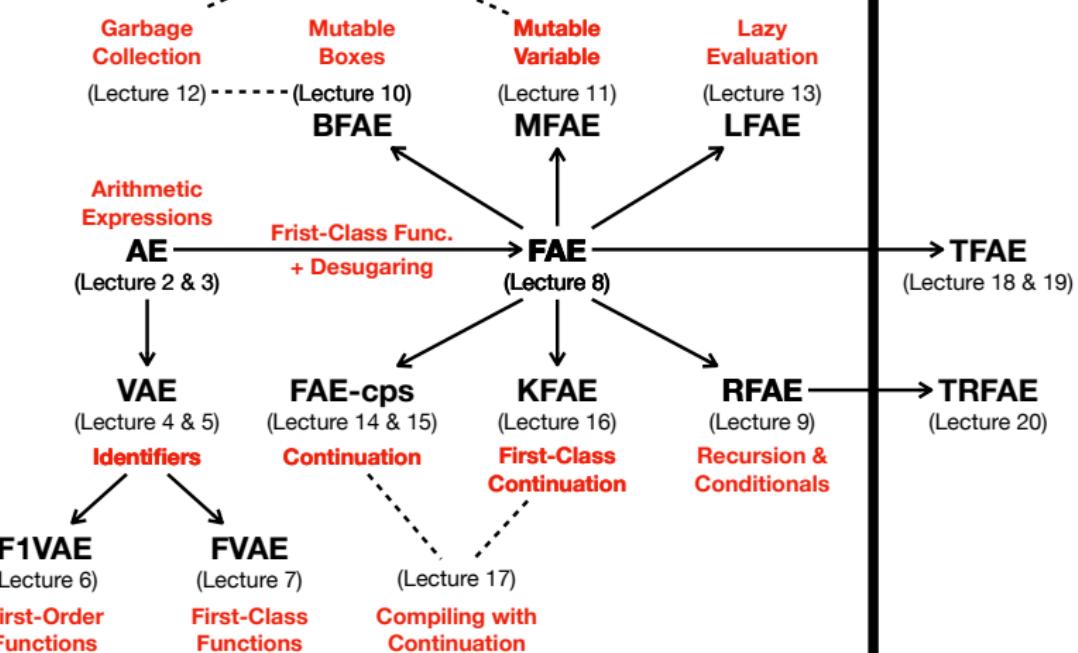


Summary



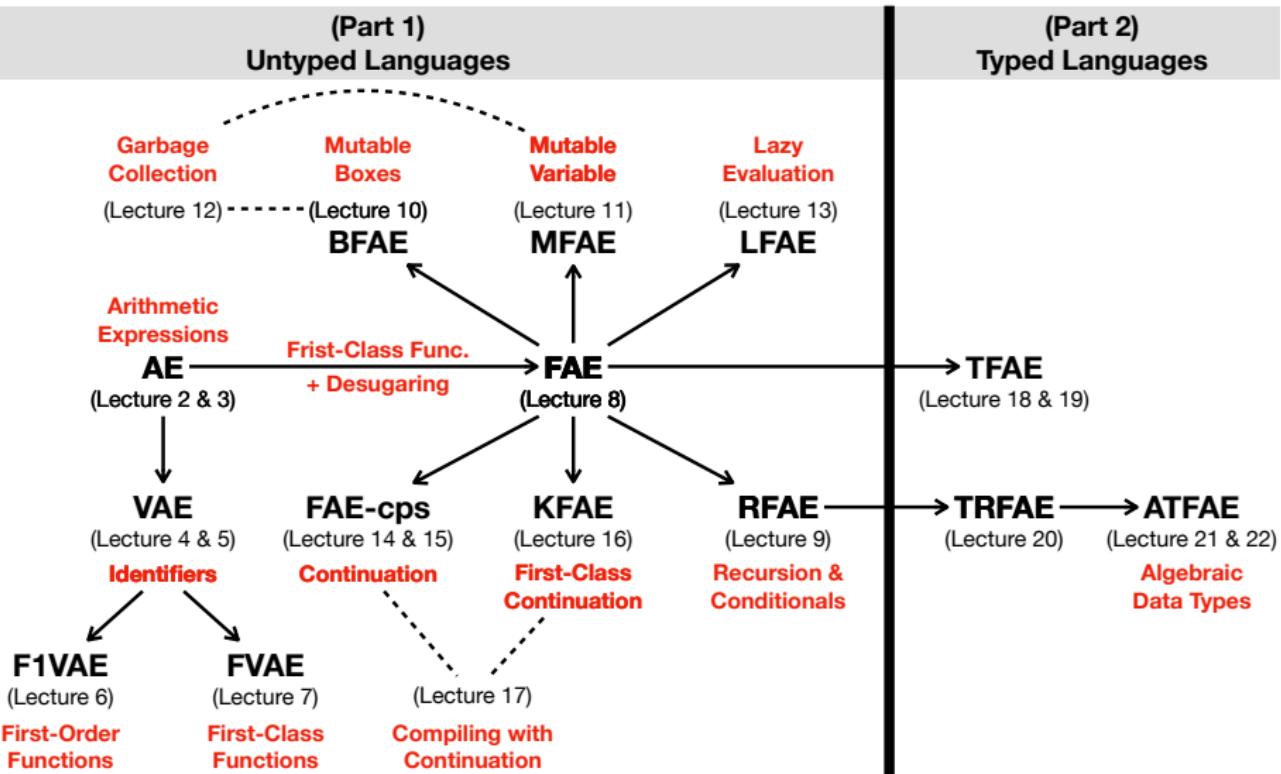
(Part 1)
Untyped Languages

(Part 2)
Typed Languages



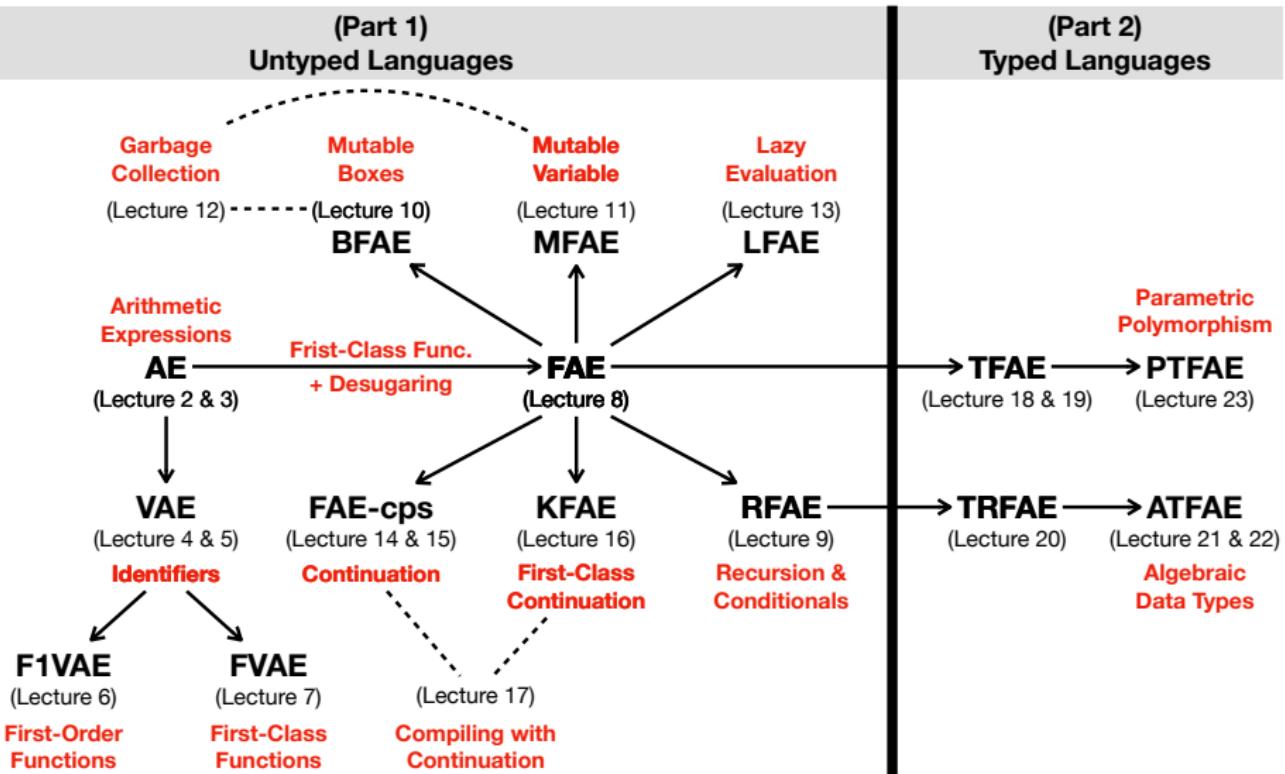
(Part 1)
Untyped Languages

