

**ĐẠI HỌC QUỐC GIA HÀ NỘI**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**

**Trương Xuân Hiếu**

**KỸ THUẬT THỰC THI TƯỢNG TRƯNG**

**VÀ ỨNG DỤNG TRONG KIỂM CHỨNG**

**CHƯƠNG TRÌNH ĐA LUỒNG**

**Khóa luận tốt nghiệp đại học hệ chính quy**

Ngành Công nghệ thông tin

Giảng viên hướng dẫn: Tô Văn Khánh

**Hà Nội - 2023**

**LỜI CẢM ƠN**

Lời đầu tiên, em xin tỏ lòng biết ơn chân thành và sâu sắc tới TS.Tô Văn Khánh – giảng viên làm việc tại trường Đại học Công Nghệ - Đại học Quốc Gia Hà Nội, người đã hướng dẫn tận tình, động viên và giúp đỡ em trong suốt quá trình thực hiện khóa luận tốt nghiệp này.

Em cũng xin gửi cảm ơn tới các quý thầy cô Khoa Công nghệ thông tin nói riêng và trường Đại học Công nghệ - Đại học Quốc gia Hà Nội nói chung, đã dìu dắt, bảo ban em trong thời gian em vẫn còn ngồi ghế nhà trường. Những kiến thức mà các quý thầy cô tận tình truyền đạt đã giúp em có được nền tảng vững chắc trên con đường học tập và làm việc.

Cuối cùng, em xin cảm ơn những tới các anh chị và các bạn, đặc biết là tập thể lớp K64-CD đã đồng hành cùng em trong suốt những năm học qua. Mọi người không chỉ tận tình chia sẻ, giúp đỡ và đưa ra những góp ý chân thành mà còn là chỗ dựa tinh thần, san sẻ niềm vui nỗi buồn, giúp em vượt qua những khó khăn trong học tập cũng như là cuộc sống.

# **Mục lục**

[**Mục lục** 3](#_Toc130171574)

[**Lời cam đoan** 5](#_Toc130171575)

[**Tóm tắt** 6](#_Toc130171576)

[**Phụ lục** 8](#_Toc130171577)

[**1.** **Danh sách kí hiệu** 8](#_Toc130171578)

[**2.** **Danh sách bảng biểu** 8](#_Toc130171579)

[**3.** **Danh sách hình vẽ** 8](#_Toc130171580)

[**4.** **Danh sách tài liệu tham khảo** 8](#_Toc130171581)

[**Chương I:** 9](#_Toc130171582)

[**Mở đầu** 9](#_Toc130171583)

[**1.** **Đặt vấn đề** 9](#_Toc130171584)

[1.1. Kiểm thử và kiểm chứng 9](#_Toc130171585)

[1.2. Thực thi đa luồng 9](#_Toc130171586)

[1.3. Xung đột đọc / ghi và ràng buộc đọc / ghi 10](#_Toc130171587)

[**2.** **Kiểm chứng** 10](#_Toc130171588)

[2.1. Kiểm chứng chương trình 10](#_Toc130171589)

[2.2. Kiểm chứng mô hình 11](#_Toc130171590)

[2.3. Kiểm chứng chương trình đa luồng 11](#_Toc130171591)

[**3.** **Kỹ thuật thực thi tượng trưng** 11](#_Toc130171592)

[**4.** **Satifiability Modulo Theories** 11](#_Toc130171593)

[**Chương II:** 12](#_Toc130171594)

[**Cơ sở lý thuyết** 12](#_Toc130171595)

[**1.** **Phương pháp thực thi tượng trưng** 12](#_Toc130171596)

[**2.** **Phương pháp kiểm chứng mô hình Yogar-CBMC** 12](#_Toc130171597)

[2.1. Event Order Graph 12](#_Toc130171598)

[2.2. Read-Write Conflict Circle 12](#_Toc130171599)

[2.3. Read-Write Constraints 12](#_Toc130171600)

[**Chương III:** 13](#_Toc130171601)

[**Xây dựng phương pháp kiểm chứng mô hình ACIO** 13](#_Toc130171602)

[**Chương IV:** 14](#_Toc130171603)

[**Thực nghiệm đánh giá** 14](#_Toc130171604)

[**Kết luận** 15](#_Toc130171605)

# **Lời cam đoan**

Tôi xin cam đoan rằng những nghiên cứu về phương pháp kiếm chứng chương trình đa luồng được trình bày trong khóa luận này là của tôi và chưa từng được nộp như một báo cáo khóa luận tại trường Đại học Công Nghệ - Đại học quốc gia Hà Nội hoặc bất kỳ trường đại học khác. Những gì tôi viết ra không sao chép từ các tài liệu, không sử dụng các kết quả của người khác mà không trích dẫn cụ thể. Tôi xin cam đoan cốt lõi của công cụ tôi trình bày trong khoá luận là do tôi tự phát triển, không sao chép mã nguồn của người khác.

