

# Дисертація

Система верифікації програмного забезпечення

Максим Сохацький, Павло Маслянко



# Зміст

---

1	Вступ	1
1.1	Системна інженерія та верифікація	1
1.2	Історія систем верифікації	2
1.3	Методи верифікації	3
1.4	Метематичне забезпечення	4
1.4.1	Теорія категорій	4
1.4.2	Алгебраїчні типи даних	6
1.4.3	Лямбда числення	7
1.4.4	Числення процесів	8
1.4.5	Інтуїціоністична теорія типів Мартіна Льюфа	9
1.4.6	Логіка та квантори	10
1.5	Дослідження середовищ виконання	11
1.5.1	Agda та M-Alonso	11
1.5.2	Coq та coq.io	11
1.5.3	Rust	11
1.5.4	Erlang	11
1.6	Архітектура компонентів системи	11
	Бібліографія	12



## Глава 1

# Вступ

---

### 1.1 Системна інженерія та верифікація

Протягом історії обчислювальної техніки було створено різні класи та способи обчислень, різні теорії та підходи до програмування таких систем, різні класи систем програмування. Зараз уже стало зрозумілим, що інженіринг систем які не піддаються до верифікації формальними методами не може бути застосований у галузях де вимоги до якості особливо підвищені, як то космонавтика, енергетика та фінанси.

Об'єктом дослідження данної роботи є системи верифікації програмного забезпечення та операційні системи які виконують обчислення в реальному часі, їх поєднання та побудова формальної системи для уніфікованого середовища, яке поєднує середовище виконання та систему верифікації у єдину систему мов.

Предметом дослідження такої системи мов є теорія типів, яка вивчає обчислювальні властивості мов. Теорія типів виділилася в окрему науку Мартіном Льюфом як запит на вакантне місце у трикутнику теорій, які відповідають ізоморфізму Каррі-Говарда (Логіки, Мови, Категорії). Інші дві це: теорія категорій та логіка вищих порядків. Сама система доведення теорем є логікою, або аспектом логіки у трикутнику. Імплементация мови програмування, яка релізує логічну семантику здійснюється завдяки теорії типів. Формалізація методів відбувається завдяки теорії категорій, яка є абстрактною алгеброю функцій, математичним інструментом для формалізації мов програмування та довільних математичних теорій які описуються логіками вищих порядків.

## 1.2 Історія систем верифікації

Перші спроби пошуку формального фундаменту для теорії обчислень були покладені Алонзо Черчем та Хаскелем Каррі у 30-х роках 20-го століття. Було запропоноване лямбда числення як апарат який може замінити класичну теорію множин та її аксіоматику, пропонуючи при цьому обчислювальну семантику. Пізніше в 1958, ця мова була втілена у вигляді LISP лауреатом премії тюрінга Джоном МакКарті, який працював в Принстоні. Ця мова була побудована на таких примітивах, як: cons, nil, eq, atom, car, cdr, lambda, apply and id. Направді це уривки індуктивних конструкцій які були структуровані пізніше і формалізовані за допомогою теорії категорій Вільяма Лавіра. До цих пір нетипізоване лямбда числення є одною з мов у які робиться екстракт з сучасних пруверів. Окрім LISP, нетипізоване лямбда числення маніфестується у такі мови як Erlang, JavaScript, Python.

Перший математичний прувер AUTOMATH (і його модифікації AUT-68 та AUT-QE), який був написаний для комп'ютерів розроблявся під керівництвом де Брейна, 1967. У цьому прувері був квантор загальності та лямбда функція, таким чином це був перший прувер побудований на засадах ізоморфізма Каррі-Говарда.

ML/LCF або метамова і логіка обчислювальних функцій був наступник крок до досягнення фундаментальної мови простору, тут впреше з'явилися алебраїчні типи даних у вигляді індуктивних типів, поліноміальних функторів або терміновані (well-founded) дерев. Пізніше були побудовані категорні моделі Татсоа Хагіно (CPL) та Крокетом (Charity). Роберт Мілнер, асистований Морісом та Н'юві розробив Метамову (ML), як інструмент для побудови прувера LCF. LCF був основоположником у родині пруверів HOL88, HOL90, HOL98 та останньої версії на даний час HOL/Isabell.

