

# Концептуальна модель системи доведення теорем

Maksym Sokhatskyi <sup>1</sup> and Pavlo Maslianko <sup>1</sup>

<sup>1</sup> National Technical University of Ukraine  
Igor Sikorsky Kyiv Polytechnical Institute  
18 травня 2018 року

## Анотація

Дана робота презентує концептуальну модель системи доведення теорем, яка предствалена як система: 1) числень разом з їх категорними моделями [статика], тобто властивостями та семантикою; 2) програм та базових бібліотек [екземпляри], для програмного забезпечення; 3) програмних трансформацій між різними численнями [динаміка], компіляція, екстракт програм в інші мови програмування, оптимізація, тощо. Значна увага приділена таким проблемам як: 1) масштабування системи доведення теорем на довільні числення; 2) декомпозиції системи мовні рівні. В основу роботи покладені сучасні фундаментальні дослідження з кубічної теорії типів, що дозволяє формулювати і доводити гомотопічні моделі та їх властивості.

## Зміст

1	Вступ	3
1.1	Актуальність роботи	3
1.2	Верифікація та валідація	3
1.3	Формальна специфікація	3
2	Формальні методи верифікації	4
2.1	Історія систем доведення теорем	4
2.2	Системи моделювання та верифікації	5
2.3	Мови з залежними типами	5
2.4	Системи автоматичного доведення теорем	6
2.5	Обмеження	6
3	Формалізована постановка задачі	6
3.1	Об'єкт дослідження	6
3.2	Предмет дослідження	6
3.3	Мотивація	6
3.4	Завдання дослідження	7
3.5	Структура дослідження	7
4	Мінімальна верифікована система програмування	7
4.1	Інтерпретатор безтипового лямбда числення	7
4.2	Чиста система з однією аксіомою	8
5	Вищі мови програмування	8
5.1	Індуктивні мови	8
5.2	Гомотопічні мови	8
5.3	Мови числення процесів	8
6	Результати дослідження	9
6.1	CPS Інтерпретатор	9
6.2	Чиста мова Om	9
6.3	Гомотопічна мова Infinity	9
6.4	Гомотопічна базова бібліотека	9
7	Висновки	9

# 1 Вступ

## 1.1 Актуальність роботи

Ціна помилок в індустрії надзвичайно велика. Наведемо відомі приклади: 1) Mars Climate Orbiter (1998), помилка невідповідності типів британської метричної системи, коштувала 80 мільйонів фунтів стерлінгів. Невдача стала причиною переходу NASA повністю на метричну систему в 2007 році. 2) Ariane Rocket (1996), причини катастрофи – округлення 64-бітного дійсного числа до 16-бітного. Втрачені кошти на побудову ракети та запуск 500 мільйонів. 3) Помилка в FPU в перших Pentium (1994), збитки на 300 мільйонів. 4) Помилка в SSL (heartbleed), оцінені збитки у розмірі 400 мільйонів. 5) Помилка у логіці бізнес-контрактів EVM та DAO (неконтрольована рекурсія), збитки 50 мільйонів. Більше того, і найголовніше, помилки у програмному забезпеченні можуть коштувати життя людей.

## 1.2 Верифікація та валідація

Для унеможливлення помилок на виробництві застосовуються різні методи формальної верифікації. Формальна верифікація — доказ, або заперечення відповідності системи у відношенні до певної формальної специфікації або характеристики, із використанням формальних методів математики.

Дамо основні визначення згідно з міжнародними нормами (IEEE, ANSI)<sup>1</sup> та у відповідності до вимог Європейського Аерокосмічного Агентства<sup>2</sup>. У відповідності до промислового процесу розробки, верифікація та валідація програмного забезпечення є частиною цього процесу. Програмне забезпечення перевіряється на відповідність функціональних властивостей згідно вимог.

Процес валідації включає в себе перегляд (code review), тестування (модульне, інтеграційне, властивостей), перевірка моделей, аудит, увесь комплекс необхідний для доведення, що продукт відповідає вимогам висунутим при розробці. Такі вимоги формуються на початковому етапі, результатом якого є формальна специфікація.

