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

1st Zhige Chen
Dept. of Computer Science and Engineering
Southern University of Science and Technology
Shenzhen, China
12413315@mail.sustech.edu.cn

2nd Junqi Huang
Dept. of Computer Science and Engineering
Southern University of Science and Technology
Shenzhen, China
12212226@mail.sustech.edu.cn

3rd Zhiyuan Cao
Dept. of Computer Science and Engineering
Southern University of Science and Technology
Shenzhen, China
12311109@mail.sustech.edu.cn

***Abstract*—In this project, we introduce the design and implementation of a simple functional programming language with a type system that enforces secure information flow. Building on the simply-typed lambda calculus (STLC), we extend the type system with a graded modality with a security semiring. Inspired by *Graded Modal Dependent Type Theory* [1], our system statically prevents unauthorized flows of sensitive data into public outputs, ensuring the noninterference property [2].**

***Index Terms*—Computer security, information flow, noninterference, security-type systems**

I. INTRODUCTION

Modern software often handles sensitive data that must remain confidential. The *informational flow control* (or IFC) aims to ensure that high-security inputs of the program do not improperly influence low-security outputs. For example, private user information like passwords should never be leaked to public logs. Formally, the security policy of *noninterference* requires that variations in high-security inputs have no observable effect on low-security outputs. Intuitively, this means that changing a secret value should never cause any change in what an outside observer at low level sees. Violating the IFC principle can lead to information leak, so enforcing noninterference is critical for system security.

Type systems [3] have been proved effective at enforcing noninterference: they associate each value with aa security label and constrain operations so that a well-typed program is guaranteed not to leak high-security information. For example, a simple rule is that the result of a computation must have the hightest security label of any input used in its computation. If a low-security result depends on a high-security input, the type system would reject the program. Thus, those security type systems can automatically verify that programs adhere to confidentiality policies.

However, existing IFC type systems can be rather inflexible or coarse-grained. Many system treat security labels in a rigid, lattice-based way [4], and often difficult to coexist with other sophisticated type system constructs. To address these issues, we propose the FLOWSTLC, a more flexible and extensible type system that track informational flows with *graded modalities*. More specifically, we will design a simply typed lambda-calculus with integrated *graded necessity* for the semiring $\{\texttt{Public} \sqsubseteq \texttt{Secret}\}$. This system will allow us precisely control how the values interact as a type of coeffects, while remain easy to extend with other program reasoning constructs like the *usage analysis* [5].

II. FLOWSTLC: THE CORE CALCULUS

A. Syntax

B. The Security Semiring

C. Typing

D. Evaluation

E. Graded Modality

III. METATHEORY

A. Substitution

Lemma 1 (Permutation)**.** *If $\Gamma \vdash t : T$ and Δ is a permutation of Γ, then $\Delta \vdash t : T$ and the derivation depth of the latter is the same as the former.*

The proof for the permutation lemma is trivial since assumptions in contexts are unrelated in a simply-typed context. The permutation lemma enables us to state the two substitution lemmas in a cleaner form, we first prove the well-typedness of substitution through regular assumptions:

Lemma 2 (Well-typed Substitution)**.** *If $\Gamma, x : S \vdash t : T$ and $\Delta \vdash s : S$, then $\Gamma, \Delta \vdash [x \mapsto s]t : T$.*

Lemma 3 (Well-typed Graded Substitution)**.** *If $\Gamma, x : [S]_\ell \vdash t : T$ and $[\Delta] \vdash s : S$, then $\Gamma, \ell \cdot \Delta \vdash [x \mapsto s]t : T$.*

B. Type Preservation

Theorem 1 (Type Preservation)**.**

C. Progression

D. Strong Normalization

E. Noninterference

F. Type Checking Algorithm

IV. Implementation and Examples

V. Related Work

Language-based IFC has a rich literature. Sabelfeld and Myers [3] survey a variety of security type systems that enforce noninterference via typing. Seminal work by Volpano *et al.* [6] introduced a static type system for a simple imperative language, ensuring well-typed programs satisfy noninterference. Extensions include JFlow [7] and JIF [8] for Java, which associate security labels with Java types to track flows, and FlowCaml [9] for OCaml, which similarly adds a security type-checker. Generally, these systems label each variable as high or low and enforce rules so that no operation can use a high value in a low context.

More recently, graded type theories have been proposed to generalize such analyses. Graded type systems annotate types with additional information (”grades”) to capture various properties of programs. For example, the Granule language [5] uses graded modalities to track the effects and coeffects of program. In information flow use-cases, types are graded by a security lattice, allowing automatic enforcement of noninterference. Moon *et al.* introduce the Graded Modal Dependent Type Theory (GrTT) [1], which extends dependent type system with graded modality and show that the grades can be effective at reasoning of programs. This work demonstrates that graded modality can capture fine-grained flow policies in types. Similarly, Marshall and Orchard 10 demonstrated a graded-modal framework that can enforce confidentiality and even integrity simultaneously.

