ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

TRẦN NGUYÊN HƯƠNG

CẢI TIẾN PHƯƠNG PHÁP SINH DỮ LIỆU KIỂM THỬ TỰ ĐỘNG TỪ MÃ NGUỒN

LUẬN ÁN TIẾN SĨ KỸ THUẬT PHẦN MỀM

Hà Nội - 2022

Công trình được hoàn thành tại:

Trường Đại học Công nghệ, Đại học Quốc gia Hà Nội.

Người hướng dẫn khoa học: PGS.TS. Phạm Ngọc Hùng

Phản biện 1:

Phản biện 2:

Phản biện 3:

Luận án sẽ được bảo vệ tại Hội đồng chấm luận án cấp Cơ sở họp tại: Trường Đại học Công nghệ – Đại học Quốc gia Hà Nội vào hồi... ngày...tháng...năm...

Chương 1

GIỚI THIỆU

1.1. Đặt vấn đề

Ngày nay, phần mềm được ứng dụng rộng rãi trong hầu hết các lĩnh vực của khoa học kỹ thuật và trong đời sống kinh tế xã hội. Các phần mềm đặc biệt là phần mềm hệ thống, phần mềm điều khiển thường yêu cầu cao về chất lượng nên đòi hỏi quá trình kiểm thử thực hiện khắt khe và nghiêm ngặt. Kiểm thử là một quá trình rất tốn thời gian, công sức và chi phí có thể chiếm 40% – 60% tổng chi phí trong toàn bộ quá trình phát triển phần mềm. Có nhiều mức kiểm thử trong giai đoạn này, bao gồm kiểm thử đơn vị, kiểm thử tích hợp, kiểm thử hệ thống và kiểm thử chấp nhận. Trong các mức trên, kiểm thử đơn vị thường được đánh giá là mức quan trọng nhất.

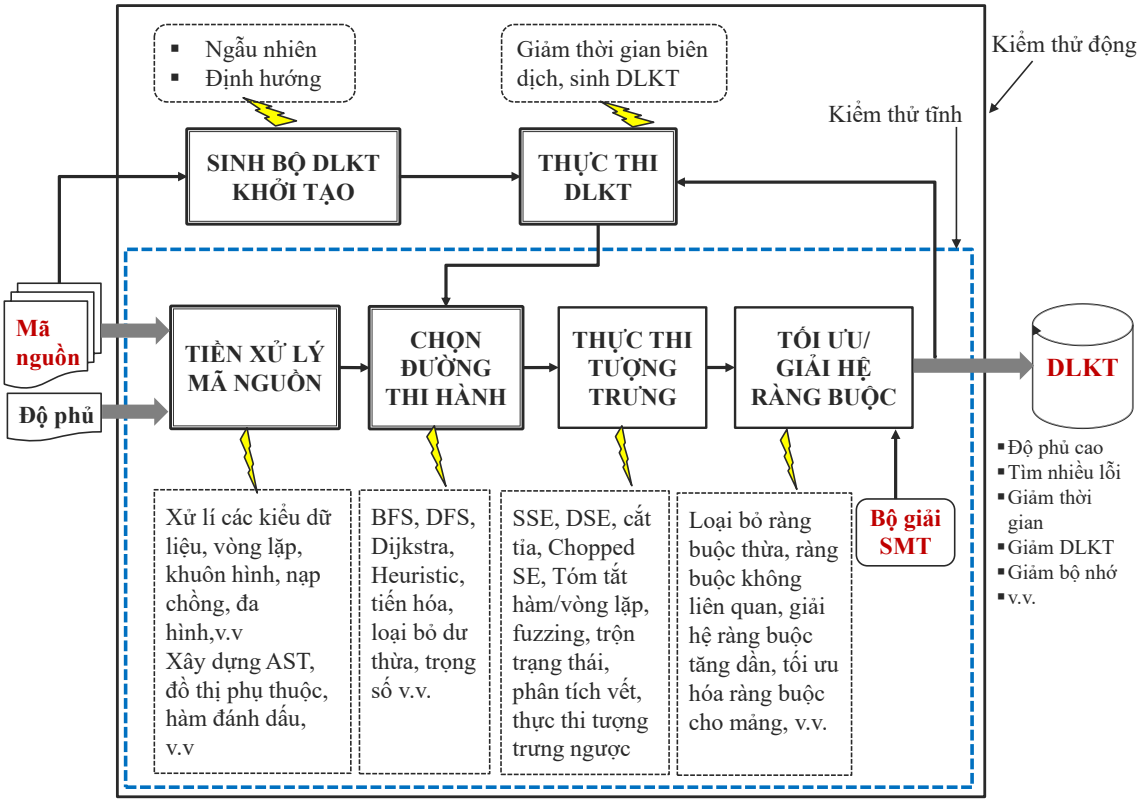
Quá trình kiểm thử hiện nay đang được tự động hóa, đây là một giải pháp hiệu quả nhằm giúp kiểm thử viên bớt nhàm chán, giảm thiểu thời gian, công sức và chi phí trong khi vẫn đảm bảo độ tin cậy cao của phần mềm. Ngoài ra, kiểm thử tự động không chỉ có ý nghĩa khi dự án không đủ tài nguyên mà còn trong kiểm thử hồi quy khi phần mềm cần sửa đổi hoặc nâng cấp.

Hiện nay, đa số quá trình tự động hóa thường tập trung vào việc thực thi ca kiểm thử mà ít quan tâm đến việc thiết kế ca kiểm thử. Về bản chất, việc phát hiện lỗi phần mềm phụ thuộc vào chất lượng các ca kiểm thử chứ không phụ thuộc vào thực thi ca kiểm thử. Do vậy, thiết kế các ca kiểm thử có chất lượng tốt là việc làm quan trọng. Thiết kế ca kiểm thử bao gồm sinh dữ liệu kiểm thử và thiết kế kết quả đầu ra mong đợi. Thực tế cho thấy, thiết kế đầu ra mong đợi rất khó tự động hóa nên người ta chỉ hướng đến sinh dữ liệu kiểm thử tự động.

Với bài toán sinh dữ liệu kiểm thử từ mã nguồn, đầu vào là mã nguồn của một dự án, ta cần phải sinh các bộ dữ liệu kiểm thử để kiểm tra tính đúng đắn của chương trình. Có hai kĩ thuật chính để sinh dữ liệu kiểm thử là kĩ thuật

kiểm thử tĩnh và kiểm thử động. Kiểm thử tĩnh là kĩ thuật tự động phân tích cú pháp của mã nguồn để sinh dữ liệu kiểm thử mà không cần chạy chương trình. Kiểm thử tĩnh thường được thực hiện sớm hơn kiểm thử động trong vòng đời phát triển phần mềm. Ngược lại, kiểm thử động được thực hiện khi mã nguồn đang ở chế độ thực thi.

Quá trình sinh dữ liệu kiểm thử tự động từ mã nguồn được mô hình hóa như Hình 1.1. Mô hình này có đầu vào là mã nguồn của dự án và tiêu chí bao phủ, đầu ra là các bộ dữ liệu kiểm thử thỏa mãn một hoặc nhiều tiêu chí về độ phủ, thời gian, khả năng phát hiện lỗi, số lượng dữ liệu kiểm thử, v.v. Quá trình gồm sáu bước chính: sinh bộ dữ liệu kiểm thử khởi tạo, tiền xử lý mã nguồn, chọn đường thi hành, thực thi tượng trưng, tối ưu hóa/giải hệ ràng buộc và thực thi dữ liệu kiểm thử. Tùy theo tiêu chí của việc sinh dữ liệu kiểm thử, các bước trong quá trình trên được chọn ra để nghiên cứu phát triển và cải tiến.



Hình 1.1: Tổng quan về sinh dữ liệu kiểm thử tự động từ mã nguồn.

Mặc dù đã có nhiều nghiên cứu cải tiến quá trình sinh dữ liệu kiểm thử tự động từ mã nguồn, ta thấy bài toán tự động sinh dữ liệu kiểm thử vẫn cần tiếp tục cải tiến để đạt hiệu suất tốt hơn, phát hiện lỗi và đạt được mục đích khác. Các cải tiến có thể thực hiện bao gồm kĩ thuật tiền xử lý mã nguồn, sinh ca kiểm thử khởi tạo, chọn đường thi hành, thực thi tượng trưng, tối ưu hóa và giải hệ ràng buộc, lựa chọn bộ giải và kết hợp bộ giải, giảm thời gian thực thi/biên

dịch dữ liệu kiểm thử. Tùy theo mục đích kiểm thử, một hoặc nhiều kĩ thuật đó sẽ được nghiên cứu để sinh dữ liệu kiểm thử tự động từ mã nguồn.

Từ các phân tích trên, luận án nghiên cứu cải tiến và đề xuất các phương pháp sinh dữ liệu kiểm thử tự động từ mã nguồn cho các dự án phần mềm và thực nghiệm trên ngôn ngữ lập trình C/C++. Mục tiêu chung của luận án là xây dựng được các phương pháp nhằm tự động hóa quá trình kiểm thử, đảm bảo tính đúng đắn, tính hiệu quả và hướng tới việc nâng cao chất lượng phần mềm. Mục tiêu cụ thể của các phương pháp nhằm làm tăng khả năng tiền xử lý mã nguồn, tăng độ phủ mã nguồn, tăng khả năng tìm lỗi, giảm thời gian sinh dữ liệu kiểm thử và giảm số bộ dữ liệu kiểm thử.

Đối tượng nghiên cứu của luận án là mã nguồn của các dự án C/C++ (chủ yếu là các hàm đơn vị) và các phương pháp sinh dữ liệu kiểm thử tự động từ mã nguồn (kiểm thử tĩnh và kiểm thử động định hướng). Trong khuôn khổ luận án này, mã nguồn được tiền xử lý để sinh ra đồ thị dòng điều khiển. Từ đồ thị dòng điều khiển, sẽ tiến hành các kĩ thuật tìm đường thi hành, thực thi tượng trưng, giải hệ ràng buộc và sinh dữ liệu kiểm thử.

Để đạt được mục tiêu nghiên cứu, các phương pháp sinh dữ liệu kiểm thử tự động từ mã nguồn sẽ được tiến hành khảo sát. Từ đó, các khoảng trống nghiên cứu được xác định làm cơ sở để đề xuất các phương pháp mới hoặc cải tiến các phương pháp hiện có. Các công cụ hỗ trợ thực nghiệm cũng được xây dựng để minh chứng cho tính đúng đắn và tính hiệu quả của các đề xuất và cải tiến.