## 1.3 Формальна специфікація

Для спрощення процесу верифікації та валідації застосовується математична техніка формалізації постановки задачі — формальна специфікація. Формальна специфікація — це математична модель, створена для опису систем, визначення їх основних властивостей, та інструментарій для перевірки властивостей (формальної верифікації) цих систем, побудованих на основі формальної специфікації.

<sup>1</sup>IEEE Std 1012-2016 — V&V Software verification and validation

<sup>2</sup>ESA PSS-05-10 1-1 1995 – Guide to software verification and validation

Існують два фундаментальні підходи до формальних специфікацій: 1) Агребраїчний підхід, де система описується в термінах операцій, та відношень між ними, та 2) Модельно-орієнтований підхід, де модель створена конструктивними побудовами, як то на базі теорії множин, чи інакше, а системні операції визначаються тим, як вони змінюють стан системи. Також були створені сімейства послідованих та розподілених мов.

Найбільш стандартизована та прийнята в області формальної верифікації — це нотація  $Z^3$  (Spivey, 1992), приклад модельно-орієнтованої мови Названа у честь Ернеста Цермело, роботи якого мали вплив на фундамент математики та аксіоматику теорії множин. Саме теорія множин, та логіка предикатів першого порядку є теорією мови  $Z$ .

Інша відома мова формальної специфікації як стандарт для моделювання розподілених систем, таких як телефонні мережі та протоколи, це LOTOS<sup>4</sup> (Bolognesi, Brinksma, 1987), як приклад алгебраїчного підходу. Ця мова побудована на темпоральних логіках, та поведінках залежних від спостережень. Інші темпоральні мови специфікацій, які можна відзначити тут — це TLA+<sup>5</sup>, CSP (Hoare, 1985), CCS<sup>6</sup> (Milner, 1971), Actor Model, Reactive Streams, etc.

## 2 Формальні методи верифікації

### 2.1 Історія систем доведення теорем

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 конструктивних примітивах, які пізніше виявилися компонентами індуктивних конструкцій та були формалізовані за допомогою теорії категорій Вільяма Лавіра. Окрім LISP, нетипізоване лямбда числення маніфестується у такі мови як Erlang, JavaScript, Python. До цих пір нетипізоване лямбда числення є одною з мов у які робиться конвертація доведених програм (екстракція).

Перший математичний прuver AUTOMATH (і його модифікації AUT-68 та AUT-QE), який був написаний для комп'ютерів розроблявся під керівництвом де Брейна, 1967. У цьому прuverі був квантор загально-

<sup>3</sup>ISO/IEC 13568:2002 — Z formal specification notation

<sup>4</sup>ISO 8807:1989 — LOTOS — A formal description technique based on the temporal ordering of observational behaviour

<sup>5</sup>The TLA+ Language and Tools for Hardware and Software Engineers

<sup>6</sup>J.C.M. Baeten. A Brief History of Process Algebra.

сті та лямбда функція, таким чином це був перший прuver побудований на засадах ізоморфізма Каррі-Говарда-Ламбека.

ML/LCF або метамова і логіка обчислювальних функцій були наступним кроком до досягнення фундаментальної мови простору, тут вперше з'явилися алебраїчні типи даних у вигляді індуктивних типів, поліноміальних функторів або термінованих (well-founded) дерев. Роберт Мілнер, асистований Морісом та Н'юві розробив Метамову (ML), як інструмент для побудови прuverа LCF. LCF був основоположником у родині прuverів HOL88, HOL90, HOL98 та останньої версії на даний час HOL/Isabell. Пізніше були побудовані категорні моделі Татсою Хагіно (CPL, Японія) та Робіна Кокета (Charity, Канада).

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).

