Giới thiệu về lý thuyết hình thái trong khoa học máy tính

Ngày 21 tháng 11 năm 2023

Bế Trọng Nghĩa
github.com/ichxorya

Curly Howard (1903–1952), tên khai sinh là Jerome Lester Horwitz.
Là nghệ sĩ hài và diễn viên người Mỹ, không liên quan đến lý thuyết hình thái.
Tuy nhiên, tên ông giống Curry-Howard (tên của hai nhà nghiên cứu lớn trong lĩnh vực này).

TÓM TẮT

Lý thuyết hình thái [hiểu theo nghĩa tổng quát] là ngành nghiên cứu về các hệ thống kiểu. Khởi nguồn với lý thuyết hình thái của Russell trong quá trình tìm ra giải pháp cho khủng hoảng nền tảng toán học, lý thuyết hình thái [nói chung] đã trở thành một trong những thành phần cốt lõi của ngành khoa học máy tính. Tuy nhiên, lĩnh vực này ở Việt Nam chưa được chú trọng vì nhiều lí do. Bài nghiên cứu này có mục tiêu giới thiệu lý thuyết hình thái đến với mọi người ở hầu như mọi phương diện: từ lịch sử và triết học cho đến lý thuyết và ứng dụng; đồng thời hi vọng từ sự chia sẻ đó, mọi người có thể quan tâm đến lĩnh vực này hơn, và quan trọng hơn cả là tiếp thu được những kiến thức có ích để ứng dụng vào quá trình học tập và lao động.

Từ khóa: *Lý thuyết hình thái, hệ thống kiểu, khủng hoảng nền tảng toán học, khoa học máy tính.*

Nội dung chính

Phụ lục

1. Mở đầu

Lý thuyết hình thái - một cái tên dường như vô cùng xa lạ kể cả với những người hàng ngày làm việc với những gì xoay quanh nó. Là một trong những trụ cột của ngành khoa học máy tính, lý thuyết hình thái (về cơ bản) là lĩnh vực nghiên cứu các hệ thống kiểu; tuy nhiên ứng dụng của nó không dừng ở phạm vi **lý thuyết ngôn ngữ lập trình** mà còn ảnh hưởng sâu sắc đến lĩnh vực **"cơ giới hóa" toán học** (đặc biệt bao gồm **các phương pháp hình thức** - vốn có nhiều ứng dụng thực tiễn trong các bài toán yêu cầu tính "bảo đảm" cao).

Tuy nhiên, có một thực trạng đáng buồn tại Việt Nam, **khoa học máy tính lý thuyết** nói chung, cũng như lý thuyết hình thái nói riêng, vì nhiều lý do khách quan đã không có điều kiện phát triển. Việc nghiên cứu khoa học lý thuyết mà chưa khai thác được ứng dụng thực tiễn, cộng thêm sự trỗi dậy của trí tuệ nhân tạo trong những năm gần đây, có thể được coi là lời giản thích cơ bản cho gốc rễ vấn đề này. Bản thân lý thuyết hình thái cũng chưa được nghiên cứu và công bố (theo tìm hiểu của người viết, tại Việt Nam mới chỉ có một bài báo của Viện Toán học và Khoa học ứng dụng Thăng Long giới thiệu về dự án *Formal Abstract in Mathematics* vào năm 2021 [1]); cũng vì vậy nên không có tài liệu giảng dạy hay sách tham khảo, chuyên khảo bằng tiếng Việt. Ngoài ra, trên internet tồn tại một số rất ít bài viết về lĩnh vực này, nhưng hầu hết xoay quanh góc độ triết học của lý thuyết hình thái [do Russell đề xuất] (ví dụ [2], [3], [4], [5], [6]); hoặc nếu có liên quan đến lý thuyết hình thái trong khoa học máy tính, thì chỉ có duy nhất 2 bài viết trên Wikipedia [7], [8] .

Bài báo cáo này được viết bằng ngôn ngữ tiếng Việt, với hi vọng có thể giúp làm giàu kho tư liệu về lý thuyết hình thái; cũng như mong muốn rằng đề tài này sẽ trở nên dễ dàng tiếp cận hơn với người Việt. Một khía cạnh cốt lõi của khoa học máy tính mà chưa có từ vựng để biểu diễn (thậm chí ngay cả cái tên lý thuyết hình thái cũng chưa có cách gọi thống nhất!); tốt hơn hết nên được hoàn thiện. Dẫu biết rằng truyền thống ngành [khoa học máy tính] chúng ta có sử dụng những từ "chưa Việt hóa" (ví dụ: *logic học* thay vì *luận lý học*) vì đặc thù chuyên ngành cũng như văn hóa, ngôn ngữ...; song, có thể dễ dàng thấy, việc chuyển ngữ/dịch thuật trong khoa học, hay thậm chí là "Việt hóa" khoa học đã từ lâu là một đề tài làm "đau đầu" các nhà nghiên cứu. Tuy nhiên, khó không có nghĩa là từ bỏ; và bài báo cáo này dù chỉ là một bài tập trên lớp, song vẫn hi vọng có thể là tiền đề của những bước tiến về sau của lý thuyết hình thái tại Việt Nam.

2. Lịch sử của lý thuyết hình thái

Phần này chủ yếu tham khảo cuốn sách *A Modern Perspective On Type Theory* của Fairouz Kamareddine và các cộng sự [9]; với nội dung đề cập đến sự phát triển của lý thuyết hình thái từ trước khi nó được định nghĩa cho đến những sự phát triển về sau.