1.2. Các đóng góp chính của luận án

Luận án đã đạt được ba kết quả chính: (i) sinh dữ liệu kiểm thử dựa trên đồ thị dòng điều khiển có trọng số và dựa trên phân tích giá trị biên, (ii) cải tiến kĩ thuật sinh dữ liệu kiểm thử ngẫu nhiên có định hướng, (iii) sinh dữ liệu kiểm thử bằng phương pháp giả lập đơn vị mã nguồn. Chi tiết các đóng góp của luận án như sau:

Thứ nhất, luận án đề xuất phương pháp sinh dữ liệu kiểm thử sử dụng đồ thị dòng điều khiển có trọng số. Theo phương pháp này, sau khi khởi tạo trọng số cho đồ thị dòng điều khiển, đường thi hành có tổng trọng số cao nhất sẽ được chọn để sinh dữ liệu kiểm thử. Nếu hệ ràng buộc sinh ra từ đường thi hành có nghiệm thì đường thi hành đó được đánh dấu là đã được thăm và trọng số trên các cạnh của đường thi hành đó sẽ được tăng thêm một đơn vị. Phương pháp này có thời gian sinh dữ liệu kiểm thử nhanh và có khả năng tìm được mã nguồn chết. Tiếp theo, để tăng cường khả năng sinh dữ liệu kiểm thử tại biên, phương pháp sinh dữ liệu kiểm thử tại biên có tên là BVTG và phương pháp cải tiến sinh dữ liệu kiểm thử tại biên có tên là IBVTG được đề xuất. Với phương pháp

BVTG, các điều kiện đơn được chọn ra tại các điểm quyết định của đồ thị dòng điều khiển được chuẩn hóa và đưa vào bộ giải hệ ràng buộc SMT. Phương pháp này sinh được các bộ dữ liệu kiểm thử có khả năng phát hiện lỗi tại biên, tuy nhiên cần nhiều thời gian do phải sử dụng bộ giải. Phương pháp IBVTG là cải tiến của phương pháp BVTG bằng cách sinh trực tiếp dữ liệu kiểm thử tại các điều kiện đơn mà không dùng bộ giải SMT. Do vậy, thời gian sinh dữ liệu kiểm thử giảm đi rất nhiều. Cuối cùng, để có một phương pháp vừa có khả năng sinh dữ liệu kiểm thử đảm bảo độ phủ, vừa có khả năng phát hiện lỗi tại biên và giảm thời gian sinh dữ liệu kiểm thử, luận án đề xuất phương pháp tích hợp WCFT và IBVTG có tên là Hybrid.

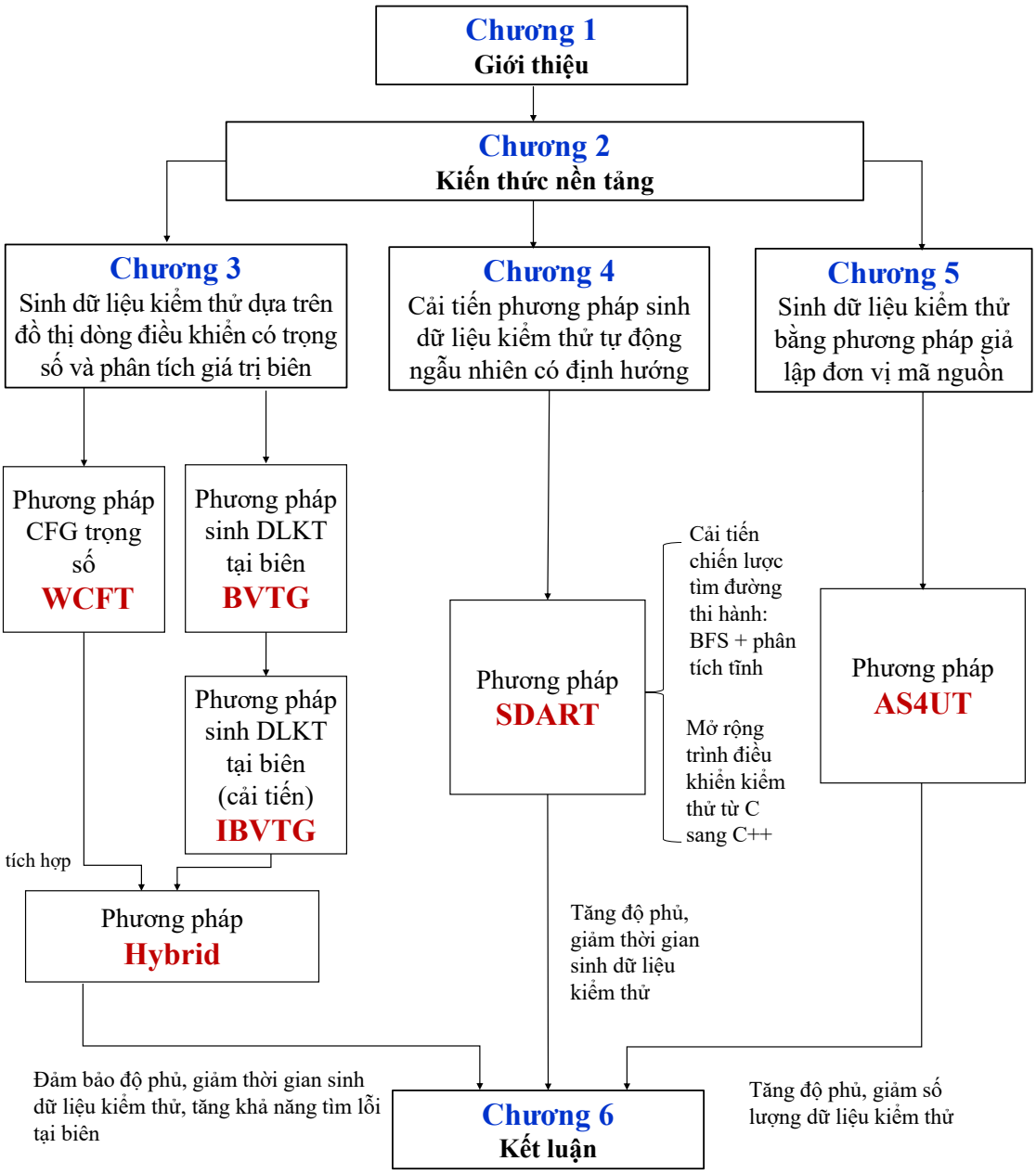
Thứ hai, luận án đề xuất cải tiến kĩ thuật tìm kiếm theo chiều rộng được đề xuất trong DART bằng cách kết hợp tìm kiếm theo chiều rộng với phân tích tĩnh. Cải tiến này nhằm cải thiện quá trình sinh dữ liệu kiểm thử khởi tạo, sinh các bộ dữ liệu kiểm thử đạt độ phủ cao và số lượng dữ liệu kiểm thử sinh ra ít. Khi áp dụng kĩ thuật tìm kiếm theo chiều rộng, sau một vài lần lặp mà không làm tăng độ phủ mã nguồn thì phương pháp phân tích tĩnh được thay thế. Tiếp theo, để giảm thời gian sinh dữ liệu kiểm thử, một trình điều khiển kiểm thử tổng quát được đề xuất. Trình biên dịch này chỉ thực hiện biên dịch một lần và có thể thực thi được rất nhiều bộ dữ liệu kiểm thử. Trình biên dịch được mở rộng để xử lý nhiều kiểu dữ liệu khác nhau của C++ thay vì chỉ xử lý được các kiểu dữ liệu của C như trong DART. Công cụ được cài đặt và thực thi trên một số mã nguồn tiêu biểu đã cho thấy độ phủ mã nguồn, số lần gọi đến bộ giải, số bộ dữ liệu kiểm thử có ý nghĩa, v.v, được cải thiện đáng kể.

Thứ ba, luận án đề xuất phương pháp sinh dữ liệu kiểm thử bằng phương pháp giả lập đơn vị mã nguồn. Phương pháp này được thực hiện ở bước tiền xử lý đồ thị dòng điều khiển. Theo đó, mỗi nút là lời gọi hàm trên đồ thị dòng điều khiển sẽ được thay thế bằng một biến giả lập tương ứng. Các đường thi hành sinh ra từ đồ thị dòng điều khiển sẽ được thực thi tượng trưng, sinh hệ ràng buộc và đưa vào bộ giải để sinh dữ liệu kiểm thử. Kết quả thực nghiệm cho thấy các bộ dữ liệu kiểm thử sinh ra đạt độ phủ mã nguồn cao hơn, số bộ dữ liệu kiểm thử sinh ra ít hơn phương pháp kiểm thử động định hướng.

1.3. Cấu trúc của luận án

Luận án gồm sáu chương, được bố cục như Hình 1.1. Chương 1 giới thiệu tổng quan về bài toán nghiên cứu, các khó khăn thách thức đặt ra và những đóng góp chính của luận án. Chương 2 giới thiệu các kiến thức nền tảng sẽ được sử dụng trong các nghiên cứu của luận án. Chương 3 trình bày phương pháp sinh dữ liệu kiểm thử đảm bảo độ phủ bằng đồ thị dòng điều khiển có trọng số và sinh dữ liệu kiểm thử có khả năng phát hiện lỗi tại biên. Một số cải tiến của

phương pháp sinh dữ liệu kiểm thử ngẫu nhiên có định hướng được trình bày trong Chương 4.



Hình 1.2: Bố cục của luận án.

Chương 5 trình bày phương pháp giả lập đơn vị mã nguồn để sinh dữ liệu kiểm thử cho các hàm đơn vị có lời gọi đến một hoặc nhiều hàm đơn vị khác. Cuối cùng, tổng kết các kết quả nghiên cứu của luận án và các hướng nghiên cứu tiếp theo được trình bày trong Chương 6.

Chương 2

KIẾN THỨC NỀN TẢNG

2.1. Các chiến lược kiểm thử

Phần này trình bày về Kiểm thử hộp đen và Kiểm thử hộp trắng.

2.2. Các mức kiểm thử

Các mức kiểm thử được trình bày trong bao gồm kiểm thử đơn vị, kiểm thử tích hợp, kiểm thử hệ thống và kiểm thử chấp nhận.

2.3. Tiêu chí bao phủ

Định nghĩa 2.1 (Độ phủ câu lệnh) *Độ phủ câu lệnh là độ phủ cho biết tỉ lệ lệnh của hàm đã được thực thi trên tổng số lệnh của hàm đang xét.*

Định nghĩa 2.2 (Độ phủ nhánh) *Độ phủ nhánh cho biết tỉ lệ số nhánh trong hàm được thực thi trên tổng số nhánh của hàm.*

Định nghĩa 2.3 (Độ phủ điều kiện con) *Độ phủ điều kiện con là tỉ lệ số nhánh đúng/sai của các điều kiện con được thực thi trên tổng số nhánh đúng/sai của các điều kiện con trong hàm.*

2.4. Dữ liệu kiểm thử

Dữ liệu kiểm thử là tập các giá trị thực (thỏa mãn tiêu chí bao phủ dữ liệu đã chọn) được xác định chỉ rõ là đầu vào để thực hiện các ca kiểm thử trong quá trình kiểm thử. Trong kiểm thử hàm đơn vị, dữ liệu kiểm thử là bộ các giá trị của các tham số đầu vào của hàm. Một cách hình thức, ta trình bày dữ liệu kiểm thử như trong Định nghĩa 2.4.