## 2.2 Системи моделювання та верифікації

Можна виділити три підходи до верифікації. Перший застосовується де вже є певна програма написана на певній мові програмування і потрібно довести ізоморфізм 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: 1) VST (CompCert, сертифікація C програм), 2) NuPRL (Cornell University, розподілені системи, залежні типи), 3) TLA+ (Microsoft Research, Леслі Лампорт), 4) Twelf (для верифікації мов програмування), 5) SystemVerilog (для чіпрограминого та апаратного забезпечення).

## 2.3 Мови з залежними типами

Другий підхід можна назвати підходом вбудованих мов. Компілятор основної мови перевіряє модель закодовану у ній же. Можливо моделювання логік вищого порядку, лінійних логік, модальних логік, категорний та гомотопічних логік. Процес специфікації та верифікації відбувається в основній мові, а сертифіковані програми автоматично екстрагуються в довільні мови. Приклади таких систем: 1) Coq побудована на мові OCaml від науково-дослідного інституту Франції INRIA; 2) Agda побудовані на мові Haskell від шведського інституту технологій Чалмерс; 3) Lean побудована на мові C++ від Microsoft Research та Університету Каргені-Мелона; 4) Idris побудована на мові Haskell Едвіна Бреді з шотландського Університету ім. св. Андрія; 5) F\* – окремий проект Microsoft Research.

## 2.4 Системи автоматичного доведення теорем

Третій підхід полягає в синтезі конструктивного доведення для формальної специфікації. Це може бути зроблено за допомогою асистентів доведення теорем, таких як HOL/Isabell, Coq, ACL2, або систем розв'язку задач виконуваності формул в теоріях (Satisfiability Modulo Theories, SMT).

## 2.5 Обмеження

Незалежно від піходу до верифікації, формальна верифікація неможлива, якщо мова програмування моделі формально не визначена. Це означає, що значна міра програмного забезпечення може бути автоматично верифікована тільки для тих мов, формальні моделі яких побудовані, на даний момент це тільки мова C. Більше того, не завжди можна також формально довести те, що програма завершиться, потрібно звужувати клас програм, якщо формальні специфікації містять такі властивості.

# 3 Формалізована постановка задачі

## 3.1 Об'єкт дослідження

Об'єктом дослідження даної роботи є: 1) системи верифікації програмного забезпечення; 2) системи доведення теорем; 3) мови програмування; 4) операційні системи, які виконують обчислення в реальному часі; 5) їх поєднання, побудова формальної системи для уніфікованого середовища, яке поєднує середовище виконання та систему верифікації у єдину систему мов та засобів.

## 3.2 Предмет дослідження

Предметом дослідження такої системи мов є теорія типів, яка вивчає обчислювальні властивості мов. Теорія типів виділилася в окрему науку завдяки роботам Бішоп та Мартіна-Льофа, як запит на вакантне місце у трикутнику теорій, які відповідають ізоморфізму Каррі-Говарда-Ламбека (Логіки, Мови, Категорії), або інтерпретації Брауера-Гейтінга-Колмогорова. Інші дві це: теорія категорій та логіки вищих порядків.

## 3.3 Мотивація

Одна з причин низького рівня впровадження у виробництво систем верифікації – це висока складність таких систем. Складні системи верифікуються складно. Ми хочемо запропонувати спрощений підхід до верифікації – оснований на концепції компактних та простих мовних

ядер для створення специфікацій, моделей, перевірки моделей, доведення теорем у теорії типів з кванторами.

### 3.4 Завдання дослідження

Завдання дослідження полягає у: 1) декомпозиції типових систем, побудові єдиної категорії мов програмування, де об'єкти — це мови програмування, а морфізми — це мовні перетворення; 2) виділенні мінімальної конфігурації яка дозволяє доводити властивості програмного забезпечення на усіх етапах від проектування до виконання; 3) побудові гнучкої поліморфної системи мовних розширень до мінімального ядра для забезпечення потреб масштабування системи; 4) реалізації мінімального ядра системи та вищих мовних розширень разом із базовими бібліотеками.