2.1. Thời "tiền sử"

Trước khi lý thuyết hình thái được đề xuất lần đầu tiên bởi Russell, những ý tưởng gần với **"hệ [thống] hình thái"** (hay **hệ thống kiểu**) đã được áp dụng ngầm định trong toán học, logic học và triết học. Có thể chỉ đơn giản như *"tam đoạn luận"* của Aristotle (có thể hiểu *khái niệm* `Socrates` đã được gán kiểu là `con người`, do đó `Socrates` phải tồn tại các thuộc tính của `con người`). Ở đây, **cơ chế cốt lõi** của lý thuyết hình thái là **sự phân lớp các đối tượng [toán học]**.

Mọi `con người` *rồi cũng chết.*

`Socrates` là `con người`.

Vì vậy: `Socrates` *rồi cũng chết.*

[10]

Ví dụ cụ thể được trình bày trong tài liệu này là về *hình học Euclid* và bộ sách *Cơ sở*.

2.1.1. Ý tưởng cốt lõi và sự trực quan của toán học sơ khai

Euclid (350 TCN - ?) là nhà toán học vĩ đại thời Hy Lạp cổ đại, và được mệnh danh là "cha đẻ của hình học". Ông đã hệ thống hóa hình học phẳng trong bộ sách *Cơ sở*; xét theo tư tưởng của lý thuyết hình thái, ông đã định nghĩa những khái niệm về các đối tượng hình học bằng những thuộc tính của chúng. Một phần, điều này giúp ông **phân loại các đối tượng**; mặt khác, ông định nghĩa được **các đối tượng thuộc lớp nào**, giúp nhận biết những gì có thể xảy ra, những đối tượng nào có thể được xuất hiện trong hệ thống. Xét một vài ví dụ:

1. *Điểm* là cái không [thể] chia nhỏ.
2. *Đường* là một lượng dài không có chiều rộng.
3. Một *đường thẳng* là một *đường* (bất kỳ) mà trên đó các *điểm* nằm ngang bằng. [11]
4. Ta cũng biết được rằng "giao của hai đường thẳng là một điểm", nhưng không tồn tại định nghĩa cho "giao của hai điểm".

Việc ngầm định về *hình thái* hiệu quả với hình học mà Euclid giới thiệu. Bởi lẽ, tính trừu tượng trong lĩnh vực này chưa quá cao, khi lý thuyết lại gắn liền với thực tiễn – một người dù có tư duy tốt hay không, dù có khả năng hình dung một hệ hình học phẳng trong não bộ hay không, thì vẫn có thể nhận diện được sự tồn tại của hình học phẳng trong thực tế. Cũng như hình học Euclid, nhiều nhánh khác của toán học cũng tận dụng được sự trực quan này (và cũng đồng thời ngầm định giới thiệu các *hình thái* cho những khái niệm liên quan).

2.1.2. Ảnh hưởng tiêu cực của sự gia tăng tính trừu tượng trong toán học

Tuy nhiên, khi toán học đã phát triển hơn, người ta nhận thấy tính trực quan trong toán học không còn thể hiện rõ như trước nữa. Để mô hình hóa những góc độ "thuần túy toán học", người ta đưa ra những khái niệm dường như nằm ngoài khả năng cảm nhận của con người; và kể từ thế kỷ XIX, các hệ thống toán học ngày càng trở nên thiếu "trực quan" hơn.

Với ví dụ tiêu biểu là *giải tích toán học* và *hình học phi Euclid*, **tính trừu tượng toán học** ở giai đoạn này đang có những sự thay đổi lớn, đặc biệt là về **tinh thần toán học**. Ngay từ trước thế kỷ XIX, những cuộc tranh cãi xung quanh các khái niệm trong giải tích đã diễn ra,

bởi các khái niệm dường như đi ngược lại một số quan điểm triết học (ví dụ sự phê phán nền tảng của giải tích (đặc biệt là về *thông lượng* (hay *đạo hàm*) của Newton và *vô cùng bé* của Leibniz) trong cuốn *The Analyst (Nhà phân tích)* của George Berkeley) [12]. Về hình học phi Euclid, trước tiên cần nhắc lại *5 định đề* do ông đưa ra:

1. Cùng quy ước rằng có thể vẽ một đoạn thẳng nối hai điểm bất kỳ.
2. Và có thể kéo dài liên tục một đoạn thẳng thành đường thẳng.
3. Có thể vẽ một hình tròn với tâm và bán kính bất kỳ.
4. Tất cả các góc vuông đều bằng nhau.
5. Nếu một đường thẳng (gốc) cắt hai đường thẳng khác tạo thành các góc trong về cùng một phía [với nó] có tổng nhỏ hơn hai góc vuông, thì hai đường thẳng bị cắt khi được kéo dài ra vô hạn sẽ cắt nhau ở phía của đường thẳng gốc mà tổng hai góc trong nhỏ hơn hai góc vuông (chứ không cắt lại ở phía bên kia). [11]

Định đề thứ 5 phức tạp hơn các định đề còn lại, có thể hiểu cơ bản rằng "nếu hai đường thẳng cùng cắt qua một đường thẳng mà cả hai đều không vuông góc, thì kéo dài vô tận chúng sẽ giao nhau". Từ xưa, người ta đã cố chứng minh rằng định đề thứ 5 là định lý có thể được chứng minh từ 4 định đề còn lại. Tuy nhiên, cho đến đầu thế kỷ XIX, một số nhà toán học đã phủ định việc này và đặt nền móng cho hình học phi Euclid; có thể kể đến Carl Friedrich Gauss, Nikolai Ivanovich Lobachevsky và János Bolyai. Nếu như hình học [phẳng] Euclid có thể mô hình hóa những góc độ trực quan của thế giới, thì những loại hình học trừu tượng này có "đúng" với thế giới không? Hình học nào mới là hình học "chuẩn"?

