

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**.

(Part 1)
Untyped Languages

(Part 2)
Typed Languages

**Arithmetic
Expressions**

AE

(Lecture 2 & 3)

(Part 1)
Untyped Languages

(Part 2)
Typed Languages

**Arithmetic
Expressions**

AE
(Lecture 2 & 3)

↓

VAE
(Lecture 4 & 5)

Identifiers

(Part 1)
Untyped Languages

(Part 2)
Typed Languages

**Arithmetic
Expressions**

AE

(Lecture 2 & 3)



VAE

(Lecture 4 & 5)

Identifiers



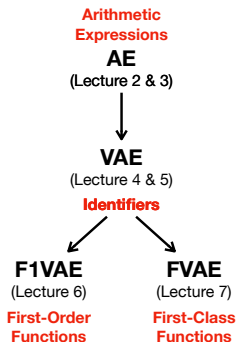
F1VAE

(Lecture 6)

**First-Order
Functions**

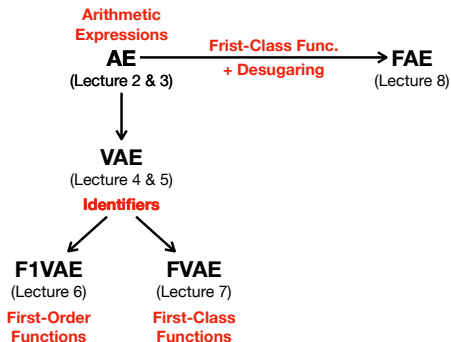
(Part 1)
Untyped Languages

(Part 2)
Typed Languages



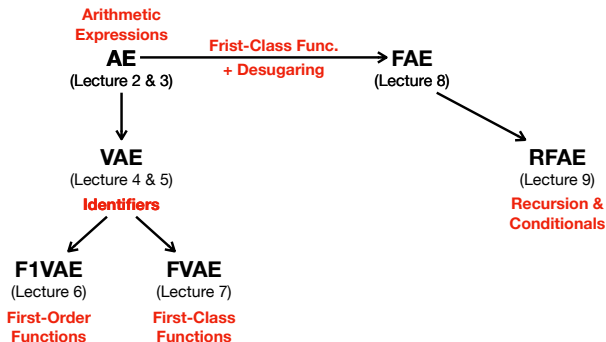
(Part 1) Untyped Languages

(Part 2) Typed Languages



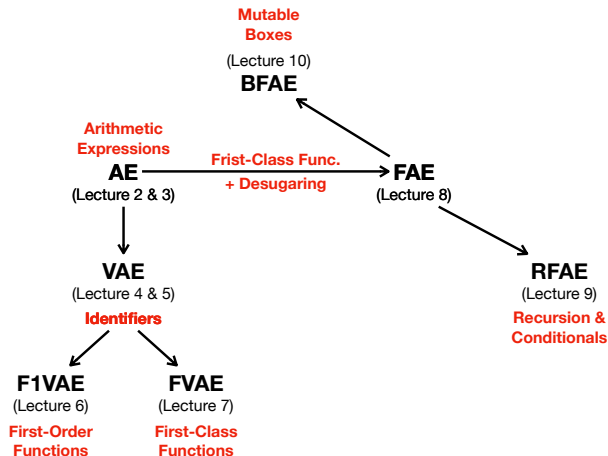