Định nghĩa 2.4 (Dữ liệu kiểm thử) *Một bộ dữ liệu kiểm thử của hàm được định nghĩa* $T = \{v_1, v_2, \ldots, v_k | k >= 1\}$*, trong đó* k *là số lượng biến truyền vào hàm,* v_i *là giá trị của biến thứ* i *trong danh sách biến. Biến truyền vào hàm có thể là tham số hoặc biến ngoài. Kiểu dữ liệu của biến là kiểu cơ bản, kiểu dẫn xuất, kiểu con trỏ, hoặc kiểu mảng.*

2.5. Đồ thị dòng điều khiển

Định nghĩa 2.5 (Đồ thị dòng điều khiển - CFG) *Đồ thị dòng điều khiển CFG là đồ thị có hướng với một cặp* $G = (V, E)$*, trong đó* $V = \{v_0, v_1, .., v_n\}$ *là tập các đỉnh đại diện cho các câu lệnh,* $E = \{(v_i, v_j) | v_i, v_j \in V\}$ *là danh sách các cạnh. Mỗi cạnh* (v_i, v_j) *biểu diễn dòng điều khiển, có cạnh từ đỉnh* v_i *đến đỉnh* v_j *nếu câu lệnh tương ứng của đỉnh* v_j *có thể được tiến hành ngay sau câu lệnh tương ứng với đỉnh* v_j*.*

Trong một số kỹ thuật, đồ thị dòng điều khiển sẽ được gán trọng số để thực hiện cho mục đích tính toán. Trong Chương 3 của luận án sẽ sử dụng đồ thị này với phương pháp WCFT để lựa chọn đường thi hành.

Định nghĩa 2.6 (Đồ thị dòng điều khiển có trọng số - Weighted CFG) *Đồ thị dòng điều khiển có trọng số là đồ thị có hướng* $G = (V, E)$*, trong đó* $V = \{v_0, v_1, .., v_n\}$ *là tập các đỉnh đại diện cho các câu lệnh,* $E = \{(v_i, v_j) | v_i, v_j \in V\}$ *là danh sách các cạnh. Mỗi cạnh* (v_i, v_j) *biểu diễn dòng điều khiển và được gán một giá trị được gọi là trọng số trên cạnh đó.*

Trọng số trên cạnh (v_i, v_j) thể hiện số lần chương trình thực thi câu lệnh ứng với đỉnh v_j ngay sau câu lệnh ứng với đỉnh v_j.

Định nghĩa 2.7 (Đường thi hành) *Cho một CFG* $G = (V, E)$*, một đường thi hành là một đường thi qua các đỉnh* $\{v_0, v_1, .., v_n | (v_i, v_{i+1}) \in E\}$*, ở đây* $0 \leq i \leq n-1$*,* v_0 *và* v_n *tương ứng là đỉnh khởi đầu và đỉnh kết thúc của đồ thị CFG.*

Ràng buộc đường thi hành (Path Constraint - PC) là tập các ràng buộc thu thập được trên đường thi hành. Ràng buộc đường thi hành được trình bày trong Định nghĩa 2.8.

Định nghĩa 2.8 (Ràng buộc đường thi hành) *Cho một đường thi hành của một hàm fn, ràng buộc đường thi hành là biểu thức lôgic, được định nghĩa như sau:*

$$PC = pc_0 \wedge pc_2 \wedge ... \wedge pc_{n-1}$$

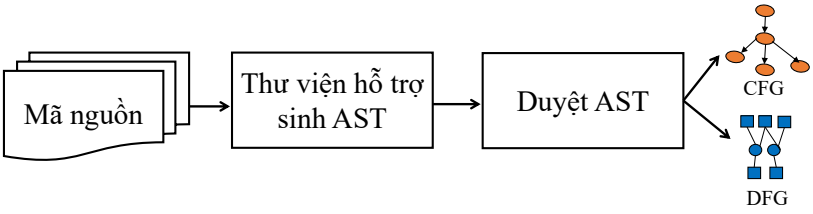
ở đây n là số điều kiện trên đường thi hành; pc_i là một ràng buộc ($0 \leq i \leq n-1$); pc_0 và pc_{n-1} là một ràng buộc đường thi hành tương ứng với điều kiện đầu tiên và điều kiện cuối cùng trong đường thi hành.

Định nghĩa 2.9 (Điều kiện đơn) *Một điều kiện đơn là biểu thức có dạng $x\ \theta\ k$, với $\theta \in \{>; \geq; <; \leq; =; \neq\}$ là các toán tử so sánh, x là một tham số đầu vào của hàm và k là một giá trị cụ thể (gọi là giá trị biên).*

Một điều kiện chứa hai hoặc nhiều tham số đầu vào được gọi là điều kiện phức.

2.6. Phân tích mã nguồn

Hình 2.1 trình bày chi tiết phương pháp phân tích và duyệt cây cú pháp trừu tượng (Abstract Syntax Tree - AST) để tạo thành CFG của đơn vị cần kiểm thử. CFG là thành phần cốt lõi cho các nội dung trong các chương tiếp theo về việc sinh dữ liệu kiểm thử một cách tự động trong việc kiểm thử đơn vị.



Hình 2.1: Tổng quan về phân tích mã nguồn

2.6.1. Cây cú pháp trừu tượng

Định nghĩa 2.10 (Cây cú pháp trừu tượng) *Cho mã nguồn P của chương trình, một cây cú pháp trừu tượng của mã nguồn P được định nghĩa là $T = (N, D)$*

với $N = \{n_1, n_2, n_3, ..., n_m\}$ *là tập hợp các nút đại diện cho các cấu trúc trong mã nguồn như: toán hạng, toán tử, biến, hằng số, mảng, con trỏ, câu lệnh điều kiện, lệnh lặp, v.v;* $D = \{(n_i, n_j) | n_i, n_j \in N\}$ *là tập hợp các cạnh. Mỗi cạnh* (n_i, n_j) *biểu diễn mỗi quan hệ của* n_i *và* n_j *hay* n_i *phụ thuộc vào* n_j*.*

2.6.2. Cây cấu trúc

Định nghĩa 2.11 (Cây cấu trúc) *Cho dự án phần mềm, cây cấu trúc tương ứng của nó, ký hiệu là S, được định nghĩa như sau:* $S = (V_k, E)$*. Trong đó k là số node của cây cấu trúc;* $V_k = \{nd_0, nd_1, ..., nd_{k-1}\}$ *là một danh sách các node;* $E = \{(nd_i, nd_j)^*\} \subset V_k \times V_k$ *là tập các cạnh (*$0 \leq i, j \leq k-1$*,* $nd_i, nd_j \in V_k$*). Với cạnh* (nd_i, nd_j) *thì node* nd_j *là một thành phần con của node* nd_i*.*

2.6.3. Mã đánh dấu

Ý tưởng chính của phương pháp đánh dấu mã là ta thêm một câu lệnh đơn giản ghi lại thông tin của câu lệnh đang xét như vị trí bắt đầu và kết thúc trong file mã nguồn, vị trí kí tự bắt đầu và vị trí kí tự kết thúc của câu lệnh, v.v.

2.7. Trình điều khiển kiểm thử

Trình điều khiển kiểm thử là một chương trình nhỏ có khả năng thực thi nhằm mục đích chạy hàm đang cần kiểm thử với một dữ liệu kiểm thử cho trước. Một trình điều khiển kiểm thử thông thường gồm bốn thành phần chính như sau: Các thành phần hỗ trợ, Thành phần thiết lập, Thành phần gọi hàm, Thành phần giải phóng

2.8. Thực thi tượng trưng

Thực thi tượng trưng là quá trình chuyển đổi các đường thi hành thành các biểu thức ràng buộc lôgic. Các biểu thức ràng buộc này được chuyển sang định dạng SMT-Lib làm đầu vào cho các bộ giải SMT. Sau khi các bộ giải SMT giải các ràng buộc đó, nếu có nghiệm, ta có thể sinh dữ liệu kiểm thử từ nghiệm được trả về này.

2.9. Lựa chọn bộ giải hệ ràng buộc

Phần này trình bày về cách chọn bộ giải trong thực thi tượng trưng

2.10. Kiểm thử giá trị biên

Phần này trình bày về cách tạo dữ liệu kiểm thử từ các giá trị biên cho các tham số đầu vào của hàm.

2.11. Kiểm thử dựa trên độ đo

Kiểm thử dựa trên độ đo là phương pháp thực thi mã nguồn sao cho bao phủ một độ đo nào đó.

2.12. Kiểm thử vòng lặp

Phần này trình bày về kiểm thử vòng lặp,lệnh lặp đơn giản, Lệnh lặp liền kề, lệnh lặp lồng nhau.

Chương 3

SINH DỮ LIỆU KIỂM THỬ DỰA TRÊN ĐỒ THỊ DÒNG ĐIỀU KHIỂN CÓ TRỌNG SỐ VÀ PHÂN TÍCH GIÁ TRỊ BIÊN

3.1. Giới thiệu

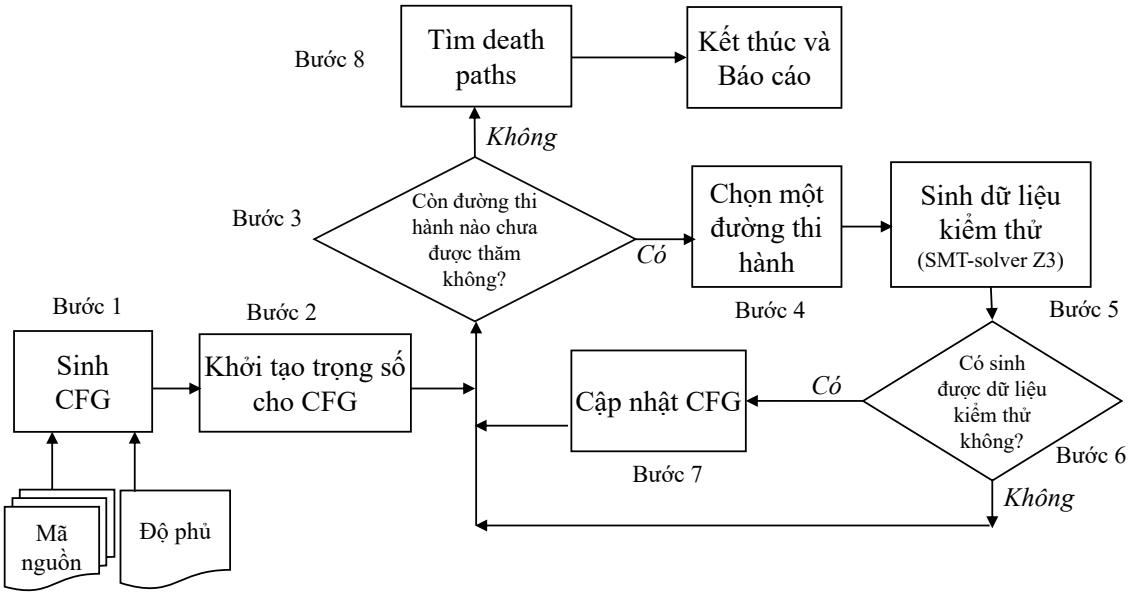
Chương này đề xuất phương pháp sinh dữ liệu kiểm thử tại các điểm biên dựa trên đồ thị dòng điều khiển (CFG) của hàm cần kiểm thử. Các phương pháp cải tiến sinh dữ liệu kiểm thử tại biên bao gồm BVTG, IBVTG. Cuối cùng phương pháp sinh dữ liệu kiểm thử kết hợp (có tên là Hybrid) để kết hợp IBVTG và WCFT. Hybrid có thể sinh dữ liệu kiểm thử với tốc độ tương đương với phương pháp WCFT đồng thời có khả năng phát hiện lỗi cao hơn. Công cụ HybridCFT4Cpp được tạo ra để đánh giá hiệu quả của phương pháp Hybrid.