(Part 2)
Typed Languages

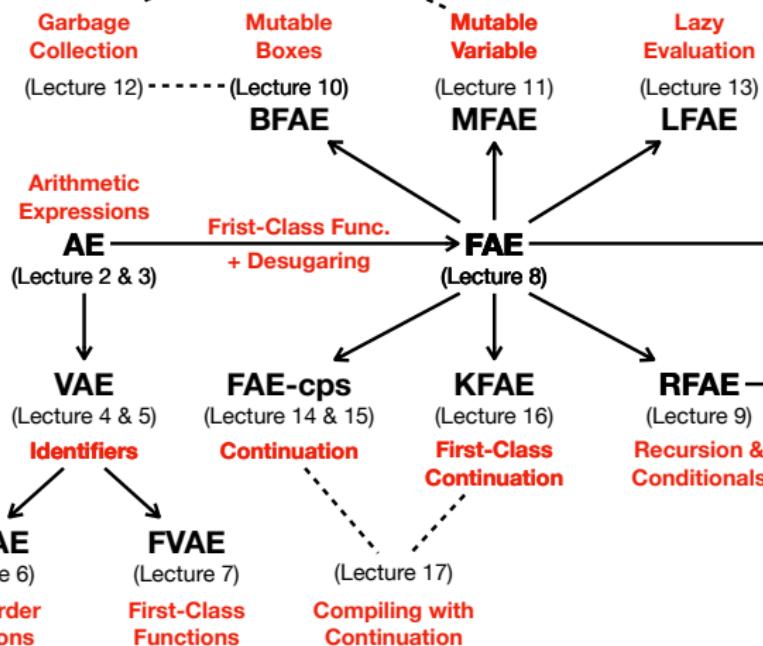


(Part 1)
Untyped Languages

(Part 2)
Typed Languages

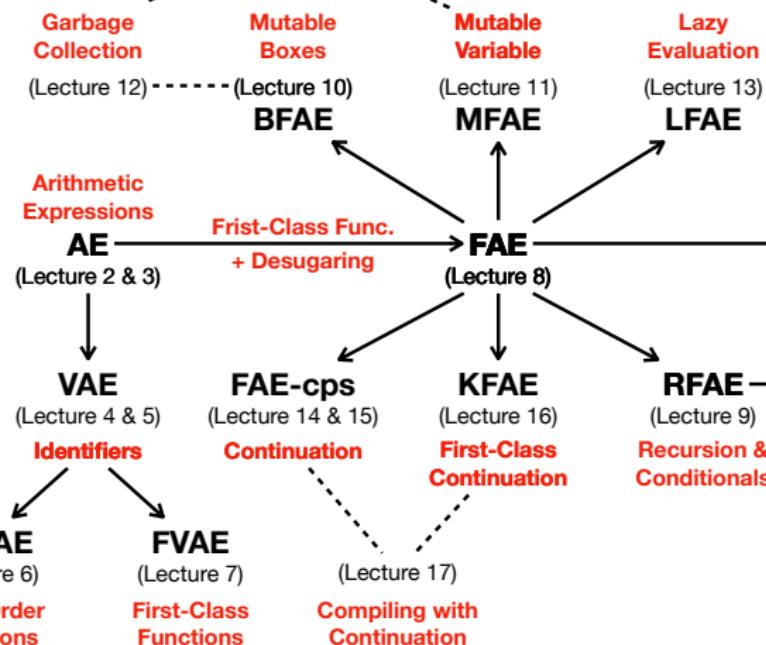


(Part 1)
Untyped Languages



(Part 2)
Typed Languages

(Part 1)
Untyped Languages



A deeper understanding of programming languages can help you in:

- Static/Dynamic Analysis
- Vulnerability Detection
- Automated Testing
- Program Synthesis
- Explainable AI
- etc.

