Em xin tự giới thiệu, em tên là Trương Xuân Hiếu, sinh viên khóa 64 trường Đại học Công Nghệ. Trong buổi thuyết trình ngày hôm nay, em xin được phép trình bày về đề tài nghiên cứu “Phân tích tĩnh chương trình đa luồng” của em.

Lời đầu tiên, em xin gửi lời chào và lời cảm ơn các quý thầy cô đã tham gia vào buổi thuyết trình về đề tài nghiên cứu của em ngày hôm nay. Em cũng xin gửi lời cảm ơn tới tiến sĩ Tô Văn Khánh, người đã hỗ trợ em rất nhiều trong quá trình nghiên cứu đề tài này.

Trước tiên, em xin khái quát sơ lược về bài thuyết trình của mình. Bài thuyết trình của em sẽ gồm 3 mục chính, lần lượt là kiểm chứng chương trình đa luồng, xây dựng phương pháp ACIO và thực nghiệm đánh giá. Trong mục đầu tiên, kiểm chứng chương trình đa luồng, chúng ta sẽ tìm hiểu về sự cần thiết của chương trình đa luồng và vấn đề của kiểm chứng chương trình đa luồng. Trong mục tiếp theo, chúng ta sẽ đi vào các lý thuyết và kỹ thuật được sử dụng của phương pháp kiểm chứng chương trình đa luồng ACIO. Mục cuối cùng sẽ là thực nghiệm và đánh giá kết quả kiểm chứng của công cụ được xây dựng dựa trên phương pháp ACIO.

Đến với mục đầu tiên, chúng ta sẽ trả lời cho câu hỏi “Tại sao kiểm chứng chương trình đa luồng là cần thiết?” Có thể thấy rằng, với sự bùng nổ của ngành công nghiệp phần mềm, các phần mềm càng ngày càng trở nên phức tạp, kéo theo đó là sự gia tăng chi phí về mặt thời gian trong việc thực thi. Đó là lý do mà khái niệm thực thi đa luồng ra đời. Thực thi đa luồng giúp phần mềm có thể thực hiện nhiều tác vụ cùng một lúc bằng cách phân chia công việc cho các luồng khác nhau, giúp giảm thiểu thời gian thực thi đi rất nhiều.

Lấy một ví dụ là một đoạn mã gồm 6 dòng lệnh như trên. Cho rằng mỗi dòng lệnh thực thi tiêu hao 1 đơn vị thời gian (đvtg). Khi đó thực thi đoạn mã này sẽ tiêu hao 6 đvtg. Phân chia đoạn mã thành 2 phần, mỗi phần gồm 3 dòng lệnh rồi thực thi đa luồng 2 đoạn mã mới này. Cho rằng hai đoạn mã thực thi hoàn toàn song song với nhau, việc thực thi toàn bộ 6 dòng lệnh sẽ chỉ tiêu hao 3 đvtg. Tương tự, nếu ta chia 6 dòng lệnh thành 3 phần, mỗi phần gồm 2 dòng lệnh thì việc thực thi sẽ chỉ mất 2 đvtg.

Nhưng, đánh đổi với lợi thế về mặt thời gian thực thi, chương trình đa luồng lại rất phức tạp trong việc xác định trình tự thực thi vì thứ tự thực thi tương đối giữa các dòng lệnh trong các luồng khác nhau là không cố định. Mỗi khi thứ tự thực thi tương đối giữa các dòng lệnh thay đổi cũng sẽ làm trình tự thực thi thay đổi theo. Điều đó trực tiếp đặt gánh nặng lên quy trình đảm bảo chất lượng phần mềm, nói cách khác là quy trình kiểm thử và kiểm chứng do phải kiểm tra toàn bộ các trình tự thực thi khả thi.