Giai đoạn này cũng bắt đầu những công trình **hình thức hóa** toán học, cùng với *phái hình thức* (là những người theo **chủ nghĩa hình thức**). Chủ nghĩa hình thức là tư tưởng cho rằng mệnh đề (toán học hay logic học) có thể được chứng minh chỉ dựa trên "hình thức" (tức là những luật suy diễn, được thao tác trên những ký hiệu toán học); có thể hiểu đơn giản rằng toán học/logic học được đặt trong một môi trường thuần túy hình thức. Việc hình thức hóa toán học có những ưu điểm nhất định, đặc biệt là việc giúp người ta có khả năng hiểu rõ hơn về thứ toán học mà họ đang nghiên cứu. Tuy nhiên, vì "hình thức" có phần xa rời, thậm chí tách biệt với "thực tại", cho nên tính trừu tượng toán học được gia tăng đáng kể.

Sau này, đến thế kỷ XX, **máy tính điện tử ra đời** với những ứng dụng thiết thực, đặc biệt là trong lĩnh vực toán học. Tuy nhiên, việc sử dụng máy tính như một công cụ toán học cũng đặt ra những thách thức nhất định, bởi việc thao tác trên các hệ thống toán học sẽ phải qua một lớp trung gian (máy tính), do đó đòi hỏi việc thay đổi tư duy toán học, cũng như đảm bảo rằng toán học triển khai trên máy tính là "chuẩn".

2.1.3. Sự lung lay của nền tảng toán học

Như đã nêu trên, toán học đã trở nên trừu tượng hơn rất nhiều. Việc thiếu trực quan khiến những định nghĩa toán học trở nên khó "xác định" hơn bao giờ hết; và đây chính là tiền đề của **những mâu thuẫn lớn trong toán học**.

Bên cạnh sự lung lay về hình học bởi những phát kiến mới (như hình học phi Euclid nêu trên), số học cũng bị ảnh hưởng ít nhiều sau khi nền tảng toán học dựa trên logic (dưới dạng dựa trên *lý thuyết tập hợp*) của họ được chứng minh là có mâu thuẫn. Trường hợp cụ thể của nhà triết học, toán học Gotlob Frege là một ví dụ kinh điển. Năm 1879, ông xuất bản cuốn sách *Begriffsschrift (Hệ ghi ý)*, đặt nền móng cho sự ***hình thức hóa logic học***. Sau đó, ông tiếp tục nghiên cứu và xuất bản nhiều cuốn sách quan trọng như *Die Grundlagen der Arithmetik*

(Nền tảng số học) và *Grundgesetze der Arithmetik (Các định luật cơ bản của số học)*, củng cố lập luận rằng: ***Số học là một nhánh của logic học và có thể được suy ra từ logic.*** Tuy nhiên, trong hệ thống toán học của ông, có một định luật như sau:

Định luật cơ bản V: $\varepsilon' f(\varepsilon) = \varepsilon' g(\varepsilon) \Leftrightarrow \forall x(f(x) = g(x))$

"Khoảng giá trị" của hàm $f(\varepsilon)$ bằng "khoảng giá trị" của hàm $g(\varepsilon)$
khi và chỉ khi với mọi x ta có $f(x) = g(x)$.

Xét lý thuyết tập hợp; về mặt logic, một *tập hợp* là một nhóm các đối tượng *thỏa mãn những đặc tính (vị từ) nhất định*, và về cơ bản *lý thuyết tập hợp* được định nghĩa bởi các *luật suy diễn logic* và *tiên đề tập hợp*. Tuy nhiên, lý thuyết tập hợp thời điểm đó mới chỉ là *"lý thuyết tập hợp ngây thơ"*, tức là chưa hoàn toàn chặt chẽ. Bertrand Russell đã nhận ra được kẽ hở trong đó, và vô tình "đánh đổ" niềm tin của Frege (dù Russell là người ngưỡng mộ Frege) cùng nhiều người đương thời về một nền tảng toán học dựa trên logic.

Đó là khi Russell phát hiện ra một nghịch lý [có thể được phát biểu] như sau:

Cho tập hợp T gồm các tập hợp x không phải là phần tử của chính nó.

$$T = \{x \mid x \text{ là tập hợp}, x \notin x\}$$

T có phải là phần tử của chính nó không?

Nếu T là phần tử của chính nó,
thì T phải không là phần tử của chính nó.

$$T \in T \Leftrightarrow T \notin T$$

Nếu T là không phải phần tử của chính nó,
thì T là phần tử của chính nó.

$$T \notin T \Leftrightarrow T \in T$$

Đây chính là **nghịch lý Russell** kinh điển; và không chỉ riêng nghịch lý này, nhiều nghịch lý toán học khác cũng xảy ra vì sự *tùy tiện* trong việc *tự tham chiếu* đến các khái niệm liên quan. Russell đã đưa ra kết luận: *Để tránh mâu thuẫn, bất cứ thứ gì liên quan đến toàn bộ tập hợp thì không được là một trong những tập hợp đó.*

Dù sao thì, toán học làm sao có thể "chính xác" nếu có nền tảng dựa trên một lý thuyết mâu thuẫn? Do vậy, Russell đã nghiên cứu và phát triển một lý thuyết khác, đó chính là **lý thuyết hình thái**. [9]

2.2. Sự hình thành và phát triển

2.2.1. Giải pháp của Russell

Về cơ bản, ban đầu Russell đã loại bỏ định luật cơ bản V của Frege, cũng như quy định **sự phân tầng** các mệnh đề logic, nhằm đảm bảo **việc "tham chiếu" bị giới hạn** trong những khoảng phù hợp, tránh xảy ra những nghịch lý liên quan đến sự tự tham chiếu [9], [13], [14] .