(Part 1)
Untyped Languages

(Part 2)
Typed Languages



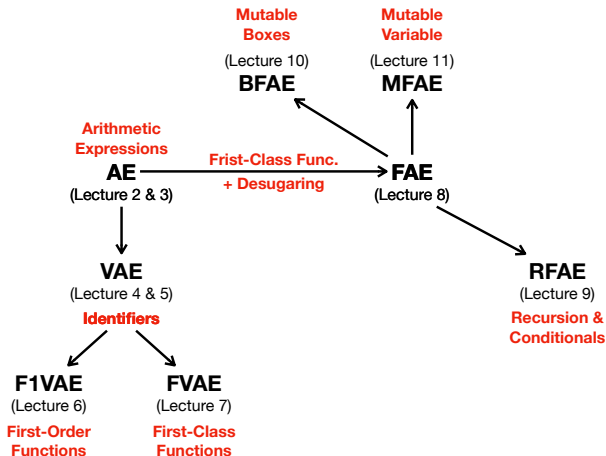
(Part 1)
Untyped Languages

(Part 2)
Typed Languages



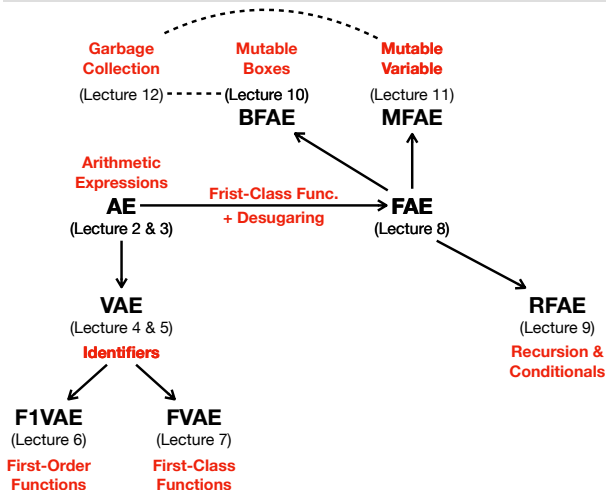
(Part 1) Untyped Languages

(Part 2) Typed Languages



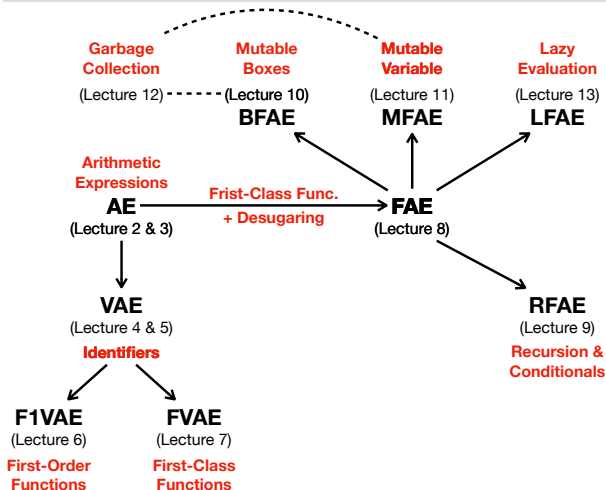
(Part 1) Untyped Languages

(Part 2) Typed Languages



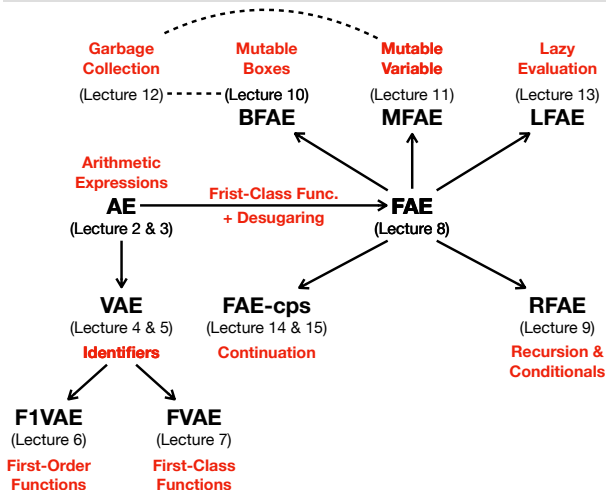