A deeper understanding of programming languages can help you in:

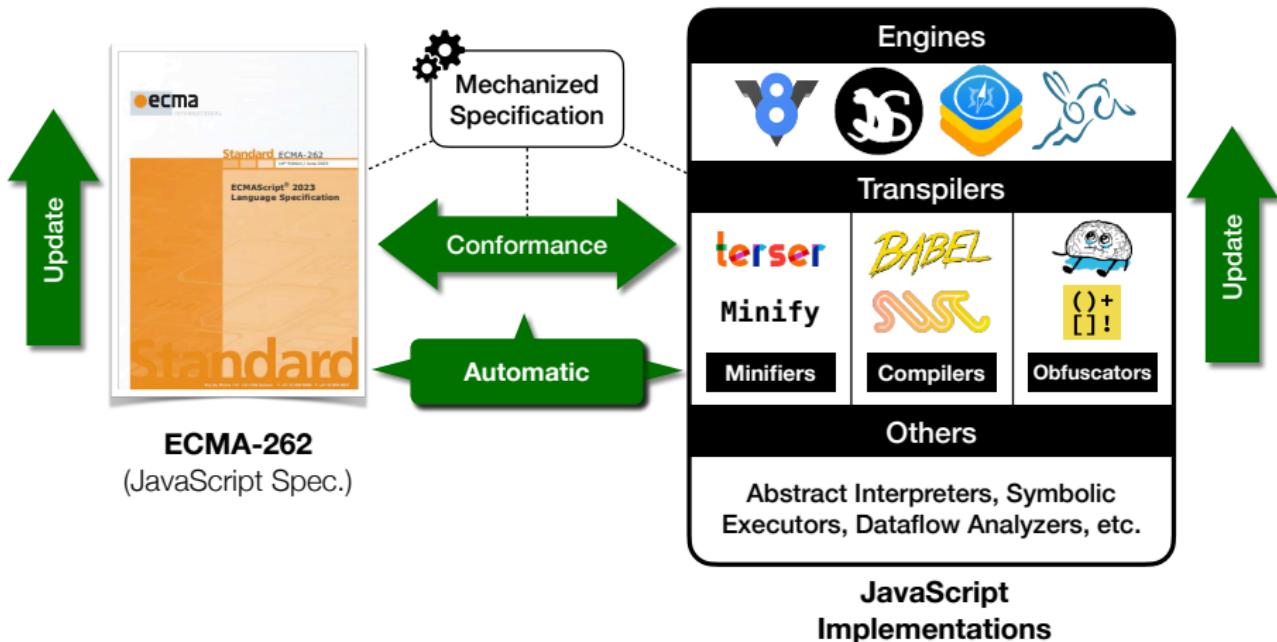
- Static/Dynamic Analysis
- Vulnerability Detection
- Automated Testing
- Program Synthesis
- Explainable AI
- etc.

In the rest of this lecture, I will introduce some of the applications developed in our **programming languages research group (PLRG)**:

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

Application 1 – Mechanized Specification

ECMA-262 is the official specification of JavaScript written in English.¹
ESMeta is a **mechanized** version of ECMA-262 developed by our lab.²