Như đã nêu trên, Russell đưa ra lý thuyết hình thái như một nền tảng toán học **nhất quán** – tức là *được chứng minh là không có mâu thuẫn*. Tuy nhiên, tuy nhiên điều này đã được chứng minh là không thể theo *những định lý bất toàn* của Gödel.

– Định lý 1: Một hệ hình thức **nhất quán** có khả năng biểu diễn *số học cơ bản* thì sẽ **có những mệnh đề không chứng minh được.**

– Định lý 2: Một hệ hình thức **nhất quán** không thể **tự chứng minh nó nhất quán.**

Ngoài ra, sau này vai trò nền tảng toán học được đặt cho các lý thuyết khác, đặc biệt là *lý thuyết tập hợp ZFC* của Ernst Zermelo và Abraham Fraenkel (chữ *ZF* là tên viết tắt của hai ông, còn chữ *C* là viết tắt của *Choice (tiên đề lựa chọn)*).

Tuy nhiên, lý thuyết hình thái không vì thế mà trở thành dĩ vãng; đặc biệt, trong ngành khoa học máy tính, lý thuyết hình thái là tiền đề cũng như lý thuyết cho nhiều ứng dụng về sau. Phần tiếp theo sẽ giới thiệu một số cột mốc phát triển của lý thuyết hình thái [trong khoa học máy tính] sau phát kiến của Russell.

2.2.2. Một số cột mốc tiêu biểu sau đó

Phần này tham khảo chủ yếu từ Wikipedia [15]; với nội dung đề cập đến các bước phát triển lớn của lý thuyết hình thái từ Russell cho đến các nghiên cứu sau này.

2.2.2.1. *Tương ứng Curry–Howard* của Curry và Howard (1934–1969)

Tương ứng Curry–Howard thể hiện mối quan hệ trực tiếp giữa **chương trình máy tính** và **chứng minh toán học**; và góc độ nhìn nhận này đã được Curry Haskell và William Alvin Howard phát hiện ra. Một lời giải thích đơn giản về quan sát này có thể được đưa ra như sau:

– Đặt trong ngữ cảnh logic, mệnh đề $A \to B$ được phát biểu là "nếu có mệnh đề A thì chứng minh được mệnh đề B".

– Đặt trong ngữ cảnh lập trình, mệnh đề trên có thể được phát biểu là "nếu có giá trị kiểu A thì có thể tính được giá trị kiểu B".

– Điều cốt lõi ở đây là B có được chứng minh/tính toán hay không. [16]

Do đó, tương ứng Curry–Howard còn được biết đến với cái tên **Chứng minh-là-chương trình** hoặc **Mệnh đề-là-kiểu**.

Đây là nền tảng của những liên hệ toán–tin, tiêu biểu là *Tương ứng Curry–Howard–Lambek* (mở rộng thêm *lý thuyết phạm trù*). Những liên hệ này có thể được được phát biểu là "mọi chứng minh toán học đều có thể biểu diễn bằng một chương trình máy tính, và ngược lại (và còn hơn thế nữa nếu đặt trong ngữ cảnh **ba góc nhìn lớn về [cách diễn giải khái niệm] tính toán** (xem thêm [17], [18])".

2.2.2.2. *Phép tính lambda* của Church (1936-1940)

Phép tính lambda ra đời từ những nghiên cứu của Alonzo Church; và trước khi mối quan hệ với **ngôn ngữ lập trình** được làm rõ [vào những năm 60 của thế kỷ XX], đây vẫn chỉ là một phạm trù đơn thuần thuộc về toán học. Ảnh hưởng về nhiều mặt (toán học, logic học, ngôn ngữ học [19]) và quan trọng nhất là khoa học máy tính (ngôn ngữ lập trình).

Có thể tóm tắt một số ý chính về phép tính lambda như sau:

– Là một *ngôn ngữ lập trình* đơn giản, đồng thời là *mẫu hình tính toán* giống như **máy Turing**.

$$\text{Hàm } f \text{ khả tính lambda}$$
$$\Leftrightarrow \text{Hàm } f \text{ khả tính Turing}$$

– Là nền tảng cho nhiều ngôn ngữ lập trình sau này (đặc biệt là ***ngôn ngữ lập trình hàm*** [bên cạnh **lý thuyết phạm trù**]).

2.2.2.3. *Automath* của de Brujin (1967–2003)

Automath là một *ngôn ngữ lập trình* được thiết kế bởi Nicolaas Govert de Bruijn để biểu diễn các chứng minh toán học trong máy tính; với sự kiểm định thông qua thực nghiệm hình thức hóa các lý thuyết toán học từ cuốn sách *Grundlagen der Analysis (Nền tảng giải tích)* của Edmund Landau. Đây là **dự án tiên phong** cho những **công cụ hỗ trợ chứng minh định lý** sau này. [20]

2.2.2.4. *Lý thuyết hình thái trực giác* của Martin-Löf (1971–1984)

Lý thuyết hình thái trực giác (hay **lý thuyết hình thái kiến thiết**, hoặc **lý thuyết hình thái Martin-Löf**) được nghiên cứu và phát triển bởi Per Martin-Löf; đây là loại lý thuyết hình thái được xây dựng dựa trên cơ sở **toán học kiến thiết**. [21]

Chứng minh kiến thiết yêu cầu một chứng minh phải tồn tại **đối tượng làm chứng** (hay **vật chứng**); **vật chứng** là thứ "quan sát" được **giá trị chân lý của chứng minh**.
Có thể so sánh chứng minh "truyền thống" với chứng minh kiến thiết qua một ví dụ vui như sau:

Chứng minh rằng tồn tại $x > 2$ sao cho $2^x = x^2$.