Nếu sai tôi hoàn toàn chịu trách nhiệm theo quy định của trường Đại Học Công Nghệ - Đại Học Quốc Gia Hà Nội.

Hà Nội, ngày....... tháng....... năm.............

Sinh viên

Trương Xuân Hiếu

# **Tóm tắt**

Sự phát triển bùng nổ của ngành công nghiệp phần mềm trong những năm gần đây đã kéo theo rất nhiều hệ quả. Các phần mềm càng ngày càng lớn dần, càng trở nên phức tạp, việc thực thi các phần mềm đó cũng càng ngày càng tiêu tốn nhiều thời gian.

Để giảm thiểu chi phí về thời gian, khái niệm thực thi đa luồng (multi threading) đã xuất hiện. Thực thi đa luồng giúp phần mềm có thể thực hiện nhiều tác vụ cùng một lúc, giúp giảm thiểu chi phí thời gian đi rất nhiều.

Tuy nhiên, bên cạnh đó, thực thi đa luồng lại vô cùng phức tạp trong trình tự thực thi vì các câu lệnh của chương trình chạy đan xen (interleave) vào nhau, không có một thứ tự cố định, kéo theo đó là sự gia tăng của chi phí kiểm thử. Ngoài ra, các phương pháp kiểm thử truyền thống không thể đảm bảo rằng toàn bộ các trình tự thực thi đều được kiểm thử, dẫn đến việc kiểm thử thiếu, làm kết quả kiểm thử bị sai lệch so với thực tế.

Do đó, nghiên cứu này đề xuất một phương pháp kiểm chứng chương trình đa luồng có tên là All-Conflicts-In-One (ACIO) để thay cho các phương pháp kiểm thử. Phương pháp kiểm chứng này được xây dựng dựa trên phương pháp kiểm chứng Yogar-CBMC.

Phương pháp kiểm chứng Yogar-CBMC sử dụng đồ thị để biểu diễn các hành động đọc/ghi giá trị (Event Order Graph - EOG) của chương trình đa luồng và tìm ra điểm xung đột đọc – ghi giữa các biến dùng chung của các luồng, từ đó xây dựng nên các ràng buộc. Tuy nhiên, với mỗi vòng lặp thực thi, phương pháp Yogar-CBMC chỉ có thể tìm được nhiều nhất một điểm xung đột, do đó đối với các chương trình đa luồng có nhiều điểm xung đột và các điểm xung đột là phức tạp, số vòng lặp thực thi lớn sẽ làm giảm độ hiệu quả và tin cậy của phương pháp này.

Phương pháp kiểm chứng ACIO cũng sử dụng đồ thi EOG làm cơ sở để xác minh các xung đột. Tuy nhiên thay vì tìm kiếm lần lượt các điểm xung đột, phương pháp ACIO sẽ xây dựng nên tập hợp các ràng buộc cho tất cả các hành động đọc – ghi và tìm ra toàn bộ các điểm xung đột chỉ trong một vòng lặp thực thi duy nhất. Chi tiết về phương pháp kiểm chứng ACIO sẽ được trình bày chi tiết trong tài liệu này.

# **Phụ lục**

1. **Danh sách kí hiệu**

* **AST** Abstract Syntax Tree
* **CFG** Control Flow Graph
* **EOG** Event Order Graph
* **SMT** Satifiability Modulo Theories
* **SV-COMP** Software Verification Competition
* **ACIO** All-Conflicts-In-One

1. **Danh sách bảng biểu**
2. **Danh sách hình vẽ**

1. **Danh sách tài liệu tham khảo**

|  |  |
| --- | --- |
| [1] | S.-C. 2017, "SV-COMP 2017 - 6th International Competition on Software Verification (sosy-lab.org)," 2017. [Online]. Available: https://sv-comp.sosy-lab.org/2017/results/results-verified/. |

# **Chương I:**

# **Mở đầu**

1. **Đặt vấn đề**
   1. Kiểm thử và kiểm chứng

Sự phát triển của công nghệ phần cứng kéo theo sự bùng nổ của ngành công nghiệp phần mềm khiến cho nhu cầu đảm bảo chất lượng phần mềm càng ngày càng được chú ý, đặc biệt là những phần mềm phức tạp, sử dụng kiến trúc đa luồng.

Phần lớn các quy trình đảm bảo chất lượng phần mềm hiện nay được xây dựng dựa trên phương pháp kiểm thử. Có thể nói kiểm thử là một phương pháp khá hiệu quả trong việc kiểm tra tính đúng đắn của phần mềm với các đầu vào cụ thể và phổ biến. Tuy nhiên, các đầu vào cụ thể và phổ biến chỉ chiếm một phần rất nhỏ trong toàn bộ các đầu vào khả thi, và dù rằng xác xuất xảy ra của chúng là “phổ biến” không có nghĩa là chúng luôn luôn xảy ra. Điều đó có thể dẫn đến việc bỏ sót nhiều lỗi của chương trình khi thực hiện kiểm thử.