(Part 1) Untyped Languages

(Part 2) Typed Languages



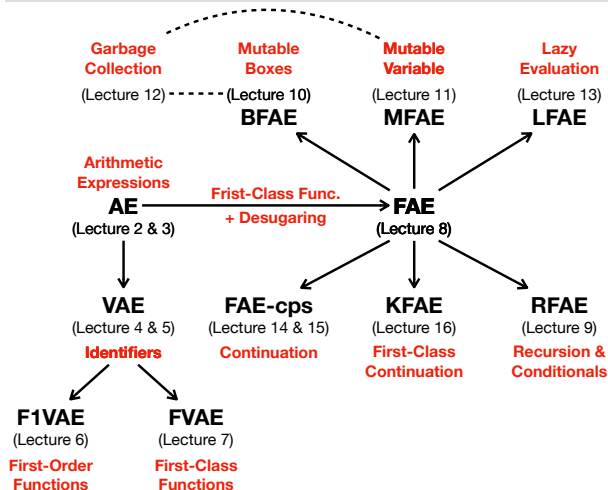
(Part 1) Untyped Languages

(Part 2) Typed Languages



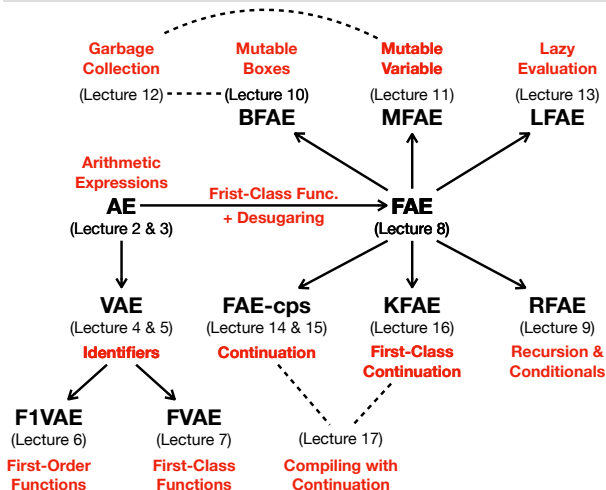
(Part 1) Untyped Languages

(Part 2) Typed Languages



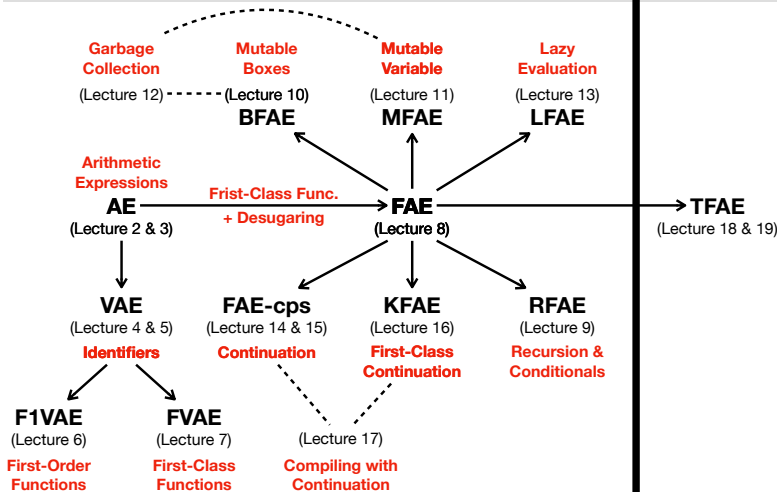
(Part 1) Untyped Languages

(Part 2) Typed Languages



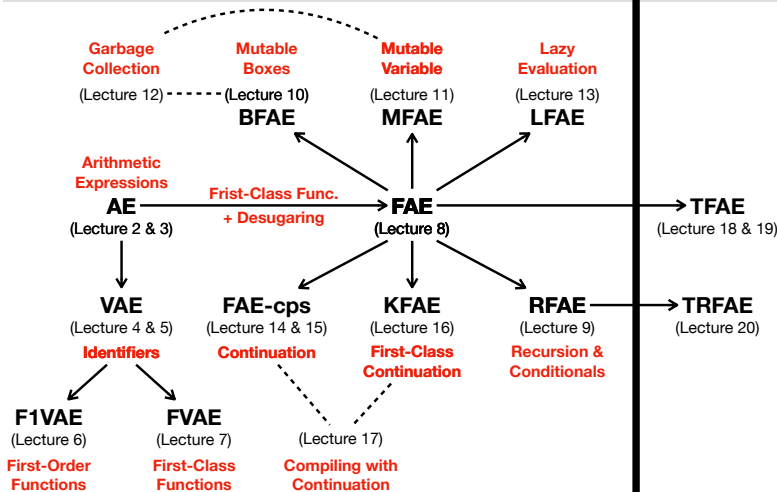