– Chứng minh phi kiến thiết: Diễn giải từng bước lập luận, "phải tồn tại x vì…"

– Chứng minh kiến thiết: Biết $x = 4$ thỏa mãn $2^x = x^2$ và $4 > 2$; vậy tồn tại $x > 2$ thỏa mãn. [22]

Đặc điểm nổi bật của lý thuyết hình thái Martin-Löf là việc sử dụng **kiểu phụ thuộc** – đây là thành phần quan trọng trong *lý thuyết ngôn ngữ lập trình* và nền tảng của nhiều ngôn ngữ lập trình sau này.

2.2.2.5. *Phép tính kiến thiết* của Coquand và Huet (1967–2003)

Phép tính kiến thiết được tạo ra bởi Thierry Coquand và Gérard Huet; cũng như *lý thuyết hình thái Martin-Löf*, phép tính kiến thiết có quan hệ với *toán học kiến thiết*. Đây là lý thuyết nền tảng của nhiều *công cụ hỗ trợ chứng minh định lý* (như Agda [23], F* [24], *Lean* [25]…); đặc biệt là công cụ đi đôi với sự phát triển lý thuyết này là **Coq**.

2.2.2.6. *Lý thuyết hình thái đồng luân* của Voevodsky (2006–2013+)

Lý thuyết hình thái đồng luân, cùng với ***nền tảng đơn diệp*** là những khái niệm được đưa ra bởi Vladimir Voevodsky – người được trao huy chương Fields năm 2002 nhờ nghiên cứu về *đối đồng luân của các đa tạp đại số*. Lý thuyết hình thái đồng luân, một mặt là **bước phát triển** của **lý thuyết hình thái trực giác**; mặt khác, là **sự mở rộng liên hệ toán–tin**, cũng như [cùng với *nền tảng đơn diệp*] đưa lý thuyết hình thái trở về vai trò **nền tảng toán học**. Đây là lĩnh vực mới trong lý thuyết hình thái và được quan tâm nghiên cứu sâu sắc [26].

2.2.2.7. Một số nghiên cứu khác

Bên cạnh lĩnh vực được nghiên cứu mạnh mẽ là *lý thuyết hình thái đồng luân*, một số lý thuyết hình thái khác cũng được nghiên cứu dưới góc độ hàn lâm hoặc dưới góc độ ứng dụng thực tế. Dưới đây là một vài ví dụ:

– **Lý thuyết hình thái tình thái đồng luân** nghiên cứu về sự mở rộng **lý thuyết hình thái tình thái** (là lý thuyết hình thái liên quan đến **logic tình thái**) và **lý thuyết hình thái đồng luân**); bổ sung những đóng góp về nhiều lĩnh vực bên cạnh toán–tin [27], [28] .

– **Lý thuyết hình thái định lượng** nghiên cứu về sự kết hợp **kiểu phụ thuộc** và **kiểu tuyến tính**, được ứng dụng trong xây dựng **hệ thống kiểu** của ngôn ngữ lập trình Idris 2 [29], [30] .

3. Một số khái niệm cơ bản trong lý thuyết hình thái

Phần này tham khảo chủ yếu từ Wikipedia [31], vì không có quá nhiều nguồn giải thích về "lý thuyết hình thái" nói chung (vì có nhiều loại lý thuyết hình thái). Danh sách được đề cập trong bài viết này chưa đầy đủ vì tính phức tạp của khái niệm cần trình bày.

3.1. Kiểu và khái niệm

Khái niệm trong lý thuyết hình thái, có thể hiểu là các đối tượng toán học được **gắn kiểu** (mỗi khái niệm đều có một kiểu); ký hiệu là *khái niệm : kiểu*.

Trong ngLuật tính toánn, h ngữ lập trình, khái niệm có thể là *biến, hằng, hàm...* Sau đây là một ví dụ trong C++:

```
int life = 42;
```

Trong ví dụ này, `life` là *khái niệm (biến)*, và `int` là *kiểu*.

3.2. Luật tính toán

Luật tính toán (còn được gọi là **luật rút gọn**), là phương thức biểu diễn sự tính toán trong lý thuyết hình thái.
Xét hai ví dụ:

$$1 + 4 : \mathbb{N} \text{ và } 5 : \mathbb{N}$$

Có thể nói $1 + 4 : \mathbb{N}$ rút gọn thành $5 : \mathbb{N}$. Những khái niệm không rút gọn được gọi là **khái niệm chính tắc** (ví dụ như $5 : \mathbb{N}$).

3.3. Một số kiểu đặc biệt

– **Kiểu rỗng**: Không có khái niệm nào [trong hệ hình thái] thuộc kiểu này.

Ứng với khái niệm *sai* trong logic, và $\emptyset$ trong lý thuyết tập hợp.

– **Kiểu đơn vị**: Có duy nhất một khái niệm thuộc kiểu này.

Ứng với khái niệm *tập chỉ có 1 phần tử* trong lý thuyết tập hợp.

– **Kiểu tích**: Xét kiểu tích của hai kiểu A và B.