У 80-90 роках були створені інші системи автоматичного доведення теорем, такі як Mizar (Трибулек, 1989). PVS (Оур, Рушбі, Шанкар, 1995), ACL2 на базі Common Lisp (Боєр, Кауфман, Мур, 1996), Otter (МакКюн, 1996).

### 1.3 Методи верифікації

Можна виділити два підходи до верифікації. Перший застосовується де вже є певна програма написана на певній мові програмування і потрібно довести ізоморфність цієї програми до доведеної моделі. Ця задача вирішується у побудові теоретичної моделі для певної мови програмування, потім програма на цій мові переводиться у цю теоретичну модель і доводить ізоморфізм цієї програми у побудованій моделі до доведеної моделі. Приклади таких систем та піходів. VST (CompCert, сертифікація Cі-програм), NuPRL (Cornell University, розподілені системи, залежні типи), TLA+ (Microsoft Reseach, Леслі Лампорт), Twelf (для верифікації мов програмування).

Інший підхід можна назвати підходом вбудованих DSL. Усе моделювання відбувається в основній мові, а сертифіковані програми автоматично екстрактяться в довільні мови. Приклади таких систем: Coq побудована на мові OCaml від науково-дослідного інституту Франції INRIA; Agda побудовані на мові Haskell від шведського інституту технологій Чалмерс; Lean побудована на мові C++ від Microsoft Research та Універсистету Каргені-Мелона; Idris подудована на мові Haskell Едвіна Бреді з шотландського Університету ім. св. Андрія; F\* – окремий проект Microsoft Research.

Завдання цього дослідження є побудова єдиної системи, яка поєднує середовище викодання та систему верифікації програмного забезпечення. Це прикладне дослідження, яке є фьюжином фундаментальної математики та інженерних систем з формальними методами верицікації. Методи цього дослідження є суто теоретичними.

## 1.4 Метематичне забезпечення

### 1.4.1 Теорія категорій

Теорія категорій широко застосовується як інструмент для математиків у тому числі і при аналізі програмного забезпечення. Теорію категорій можна вважати абстрактною алгеброю функцій. Дамо конструктивне визначення категорії. Категорії (програми) визначаються переліком своїх об'єктів (типів) та своїх морфізмів (функцій), а також бінарною операцією композиції, що задовольняє закону асоціативності, та з тотожнім морфізмом (тотальною функцією — одиницею) який існує для кожного об'єкту (типу) категорії. Аксиоми формації об'єктів не приводяться та постулюються в нижніх аксіомах. Аксиома формації морфізмів буде даватися як введення експоненти після визначення декартового добутку. Поки що тут буде визначатися тільки композиція морфізмів. Об'єкти  $A$  та  $B$  морфізма  $f : A \rightarrow B$  називаються домен та кодомен відповідно.

Інтро аксиоми — асоціативність композиції та права і ліва композиції одиниці показують, що категорії є типизованими моноїдами, що складаються з морфізмів та операції композиції. Є різні мови, у тому числі і графічні, представлення категорної семантики, однак у цій роботі ми будемо використовувати теоретико-логічні формулювання.

$$\frac{\Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash g : B \rightarrow C}{\Gamma \vdash g \circ f : A \rightarrow C}$$

$$\frac{\Gamma \vdash f : B \rightarrow A \quad \Gamma \vdash g : C \rightarrow B \quad \Gamma \vdash h : D \rightarrow C}{\Gamma \vdash (f \circ g) \circ h = f \circ (g \circ h) : D \rightarrow A}$$

$$\frac{}{\Gamma \vdash \text{id}_A : A \rightarrow A}$$

$$\frac{\Gamma \vdash f : A \rightarrow B}{\Gamma \vdash f \circ \text{id}_A = f : A \rightarrow B}$$

$$\frac{\Gamma \vdash f : A \rightarrow B}{\Gamma \vdash \text{id}_B \circ f = f : A \rightarrow B}$$



Композиція показує можливість зв'язувати область значень попереднього обчислення (кодомен) та область визначення наступного обчислення (домен). Композиція є фундаментальною властивістю морфізмів.