Other approaches include dynamic IFC (e.g. LIO monad in Haskell [11]) or hybrid systems, but our focus is purely static. In summary, while classical security type sytems (JIF, FlowCaml, etc.) enforce noninterference via fixed lattice-labels. Our project draws on these ideas: we adapt graded modalities to a simple functional language to track information flows more flexibly.

VI. Further Work and Conclusion

Acknowledgment

References

[1] B. Moon, H. Eades III, and D. Orchard, “Graded modal dependent type theory,” in *European Symposium on Programming*. Springer International Publishing Cham, 2021, pp. 462–490.

[2] G. Smith, “Principles of secure information flow analysis,” in *Malware Detection*. Springer, 2007, pp. 291–307.

[3] A. Sabelfeld and A. C. Myers, “Language-based information-flow security,” *IEEE Journal on selected areas in communications*, vol. 21, no. 1, pp. 5–19, 2003.

[4] D. E. Denning, “A lattice model of secure information flow,” *Communications of the ACM*, vol. 19, no. 5, pp. 236–243, 1976.

[5] D. Orchard, V.-B. Liepelt, and H. Eades III, “Quantitative program reasoning with graded modal types,” *Proceedings of the ACM on Programming Languages*, vol. 3, no. ICFP, pp. 1–30, 2019.

[6] D. Volpano, C. Irvine, and G. Smith, “A sound type system for secure flow analysis,” *Journal of computer security*, vol. 4, no. 2-3, pp. 167–187, 1996.

[7] A. C. Myers, “Jflow: Practical mostly-static information flow control,” in *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 1999, pp. 228–241.

[8] K. Pullicino, “Jif: Language-based information-flow security in java,” *arXiv preprint arXiv:1412.8639*, 2014.

[9] V. Simonet and I. Rocquencourt, “Flow caml in a nutshell,” in *Proceedings of the first APPSEM-II workshop*, 2003, pp. 152–165.

[10] D. Marshall and D. Orchard, “Graded modal types for integrity and confidentiality,” *arXiv preprint arXiv:2309.04324*, 2023.

[11] D. Stefan, A. Russo, J. C. Mitchell, and D. Mazières, “Flexible dynamic information flow control in haskell,” in *Proceedings of the 4th ACM Symposium on Haskell*, 2011, pp. 95–106.

APPENDIX

A. Complete Specification of FLOWSTLC

1) Syntax:

Term	t	::=	x	*variable*
		|	$t\ t$	*application*
		|	$\lambda x.t$	*abstraction*
		|	$[t]$	*packing*
		|	**let** $[x] = t : T$ **in** t	*unpacking*
Type	T	::=	$T \to T$	*function type*
		|	$\Box_r T$	*graded modality*
Value	v	::=	$\lambda x : T.t$	*abstraction value*
Context	Γ	::=	$\emptyset$	*empty context*
		|	$\Gamma, x : T$	*assumption*
		|	$\Gamma, x : [T]_r$	*graded assumption*
Security	ℓ	::=	`Secret`	
		|	`Public`	

2) Security Level Semiring:

Definition 1 (Security Level Semiring)**.** *The security level semiring is a two-point lattice of security levels* $\{\texttt{Public} \sqsubseteq \texttt{Secret}\}$ *with*

- $0 = \texttt{Secret}$
- $1 = \texttt{Public}$
- *Addition as the meet:* $r + s = r \sqcap s$
- *Multiplication as the join:* $r \cdot s = r \sqcup s$

See Appendix B for a proof that this algebra is a indeed a semiring.

3) Auxiliary Definitions:

Definition 2 (Context Concatenation)**.**

$$\emptyset + \Gamma = \Gamma$$
$$\Gamma + \emptyset = \Gamma$$
$$(\Gamma, x : T) + \Gamma' = (\Gamma + \Gamma'), x : T \text{ iff } x \notin dom(\Gamma')$$
$$\Gamma + (\Gamma', x : T) = (\Gamma + \Gamma'), x : T \text{ iff } x \notin dom(\Gamma')$$
$$(\Gamma, x : [T]_r) + (\Gamma', x : [T]_s) = (\Gamma + \Gamma'), x : [T]_{r+s}$$

Definition 3 (Context Scalar Multiplication)**.**

$$r \cdot \emptyset = \emptyset$$
$$r \cdot (\Gamma, x : [T]_s) = (r \cdot \Gamma), x : [T]_{(r \cdot s)}$$

4) Typing Rules:

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T}\ \text{T-VAR}$$

$$\frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \lambda x : T_1.t : T_1 \to T_2}\ \text{T-ABS}$$

$$\frac{\Gamma_1 \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma_2 \vdash t_2 : T_{11}}{\Gamma_1 + \Gamma_2 \vdash t_1\ t_2 : T_{12}}\ \text{T-APP}$$

$$\frac{\Gamma \vdash t : T}{\Gamma, \Gamma' \vdash t : T}\ \text{T-WEAK}$$

$$\frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma, x : [T_1]_{\texttt{Public}} \vdash t : T_2}\ \text{T-DER}$$

$$\frac{[\Gamma] \vdash t : T}{\ell \cdot [\Gamma] \vdash [t] : \Box_\ell T}\ \text{T-PRO}$$

where $[\Gamma]$ denotes a context containing only graded assumptions.

$$\frac{\Gamma_1 \vdash t_1 : \Box_\ell T_1 \qquad \Gamma_2, x : [T_1]_\ell \vdash t_2 : T_2}{\Gamma_1 + \Gamma_2 \vdash \textbf{let}\ [x] = t_1\ \textbf{in}\ t_2 : T_2}\ \text{T-LET}$$

$$\frac{\Gamma, x : [T_2]_{\ell_1} \vdash t : T_1 \qquad \ell_1 \sqsubseteq \ell_2}{\Gamma, x : [T_2]_{\ell_2} \vdash t : T_1}\ \text{T-APPROX}$$

5) Evaluation Rules:

$$\frac{t_1 \to t_1'}{t_1\ t_2 \to t_1'\ t_2}\ \text{E-APP1}$$

$$\frac{t_2 \to t_2'}{v_1\ t_2 \to v_1\ t_2'}\ \text{E-APP2}$$

$$(\lambda x : T.t)\ v \to [x \mapsto v]t \quad \text{E-APPABS}$$

$$\frac{t \to t'}{[t] \to [t']}\ \text{E-PRO}$$

$$\frac{t_1 \to t_1'}{\textbf{let}\ [x] = t_1\ \textbf{in}\ t_2 \to \textbf{let}\ [x] = t_1'\ \textbf{in}\ t_2}\ \text{E-LET-EVAL}$$

$$\textbf{let}\ [x] = v\ \textbf{in}\ t \to [x \mapsto v]t \quad \text{E-LET-UNBOX}$$

B. Proofs

1) Security Level Semiring:

Proof.

- **Associativity of addition**: $(a + b) + c = a + (b + c)$
  This is trivial since the semiring addition is meet, and meet is associative in a lattice.
- **Commutativity of addition**: $a + b = b + a$
  This is also trivial since meet is commutative in a lattice.
- **Additive identity**: $a + 0 = a$ for all a
  Since $0 = \texttt{Secret}$ and $\texttt{Public} \sqsubseteq \texttt{Secret}$, we have
  $$\begin{aligned} \texttt{Public} \sqcap \texttt{Secret} &= \texttt{Public} \\ \texttt{Secret} \sqcap \texttt{Secret} &= \texttt{Secret} \end{aligned} \tag{1}$$
- **Associativity of multiplication**: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
  This is trivial since the semiring multiplication is join, and join is associative in a lattice.
- **Multiplication distributes over addition**: $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ and $(a + b) \cdot c = (a \cdot c) + (b \cdot c)$
  Since the lattice is a chain (and thus distributive), this holds.
- **Multiplicative identity**: $1 \cdot a = a \cdot 1 = a$ for all a
  Since $1 = \texttt{Public}$ and $\texttt{Public} = \texttt{Secret}$, we have
  $$\begin{aligned} \texttt{Public} \sqcup \texttt{Public} &= \texttt{Public} \\ \texttt{Public} \sqcup \texttt{Secret} &= \texttt{Secret} \\ \texttt{Secret} \sqcup \texttt{Public} &= \texttt{Secret} \end{aligned} \tag{2}$$
- **Multiplication by 0 annihilates**: $0 \cdot a = a \cdot 0 = 0$ for all a

Since $0 = \texttt{Secret}$ and $\texttt{Public} \sqsubseteq \texttt{Secret}$, we have

$$\begin{aligned} \texttt{Secret} \sqcup \texttt{Public} &= \texttt{Secret} \\ \texttt{Secret} \sqcup \texttt{Secret} &= \texttt{Secret} \\ \texttt{Public} \sqcup \texttt{Secret} &= \texttt{Secret} \end{aligned} \tag{3}$$

Thus the security level algebra is a semiring. □

2) Well-typed substitution:

Proof. By induction on a derivation of the form $\Gamma, x : S \vdash t : T$. For a given derivation, we proceed by cases on the final typing rule used in the proof.