3.2. Các nghiên cứu liên quan

Phần này trình bày các nghiên cứu liên quan của luận án

3.3. Sinh dữ liệu kiểm thử từ CFG có trọng số

Tổng quan về WCFT được trình bày trong Hình 3.1.



Hình 3.1: Tổng quan về phương pháp WCFT.

3.3.1. Sinh đồ thị dòng điều khiển cho hàm đơn vị

3.3.2. Sinh các đường thi hành từ đồ thị dòng điều khiển

3.3.3. Cập nhật trọng số cho đồ thị dòng điều khiển và sinh dữ liệu kiểm thử

3.3.4. Thu thập đường thi hành chết

3.4. Sinh dữ liệu kiểm thử tại biên

Phương pháp sinh ra CFG với độ phủ C3 để tách các điều kiện phức hợp thành các điều kiện đơn. Ý tưởng chính của phương pháp là: Tìm các điều kiện đơn trong CFG độ phủ C3; chuẩn hóa điều kiện đơn $x\theta k$, θ được thay bằng toán tử so sánh bằng "= =". Thay các điều kiều đơn bằng các điều kiện x==k+*step* (với *step* chạy từ *start* đến *end*) và đưa vào bộ giải SMT để tìm nghiệm testData. Thêm ngẫu nhiên các tham số đầu vào bị thiếu và testData.

Ưu điểm: Có khả năng tìm lỗi tại biên

Hạn chế: Tốn thời gian sinh dữ liệu do phải sử dụng bộ giải

Phương pháp này chỉ xử lý các kiểu dữ liệu nguyên thủy là kiểu số như short, int, long, f loat, double, v.v. Với kiểu dữ liệu bool chỉ có hai giá trị (true, false).

3.5. Sinh dữ liệu kiểm thử tại biên cải tiến

Phương pháp BVTG trình bày trong Mục 3.4 có ưu điểm là sinh được các bộ dữ liệu kiểm thử có khả năng tìm lỗi tại biên. Tuy nhiên, nó có hạn chế là phải sử dụng bộ giải để sinh dữ liệu kiểm thử, dẫn đến thời gian sinh dữ liệu kiểm thử chậm. Do đó cần phải nghiên cứu cải tiến. Phương pháp này có ý tưởng như sau:

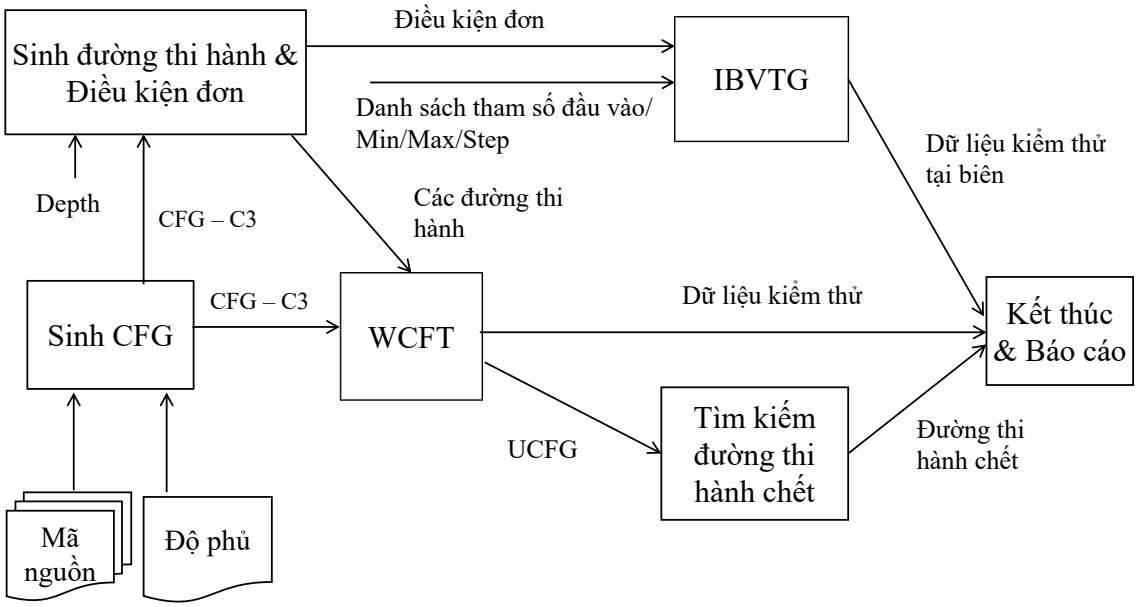
- Tìm các điều kiện đơn liên quan đến tham số đầu vào x trong CFG độ phủ C3
- Sinh trực tiếp giá trị biên tại các điều kiện đơn của tham số đầu vào x, đưa vào danh sách ValueList. $ValueList(x) = \{x_1, x_2, ...x_n\}$.
- Sắp xếp các giá trị trong ValueList(x) theo chiều tăng dần, sau đó thêm giá trị nhỏ nhất (min) và lớn nhất (max) trong miền giá trị và *ValueList*.

  VD: điều kiện đơn $x\theta k$, θ với a<=x<=b sẽ sinh ra các giá trị ValueList(x)={a, a+step, k-step, k, k+step, b-step, b}
- Tạo tập ListNorm chứa tất cả các giá trị trung bình ($norm$) của các biến đầu vào.
- Tạo test data: Lấy mỗi giá trị $norm$ của biến x_i trong ListNorm kết hợp với giá trị của các biến khác trong ValueList.

3.6. Sinh dữ liệu kiểm thử kết hợp

Để tạo thành một phương pháp hoàn chỉnh vừa có khả năng sinh dữ liệu kiểm thử đảm bảo độ phủ, vừa có khả năng phát hiện lỗi tại biên, luận án đề xuất phương pháp Hybrid. Phương pháp này là sự kết hợp phương pháp WCFT và IBVTG đã được trình bày trong các Mục 3.3 và Mục 3.5 tương ứng.

Ý tưởng của phương pháp Hybrid trích xuất ra các thông tin với cả hai phương pháp WCFT và IBVTG ở cùng một thời điểm. Khi duyệt qua CFG để tìm đường thi hành của phương pháp WCFT, Hybrid sẽ thu thập các điều kiện đơn của đường thi hành đó. Kết quả là sẽ thu được hai tập dữ liệu kiểm thử ứng với phương pháp WCFT và phương pháp IBVTG. Hai tập dữ liệu kiểm thử này được hợp nhất để có tập dữ liệu kiểm thử tạo bằng phương pháp Hybrid. Tổng quan về phương pháp Hybrid được trình bày trong Hình 3.2.



Hình 3.2: Tổng quan về phương pháp Hybrid.

3.6.1. Duyệt đồ thị dòng điều khiển

Phần này trình bày phương pháp duyệt CFG của hàm đơn vị cần kiểm thử, để chuyển nó thành một danh sách các đường thi hành và một danh sách các điều kiện đơn.

3.6.2. Phương pháp sinh dữ liệu kiểm thử Hybrid

Hybrid là phương pháp sinh dữ liệu kiểm thử mà nó tích hợp của hai phương pháp WCFT và IBVTG. Việc tích hợp nằm trong bước duyệt CFG đã cho để sinh ra một danh sách các đường thi hành và một danh sách các điều kiện đơn.

3.7. Mô phỏng hoạt động của kiểm thử kết hợp

Ví dụ này sẽ tập trung giải thích hoạt động của IBVTG và WCFT để sinh ra dữ liệu kiểm thử và thu thập mã chết .

3.8. Thực nghiệm

3.8.1. Công cụ thực nghiệm

Công cụ HybridCFT4Cpp viết bằng ngôn ngữ lập trình Java, cài đặt các phương pháp STCFG, Concolic, WCFT, BVTG, IBVTG và Hybrid.

3.8.2. So sánh giữa WCFT và STCFG

Độ phủ của hai phương pháp bằng nhau, trong khi thời gian sinh dữ liệu kiểm thử bằng WCFT ít hơn STCFG.

3.8.3. So sánh khả năng phát hiện lỗi của các phương pháp BVTG, STCFG và Concolic

BVTG tìm được nhiều lỗi hơn 2 phương pháp STCFG và Concolic.

3.8.4. So sánh giữa BVTG và IBVTG

Trong thực nghiệm, độ phủ của cả hai phương pháp là C3. Thực nghiệm so sánh hai phương pháp theo 3 tiêu chí: tổng số bộ dữ liệu kiểm thử, khả năng phát hiện lỗi, và thời gian sinh dữ liệu.

Số lượng dữ liệu kiểm thử: Trong hàm chỉ có các điều kiện đơn: số lượng dữ liệu kiểm thử do BVTG thường nhỏ hơn số lượng dữ liệu kiểm thử sinh bởi IBVTG. Trong hàm có cả điều kiện đơn và cả các điều kiện phức hợp giữa các tham số đầu vào: nếu số điều kiện đơn nhiều hơn số điều kiện phức thì số dữ liệu kiểm thử của IBVTG nhiều hơn số dữ liệu kiểm thử của BVTG. Ngược lại, nếu số điều kiện đơn ít hơn số điều kiện phức, số dữ liệu kiểm thử sinh bởi IBVG ít hơn BVTG. Hàm có các tham số đầu vào tạo ra các điểm biên trùng nhau, IBGV chỉ giữ lại một giá trị biên đại diện, do vậy số dữ liệu kiểm thử trong phương pháp IBVTG giảm đi đáng kể so với BVTG. Hàm không chứa điều kiện đơn: Các giá trị biên chỉ là các điểm nhỏ nhất (Min), lớn nhất (Max) của miền giá trị hợp lệ. Do đó số lượng dữ liệu kiểm thử sinh bởi IBVTG thường nhỏ hơn BVTG.