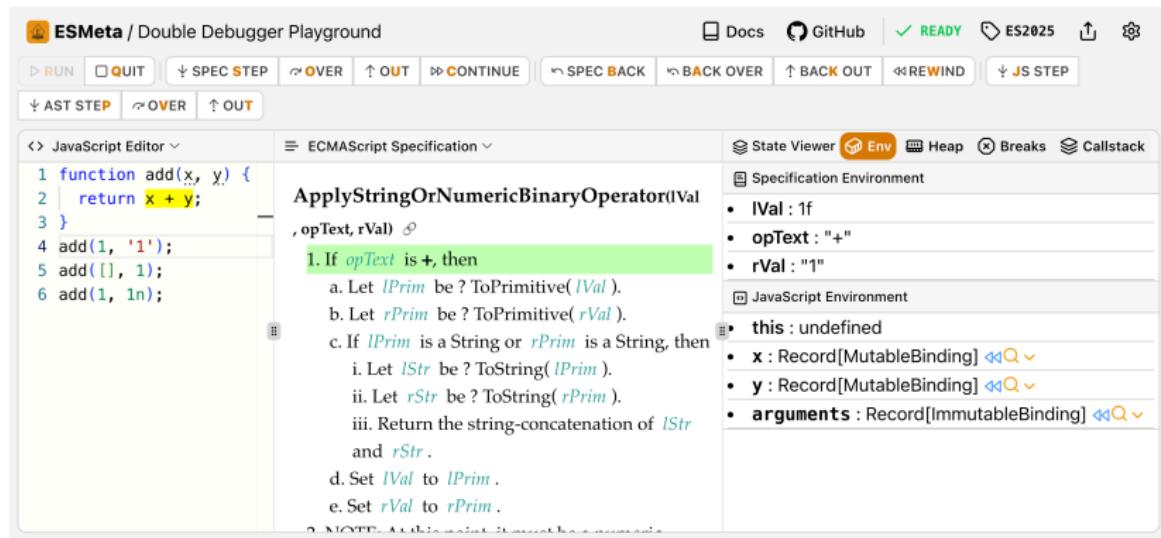


¹<https://tc39.es/ecma262/2025/multipage/>

²<https://github.com/es-meta/esmeta>

Application 2 – Double Debugger

We can **visualize** the execution of JavaScript programs based on the mechanized specification to help **understand/debug** complex features.³



The screenshot shows the ESMeta Double Debugger Playground interface. On the left, a code editor displays the following JavaScript code:

```
1 function add(x, y) {  
2 |   return x + y;  
3 }  
4 add(1, '1');  
5 add([], 1);  
6 add(1, 1n);
```

In the middle pane, the ECMAScript Specification is being analyzed for the expression `x + y`. A green box highlights the rule `ApplyStringOrNumericBinaryOperator(lVal, opText, rVal)`. Below it, a list of steps for the `+` operator is shown:

- Let `lPrim` be `ToPrimitive(lVal)`.
- Let `rPrim` be `ToPrimitive(rVal)`.
- If `lPrim` is a String or `rPrim` is a String, then
 - Let `lStr` be `ToString(lPrim)`.
 - Let `rStr` be `ToString(rPrim)`.
 - Return the string-concatenation of `lStr` and `rStr`.
- Set `lVal` to `lPrim`.
- Set `rVal` to `rPrim`.

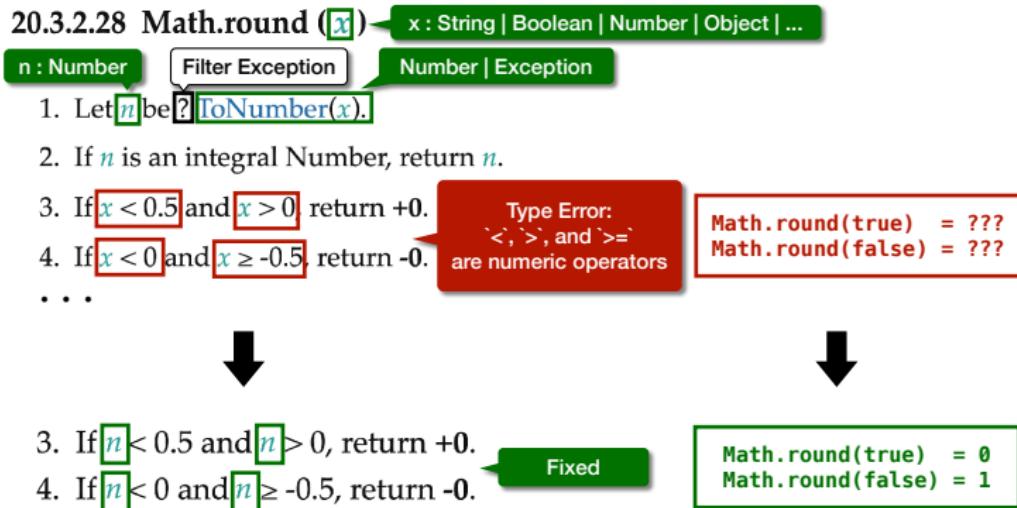
The right pane shows the State Viewer with sections for the Specification Environment and the JavaScript Environment. The Specification Environment contains objects `lVal`, `opText`, and `rVal`. The JavaScript Environment contains objects `this`, `x`, `y`, and `arguments`.