1.  $A : *$
2.  $A : *, B : * \implies f : A \rightarrow B$
3.  $f : B \rightarrow C, g : A \rightarrow B \implies f \circ g : A \rightarrow C$
4.  $(f \circ g) \circ h = f \circ (g \circ h)$
5.  $A \implies \text{id} : A \rightarrow A$
6.  $f \circ \text{id} = f$
7.  $\text{id} \circ f = f$

### 1.4.2 Алгебраїчні типи даних

Після операції композиції, як способу конструювання нових об'єктів за допомогою морфізмів далі йде операція конструювання добутка двох об'єктів певної категорії, разом з добутком морфізмів зі спільним доменом, необхідних для визначення декартового добутка  $A \times B$ .

Це є внутрішня мова декартової категорії, у якій для будь яких двох доменів існує їх декартова сума (кодобутку) та декартовий добуток (косума, кортеж), за допомогою яких конструюються суми-протоколи та добутки-повідомлення, а також існує  $\perp$  тип-термінал, та  $\top$  тип-котермінал. Термінальними типами зручно термінувати рекурсивні типи даних, такі як списки. Ми будемо розглядати тільки категорії які мають добутки та суми.

Добуток має природні елімінатори  $\pi$  зі спільним доменом, які є морфізмами-проекціями об'єктів добутка. Сума має обернені елімінатори  $\sigma$  зі спільним кодомом. Як видно добуток є дуальний до суми з точністю до направлення стрілок, таким чином елімінатори  $\pi$  та  $\sigma$  є оберненими, тобто  $\pi \circ \sigma = \sigma \circ \pi = \text{id}$ .

$$\begin{array}{c}
 \frac{\Gamma x : A \times B}{\Gamma \vdash \pi_1 : A \times B \rightarrow A; \Gamma \vdash \pi_2 : A \times B \rightarrow B} \quad \frac{}{\Gamma \vdash \top} \\
 \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash (a, b) : A \times B} \quad \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash a | b : A \otimes B} \\
 \frac{}{\Gamma \vdash \perp} \quad \frac{\Gamma x : A \otimes B}{\Gamma \vdash \sigma_1 : A \rightarrow A \otimes B; \Gamma \vdash \sigma_2 : B \rightarrow A \otimes B}
 \end{array}$$

Також додамо тут аксіому множення морфізмів, яка впливає з визначення добутка, яка необхідна для забезпечення аплікативного програмування.

$$\begin{array}{c}
 \frac{\Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash g : A \rightarrow C \quad \Gamma \vdash B \times C}{\Gamma \vdash \langle f, g \rangle : A \rightarrow B \times C} \\
 \pi_1 \circ \langle f, g \rangle = f \\
 \pi_2 \circ \langle f, g \rangle = g \\
 \langle f \circ \pi_1, f \circ \pi_2 \rangle = f \\
 \langle f, g \rangle \circ h = \langle f \circ h, g \circ h \rangle \\
 \langle \pi_1, \pi_2 \rangle = \text{id}
 \end{array}$$

## 1.4.3 Лямбда числення

Будучи внутрішньою мовою декартово-замкненої категорії лямбда числення окрім змінних та констант у вигляді термів пропонує операції абстракції та аплікації, що визначає достатньо лаконічну та потужну структуру обчислень з функціями вищих порядків, та метатипизаціями, такими як System F, яка була запропонована вперше Робіном Мілнером в мові ML, та зараз присутня в більш складних, таких як System F $\omega$ , системах Haskell та Scala.

Щоб пояснити функції з категоріальні точки зору потрібно пояснити категоріальні експоненти  $f : A^B$ , які є аналогами функціональних просторів  $f : A \rightarrow B$ . Так як ми вже визначили добутки та термінали, то ми можемо визначити і експоненти, опускаючи усі категоріальні подробиці ми визначимо конструювання функції (операція абстракції), яка параметризується змінною  $x$  у середовищі  $\Gamma$ ; та її елімінатора – операції аплікації функції до аргументу. Так визначається декартово-замкнена категорія. Визначається також рекурсивний механізм виклику функції з довільною кількістю аргументів.