Khả năng phát hiện lỗi: Hàm chỉ có điều kiện đơn hoặc hàm có nhiều điều kiện đơn hơn điều kiện phức: Số lỗi phát hiện được của IBVTG thường lớn hơn

hoặc bằng số dữ liệu kiểm thử phát biện bởi BVTG. Hàm chỉ có điều kiện phức, số lỗi phát hiện được của IBVG thường rất thấp, thậm chí bằng 0.

Thời gian sinh dữ liệu kiểm thử: Thời gian sinh dữ liệu kiểm thử bằng IBVTG nhỏ hơn thời gian sinh dữ liệu kiểm thử của BVTG. Nguyên nhân là dữ liệu kiểm thử sinh bằng BVTG phải sử dụng bộ giải SMT để giải hệ ràng buộc.

3.8.5. So sánh giữa STCFG và Hybrid

Về độ phủ: Độ phủ của hai phương pháp là như nhau. Lý do là quá trình sinh dữ liệu kiểm thử của cả hai phương pháp đều dựa trên cùng một CFG và một bộ giải SMT (Z3)

Về khả năng phát hiện lỗi: Số lượng lỗi được phát hiện bởi phương pháp Hybrid thường lớn hơn hoặc bằng số lỗi tìm được của STCFG. Lý do là tập dữ liệu kiểm thử được sinh ra bằng phương pháp Hybrid bao gồm cả dữ liệu kiểm thử được tạo từ CFG tương ứng và từ các giá trị biên. Trong khi đó, STCFG chỉ tạo ra dữ liệu kiểm thử từ CFG đã cho.

Về thời gian sinh dữ liệu kiểm thử: Thời gian sinh dữ liệu kiểm thử bằng phương pháp STCFG thường lớn hơn so với Hybrid trong hầu hết các trường hợp. Lý do là phương pháp STCFG kiểm tra tất cả các đường thi hành con trong quá trình tìm kiếm đường thi hành đầy đủ. Khi kiểm tra, các ràng buộc đường thi hành sinh ra từ đường thi hành con đều đưa vào bộ giải SMT để thực hiện. Điều này làm cho phương pháp STCFG tốn thời gian để kiểm tra điều kiện; trong khi đó phương pháp WCFT chỉ đưa vào bộ giải hệ ràng buộc một lần cho một đường thi hành đầy đủ. Thêm nữa, vì phương pháp IBVTG có tốc độ nhanh hơn nhiều so với BVTG nên việc bổ sung thêm thời gian sinh dữ liệu kiểm thử tại biên IBVTG vào Hybrid cũng không làm cho Hybrid chậm hơn STCFG.

3.9. Tổng kết chương

Chương này trước hết đề xuất phương pháp sinh dữ liệu kiểm thử sử dụng đồ thị dòng điều khiển có trọng số (WCFT) và sinh dữ liệu kiểm thử tại biên (BVTG và IBVTG). Sau đó để xuất phương pháp Hybrid là sự kết hợp của phương pháp IBVTG và WCFT.

Kết quả nghiên cứu trên được công bố tại Hội nghị KSE lần thứ 12 năm 2020 và Tạp chí Khoa học "Computer Science and Communication Engineering" số của Đại học Quốc gia Hà Nội (accepted).

Chương 4

CẢI TIẾN PHƯƠNG PHÁP SINH DỮ LIỆU KIỂM THỬ TỰ ĐỘNG NGẪU NHIÊN CÓ ĐỊNH HƯỚNG

4.1. Giới thiệu

DART là một phương pháp nổi tiếng được đề xuất cho kiểm thử tự động định hướng. Tuy nhiên phương pháp này vẫn có hạn chế liên quan đến sinh dữ liệu kiểm thử ban đầu. Ngoài ra, DART hiện cung cấp cơ chế biên dịch dữ liệu kiểm thử nhanh để giảm chi phí tính toán cho việc sinh dữ liệu kiểm thử, nhưng chỉ hỗ trợ các dự án C. Do đó, nghiên cứu trong chương này đề xuất hai kĩ thuật để cải thiện các hạn chế đã đề cập. Đầu tiên, để giảm số lượng dữ liệu kiểm thử, chương này đã cải thiện chiến lược tìm kiếm theo chiều rộng được đề xuất trong DART bằng cách kết hợp chiến lược này với chiến lược sinh dữ liệu kiểm thử tĩnh. Thứ hai, nghiên cứu mở rộng ý tưởng biên dịch dữ liệu kiểm thử cho các dự án C để sử dụng với các dự án C++ bằng cách sử dụng trình điều khiển kiểm thử tổng quát. Về bản chất, ý tưởng về trình điều khiển kiểm thử C++ chung tương tự như trình điều khiển kiểm thử được sử dụng trong quá trình biên dịch dữ liệu kiểm thử được đề xuất trong DART. Tuy nhiên, trình điều khiển kiểm thử tổng quát cho ngôn ngữ C++ được trình bày tổng quát hơn bằng cách sử dụng các mẫu (template). Khi sử dụng các mẫu, trình điều khiển kiểm thử tổng quát C++ sẽ linh hoạt hơn và có thể mở rộng để hỗ trợ nhiều kiểu dữ liệu khác nhau.

4.2. Các nghiên cứu liên quan

Phần này trình bày việc sinh dữ liệu kiểm thử cho các dự án C/C++ bao gồm cải tiến trình biên dịch, kiểm thử tích hợp, thực thi tượng trưng, tối ưu hóa

ràng buộc kết hợp bộ giải SMT, và sinh bộ dữ liệu khởi tạo.

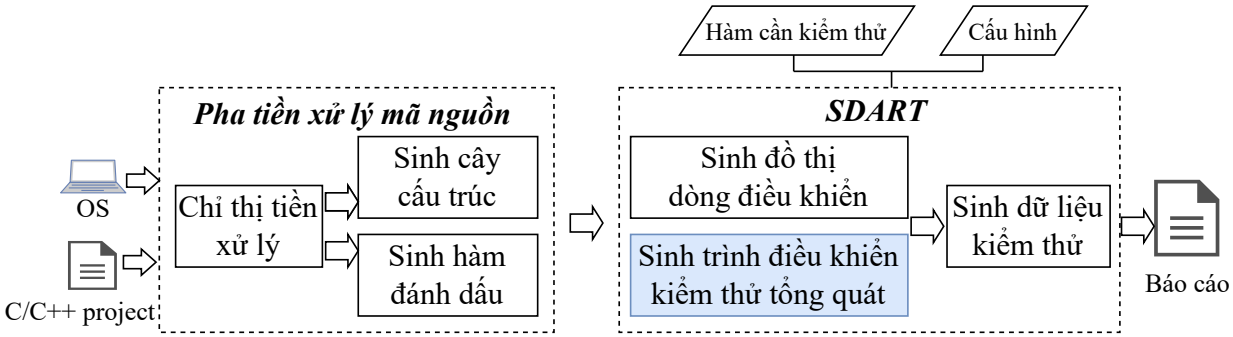
4.3. Kiểm thử tự động ngẫu nhiên có định hướng

Phần này trình bày tổng quan về phương pháp DART, phương pháp này sử dụng kỹ thuật Concolic Testing, tìm kiếm đường thi hành theo chiều rộng. Hạn chế của DART: Sinh dữ liệu kiểm thử đầu tiên: Chạy lỗi/lặp vô hạn, chỉ biên dịch các chương trình của ngôn ngữ C.

4.4. Tổng quan về phương pháp SDART

4.4.1. Tổng quan về phương pháp

Phương pháp đề xuất bao gồm hai giai đoạn chính là *giai đoạn tiền xử lý mã nguồn* và *giai đoạn sinh dữ liệu kiểm thử (SDART)* (Hình 4.1).



Hình 4.1: Tổng quan về phương pháp SDART.

4.4.2. Sinh cây cấu trúc

Giai đoạn đầu tiên của pha tiền xử lý mã nguồn là xây dựng một cây cấu trúc biểu diễn dự án C++ đã cho.

4.4.3. Chèn câu lệnh đánh dấu vào hàm

Phần này trình bày các thuật toán tạo hàm đánh dấu.

4.5. Cải tiến phương pháp DART

Cải thiện chính của SDART là khi nhận thấy khả năng tăng độ phủ chậm, việc sinh dữ liệu kiểm thử tĩnh sẽ được chọn để thay thế. Cụ thể, giá trị ngưỡng $THRESHOLD$ được người dùng tạo ra để đánh giá mức độ tăng độ phủ. Khi dãy các dữ liệu kiểm thử sinh ra làm cho độ phủ tăng với mức độ nhỏ hơn $THRESHOLD$, SDART sẽ sinh đường thi hành bộ phận qua các nhánh ít được thăm.

4.5.1. Sinh ràng buộc đường thi hành

Phần này mô tả quá trình sinh ràng buộc đường thi hành bằng cách áp dụng kĩ thuật thực thi tượng trưng. Bao gồm quá trình thực thi tượng trưng trên một đường thi hành và quá trình viết lại câu lệnh.

4.5.1.1. Thực thi tượng trưng

Quá trình này có sử dụng mô hình bộ nhớ (M) và ánh xạ tượng trưng (S). Ta sử dụng M để lưu địa chỉ của các biến được sử dụng trong quá trình thực thi tượng trưng. Ánh xạ tượng trưng S được dùng để lưu giá trị tượng trưng của các biến.

4.5.1.2. Viết lại câu lệnh

Viết lại câu lệnh hoặc quá trình đơn giản hóa nhằm mục đích chuyển đổi một câu lệnh sang một hình thức đơn giản hơn bằng cách thay thế các biến được sử dụng trong câu lệnh này bằng các giá trị tượng trưng hoặc địa chỉ của nó.

