ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

VŨ THỊ ĐÀO

CÁC KỸ THUẬT SINH TỰ ĐỘNG DỮ LIỆU KIỂM THỬ DỰA TRÊN CÁC BIỂU ĐỒ UML

Chuyên ngành: Kỹ thuật Phần mềm
Mã số: 62.48.01.03

TÓM TẮT LUẬN ÁN TIẾN SĨ NGÀNH CÔNG NGHỆ THÔNG TIN

Hà Nội – 2018

Công trình được hoàn thành tại:

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

Người hướng dẫn khoa học: PGS. TS Nguyễn Việt Hà

Phản biện 1:

Phản biện 2:

Phản biện 2:

Luận án tiến sĩ sẽ được bảo vệ trước hội đồng cấp Đại học Quốc gia chấm luận án tiến sĩ họp tại……

Vào hồi 14h giờ 00 ngày 28 tháng 11 năm 2017

Có thể tìm hiểu luận án tại:

- Thư viện Quốc gia Việt Nam
- Trung tâm Thông tin – Thư viện, Đại học Quốc gia Hà Nộ

Chương 1

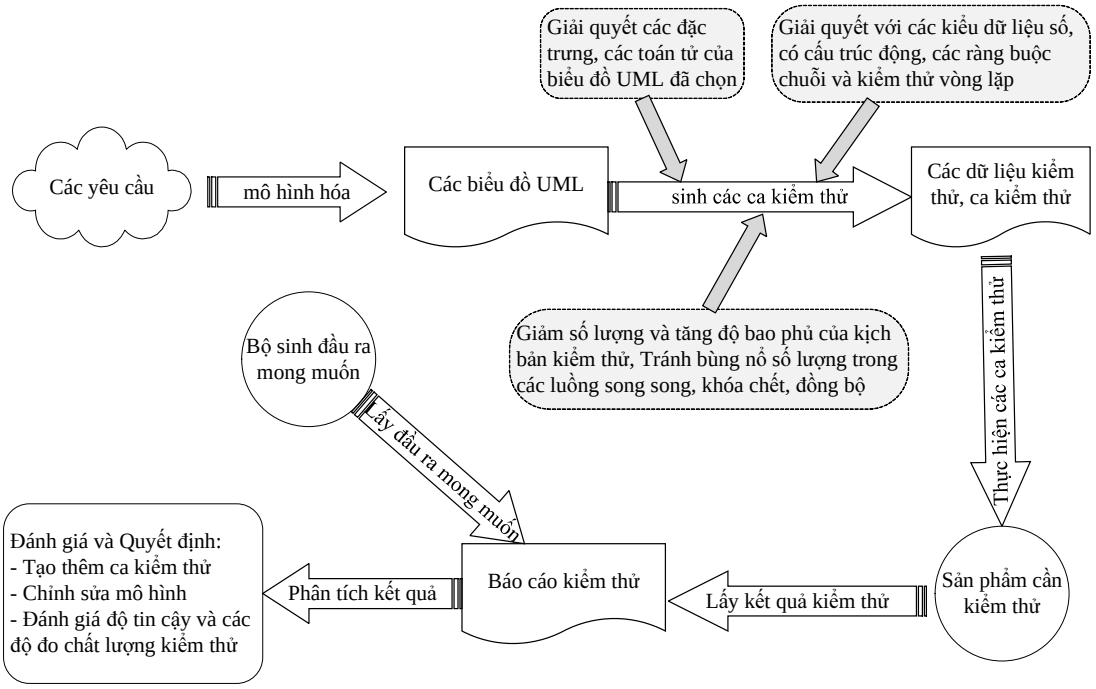
Giới thiệu

1.1 Đặt vấn đề

Trong quá trình phát triển phần mềm, kiểm thử là giai đoạn quan trọng và thực sự cần thiết để tạo ra một hệ thống phần mềm có chất lượng cao. Các công ty phần mềm cũng như các tổ chức phát triển phần mềm hướng tới giải pháp tự động hóa quá trình kiểm thử. Tuy nhiên, đa số quá trình thực hiện tự động đều tập trung vào việc thực thi kịch bản và dữ liệu kiểm thử mà không quan tâm nhiều đến việc thiết kế chúng. Hơn nữa, việc phát hiện lỗi phần mềm chủ yếu là do chất lượng của các kịch bản và dữ liệu kiểm thử được thiết kế. Vì vậy, luận án tập trung giải quyết ở giai đoạn thiết kế kiểm thử: sinh các kịch bản và dữ liệu kiểm thử từ các biểu đồ UML (Unified Modeling Language) và các ràng buộc OCL (Object Constraint Language). Luận án đề xuất các giải pháp hỗ trợ trong việc giải quyết các vấn đề của bài toán trên. Thứ nhất, luận án đề xuất một quy trình sinh dữ liệu kiểm thử từ biểu đồ tuần tự UML và các ràng buộc OCL. Biểu đồ tuần tự UML 2.0 có thể áp dụng cho tất cả mười hai toán tử, có cấu trúc phức tạp, các khối lồng ghép. Và phương pháp áp dụng cho các ràng buộc kiểu dữ liệu số và cấu trúc động. Thứ hai, luận án đề xuất phương pháp sinh dữ liệu kiểm thử tự động từ các biểu đồ tuần tự UML 2.0 và biểu đồ lớp trong trường hợp vòng lặp và các ứng dụng tương tranh, giải quyết vấn đề bùng nổ số kịch bản kiểm thử. Thứ ba, luận án đưa ra phương pháp cải tiến việc sinh dữ liệu kiểm thử tự động từ các biểu đồ tuần tự UML 2.0 và biểu đồ lớp với các ràng buộc chuỗi. Các kết quả nghiên cứu trên không phải là các kết quả rời rạc mà chúng có mối liên hệ chặt chẽ trong việc tích hợp với nhau tạo thành giải pháp cho bài toán kiểm thử dựa trên mô hình.