[FSE'25 Demo] M. Choe*, K. Song*, H. Kim, and J. Park, “JSSpecVis: A JavaScript Language Specification Visualization Tool”

³<https://es-meta.github.io/playground/>

Application 3 – Type Analysis

We can detect **specification errors** using **type analysis** based on the mechanized specification. It is officially used in language design process.⁴



[ASE'21] J. Park, S. An, W. Shin, Y. Sim, and S. Ryu, “JSTAR: JavaScript Specification Type Analyzer using Refinement”

⁴<https://github.com/tc39/ecma262/actions/workflows/esmeta-typecheck.yml>

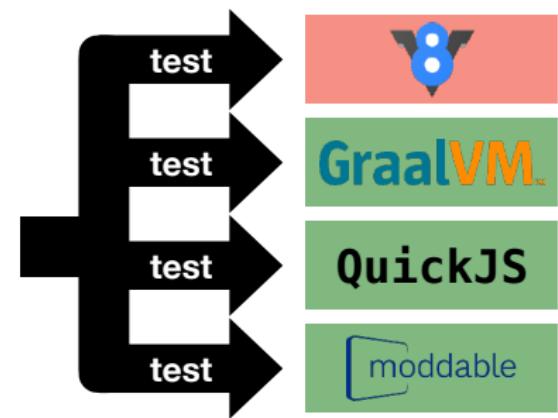
Application 4 – Program Synthesis

We can **synthesize JS programs** with assertions from the mechanized specification to **test real-world JS interpreters** (e.g., V8 for Chrome).



ECMA-262
(JavaScript Spec.)

Synthesize → Test



[PLDI'23] J. Park, D. Youn, K. Lee, and S. Ryu, “Feature-Sensitive Coverage for Conformance Testing of Programming Language Implementations”

Application 4 – Program Synthesis



For example, we found a bug in the SpiderMonkey JavaScript engine (v107.0b4) used in Firefox.⁵

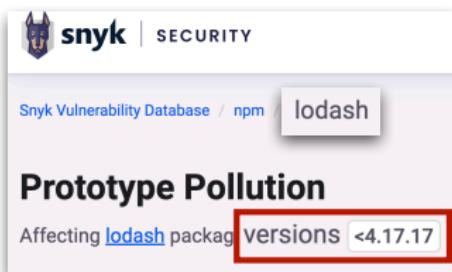
```
var x = (async function () {})();  
// Assertions  
...  
$assert.sameValue(Object.getPrototypeOf(x), Promise.prototype);  
$assert.sameValue(Object.isExtensible(x), true);  
$assert.notCallable(x);  
$assert.notConstructable(x);  
...
```

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

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

Application 5 – Vulnerability Detection

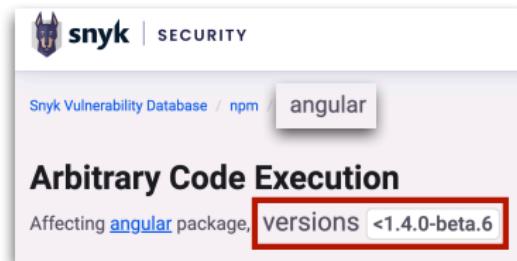
Detecting vulnerable JS libraries on the web is important for web security.



Snyk Vulnerability Database / npm / lodash

Prototype Pollution

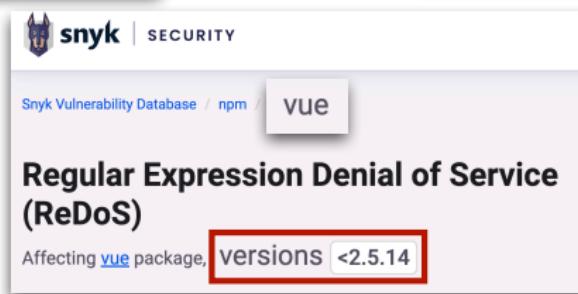
Affecting [lodash](#) package, versions <4.17.17