Ta có: $(a, b) : A \times B$ là một cặp có thứ tự, trong đó $a : A$ và $b : B$.

Ứng với khái niệm $\wedge$ trong logic và $\cap$ trong lý thuyết tập hợp.

– **Kiểu tổng**: Xét kiểu tổng của hai kiểu A và B.

Khái niệm thuộc kiểu $A + B$ sẽ tồn tại dưới dạng $a : A$ hoặc $b : B$.

Ứng với khái niệm $\vee$ trong logic và $\cup$ trong lý thuyết tập hợp.

Trong các ngôn ngữ lập trình:

– *Kiểu tích* thường được biểu diễn dưới dạng `struct` (một phần tử thuộc kiểu `struct`, cũng là thuộc một kiểu bao gồm nhiều kiểu).

– *Kiểu tổng* thường được biểu diễn dưới dạng `enum` (một phần tử thuộc kiểu `enum`, cũng là thuộc một kiểu trong nhiều kiểu khác nhau)

4. Ứng dụng của lý thuyết hình thái

Phần này chỉ nói về hai ứng dụng chính của lý thuyết hình thái [ở góc độ toán–tin]; ngoài ra lý thuyết hình thái [nói chung] cũng có những ứng dụng ở những lĩnh vực khác (như đã trình bày).

4.1. Hệ thống kiểu cho ngôn ngữ lập trình

4.1.1. Sơ lược về hệ thống kiểu

Hệ thống kiểu là thành phần không thể thiếu với mọi ngôn ngữ lập trình hiện đại. Về cơ bản (như đã trình bày), một hệ thống kiểu bao gồm các luật liên quan đến **kiểu** và **khái niệm** *(ví dụ như biến, hàm, biểu thức…)*.
Sau đây là một vài ví dụ về các "luật":

– **Hành động nào** được phép thực hiện lên **khái niệm**?
Ví dụ: Trong C++, `double x1` *có thể được* ***ép kiểu*** *sang* `int x2`.

– **Khái niệm** được phép **có giá trị nào**?
Ví dụ: Trong Java, hàm `main` *hợp lệ phải có kiểu* `void`.

Phần tiếp theo sẽ giới thiệu qua về hệ thống của của ngôn ngữ lập trình Rust – một ngôn ngữ có hệ thống kiểu *tĩnh*, *mạnh* và được thiết kế tốt.

4.1.2. Hệ thống kiểu của Rust

Rust là ngôn ngữ lập trình *đa năng*, *đa mẫu hình*, với mục tiêu đem đến cho người sử dụng ***hiệu năng***, ***tính đảm bảo*** *và* ***hiệu suất làm việc***.

– Về **hiệu năng:** Rust là ngôn ngữ lập trình hệ thống, cho nên không sử dụng *môi trường thực thi* hay *bộ thu gom rác*, hỗ trợ **giảm lượng tài nguyên cần dùng**.

– Về **tính đảm bảo:** Rust có *hệ thống kiểu* được thiết kế tốt, cùng với việc sử dụng cơ chế **"quyền sở hữu"** để quản lý bộ nhớ, giúp **đảm bảo an toàn cho tác vụ đa luồng và quản lý bộ nhớ.**

Sau đây là các khái niệm liên quan đến *"quyền sở hữu"*:

– **Quyền sở hữu:** Mọi **giá trị** đều thuộc **sở hữu** bởi một **biến**.

```rust
fn main() {
  let life = String::from("42");
}
```

Trong ví dụ này, biến `life` sở hữu giá trị thuộc kiểu `String` là `"42"`.

– **Nhượng quyền: Quyền sở hữu** có thể được chuyển giữa các **biến/hàm**, với điều kiện **chỉ có một chủ sở hữu ở mọi thời điểm.**

```rust
fn main() {
  let life = String::from("42");
  let anotherlife = life;

  println!("Another Life: {}", anotherlife);
  println!("Life: {}", life); // hetcuu💀💀💀
}
```

Trong ví dụ này, biến `life` đã **nhượng quyền sở hữu** cho biến `anotherlife`.

Câu lệnh cuối không thể được thực hiện, bởi quyền sở hữu giá trị đã bị chuyển cho biến khác.

Ngoài ra, với những biến có kiểu dữ liệu được cài đặt **cơ chế sao chép** (VD: kiểu nguyên thủy), **việc gán sẽ không nhượng quyền**.

– **Cho mượn:** Nếu không **nhượng quyền**, giá trị vẫn có thể được **"mượn"** thông qua **tham chiếu.**

```
fn main() {
    let life = String::from("42");
    let borrowlife = &life;

    println!("Borrow Life: {}", borrowlife);
    println!("Life: {}", life);
}
```

Trong ví dụ này, biến `borrowlife` đã được **mượn** quyền truy cập vào biến `life` để **đọc**.

– **Thời gian tồn tại:** Khi **ra ngoài phạm vi** cho phép, **biến** sẽ bị **hủy**.

```
fn main() {
  {
    let bigchunggus = String::from("420");
  }
  println!("{} 🔥🔥🔥", bigchunggus); // hetcuu💀💀💀
}
```

Trong ví dụ này, biến `bigchunggus` đã **hết thời gian tồn tại** do đã **vượt ra ngoài phạm vi** khai báo, do đó câu lệnh cuối không thể được thực hiện.