Lấy ví dụ là một chương trình gồm 3 luồng thread 1, thread 2 và thread 3, mỗi luồng có 2 dòng lệnh. Với điệu kiện 2 dòng lệnh trong thread 1 luôn được thực thi đầu tiên, chúng ta đã có thể xây dựng được tới 6 trình tự thực thi khả thi. Và trong thực tế khi không có bất kì điều kiện nào về dòng lệnh thực thi đầu tiên, số lượng trình tự thực thi khả thi sẽ lớn hơn 6 rất nhiều.

Để giải quyết bài toán đảm bảo chất lượng phần mềm đa luồng, em xin đề xuất phương pháp kiểm chứng tự động All-Constraints-In-One ACIO. Trong chương 2 Xây dựng phương pháp kiểm chứng, chúng ta sẽ trả lời cho câu hỏi phương pháp ACIO kiểm chứng chương trình đa luồng bằng cách nào.

Đầu tiên chúng ta sẽ đi vào tổng quan về phương pháp ACIO. Phương pháp này cung cấp một hướng giải quyết cho bài toán kiểm chứng tự động chương trình đa luồng. Nó được xây dựng dựa trên cơ sở lý thuyết thực thi tượng trưng kết hợp với đồ thị thứ tự sự kiện Event Order Graph - EOG.

Trước tiên chúng ta sẽ nói đến kỹ thuật thực thi tượng trưng. Thực thi tượng trưng gồm hai khái niệm cơ bản là AST và CFG. AST là một cấu trúc dữ liệu phân cấp biểu diễn cú pháp trừu tượng của mã nguồn trong một ngôn ngữ lập trình nhất định trong khi CFG là cấu trúc dữ liệu dạng đồ thị biểu diễn luồng điều khiển của một chương trình. Kỹ thuật thực thi tượng trưng sẽ trừu tượng hóa mã nguồn thành AST, rồi từ AST xây dựng nên CFG. Với mỗi công đoạn, chương trình sẽ dần được trừu tượng hóa từ thuần mã lập trình về thuần mã điều khiển.

Tiếp đến, chúng ta sẽ nói về EOG. EOG là cấu trúc dữ liệu dạng đồ thị biểu diễn các sự kiện đọc ghi giá trị của biến trong chương trình được xây dựng từ CFG. Về cơ bản, việc thực thi chương trình trong máy tính là việc đọc hoặc ghi giá trị trong các ô nhớ, do đó, việc biểu diễn chương trình dưới dạng EOG đảm bảo viêc mô tả đúng và đủ các sự kiện xảy ra trong quá trình thực thi chương trình.

Bước cuối cùng của kỹ thuật thực thi tượng trưng, ta phải tạo ra các biểu thức (expression), gọi là các điều kiện ràng buộc, mô tả các sự kiện trong chương trình rồi đưa vào bộ giải SMT để tính toán kết quả. Trong nghiên cứu của em, các điều kiện ràng buộc được chia ra làm 3 nhóm chính: ràng buộc độc lập trên các luồng thể hiện các khả năng có thể của các hành động ghi giá trị, ràng buộc cho các biến đọc/ghi thể hiện các khả năng có thể của các hành động đọc giá trị và ràng buộc đảm bảo thứ tự thực thi thể hiện quan hệ của các hành động đọc và ghi trên phương diện thứ tự thực thi. Các ràng buộc này lần lượt được xây dựng rồi kết hợp lại để tạo ra tập ràng buộc hoàn chỉnh.