4.5.1.3. Thảo luận

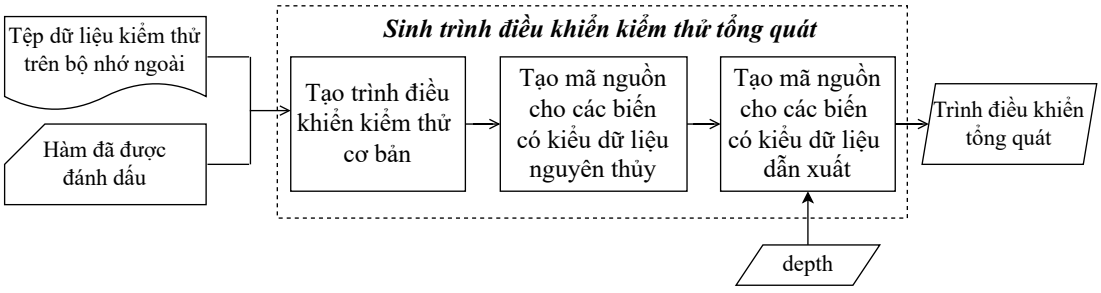
Phần này thảo luận về Phân giải tên, Sử dụng các biến không tường minh, Sinh biểu thức chuẩn SMT-Lib.

4.5.2. Sinh biểu thức chuẩn SMT-Lib

Định dạng của các ràng buộc đường thi hành (biểu thức lôgic) sinh ra từ CFG hiện không tương thích với định dạng đầu vào của bộ giải SMT. Do đó, nó cần phải đưa về dạng chuẩn đầu vào của bộ giải SMT

4.5.3. Sinh trình điều khiển tổng quát cho ngôn ngữ C++

Trình điều khiển kiểm thử được đề xuất trong DART có hạn chế là chỉ biên dịch được các kiểu dữ liệu cơ bản của ngôn ngữ C. Do vậy, luận án đề xuất mở rộng trình điều khiển để biên dịch được nhiều kiểu dữ liệu của ngôn ngữ C++ Hình 4.2 trình bày tổng quan về phương pháp thực thi dữ liệu kiểm thử cải tiến do luận án đề xuất. Đầu vào bao gồm một hàm đánh dấu và đường dẫn đến tệp lưu trữ dữ liệu kiểm thử trên bộ nhớ ngoài. Hàm cần kiểm thử có thể được định nghĩa trong class/namespace hoặc đứng độc lập. Quá trình tạo trình điều khiển kiểm thử tổng quát được mô tả như sau. Ban đầu, trình điều khiển kiểm thử cơ bản được tạo tự động có chứa một số thao tác chẳng hạn như đọc nội dung từ tệp. Trong các bước sau, trình điều khiển kiểm thử cơ bản này sẽ được chèn thêm mã nguồn để phân tích cấu trúc của dữ liệu kiểm thử. Dữ liệu này được lưu trên bộ nhớ ngoài nhằm mục đích khởi tạo dữ liệu kiểm thử.



Hình 4.2: Sinh trình điều khiển tổng quát cho ngôn ngữ C++.

4.5.4. Sinh đồ thị dòng điều khiển

Phần này trình bày thuật toán sinh đồ thị dòng điều khiển (CFG) theo các độ phủ đã cho.

4.5.5. Tính độ phủ mã nguồn

Trong quá trình thực thi dữ liệu kiểm thử, nội dung của một câu lệnh vừa được thực thi sẽ được thêm vào cuối tệp khi thực thi hàm đánh dấu fn'. Độ phủ sau đó được tính toán để xem bộ dữ liệu kiểm thử này có làm tăng độ phủ hay không. Phần này trình bày thuật toán tính độ phủ mã nguồn.

4.6. Thực nghiệm

4.6.1. Công cụ thực nghiệm

Để chứng minh cho tính hiệu quả của phương pháp SDART, công cụ ICFT4Cpp đã được xây dựng và triển khai thực nghiệm. Công cụ này được viết bằng ngôn ngữ lập trình Java. Mục tiêu của công cụ là chứng minh tính hiệu quả của SDART so với DART với cùng cài đặt cốt lõi (ví dụ cùng công cụ thực thi tượng trưng).

4.6.2. Kết quả thực nghiệm

Quá trình tạo dự án C++ để kiểm thử thực hiện như sau: Đầu tiên, các hàm được thu thập từ các trang web khác nhau, ví dụ *programmingsimplified.com*, *geekforgeek.org* và *pathcrawler-online.com*. Một hàm được chọn nếu nó thỏa mãn hai tiêu chí *(1)* phải tương thích với cài đặt và *(2)* phải liên quan đến thuật toán. Ở đây, một hàm tương thích nếu nó được hỗ trợ đầy đủ bởi công cụ thực thi tượng trưng đã chọn. Việc chọn các hàm liên quan đến thuật toán đảm bảo rằng các hàm được chọn có mức độ phức tạp cao. Quá trình này đã chọn ra 51 hàm với tổng số dòng lệnh lên tới 1000. Hầu hết các hàm được chọn đều liên quan đến các thuật toán như *BubbleSort*, *Merge*, v.v. Bước thứ hai, các hàm được chọn này được đưa vào một dự án tạo bởi IDE Dev-Cpp. IDE Dev-Cpp được chọn vì Dev-Cpp là một IDE phổ biến cho C/C++ và được sử dụng rộng rãi trong các trường đại học.

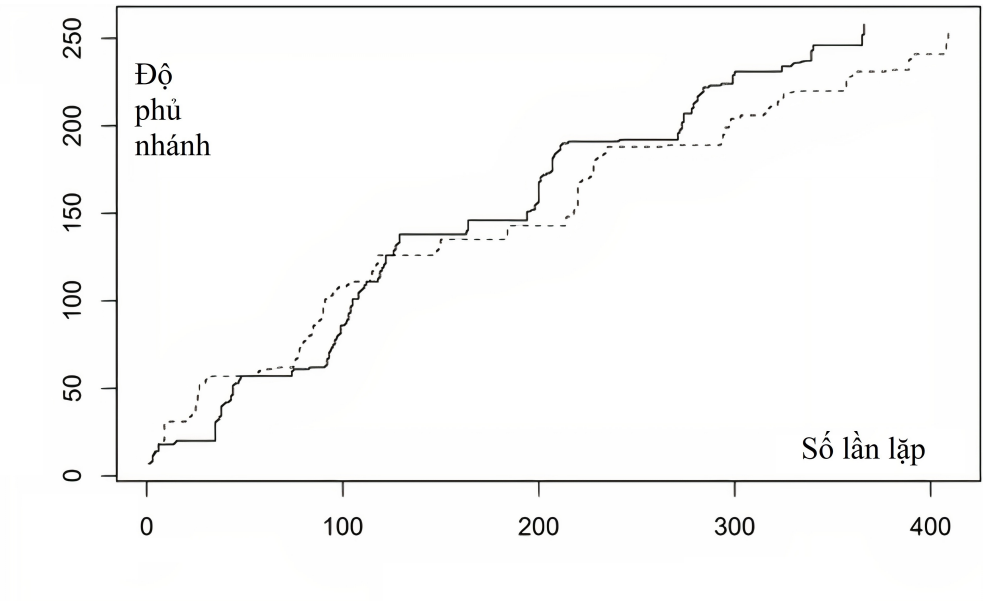
Từ danh sách các hàm thu thập được, thực nghiệm tiến hành sinh dữ liệu kiểm thử theo các kĩ thuật DART và SDART. Việc so sánh giữa DART và SDART được thực hiện theo cùng cấu hình như sau:

- **DART**. Số lần lặp tối đa (depth) của DART được khởi tạo bằng 3. Mỗi lần lặp lại thực thi 10 lần để đánh giá mức độ khách quan của phương pháp đề xuất. Do vậy, tất cả có nhiều nhất 30 lần lặp trong quá trình sinh dữ

liệu kiểm thử cho mỗi hàm. Giới hạn của các biến số nguyên là [0..30]. Giới hạn của ký tự được đặt trong phạm vi có thể quan sát được. Chiến lược lựa chọn đường thi hành là tìm kiếm theo chiều rộng.

- **SDART**. Số lần lặp lại tối đa cho mỗi hàm trong SDART bằng với của DART (30 lần lặp). Cấu hình của các biến trong SDART tương tự như DART. Ngưỡng để chuyển sang chế độ sinh dữ liệu kiểm thử tĩnh là *4*. Nghĩa là nếu tồn tại 4 bộ dữ liệu kiểm thử nghiệm ngẫu nhiên liên tục không làm tăng độ phủ, chiến lược sinh dữ liệu kiểm thử tĩnh sẽ thay thế.

Với cấu hình đã cho, có thể thấy rằng trong trường hợp xấu nhất, cả DART và SDART sẽ mất tối đa *51 hàm * 30 lần lặp = 1530 lần lặp*. Ngoài ra, trong trường hợp tốt nhất, cả hai chiến lược sẽ thực hiện ít nhất 51 lần lặp. Điều đó có nghĩa là bộ dữ liệu kiểm thử ngẫu nhiên đầu tiên sẽ đi qua tất cả các nhánh.



Hình 4.3: So sánh độ phủ nhánh giữa DART (BFS) và SDART (ngưỡng = 4)

Hình 4.3 là kết quả so sánh giữa DART (BFS) và SDART (ngưỡng = 4) với các nhánh đã thăm. DART-BFS và SDART (ngưỡng = 4) tương ứng được biểu diễn bằng đường đứt nét và đường liền nét. Từ Hình 4.3, có thể thấy rằng SDART có nhiều nhánh được thăm hơn so với DART. Mặc dù SDART không hiệu quả bằng DART trong khoảng 120 lần lặp đầu tiên, nhưng SDART dần vượt qua DART trong những lần lặp còn lại. Trong khi DART mất tổng cộng khoảng 410 lần lặp thì SDART chỉ thực hiện các hàm kiểm thử này với khoảng 370 lần lặp. Lý do chính là khi SDART phát hiện khả năng tăng độ phủ mã nguồn thấp, nó chuyển sang kĩ thuật sinh dữ liệu kiểm thử tĩnh. Với chiến lược này, SDART sẽ có nhiều nhánh được tìm thấy sớm hơn so với chiến lược lựa chọn đường thi hành hiện tại.

Bảng 4.1: Thông tin về sinh dữ liệu kiểm thử trong DART (BFS) và SDART

Tiêu chí	**SDART**	**DART (BFS)**
Số lần gọi đến bộ giải	413	1279
Số câu lệnh thực thi tượng trưng	12922	16167
Số bộ dữ liệu kiểm thử có ý nghĩa	101	102
Số lần lặp	366	409
Độ phủ nhánh	**93.48%**	92.03%
Tỷ lệ dữ liệu kiểm thử có ý nghĩa	**27.6%**	24.94%
Tổng thời gian sinh dữ liệu kiểm thử (giây)	**1586**	1779

Thông tin chi tiết về so sánh giữa DART và SDART được trình bày trong Bảng 4.1. Đầu tiên, tổng số lời gọi tới bộ giải trong DART (với 1279 lời gọi) lớn hơn đáng kể so với SDART (413 lời gọi). Lý do chính là DART tạo ra dữ liệu kiểm thử tiếp theo bằng cách phủ định các điều kiện không làm tăng độ phủ mã nguồn. Thứ hai, SDART tạo ra nhiều bộ dữ liệu kiểm thử có ý nghĩa hơn và số lần lặp ít hơn so với DART. Ở đây, bộ dữ liệu kiểm thử được coi là có ý nghĩa khi nó làm tăng độ phủ mã nguồn. Cụ thể, dữ liệu kiểm thử có ý nghĩa được tạo trong SDART chiếm 101/366 * 100% = 27,6%, trong đó 101 và 366 lần lượt là số lượng dữ liệu kiểm thử có ý nghĩa và số lần lặp. Tỷ lệ phần trăm này trong DART chỉ là 102/409 * 100% = 24,94%. Thứ ba, tổng thời gian sinh dữ liệu kiểm thử của 51 hàm bằng phương pháp SDART giảm 10,8% so với phương pháp DART.