Cơ chế *"quyền sở hữu"* giúp Rust có **hiệu năng** của một ngôn ngữ hệ thống (như C/C++), đồng thời không phải thực hiện quá trình quản lý bộ nhớ thủ công; cũng như tính **an toàn** khi **sử dụng bộ nhớ** hay **thực hiện các tác vụ đa luồng**; và việc xuất hiện cơ chế này là do hệ thống kiểu của Rust được xây dựng như một ***hệ thống kiểu affine*** [32]. Ngoài ra, hệ thống kiểu còn có các tính năng nâng cao như **kiểu phụ thuộc**, **suy diễn kiểu**... giúp tăng khả năng biểu đạt của ngôn ngữ trong việc mô hình hóa các bài toán và phát triển phần mềm.

4.2. Cơ giới hóa toán học

Cơ giới hóa, ở đây là việc phát triển các công cụ toán học trên nền tảng máy tính, cũng như tận dụng máy tính để thực hiện các tác vụ toán học. Lý thuyết hình thái (đặc biệt là lý thuyết hình thái trực giác) là cơ sở cho nhiều mặt của sự cơ giới hóa toán học, đặc biệt được ứng dụng trong **lý thuyết chứng minh** và **các phương pháp hình thức**. Những *công cụ hỗ trợ chứng minh định lý* đã giới thiệu cũng nằm trong phạm vi ứng dụng này; sau đây là một số thành tựu đạt được nhờ vào vận dụng công nghệ vào toán học.

4.2.1. Một số thành tựu về chứng minh toán học

– **Định lý bốn màu** (1976): Là **định lý lớn đầu tiên** được chứng minh bằng máy tính, bởi Appel, Haken, và Koch.

– **Cần bao nhiêu số khởi điểm để bắt đầu giải được Sodoku?** (2012): Kết quả là **17**, bởi McGuire, Tugemann, và Civario.

– **Giả thuyết Kelper** (2003–2014): Là giả thuyết hơn 400 năm sau mới được **chứng minh hình thức**, bởi Hales và cộng sự.

4.2.2. Sơ lược về các phương pháp hình thức

Phương pháp hình thức là các kỹ thuật và công cụ chặt chẽ về mặt toán học để **đặc tả**, **thiết kế** và **xác minh** hệ thống phần mềm và phần cứng. Có thể kể đến một số phương pháp thường thấy như: **đặc tả hình thức**, **kiểm chứng hình thức**, **kiểm thử dựa trên đặc tính**.

Phương pháp hình thức đã hỗ trợ **kỹ nghệ phần mềm** trong việc phát triển những hệ thống hay phần mềm đặc biệt quan tâm đến tính **đảm bảo** về sự **an toàn** hay **bảo mật**.
Sau đây là một số ví dụ tiêu biểu về ứng dụng của phương pháp hình thức:

– *seL4* là một **vi nhân hệ điều hành** mã nguồn mở hiệu năng cao và có tính đảm bảo cao. *seL4* độc đáo vì đã được **kiểm chứng hình thức** toàn diện mà không ảnh hưởng đến hiệu suất; và được sử dụng như một nền tảng đáng tin cậy để xây dựng các hệ thống quan trọng về an toàn và bảo mật. [33]

– *CPU Intel Core i7* là **CPU** dùng cho máy tính cá nhân; theo một bài nghiên cứu được công bố của nhóm nghiên cứu thuộc Intel, *cụm thực thi lõi (core execution cluster)* (thành phần chịu trách nhiệm về hoạt động chức năng của tất cả các vi lệnh) đã được đảm bảo chất lượng bằng **kiểm chứng hình thức** thay vì **kiểm thử** thông thường [34].

Tài liệu tham khảo

[1] Hà Huy Khoái và các cộng sự, "Toán học thời 4.0 và dự án "Formal abstract in mathematics" (FAB)". Available at: https://thuvienso.thanglong.edu.vn/handle/TLU/4682

[2] Bài phỏng vấn trên Edge.org (Nguyễn Tiến Văn dịch và giới thiệu), "Gôdel và bản tính của chân lý toán học". [Online]. Available at: https://tiasang.com.vn/khoa-hoc-cong-nghe/godel-va-ban-tinh-cua-chan-ly-toan-hoc-1177/

[3] Bertrand Russell (Nguyễn Văn Khoa dịch), "Toán học và lô-gic học (1919)". [Online]. Available at: https://www.ired.edu.vn/vn/so-tay-giao-duc/2856/lo-gic-hoc-va-toan-hoc-b-russell-1919

[4] Cao Dao, "Cái trác việt, cái bình phàm và cái nửa vời [đối thoại]". [Online]. Available at: https://tienve.org/home/activities/viewThaoLuan.do?action=viewArtwork&artworkId=11469

[5] Max Black (Nguyễn Đăng Khoa dịch), "Toán học và triết học (1933)". [Online]. Available at: https://www.phantichkinhte123.com/2021/04/toan-hoc-va-triet-hoc-m-black-1933.html#_ftn1

[6] Lê Dọn Bàn, "Russell – Đưa vào Triết học Toán học". [Online]. Available at: https://chuyendaudau.blogspot.com/2021/08/russell-gioi-thieu-triet-hoc-toan-hoc.html

[7] "Lý thuyết hình thái". [Online]. Available at: https://vi.wikipedia.org/wiki/L%C3%BD_thuy%E1%BA%BFt_h%C3%ACnh_th%C3%A1i

[8] "Lý thuyết hình thái đồng luân". [Online]. Available at: https://vi.wikipedia.org/wiki/L%C3%BD_thuy%E1%BA%BFt_h%C3%ACnh_th%C3%A1i_%C4%91%E1%BB%93ng_lu%C3%A2n

[9] Fairouz Kamareddine và các cộng sự, *A Modern Perspective On Type Theory.*

[10] John Stuart Mill, *A System Of Logic, Ratiocinative And Inductive (Vol. 1).*