Chính bởi sự khiếm khuyết đó của phương pháp kiểm thử, phương pháp kiểm chứng đã được ra đời nhằm kiểm tra toàn bộ các đầu vào khả thi của chương trình để đảm bảo chương trình hoàn toàn không có lỗi.

* 1. Thực thi đa luồng

Trong kiểm chứng chương trình, thực thi đa luồng là một trong những vấn đề thách thức nhất. Mặc dù trên lý thuyết, các luồng trong chương trình sẽ được thực thi một cách “song song”, tuy nhiên trong thực tế, các câu lệnh trong các luồng không được thực hiện song song mà sẽ đan xen theo rất nhiều cách khác nhau, tạo nên rất nhiều trình tự thực thi vô cùng phức tạp. Do đó, kiểm chứng bằng việc xây dựng các trình tự thực thi dựa trên cách xen kẽ các câu lệnh là không hề đơn giản, chưa kể đến độ phức tạp của chính bản thân các câu lệnh đó.

* 1. Xung đột đọc / ghi và ràng buộc đọc / ghi

Một giải pháp cho vấn đề này đó là mã hóa các tín hiệu đọc / ghi giá trị vào các ô nhớ để kiểm chứng thay vì sắp xếp xen kẽ các câu lệnh. Một chương trình, dù đơn giản hay phức tạp, dù có đa luồng hay đơn luồng thì theo góc nhìn của máy tính, chương trình đó cũng chỉ đọc giá trị từ một số ô nhớ, thao tác với giá trị đọc được rồi ghi lại vào một số ô nhớ nào đó. Ngoài ra, việc mã hóa các tín hiệu đọc / ghi giá trị vào các ô nhớ không chỉ rõ ràng mà việc kiểm tra xung đột giữa các tín hiệu còn đơn giản hơn rất nhiều do tập tín hiệu của chúng chỉ bao gồm 2 phần tử đọc và ghi.

Cũng chính bởi sự phức tạp của chương trình đa luồng và độ hiệu quả của việc mã hóa các tín hiệu đọc / ghi trong việc kiểm tra xung đột, đã có rất nhiều những nghiên cứu về kiểm chứng chương trình đa luồng áp dụng tín hiệu đọc / ghi được thực hiện. Một trong số đó là công cụ kiểm chứng Yogar-CBMC, sử dụng kiến trúc hình cây EOG. Công cụ cho kết quả tốt trên bộ thực nghiệm của cuộc thi SV-COMP 2017 [1]. Tuy nhiên, với chương trình có một số lượng lớn các điểm xung đột đọc / ghi phức tạp, phương pháp Yogar-CBMC cho hiệu quả không cao do phải thực thi nhiều vòng lặp. Để giải quyết vấn đề này, tôi xin đề xuất phương pháp kết hợp EOG và kỹ thuật thực thi tượng trưng để sinh ràng buộc cho toàn bộ các tín hiệu đọc / ghi. Kỹ thuật thực thi tượng trưng sẽ tạo ra AST và CFG. Sau đó, dựa trên CFG, các nút trên EOG được tạo ra và từ đó sinh ra các ràng buộc đọc / ghi để tránh việc xung đột.

1. **Kiểm chứng**
   1. Kiểm chứng chương trình

Kiểm chứng chương trình (software verification) là quá trình xác minh xem chương trình hoạt động đúng như mong đợi hay không. Trong quá trình kiểm chứng, các kỹ sư phần mềm sẽ sử dụng các kỹ thuật khác nhau để tìm ra các lỗi hoặc sai sót trong chương trình, từ đó cải thiện chất lượng và độ tin cậy của chương trình.

Kiểm chứng chương trình có thể bao gồm các phương pháp kiểm thử (testing) để đảm bảo chương trình hoạt động đúng trong các trường hợp sử dụng thực tế, kiểm tra mã nguồn (code review) để tìm lỗi cú pháp hoặc thiếu sót trong logic của chương trình, hay sử dụng các công cụ phân tích tĩnh (static analysis) để phát hiện các lỗi tiềm ẩn trong mã nguồn.

Mục tiêu của kiểm chứng chương trình là đảm bảo rằng chương trình đáp ứng được yêu cầu và hoạt động đúng, đáng tin cậy và an toàn. Nó là một phần quan trọng trong quá trình phát triển phần mềm và đảm bảo rằng chương trình sẽ không gây ra sự cố hoặc tổn thất.

* 1. Kiểm chứng mô hình
  2. Kiểm chứng chương trình đa luồng

1. **Kỹ thuật thực thi tượng trưng**
2. **Satifiability Modulo Theories**

# **Chương II:**

# **Cơ sở lý thuyết**

1. **Phương pháp thực thi tượng trưng**
2. **Phương pháp kiểm chứng mô hình Yogar-CBMC**
   1. Event Order Graph
   2. Read-Write Conflict Circle
   3. Read-Write Constraints

# **Chương III:**

# **Xây dựng phương pháp kiểm chứng mô hình ACIO**

# **Chương IV:**

# **Thực nghiệm đánh giá**

# **Kết luận**