Snyk Vulnerability Database / npm / angular

Arbitrary Code Execution

Affecting [angular](#) package, versions <1.4.0-beta.6



Snyk Vulnerability Database / npm / vue

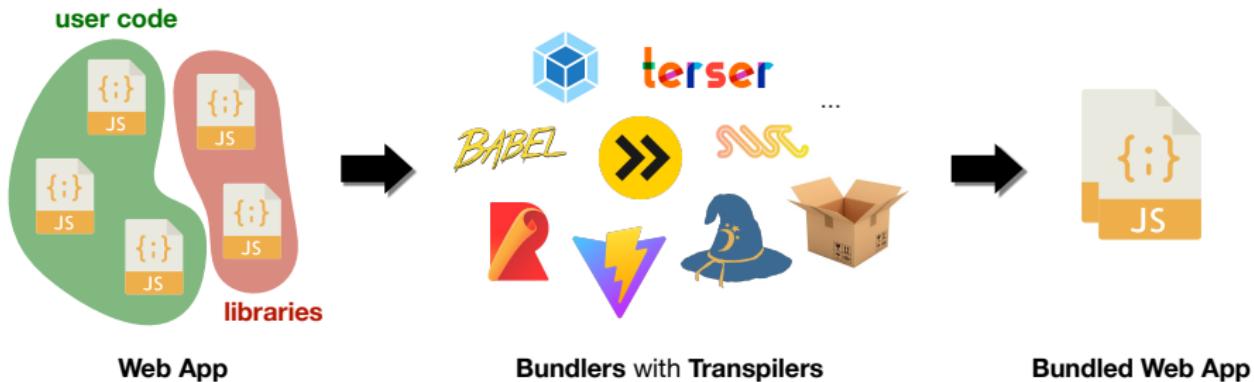
Regular Expression Denial of Service (ReDoS)

Affecting [vue](#) package, versions <2.5.14

Application 5 – Vulnerability Detection

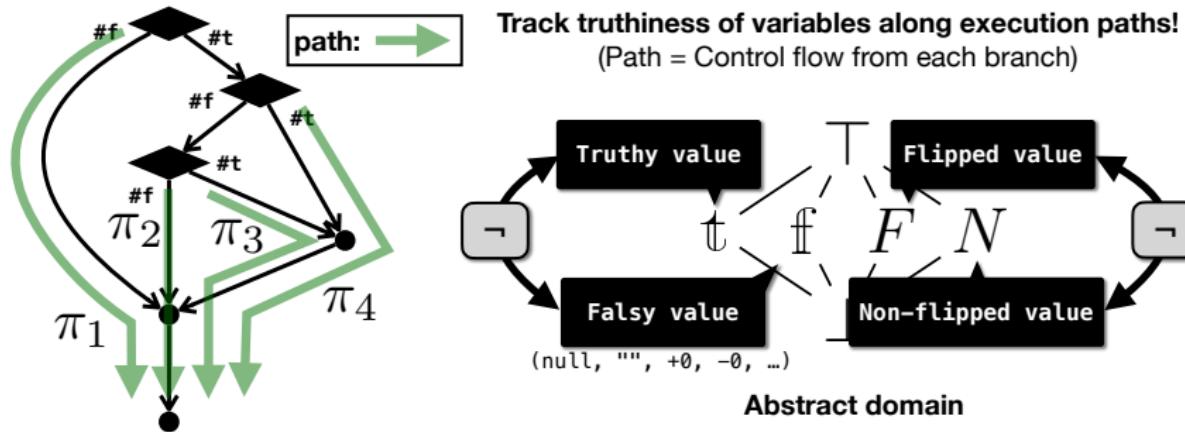
However, it is challenging to detect JS libraries accurately because:

- Bundlers **modify** and **compress** **user code** and **libraries** together using diverse **transpilers**
- It makes **difficult to detect libraries** in transpiled code in web applications



Application 5 – Vulnerability Detection

We can accurately detect vulnerable JS libraries on web using property-order graphs extracted through **static analysis** techniques.



We are recruiting motivated students who are interested in this topic!

[ASE'25] S. Kim*, S. Park*, and J. Park, “Debun: Detecting Bundled JavaScript Libraries on Web using Property-Order Graphs”

- **Date:** 18:30 – 21:00 (150 min.), December 17 (Wed.).
- **Location:** B102, IT & General Education Center (정운오IT교양관)
- **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 15 (Mon.)**.

- I hope you enjoyed the class!

Jihyeok Park

jihyeok_park@korea.ac.kr

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