[11] Vũ Thái Hà và các cộng sự dịch, *Cơ sở của hình học.*

[12] George Berkeley, *The Analyst.*

[13] Aatu, "Russell's theory of types". [Online]. Available at: https://planetmath.org/russellstheoryoftypes

[14] Thierry Coquand, "Type Theory". [Online]. Available at: https://plato.stanford.edu/entries/type-theory/

[15] "History of type theory". [Online]. Available at: https://en.wikipedia.org/wiki/History_of_type_theory

[16] Joey Eremondi, "Câu trả lời cho "Why is the Curry-Howard isomorphism?"". [Online]. Available at: https://cstheory.stackexchange.com/questions/50714/why-is-the-curry-howard-isomorphism

[17] Fewer Lacunae, "Computational Trinitarianism". [Online]. Available at: https://kevinbinz.com/2015/06/26/computational-trinitarianism/

[18] Các tác giả trên trang wiki nLab, "computational trilogy". [Online]. Available at: https://ncatlab.org/nlab/show/computational+trilogy

[19] Barbara B.H. Partee và các cộng sự, *Mathematical Methods in Linguistics*.

[20] "About Automath". [Online]. Available at: https://www.win.tue.nl/automath/

[21] Per Martin-Löf (ghi lại bởi Giovanni Sambin), "Intuitionistic type theory".

[22] Jacob Neumann, "Three for One: Logic Interpretation [Intro to HoTT, No. 1, Part 3]". [Online]. Available at: https://www.youtube.com/watch?v=2zcEP2Ny63s

[23] [Online]. Available at: https://wiki.portal.chalmers.se/agda/pmwiki.php

[24] [Online]. Available at: https://www.fstar-lang.org/

[25] [Online]. Available at: https://lean-lang.org/

[26] Nhiều tác giả, *Homotopy Type Theory: Univalent Foundations of Mathematics*. Available at: https://homotopytypetheory.org/book/

[27] David Corfield, *Modal Homotopy Type Theory*.

[28] Các tác giả trên trang wiki nLab, "modal homotopy type theory". [Online]. Available at: https://ncatlab.org/nlab/show/modal+homotopy+type+theory

[29] [Online]. Available at: https://www.idris-lang.org/

[30] Edwin Brady, "Idris 2: Quantitative Type Theory in Practice". Available at: https://arxiv.org/abs/2104.00480

[31] "Type theory". [Online]. Available at: https://en.wikipedia.org/wiki/Type_theory

[32] nbro, "What are affine types in Rust?". [Online]. Available at: https://users.rust-lang.org/t/what-are-affine-types-in-rust/23755

[33] "About seL4". [Online]. Available at: https://sel4.systems/About/home.pml

[34] Roope Kaivola và các cộng sự, "Replacing Testing with Formal Verification in Intel CoreTM i7 Processor Execution Engine Validation.".

A. Phụ lục 1: Về cái tên "lý thuyết hình thái"

Như đã trình bày, **lý thuyết hình thái** ở Việt Nam thậm chí chưa có một tên gọi thống nhất (chưa kể đến các khái niệm chưa được định nghĩa). Người viết gợi ý sử dụng tên **lý thuyết hình thái** trong ngữ cảnh chung nhất (hay **lý thuyết kiểu** nếu biểu diễn trong lĩnh vực khoa học máy tính) vì lý do dưới đây:

– **Hình thái** là đối tượng khởi xướng từ **triết học toán học** của Russell, còn **kiểu** theo định nghĩa chung gắn với **kiểu dữ liệu** của các ngôn ngữ lập trình. Việc lý thuyết hình thái mở rộng ra các phạm trù ngoài triết học và toán–tin cũng là lí do để sử dụng từ **"hình thái"** có tính phổ quá hơn. Ngoài ra, *hình thái* dịch từ *type*, với hi vọng tránh được trùng lặp từ ngữ khi nghiên cứu và chuyển ngữ một số lĩnh vực liên quan, ví dụ như *sort* trong *many-sorted logic*.

– **Lý thuyết kiểu** có thể được dùng khi nói về lý thuyết hình thái trong khoa học máy tính, đơn giản vì **kiểu** chính là đối tượng được nhắc đến khi đề cập đến **hình thái**.

B. Phụ lục 2. Bảng thuật ngữ gợi ý

Như đã đề cập rất nhiều lần trong bài viết, các thuật ngữ liên quan đến lý thuyết hình thái chưa có định nghĩa tương ứng trong tiếng Việt. Phần này xin gợi ý một "từ điển" cơ bản như sau:

Tiếng Anh	***Tiếng Việt***
Type theory	Lý thuyết hình thái
Curry–Howard correspondence	Tương ứng Curry–Howard
Computation trinitarianism	ba góc nhìn lớn về [cách diễn giải khái niệm] tính toán
Proof assistant	Công cụ hỗ trợ chứng minh định lý
Witness	Vật chứng
Dependent type	Kiểu phụ thuộc
Calculus of constructions	Phép tính kiến thiết
Homotopy type theory	Lý thuyết hình thái đồng luân
Linear type	Kiểu tuyến tính
Type	Kiểu/Hình thái
Term	Khái niệm
Computation rule	Luật tính toán
Reduction rule	Luật rút gọn

Empty type	Kiểu rỗng
Unit type	Kiểu đơn vị
Product type	Kiểu tích
Sum type	Kiểu tổng
Type system	Hệ thống kiểu/Hệ [thống] hình thái
Mechanized mathematics	Toán học cơ giới hóa