(Part 1) Untyped Languages

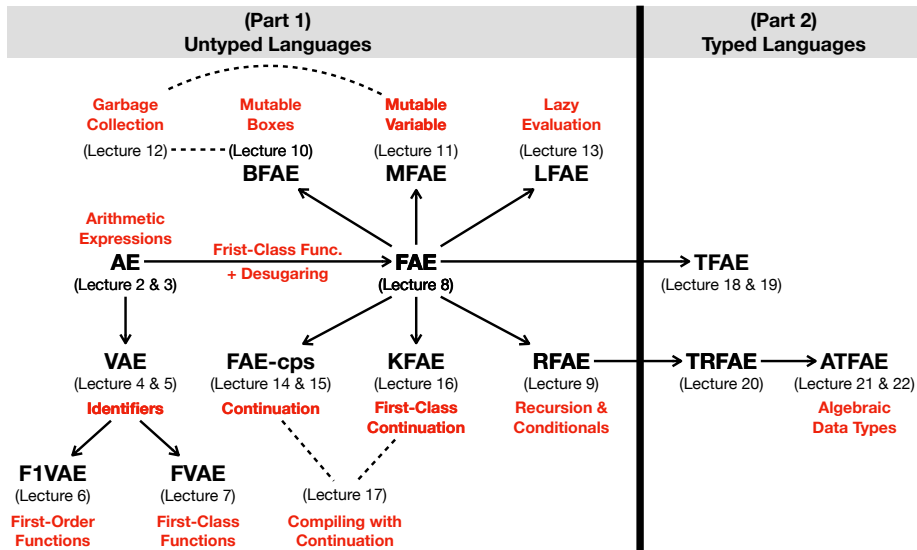
(Part 2) Typed Languages

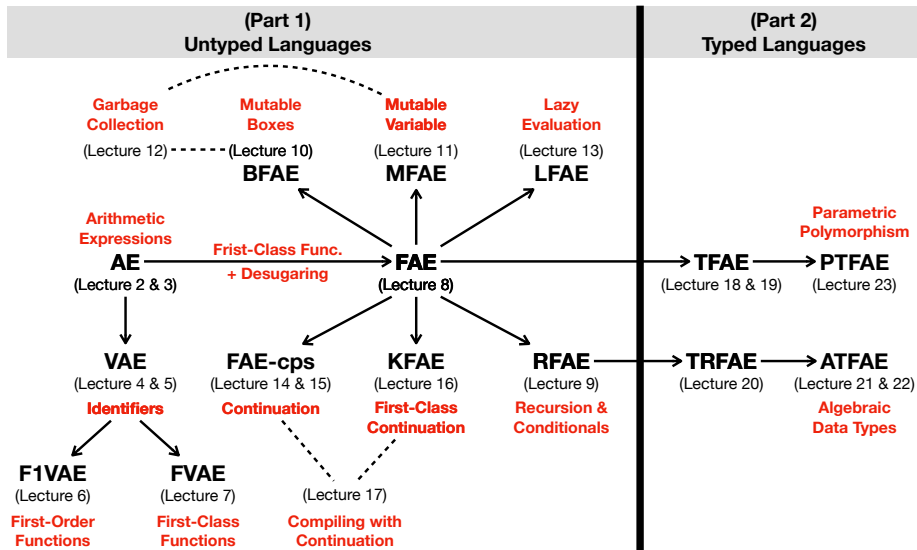


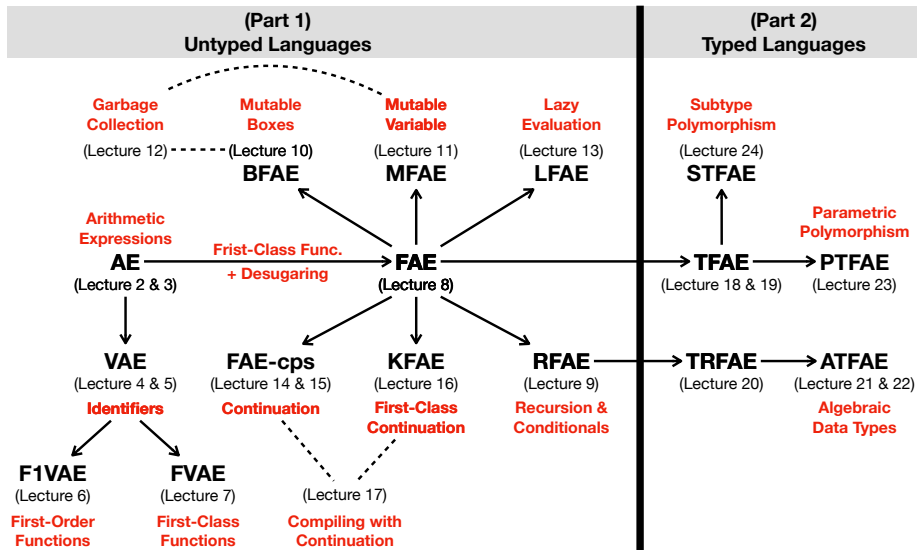
(Part 1) Untyped Languages

(Part 2) Typed Languages



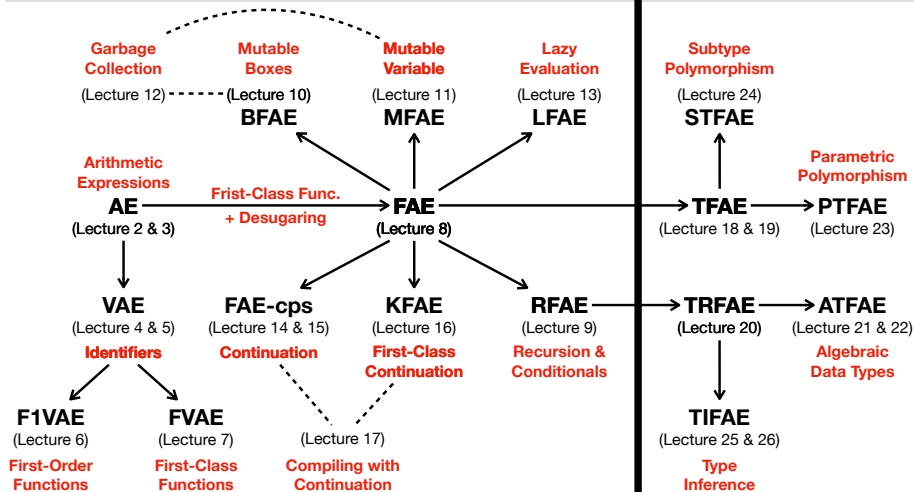






(Part 1) Untyped Languages

(Part 2) Typed 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:

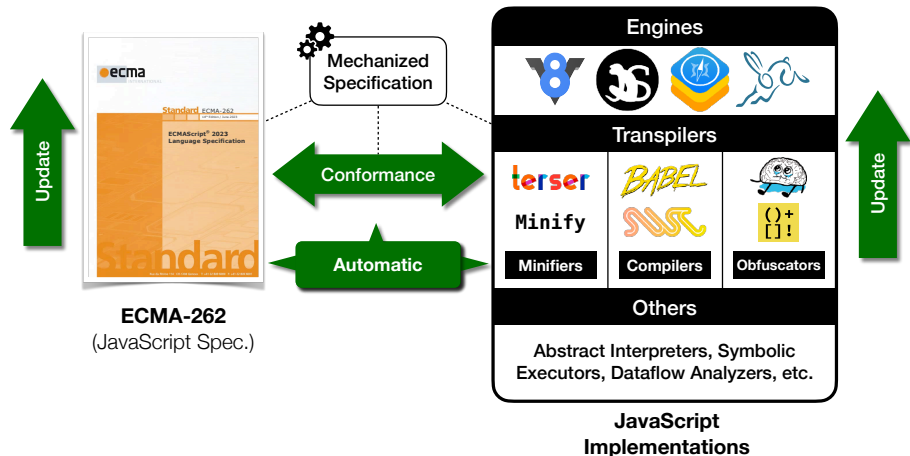
- Static/Dynamic Analysis
- Vulnerability Detection
- Automated Testing
- Program Synthesis
- Explainable AI
- etc.

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

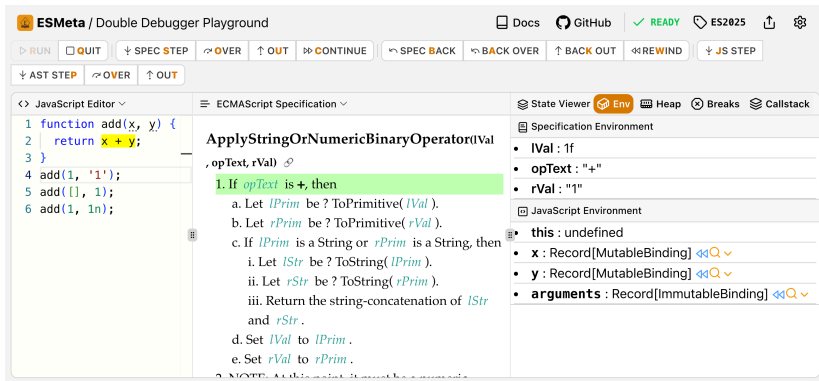


¹<https://tc39.es/ecma262/2025/multipage/>

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



ESMeta / Double Debugger Playground

Docs GitHub READY ES2025

RUN QUIT SPEC STEP OVER OUT CONTINUE SPEC BACK BACK OVER BACK OUT REWIND JS STEP

AST STEP OVER OUT

JavaScript Editor

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

ECMAScript Specification

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(*rPrim*).
 - iii. Return the string-concatenation of *lStr* and *rStr*.
 - d. Set *lVal* to *lPrim*.
 - e. Set *rVal* to *rPrim*.

NOTE: ApplyStringOrNumericBinaryOperator is a numeric operator.

State Viewer Env Heap Breaks Callstack

Specification Environment

- *lVal* : 1f
- *opText* : "+"
- *rVal* : "1"

JavaScript Environment

- *this* : undefined
- *x* : Record[MutableBinding] ⏏ ⏏ ⏏
- *y* : Record[MutableBinding] ⏏ ⏏ ⏏
- *arguments* : Record[ImmutableBinding] ⏏ ⏏ ⏏

[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/>

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

20.3.2.28 Math.round (\tilde{x})

$x : \text{String} \mid \text{Boolean} \mid \text{Number} \mid \text{Object} \mid \dots$

$n : \text{Number}$

Filter Exception

Number | Exception

1. Let \tilde{n} be $\text{ToNumber}(x)$.
2. If \tilde{n} is an integral Number, return \tilde{n} .
3. If $\tilde{x} < 0.5$ and $\tilde{x} > 0$, return $+0$.
4. If $\tilde{x} < 0$ and $\tilde{x} \geq -0.5$, return -0 .
- ...