### 3.5 Структура дослідження

Структура дослідження відображає компонентну модель системи доведення теорем: 1) інтерпретатор безтипового лямбда числення; 2) компактне ядро — система з однією аксіомою; 3) мова з індуктивними типами; 4) мова з гомотопічним інтервалом  $[0, 1]$ ; 5) уніфікована базова бібліотека.

## 4 Мінімальна верифікована система програмування

Пошук мінімальної конфігурації мов визначається мотивацією дослідження, як запорука прозорості та адаптованості до безкоштовної верифікації або верифікації малими зусиллями.

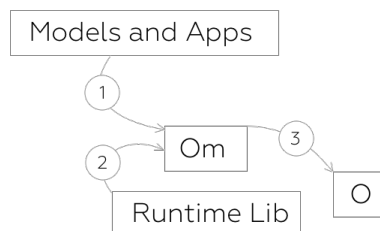


Рис. 1: The Stages of Model Verifications

### 4.1 Інтерпретатор безтипового лямбда числення

Чому саме інтерпретатор? Як можна побудувати ефективний інтерпретатор? Нормальний та зворотній порядок бета редукції. Вектори-

зація обчислень.

## 4.2 Чиста система з однією аксіомою

Чиста система. Операційна семантика. Категорна семантика залежної теорії.

## 5 Вищі мови програмування

Математична індукція. Топологічні та гомотопічні моделі.

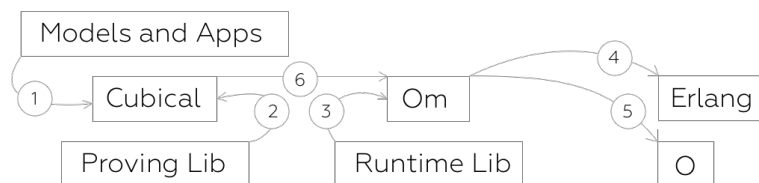


Рис. 2: The Stages of Model Verifications

### 5.1 Індуктивні мови

Категорна семантика індуктивних типів.

### 5.2 Гомотопічні мови

Розшарування CCHM. Кубічна система типів.

### 5.3 Мови числення процесів

Середовище виконання. Бісімуляція.



## 6 Результати дослідження

### 6.1 CPS Інтерпретатор

### 6.2 Чиста мова Om

### 6.3 Гомотопічна мова Infinity

### 6.4 Гомотопічна базова бібліотека

## 7 Висновки

[22] [21] [3] [10] [27] [26] [24] [4] [14] [23] [28] [16] [30] [7] [18] [12] [1] [13]  
[15] [5] [29] [20] [17] [8] [9] [19] [2] [25] [11] [6]

## Список литературы

- [1] Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '16, pages 18–29, New York, NY, USA, 2016. ACM.
- [2] Thorsten Altenkirch and Conor McBride. Towards observational type theory. 2006.
- [3] H. P. Barendregt. In S. Abramsky, Dov M. Gabbay, and S. E. Maibaum, editors, Handbook of Logic in Computer Science (Vol. 2), pages 117–309, New York, NY, USA, 1992. Oxford University Press, Inc.
- [4] Gilles Barthe. Extensions of pure type systems, pages 16–31. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.
- [5] Henning Basold and Herman Geuvers. Type theory based on dependent inductive and coinductive types. In Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, pages 327–336. ACM, 2016.
- [6] Errett Bishop. Foundations of constructive analysis. 1967.
- [7] Corrado Böhm and Alessandro Berarducci. Automatic synthesis of typed lambda-programs on term algebras. In Theoretical Computer Science, volume 39, pages 135–154, 1985.
- [8] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. In Cubical Type Theory: a constructive interpretation of the univalence axiom, volume abs/1611.02108, 2017.
- [9] Thierry Coquand. An algorithm for type-checking dependent types. In Sci. Comput. Program., volume 26, pages 167–177, 1996.