$$\frac{\Gamma x : A \vdash M : B}{\Gamma \vdash \lambda x. M : A \rightarrow B}$$

$$\frac{\Gamma f : A \rightarrow B \quad \Gamma a : A}{\Gamma \vdash \text{apply } f \ a : (A \rightarrow B) \times A \rightarrow B}$$

$$\frac{\Gamma \vdash f : A \times B \rightarrow C}{\Gamma \vdash \text{curry } f : A \rightarrow (B \rightarrow C)}$$

$$\begin{aligned} \text{apply} \circ \langle (\text{curry } f) \circ \pi_1, \pi_2 \rangle &= f \\ \text{curry } \text{apply} \circ \langle g \circ \pi_1, \pi_2 \rangle &= g \\ \text{apply} \circ \langle \text{curry } f, g \rangle &= f \circ \langle \text{id}, g \rangle \\ (\text{curry } f) \circ g &= \text{curry } (f \circ \langle g \circ \pi_1, \pi_2 \rangle) \\ \text{curry } \text{apply} &= \text{id} \end{aligned}$$

$$\begin{aligned} \text{Об'єкти} &: \perp \mid \rightarrow \mid \times \\ \text{Морфізми} &: \text{id} \mid f \circ g \mid \langle f, g \rangle \mid \text{apply} \mid \lambda \mid \text{curry} \end{aligned}$$

#### 1.4.4 Числення процесів

Теорія  $\pi$ -числення процесів Роберта Мілнера є основним формалізмом обчислювальної теорії розподілених систем та її імплементації. З часів виникнення CSP числення розробленого Хораром, Мілнеру вдалося значно розширити та адаптувати теорію до сучасних телекомунікаційних вимог, як наприклад хендвери в мобільних мережах. Основні теореми в моделі  $\pi$ -числення стосуються непротиворічливості та неблокованості у синхронному виконанні мобільних процесів. Так як сучасний Web можна розглядати як телекомунікаційну систему, тому у розробці додатків можна покладатися у тому числі і на такі моделі як  $\pi$ -числення. Також ми анонсуємо процес як фундаментальний тип даних, подібний до функції але який здатний тримати певний стан у вигляді типу координати та є морфізмом-одиноцею типу свого стану.

$$\begin{array}{c}
 \frac{\Gamma \vdash E, \Sigma, X \quad \Gamma \vdash \text{action} : \Sigma \times X \rightarrow \Sigma \times X}{\Gamma \vdash \text{spawn action} : \pi_\Sigma} \\
 \\
 \frac{\Gamma \vdash \text{pid} : \pi_\Sigma \quad \Gamma \vdash \text{msg} : \Sigma}{\Gamma \vdash \text{join msg pid} : \Sigma \times \pi_\Sigma \xrightarrow{\bullet} \Sigma; \Gamma \vdash \text{send msg pid} : \Sigma \times \pi_\Sigma \rightarrow \Sigma} \\
 \\
 \frac{\Gamma \vdash L : A + B, R : X + Y \quad \Gamma \vdash M : A \rightarrow X, N : B \rightarrow Y}{\Gamma \vdash \text{receive } L \ M \ N : L \xrightarrow{\bullet} R}
 \end{array}$$

Алгебра процесів визначає базові операції мультиплексування двох чи декількох протоколів в рамках одного процесу (добуток), а також паралельного та повністю ізольованого запуску включно зі стеком та областю пам'яті (сума) на віртуальній машині.

$$\begin{array}{ll}
 \oplus & : \quad \pi \parallel \pi \\
 \otimes & : \quad \pi \mid \pi
 \end{array}$$

### 1.4.5 Інтуїтіоністична теорія типів Мартіна Льюфа

Системи з залежними типами як верифікаційні математичні формальні моделі для доведення коректності. Система  $\Sigma$  та  $\Pi$  типів, як кванторів існування та узагальнення. Системи Mizar, Coq, Agda, Idris, F\*, Lean. Ми будемо використовувати cubicaltt, Coq та Lean для доведення MLTT моделей.

Розбудовуючи певний фреймворк чи систему конструктивними методами так чи інакше доведеться зробити певний вибір у мові та способі кодування. Так при розробці теорії абстрактної алгебри в Coq були використані поліморфні індуктивні структури. Однак Agda та Idris використовують для побудови алгебраїчної теорії типи класів, а у Idris взагалі відсутні поліморфні індуктивні структури та коіндуктивні структури. В Lean теж відсутні коіндуктивні структури проте повністю реалізована теорія HoTT на нерекурсивних поліморфних структурах що об'єднує основні чотири класи математичних теорій: логіка, топологія, теорія множин, теорія типів. Як було показано Стефаном Касом, одна з стратегій імплементації типів класів — це використання поліморфних структур. В Lean також підтримуються типи класів.