Type Error:
'<', '>', and '>='
are numeric operators

$\text{Math.round}(\text{true}) = ???$
 $\text{Math.round}(\text{false}) = ???$



3. If $\tilde{n} < 0.5$ and $\tilde{n} > 0$, return $+0$.
4. If $\tilde{n} < 0$ and $\tilde{n} \geq -0.5$, return -0 .

Fixed

$\text{Math.round}(\text{true}) = 0$
 $\text{Math.round}(\text{false}) = 1$

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

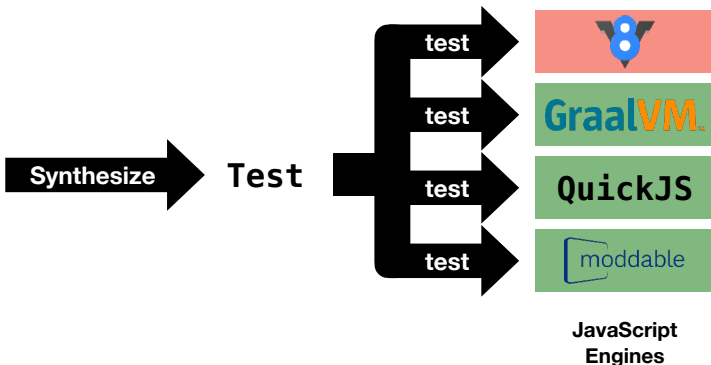
⁴<https://github.com/tc39/ecma262/actions/workflows/esmeta-typecheck.yml>

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.)



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

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

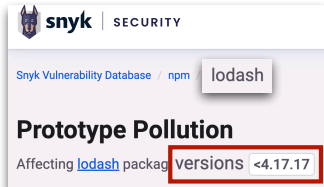
```
var x = (async function ([]) {} )();  
// Assertions  
...  
$assert.sameValue(Object.getPrototypeOf(x), Promise.prototype);  
$assert.sameValue(Object.isExtensible(x), true);  
$assert.notCallable(x);  
$assert.notConstructable(x);  
...
```

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.

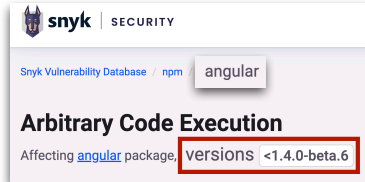


snyk | SECURITY

Snyk Vulnerability Database / npm / **lodash**

Prototype Pollution

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

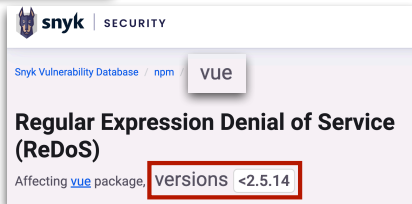


snyk | SECURITY

Snyk Vulnerability Database / npm / **angular**

Arbitrary Code Execution

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



snyk | SECURITY

Snyk Vulnerability Database / npm / **vue**

Regular Expression Denial of Service (ReDoS)

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

Application 5 – Vulnerability Detection

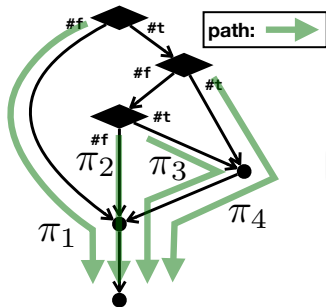
However, it is challenging to detect JS libraries accurately because:

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

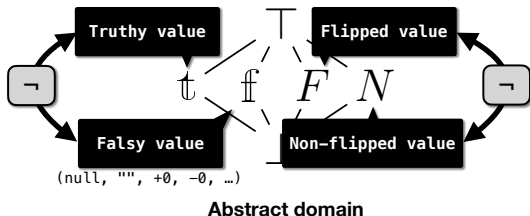


Application 5 – Vulnerability Detection

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



Track truthiness of variables along execution paths!
(Path = Control flow from each branch)



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>