- *Case* T-VAR: $t = z$ with $z : T \in (\Gamma, x : S)$
  We need to consider two subcases:
  1) if $z = x$, then $[x \mapsto s]z \to [x \mapsto s]x \to s$. So we need to show $\Gamma, \Delta \vdash s : S$. To get this we apply the T-WEAK rule and the permutation lemma to the assumption $\Delta \vdash s : S$ which will give us desired result.
  2) otherwise $[x \mapsto s]z \to z$, and the result is immediate.
- *Case* T-ABS: $t = \lambda y : T_2.t_1$, $T = T_2 \to T_1$, $\Gamma, x : S, y : T_2 \vdash t_1 : T_1$
  By convention we may assume that $x \neq y$ and $y \notin FV(s)$. Then by the permutation lemma and weakening rule, we have $\Gamma, \Delta, y : T_2, x : S \vdash t_1 : T_1$. Again we apply the two rules on the assumption $\Delta \vdash s : S$ to get $\Gamma, \Delta, y : T_2 \vdash s : S$. By the induction hypothesis, we have $\Gamma, \Delta, y : T_2 \vdash [x \mapsto s]t_1 : T_1$, then by T-ABS $\Gamma, \Delta \vdash \lambda y : T_2.[x \mapsto s]t_1 : T_2 \to T_1$, and this is our desired result.
- *Case* T-APP: $t = t_1\ t_2$, $\Gamma, x : S \vdash t_1 : T_2 \to T_1$, $\Gamma, x : S \vdash t_2 : T_2$, $T = T_1$
  By the induction hypothesis, we have $\Gamma \vdash [x \mapsto s]t_1 : T_2 \to T_1$ and $\Gamma \vdash [x \mapsto s]t_2 : T_2$. Then by T-APP, $\Gamma \vdash [x \mapsto s]t_1\ [x \mapsto s]t_2 : T_1$, i.e., $\Gamma \vdash [x \mapsto s](t_1\ t_2) : T_1$.
- *Case* T-WEAK: $t = t_1$, $\Gamma' \vdash t_1 : T$ where $\Gamma' \subseteq (\Gamma, x : S)$ and $(\Gamma, x : S) - \Gamma'$ does not contain assumptions graded by $\texttt{Public}$.
  By the induction hypothesis, we have $\Gamma' \vdash [x \mapsto s]t_1 : T$, then apply the T-WEAK, we have $\Gamma, x : S \vdash [x \mapsto s]t_1 : T$.
- *Case* T-DER: $t = t_1$, $T = T_1$, $\Gamma, x : S, y : T_2 \vdash t_1 : T_1$
  By the induction hypothesis and permutation lemma, we have $\Gamma, y : T_2 \vdash [x \mapsto s]t_1 : T_1$. Then by the T-Der rule, we immediately have our desired result $\Gamma, y : [T_2]_{\texttt{Public}} \vdash [x \mapsto s]t_1 : T_1$.
- *Case* T-PRO:
  This case is impossible since T-PRO rule requires a purely graded context, and $x : S$ would violate the premise.
- *Case* T-LET: $t = \textbf{let}\ [y] = t_1\ \textbf{in}\ t_2$, $\Gamma_1, x : S \vdash t_1 : \Box_\ell T_1$, $\Gamma_2, y : [T_1]_\ell, x : S \vdash t_2 : T_2$, $T = T_2$
  Again by convention we may assume that $x \neq y$ and $y \notin FV(s)$. First we apply the induction hypothesis to get $\Gamma_1 \vdash [x \mapsto s]t_1 : \Box_\ell T_1$ and $\Gamma_2, y : [T_1]_\ell \vdash [x \mapsto s]t_2 : T_2$. And it follows from the T-LET rule that $\Gamma_1 + \Gamma_2 \vdash$ **let** $[y] = [x \mapsto s]t_1$ **in** $[x \mapsto s]t_2 : T_2$, i.e., $\Gamma_1 + \Gamma_2 \vdash [x \mapsto s](\textbf{let}\ [y] = t_1\ \textbf{in}\ t_2) : T_2$.
- *Case* T-APPROX: $t = t_1$, $\Gamma, x : S, y : [T_2]_{\ell_1} \vdash t_1 : T_1$, $\ell_1 \sqsubseteq \ell_2$, $T = T_1$.
  From the induction hypothesis and permutation lemma we can get $\Gamma, y : [T_2]_{\ell_1} \vdash [x \mapsto s]t_1 : T_1$. Then we apply the T-APPROX rule to get our desired result $\Gamma, y : [T_2]_{\ell_2} \vdash [x \mapsto s]t_1 : T_1$.

□

3) Well-typed Graded Substitution:

Proof. □

4) Type Preservation:

Proof. □