4.7. Tổng kết chương

Nghiên cứu trình bày trong chương này đề xuất hai cải tiến để giải quyết vấn đề liên quan đến số lượng dữ liệu kiểm thử và chi phí biên dịch dữ liệu kiểm thử trong các dự án C++. Cải tiến đầu tiên có tên *SDART* nhằm mục đích tăng độ phủ mã nguồn bằng cách kết hợp chiến lược tìm kiếm theo chiều rộng của DART với phương pháp sinh dữ liệu kiểm thử tĩnh. Cải tiến thứ hai, liên quan đến chi phí tính toán của việc biên dịch dữ liệu kiểm thử, chương này đã tổng quát hóa trình điều khiển kiểm thử C++ để xử lý nhiều kiểu dữ liệu khác nhau và giảm chi phí tính toán cho việc biên dịch dữ liệu kiểm thử. Để thực hiện điều đó, phương pháp này mở rộng cách làm của DART (chỉ xử lý dự án viết bằng C) sang xử lý các dự án viết bằng C++.

Thực nghiệm cho thấy rằng SDART chỉ cần số lần lặp nhỏ hơn nhưng vẫn đạt được độ phủ nhánh cao hơn so với chiến lược tìm kiếm theo chiều rộng của DART. Kết quả nghiên cứu chính của chương này đã được công bố trên tạp chí International Journal of Software Engineering and Knowledge Engineering(ISI Indexed).

Chương 5

SINH DỮ LIỆU KIỂM THỬ BẰNG PHƯƠNG PHÁP GIẢ LẬP ĐƠN VỊ MÃ NGUỒN

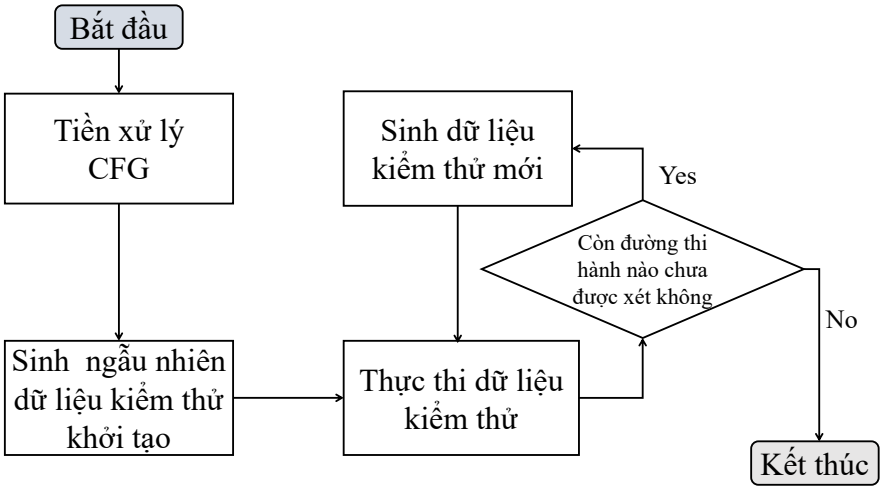
5.1. Giới thiệu

Chương này sẽ trình bày phương pháp sinh dữ liệu kiểm thử tự động khi hàm đơn vị cần kiểm thử gọi đến một hoặc nhiều hàm khác (trong lệnh điều kiện). Cách giải trước đây là bỏ qua điều kiện chứa lời gọi hàm hoặc sinh ngẫu nhiên một giá trị thay thế dẫn đến độ phủ không cao. Hàm được gọi chưa hoàn chỉnh, chưa được kiểm thử phải tạo stub thủ công.

Ý tưởng chính của phương pháp là thay mỗi lời gọi hàm bằng một biến giả lập để chỉnh sửa đồ thị dòng điều khiển; sau đó tiến hành sinh dữ liệu kiểm thử, thực thi bộ dữ liệu kiểm thử, tính toán độ phủ sử dụng CFG đã sửa đổi. Công cụ AutoStubTesting được cài đặt và tiến hành thực nghiệm để chứng minh cho tính hiệu quả của AS4UT.

5.2. Các nghiên cứu liên quan

Có nhiều nghiên cứu bao gồm các nghiên cứu Conconic testing: DART, CUTE, CREST, CAUT, v.v. Các nghiên cứu theo hướng tích hợp: Ghép cặp các tham số đầu vào, dựa trên đồ thị luồng dữ liệu; tóm tắt hàm thành biểu thức logic mệnh đề.



Hình 5.1: Tổng quan về phương pháp AS4UT

5.3. Phương pháp giả lập đơn vị mã nguồn trong kiểm thử hàm đơn vị

5.3.1. Tổng quan về phương pháp

Phương pháp AS4UT là cải tiến của phương pháp kiểm thử động định hướng cho kiểm thử hàm đơn vị trong trường hợp hàm đơn vị gọi đến hàm khác. Ý tưởng chính của phương pháp AS4UT là thay thế lời gọi hàm trong hàm đơn vị bằng một biến giả lập. Ta coi mỗi lời gọi hàm là một biến mới, từ đó thay đổi nội dung của đồ thị dòng điều khiển (hàm sang biến). Sau đó sinh đường thi hành, thực thi tượng trưng và sinh dữ liệu kiểm thử theo CFG mới. Tổng quan về phương pháp được cho như Hình 5.1. Trong hình này, AS4UT gồm 4 phần chính: Tiền xử lý CFG), Sinh ngẫu nhiên dữ liệu kiểm thử, Chạy dữ liệu kiểm thử, Sinh một dữ liệu kiểm thử mới.

5.3.2. Phương pháp AS4UT

5.3.2.1. Tiền xử lý CFG

Đây là bước đầu tiên, cũng là bước quan trọng và cốt lõi nhất của phương pháp AS4UT. Tại bước này, ta sửa đổi CFG thành CFG mới mà trên các nút của nó không có lời gọi hàm. Thay vì sử dụng các lời gọi hàm, mỗi lời gọi hàm đó được thay thế bằng một biến giả lập. Để quản lý các lời gọi hàm và kết hợp

với các biến giả lập, AS4UT tạo ra một bảng băm MAP, mà ở đó các khoá là tên của hàm được gọi và giá trị là số lần tương ứng với số lần được gọi.

5.3.2.2. Thực thi tượng trưng

5.3.2.3. Giải hệ ràng buộc đường thi hành

Sau khi thực thi tượng trưng và thu được một hệ ràng buộc giữa các biến của đường thi hành hiện tại. Các ràng buộc được viết lại theo mọt định dạng nhất định, chẳng hạn Z3, sẽ viết theo chuẩn SMT-Lib. Bộ dữ liệu kiểm thử là nghiệm của hệ ràng buộc bao gồm cả tham số đầu vào và biến giả lập.

5.4. Thực nghiệm

5.4.1. Kiến trúc của công cụ

Công cụ này cài đặt để thực nghiệm được cả hai phương pháp kiểm thử động định hướng và phương pháp giả lập đơn vị.

5.4.2. Công cụ thực nghiệm

Thực nghiệm trên dự án mã nguồn mở C-Algorithms: bao gồm ập hợp một số thuật toán (sort, compare, v.v); tập hợp một số cấu trúc dữ liệu (heap, hash map, v.v); So sánh trên 3 tiêu chí: Thời gian sinh dữ liệu kiểm thử, số lượng dữ liệu kiểm thử sinh ra và độ phủ

Kết quả thực nghiệm cho thấy phương pháp AS4UT : Độ phủ được cải thiện, số bộ dữ liệu sinh ra ít hơn, tiêu tốn nhiều thời gian hơn so với phương pháp Concolic.

5.4.3. Thảo luận

Phần này thảo luận về các kết quả: Độ phủ mã nguồn, về số bộ dữ liệu kiểm thử và về thời gian sinh dữ liệu kiểm thử.

Bảng 5.1: Kết quả thực nghiệm dự án C-Algorithms

No	File	Units	Stub	LOC	↑	Concolic Testing			AS4UT		
						Time (s)	Num	Cov (%)	Time (s)	Num	Cov (%)
1	arraylist.c	12	11	72	4	216	58	87.9	689	45	95.0
2	avl-tree.c	24	16	184	9	1,839	50	83.1	2,463	40	91.5
3	binomial_heap.c	11	10	162	6	2,832	67	74.0	3,135	48	92.6
4	binary-heap.c	6	5	64	3	106	21	86.6	374	20	96.7
5	bloom-filter.c	8	6	80	3	2,193	33	85.7	2,774	29	96.0
6	compare-int.c	2	0	14	0	29	4	88.9	48	2	88.9
7	compare-string.c	4	4	25	0	78	6	85.1	161	5	85.0
8	hash-table.c	13	8	164	6	2,545	25	61.5	2,897	23	78.1
9	list.c	20	10	183	4	1,984	33	87.6	2,654	29	94.8
10	queue.c	9	8	67	3	68	17	84.8	618	16	88.8
11	rb-tree	25	15	127	6	768	60	84.2	1,487	46	98.2
12	set.c	16	11	189	9	4,370	36	66.1	4,316	30	86.5
13	slist.c	19	8	149	3	961	21	86.8	2,938	17	89.8
14	sortarray.c	11	8	95	0	1,299	47	67.9	2,559	33	75.3
15	trie.c	14	7	190	4	1,564	31	57.1	1,992	26	61.2

5.5. Tổng kết chương

Chương này đã trình bày về phương pháp giả lập đơn vị mã nguồn nhằm hỗ trợ sinh dữ liệu kiểm thử tự động cho các thư viện và dự án được viết bằng ngôn ngữ lập trình C/C++. Ý tưởng chính của việc giả lập đơn vị mã nguồn được đề cập tới là việc chỉnh sửa đồ thị dòng điều khiển sao cho các lời gọi hàm chuyển thành các biến giả lập tương ứng. Kết quả thực nghiệm cho thấy phương pháp đề xuất có độ phủ mã nguồn cao hơn trong khi số bộ dữ liệu kiểm thử sinh ra ít hơn so với phương pháp kiểm thử động định hướng. Tuy nhiên, thời gian sinh dữ liệu kiểm thử phương pháp giả lập đơn vị nhiều hơn phương pháp kiểm thử động định hướng (có thể chấp nhận được).

