CS315 Computer Security Project Progress Report

FlowSTLC: An Information Flow Control Type System Based On Graded Modality

Zhige Chen, Junqi Huang, Zhiyuan Cao

Department of Computer Science and Engineering, SUSTech

November 18, 2025

Project Overview

- **Goal**: design and implement a simply-typed lambda calculus with graded modality to enforce secure information flow (noninterference property).
- **Core ideas**: use a graded necessity $\Box_\ell T$ on a security semiring $\{\mathtt{Pub} \sqsubseteq \mathtt{Sec}\}$ to track the security requirement of information.

The project (implementation, report draft, and Lean 4 formalization) is available at https://github.com/nofe1248/cs315-term-project/.

Current Progress

Completed:

- Lexer and parser are implemented using ANTLR4.
- AST generation completed and integrated with parse trees.
- Complete specification of the syntax, typing rules, and small-step operational semantics of our type system.

Ongoing:

- Type checker implementation.
- AST-based interpreter.
- Lean 4 formalization of the type system and relevant proofs: about halfway finished.

Next Steps

- Complete type checker and interpreter implementation (2-3 weeks).
- Complete Lean 4 formalization (3-4 weeks, proceed simultaneously with the language implementation).
- Prepare a demo and evaluation: write example programs, finish the project report, etc. (1 week).

Small Demo for Current Implementation

Figure 1: AST generation demo

```
1   fun fib : Int^Public -> Int
2   fun fib n =
3       if n <= 0 then 0
4       else (
5           if n == 1 or n == 2 then 1
6           else (fib (n - 1)) + (fib (n - 2))
7       )
8
9   fun main : Unit^Public -> Int
10  fun main = fib 10
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Projects\cs315-term-project\flowstlc-compiler\target> java -jar .\flowstlc-compiler-1.0-SNAPSHOT.jar --source ..\test\fibonacci.
                BinaryExpr op=ADD
                  left:
                    FunctionCallExpr name=fib
                      arguments:
                        [0]:
                            BinaryExpr op=SUB
                              left:
                                Identifier name=n
                              right:
                                IntLiteral value=1
                  right:
                    FunctionCallExpr name=fib
                      arguments:
                        [0]:
                            BinaryExpr op=SUB
                              left:
                                Identifier name=n
                              right:
                                IntLiteral value=2
    [1]:
        FunctionDeclaration name=main
```

Small Demo for Current Implementation

The source code:

```
1  fun fib : Int^Public -> Int
2  fun fib n =
3      if n <= 0 then 0
4      else (
5          if n == 1 or n == 2 then 1
6          else (fib (n - 1)) + (fib (n - 2))
7      )
8
9  fun main : Unit^Public -> Int
10 fun main = fib 10
```

Small Demo for Current Implementation

```
Program
  declarations:
    [0]:
      FunctionDeclaration name=fib
        parameters: n
        type:
          FunctionType
            from:
              BuiltinType kind=INT
            level: PUBLIC
            to:
              BuiltinType kind=INT
        body:
          IfExpr
          ...
```

Thanks!