Lấy ví dụ là một chương trình gồm 3 luồng như trên. Để phân biệt trạng thái giá trị của từng biến tại một thời điểm nhất định, ta đánh thêm chỉ số định danh cho chúng. Ràng buộc độc lập trên các luồng, gọi tắt là ràng buộc độc lập chính là các phép gán giá trị được thể hiện trên chính mã nguồn. Tiếp đến, ràng buộc cho các biến đọc-ghi, gọi tắt là ràng buộc đọc-ghi sẽ dựa trên ràng buộc độc lập để phân tách hành độc đọc và ghi rồi bắt cặp chúng lại thành các cặp hành động đọc-ghi tương ứng. Lấy ví dụ, hành động đọc y1 có thể đọc được giá trị của hành động ghi y2, y4 và y6. Lúc này, ta có 3 cặp hành động đọc – ghi tương ứng là y1-y2, y1-y4 và y1-y6. Tiếp đến, ràng buộc đảm bảo thứ tự thực thi sẽ được xây dựng dựa trên các cặp hành động đọc-ghi này. Các ràng buộc này thể hiện mối quan hệ giữa các cặp hành động đọc và ghi với nhau để đảm bảo trình tự thực thi không xảy ra xung đột. Xét cặp hành động đọc-ghi y1-y4 thể hiện rằng y4 sẽ xảy ra trước y1. Kết hợp với thứ tự thực thi được thể hiện trên mã nguồn và ràng buộc độc lập, ta thu được trình tự thực thi như sau. Từ trình tự này, ta có y3 xảy ra trước y4 và y2. Điều này có nghĩa là y3 không thể đọc được giá trị của y4 và y2, tức là cặp hành động đọc ghi y3-y4 và y3-y2 là không tồn tại.

Để chứng minh phương pháp ACIO được trình bày ở trên có khả năng kiểm chứng chương trình đa luồng, sau đây em xin trình bày về kết quả thực nghiệm của công cụ được xây dựng dựa trên phương pháp ACIO.

Về dữ liệu thực nghiệm, bộ dữ liệu thực nghiệm cho công cụ là 18 tiêu chuẩn được lựa chọn từ 1047 tiêu chuẩn trong bộ thực nghiệm của SV-COMP – 1 cuộc thi nổi tiếng về kiểm chứng chương trình đa luồng.

Về môi trường thực nghiệm, em sử dụng một máy tính có CPU AMD Ryzen 5 4600H, xung nhịp 3GHz, 8GB RAM với thời gian giới hạn cho việc kiểm chứng mỗi chương trình đơn lẻ là 900s.

Bảng trên đây là một số thông tin về kết quả thực nghiệm của công cụ với bộ 18 dữ liệu thực nghiệm, trong đó 3 trường quan trọng nhất là Kết quả của bộ giải SMT, Thời gian giải ràng buộc và Kết luận. Ngoài ra, các thông tin khác có thể được tìm thấy trong tệp báo cáo hoạt động được sinh ra bởi công cụ.

Công cụ được xây dựng bởi phương pháp ACIO đã kiểm chứng đúng 15/18 chương trình, 3/18 chương trình không tìm được đáp án trong thời gian giới hạn và 0 có chương trình nào bị kiểm chứng sai. Ngoài ra 12/15 chương trình được kiểm chứng đúng trong thời gian dưới 100ms.

Dù rằng vẫn còn một số chương trình chưa đưa ra đáp án trong thời gian giới hạn nhưng với việc không đưa ra bất kì kết quả kiểm chứng sai nào đã mở ra một triển vọng lớn trong việc áp dụng phương pháp ACIO vào việc kiểm chứng tự động chương trình đa luồng. Bằng việc tối ưu hóa thuật toán sinh ràng buộc được sử dụng trong phương pháp ACIO, không gian ràng buộc có thể được thu gọn lại, giúp việc kiểm chứng nhanh hơn và cho phép kiểm chứng các chương trình phức tạp hơn. Ngoài ra, ta có thể mở rộng khả năng ứng dụng của phương pháp ACIO bằng cách thêm vào các tính chất khác của mã nguồn như các kiểu dữ liệu (chuỗi, mảng, con trỏ...) hoặc các cấu trúc phức tạp (struct, class...)...

Trên đây là toàn bộ bài thuyết trình về đề tài nghiên cứu của em. Một lần nữa, em xin gửi lởi cảm ơn các quý thầy cô đã đến tham dự và lắng nghe bài thuyết trình của em, đồng thời là tiến sĩ Tô Văn Khánh, người đã hỗ trợ em hoàn thành đề tài nghiên cứu này. Mọi câu hỏi xin được đặt trực tiếp hoặc gửi về địa chỉ hòm thư [19020288@vnu.edu.vn](mailto:19020288@vnu.edu.vn).