Phương pháp này có tính ứng dụng cao trong công nghiệp phần mềm, đặc biệt là khi kiểm thử dự án mà mã nguồn của một số hàm đơn vị chưa hoàn thành. Phương pháp giúp việc kiểm thử có thể thực hiện song song với lập trình. Kết quả nghiên cứu trên được công bố tại Hội nghị KSE lần thứ 14 năm 2022.

Chương 6

KẾT LUẬN

6.1. Các kết quả đạt được

Các nghiên cứu trong luận án đã được tiến hành nhằm các mục đích tăng độ phủ mã nguồn, giảm thời gian sinh dữ liệu kiểm thử, giảm số dữ liệu kiểm thử, tăng khả năng phát hiện lỗi trong mã nguồn và biên dịch được nhiều kiểu dữ liệu trong các dự án C/C++. Luận án đã xây dựng các công cụ và tiến hành thực nghiệm để khẳng định hiệu quả của các phương pháp đề xuất. Cụ thể, các kết quả chính đã đạt được của luận án như sau:

Nguyên cứu thứ nhất được trình bày trong Chương 3. Chương này đề xuất phương pháp sinh dữ liệu kiểm thử sử dụng đồ thị dòng điều khiển có trọng số và sinh dữ liệu bằng cách phân tích giá trị biên. Thứ nhất, phương pháp sinh dữ liệu kiểm thử dựa vào đồ thị dòng điều khiển có trọng số (WCFT) được đề xuất. Theo phương pháp này, đường thi hành có tổng trọng số cao nhất sẽ được ưu tiên chọn trước để sinh dữ liệu kiểm thử. Kết quả thực nghiệm cho thấy rằng thời gian sinh dữ liệu kiểm thử ít hơn so với phương pháp kiểm thử động định hướng nhưng cả hai cùng đạt được độ phủ mã nguồn như nhau. Phương pháp này còn chỉ ra được các đường thi hành chết và mã nguồn chết, làm cơ sở cho tối ưu hóa mã nguồn. Thứ hai, chương này đề xuất hai phương pháp sinh dữ liệu kiểm thử tại các điểm biên (BVTG và IBVTG). Với phương pháp BVTG, các điều kiện đơn được chọn ra từ đồ thị dòng điều khiển sau đó chuẩn hóa và chuyển vào bộ giải SMT. Phương pháp này có khả năng phát hiện lỗi tại biên cao hơn so với phương pháp STCFG. Tiếp theo, phương pháp BVTG tiếp tục được cải tiến thành phương pháp IBVTG để làm tăng hiệu suất sinh dữ liệu kiểm thử. Cải tiến đó là các điều kiện đơn không đưa vào bộ giải SMT mà được sinh dữ liệu trực tiếp. Kết quả thực nghiệm cho thấy IBVTG có khả năng tìm lỗi cao hơn, thời gian sinh dữ liệu kiểm thử ít hơn so với BVTG và STCFG. Để tạo thành một phương pháp hoàn chỉnh, phương pháp WCFT và phương

pháp JBVTG được tích hợp với nhau thành phương pháp Hybrid. Phương pháp Hybrid ngoài khả năng đảm bảo độ phủ như phương pháp STCFG, nó còn tận dụng được ưu điểm của cả WCFT và IBVTG đó là có khả năng tìm lỗi tại biên và thời gian sinh dữ liệu kiểm thử nhanh hơn.

Nghiên cứu thứ hai được trình bày trong Chương 4. Chương này đã cải tiến kĩ thuật kiểm thử động có định hướng của DART. Nghiên cứu trình bày trong chương này đề xuất hai cải tiến để giải quyết vấn đề giảm số lượng dữ liệu kiểm thử, tăng độ phủ mã nguồn và chi phí biên dịch dữ liệu kiểm thử trong các dự án C++. Cải tiến đầu tiên có tên *SDART* nhằm mục đích tăng độ phủ mã nguồn, giảm số lượng dữ liệu kiểm thử bằng cách kết hợp chiến lược tìm kiếm theo chiều rộng của DART với phương pháp sinh dữ liệu kiểm thử tĩnh. Cụ thể, bất cứ khi nào mà có một số bộ dữ liệu kiểm thử mới liên tiếp không làm tăng độ phủ, việc sinh dữ liệu kiểm thử tĩnh sẽ được chọn thay thế. Danh sách chứa đường thi hành bộ phận bao gồm các nhánh chưa được thăm sẽ được chọn dựa trên phân tích đồ thị dòng điều khiển. Chi phí cho việc phân tích các đường thi hành bộ phận này trong công cụ thực thi tượng trưng được giảm xuống vì các đường thi hành này được tạo thành từ số lượng nhỏ các câu lệnh. Cải tiến thứ hai, liên quan đến chi phí biên dịch dữ liệu kiểm thử. Một trình điều khiển kiểm thử tổng quát cho ngôn ngữ C++ được đề xuất để xử lý nhiều kiểu dữ liệu khác nhau và giảm chi phí tính toán cho việc biên dịch dữ liệu kiểm thử. Để thực hiện điều đó, phương pháp này mở rộng cách làm của DART (chỉ xử lý dự án viết bằng C) sang xử lý các dự án viết bằng C++. Tất cả bộ dữ liệu kiểm thử sinh ra được lưu trữ trong một tệp trên bộ nhớ ngoài và trình điều khiển kiểm thử tổng quát C++ sẽ đọc các giá trị này để khởi tạo các tham số của hàm kiểm thử một cách linh hoạt. Thực nghiệm cho thấy rằng SDART chỉ cần ít lần lặp nhưng vẫn đạt được độ phủ nhánh cao hơn so với chiến lược tìm kiếm theo chiều rộng của DART. Hiện tại, phương pháp đề xuất đã được cài đặt trong công cụ có tên ICFT4Cpp. Công cụ này hiện đang được sử dụng trong công ty phần mềm Toshiba Vietnam và nhận được nhiều phản hồi tích cực.

Nghiên cứu thứ ba là đề xuất phương pháp sinh dữ liệu kiểm thử bằng phương pháp giả lập đơn vị mã nguồn. Nghiên cứu này được trình bày trong Chương 5. Chương này đề xuất phương pháp giả lập đơn vị mã nguồn (AS4UT) để kiểm thử các hàm đơn vị khi trong hàm có lời gọi đến một số hàm khác. Với các phương pháp kiểm thử động định hướng, khi gặp lời gọi hàm thì bộ giải thường bỏ qua hoặc sinh ngẫu nhiêm một giá trị để thay cho lời gọi hàm. Cách này làm cho nhiều câu lệnh không được thực thi, dẫn đến độ phủ mã nguồn chưa cao. Phương pháp AS4UT sẽ thực hiện ở pha tiền xử lý mã nguồn, theo đó các nút là lời gọi hàm trên đồ thị dòng điều khiển sẽ được thay thế bằng một biến giả lập. Các điều kiện liên quan đến biến giả lập này sẽ được đưa vào bộ giải để sinh dữ liệu kiểm thử. Phương pháp đề xuất trên được cài đặt trên công cụ AutoStubTesing và được thực nghiệm với dự án C-Algorithms. Kết quả là phương pháp AS4UT có độ phủ mã nguồn tăng lên nhưng số bộ dữ liệu kiểm

thử lại giảm xuống, tiêu tốn thời gian là chấp nhận được trong thực tế. Phương pháp này giúp cho giai đoạn kiểm thử có thể thực hiện đồng thời với giai đoạn lập trình khi mà mã nguồn của một số hàm đơn vị còn chưa được hoàn thiện.

Các nghiên cứu được trình bày trong luận án không những có ý nghĩa về phương diện lý thuyết mà còn góp phần làm cho phương pháp sinh dữ liệu kiểm thử tự động từ mã nguồn được áp dụng tốt hơn trong thực tiễn. Việc này không những giúp nâng cao chất lượng và độ tin cậy của phần mềm mà còn làm cho phương pháp dễ dàng được tiếp cận hơn trong cộng đồng nghiên cứu. Các phương pháp sinh dữ liệu kiểm thử này có thể dễ dàng áp dụng cho các dự án trong thực tiễn.

6.2. Hướng phát triển tiếp theo

Trong nghiên cứu thứ nhất: Mở rộng phương pháp này cho các kiểu phức tạp khác như con trỏ, cấu trúc, lớp, v.v.

Trong nghiên cứu thứ hai, phương pháp sẽ tiếp tục cải thiện với nhiều tính năng cho C++ hơn nữa để có thể sử dụng rộng rãi trong các dự án công nghiệp: sinh dữ liệu kiểm thử cho các khuôn mẫu lớp và đa hình, các hàm có chứa ngoại lệ, sinh trình điều khiển kiểm thử tổng quát C++ để hỗ trợ nhiều kiểu dữ liệu khác nhau trong C++ như vector, danh sách, v.v, cải thiện việc thực thi tượng trưng cần được cải thiện để phân tích các câu lệnh sử dụng cơ chế nạp chồng (overloading).

Trong nghiên cứu thứ ba, cần tiếp tục cải thiện kĩ thuật phân tích mã nguồn, kĩ thuật thực thi tượng trưng để giảm thời gian sinh dữ liệu kiểm thử của AS4UT đặc biệt là trong các trường hợp hàm có lời gọi đệ quy hoặc có vòng lặp.

Danh mục các công trình khoa học của tác giả liên quan đến luận án

1 . **T. N. Huong**, D. M. Kha, H.-V. Tran, and P. N. Hung. *Generate Test Data from C/C++ Source Code Using Weighted CFG and Boundary Values*. In 2020 12th Int. Conf. on Knowledge and Systems Engineering (KSE), pages 97–102, 2020.

2 . **Tran Nguyen Huong**, Do Minh Kha, Hoang-Viet Tran and Pham Ngoc Hung. *A Hybrid Method for Test Data Generation for Unit Testing of C/C++ Projects*. VNU Journal of Science, Vol.39, No.2 (2022) (accepted)

3 . D. A. Nguyen, **T. N. Huong**, H. D. Vo, and P. N. Hung. *Improvements of Directed Automated Random Testing in Test Data Generation for C++ Projects*. International Journal of Software Engineering and Knowledge Engineering, 29:1279–1312, 2019. (ISI Indexed)

4 . **Tran Nguyen Huong**, Le Huu Chung, Lam Nguyen Tung, Hoang-Viet Tran, and Pham Ngoc Hung. *An Automated Stub Method for Unit Testing C/C++ Projects*. In 2022 14th Int. Conf. on Knowledge and Systems Engineering (KSE), pages –, 2022.

Danh mục này gồm 04 công trình.