- [10] Thierry Coquand and Gerard Huet. The calculus of constructions. In *Information and Computation*, pages 95–120, Duluth, MN, USA, 1988. Academic Press, Inc.
- [11] Thierry Coquand, Per Martin-Löf, Vladimir Voevodsky, André Joyal, Andrej Bauer, Steve Awodey, Matthieu Sozeau, Michael Shulman, Dan Licata, Yves Bertot, Peter Dybjer, and Nicola Gambino. *Homotopy Type Theory: Univalent Foundations of Mathematics*. 2013.
- [12] P.É. Dagand, University of Strathclyde. Department of Computer, and PhD thesis Information Sciences. *A Cosmology of Datatypes: Reusability and Dependent Types*. 2013.
- [13] Peter Dybjer. In *Inductive families*, volume 6, pages 440–465. Springer, 1994.
- [14] Peng Fu and Aaron Stump. Self types for dependently typed lambda encodings. In *Rewriting and Typed Lambda Calculi - Joint International Conference, RTA-TLCA 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, pages 224–239, 2014.
- [15] Nicola Gambino and Martin Hyland. Wellfounded trees and dependent polynomial functors. In *International Workshop on Types for Proofs and Programs*, pages 210–225. Springer, 2003.
- [16] Herman Geuvers. Induction Is Not Derivable in Second Order Dependent Type Theory, pages 166–181. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.
- [17] Makoto Hamana and Marcelo P. Fiore. A foundation for gadts and inductive families: dependent polynomial functor approach. In *Proceedings of the seventh ACM SIGPLAN workshop on Generic programming, WGP@ICFP 2011, Tokyo, Japan, September 19-21, 2011*, pages 59–70, 2011.
- [18] Ralf Hinze and Nicolas Wu. Histo- and dynamorphisms revisited. In *Proceedings of the 9th ACM SIGPLAN Workshop on Generic Programming, WGP '13*, pages 1–12, New York, NY, USA, 2013. ACM.
- [19] Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory (Venice, 1995)*, volume 36 of *Oxford Logic Guides*, pages 83–111, New York, 1998. Oxford Univ. Press.
- [20] Bart Jacobs and Jan Rutten. In *A tutorial on (co) algebras and (co) induction*, volume 62, pages 222–259. EUROPEAN ASSOCIATION FOR THEORETICAL COMPUTER, 1997.

- [21] Simon Peyton Jones and Erik Meijer. Henk: A typed intermediate language. In *In Proc. First Int'l Workshop on Types in Compilation*, 1997.
- [22] Saunders MacLane. *Categories for the Working Mathematician*. Springer-Verlag, New York, 1971. Graduate Texts in Mathematics, Vol. 5.
- [23] P. Martin-Löf and G. Sambin. *Intuitionistic type theory. Studies in proof theory*. Bibliopolis, 1984.
- [24] Christian-Emil Ore. The extended calculus of constructions (ecc) with inductive types. In *Information and Computation*, volume 99, pages 231 – 264, 1992.
- [25] Ian Orton and Andrew M Pitts. Axioms for modelling cubical type theory in a topos. *arXiv preprint arXiv:1712.04864*, 2017.
- [26] Christine Paulin-Mohring. Introduction to the Calculus of Inductive Constructions. In Bruno Woltzenlogel Paleo and David Delahaye, editors, *All about Proofs, Proofs for All*, volume 55 of *Studies in Logic (Mathematical logic and foundations)*. College Publications, January 2015.
- [27] Frank Pfenning and Christine Paulin-Mohring. Inductively defined types in the calculus of constructions. In *Mathematical Foundations of Programming Semantics, 5th International Conference*, Tulane University, New Orleans, Louisiana, USA, March 29 - April 1, 1989, *Proceedings*, pages 209–228, 1989.
- [28] Aaron Stump. The calculus of dependent lambda eliminations. In *Journal of Functional Programming*, volume 27. Cambridge University Press, 2017.
- [29] Varmo Vene. *Categorical programming with inductive and coinductive types*. Tartu University Press, 2000.
- [30] Philip Wadler. *In Recursive types for free*. manuscript, 1990.