### 1.4.6 Логіка та квантори

Модель логік вищого порядку та квантори  $\forall$  та  $\exists$  які теж виражаються як конструкції типів:

$$\begin{array}{c}
 \frac{\Gamma x : A \vdash B \quad \Gamma \vdash A}{\Gamma \vdash \Pi(x : A)B} \quad \frac{\Gamma \vdash a : A \quad \Gamma x : A \vdash B \quad \Gamma b : B(x = a)}{\Gamma \vdash (a, b) : \Pi(x : A)B} \\
 \\
 \frac{\Gamma x : A \vdash B \quad \Gamma \vdash A}{\Gamma \vdash \Sigma(x : A)B} \quad \frac{\Gamma \vdash a : A \quad \Gamma x : A \vdash B \quad \Gamma b : B(x = a)}{\Gamma \vdash (a, b) : \Sigma(x : A)B} \\
 \\
 \frac{\Gamma \vdash x : A \quad \Gamma \vdash x' : A}{\Gamma \vdash \text{Id}_A(x, x')}
 \end{array}$$

рефлексивність	:	$\text{Id}_A(a, a)$
підстановка	:	$\text{Id}_A(a, a') \rightarrow B(x = a) \rightarrow B(x = a')$
симетричність	:	$\text{Id}_A(a, b) \rightarrow \text{Id}_A(b, a)$
транзитивність	:	$\text{Id}_A(a, b) \rightarrow \text{Id}_A(b, c) \rightarrow \text{Id}_A(a, c)$
конгруентність	:	$(f : A \rightarrow B) \rightarrow \text{Id}_A(x, x') \rightarrow \text{Id}_B(f(x), f(x'))$

## 1.5 Дослідження середовищ виконання

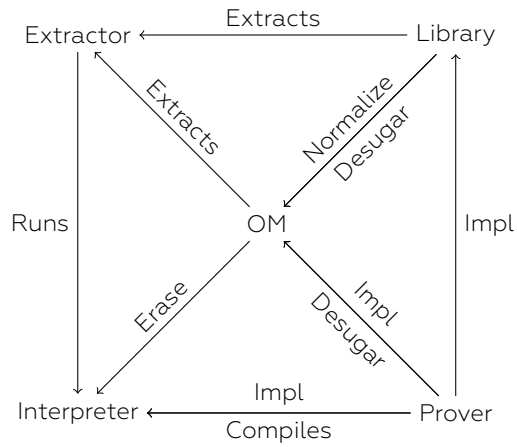
## 1.5.1 Agda та M-Alonso

## 1.5.2 Coq та coq.io

## 1.5.3 Rust

## 1.5.4 Erlang

## 1.6 Архітектура компонентів системи



# Бібліографія

---

- [1] S.MacLane Categories for the Working Mathematician 1972
- [2] W.Lawvere Conceptual Mathematics 1997
- [3] P.Curien Category theory: a programming language-oriented introduction 2008
- [4] P.Martin-Löf Intuitionistic Type Theory 1984
- [5] T.Coquand The Calculus of Constructions. 1988
- [6] E.Meijer Henk: a typed intermediate language 1997
- [7] H.Barendregt Lambda Calculus With Types 2010
- [8] F.Pfenning Inductively defined types in the Calculus of Constructions 1989
- [9] P.Wadler Recursive types for free 1990
- [10] N.Gambino Wellfounded Trees and Dependent Polynomial Functors 1995
- [11] P.Dybjer Inductive Families 1997
- [12] B.Jacobs (Co)Algebras and (Co)Induction 1997
- [13] V.Vene Categorical programming with (co)inductive types 2000
- [14] H.Geuevers Dependent (Co)Inductive Types are Fibrational Dialgebras 2015
- [15] T.Streicher A groupoid model refutes uniqueness of identity proofs 1994
- [16] T.Streicher The Groupoid Interpretation of Type Theory 1996
- [17] B.Jacobs Categorical Logic and Type Theory 1999
- [18] S.Awodey Homotopy Type Theory and Univalent Foundations 2013
- [19] S.Huber A Cubical Type Theory 2015
- [20] A.Joyal What is an elementary higher topos 2014
- [21] A.Mortberg Cubical Type Theory: a constructive univalence axiom 2017