1.2 Phương pháp và nội dung nghiên cứu

Để thực hiện các nội dung nghiên cứu trong luận án, tác giả tiến hành theo phương pháp tiếp cận từ trên xuống, phân loại và hệ thống hóa lý thuyết các phương pháp về sinh dữ liệu kiểm thử tự động từ mô hình, từ đó phân tích, phân loại và tổng hợp các phương pháp đó. Theo cách phân loại đưa ra, luận án đi sâu phân tích mỗi đặc điểm và ưu nhược điểm của từng loại. Từ phương pháp phân tích và tổng hợp trên, với mỗi khía cạnh của kiểm thử dựa trên mô hình, luận án chọn đối tượng nghiên cứu trong luận án là việc sinh tự động dữ liệu kiểm thử từ các biểu đồ UML. Từ đó với mỗi đặc trưng của biểu đồ, luận án nghiên cứu liên quan để đề xuất cải tiến hoặc đưa ra phương pháp mới về sinh các kịch bản và dữ liệu kiểm thử, với mục đích giảm số lượng, tăng độ bao phủ và tránh bùng nổ số lượng kịch bản kiểm thử được sinh ra.



Hình 1.1: Các nội dung luận án giải quyết trong bài toán kiểm thử dựa trên mô hình.

Việc sinh các dữ liệu kiểm thử được xem xét với các biến có các kiểu dữ liệu khác nhau trong các loại ràng buộc khác nhau và trường hợp kiểm thử vòng lặp. Hình 1.1 mô tả quy trình của bài toán kiểm thử dựa trên mô hình, các vấn đề luận án tập trung giải quyết là các hình chữ nhật góc tròn nét đứt.

1.3 Cấu trúc luận án

Phần còn lại của luận án được cấu trúc như sau. Chương 2 giới thiệu kiến thức nền tảng về các vấn đề nghiên cứu trong luận án. Đây là cơ sở lý thuyết cho việc xây dựng các phương pháp sinh dữ liệu kiểm thử tự động từ các biểu đồ UML trong các chương từ Chương 3 đến Chương 5. Chương 3 trình bày nội dung kết quả nghiên cứu về Phương pháp sinh dữ liệu kiểm thử tự động giải quyết cho tất cả mười hai toán tử trong biểu đồ tuần tự UML 2.0 và kiểu dữ liệu số và cấu trúc động. Phương pháp đề xuất trong việc sinh dữ liệu kiểm thử tự động với trường hợp kiểm thử vòng lặp và trong các ứng dụng tương tranh được trình bày trong Chương 4. Chương 5 trình bày nội dung kết quả nghiên cứu về Phương pháp sinh dữ liệu kiểm thử tự động với việc cải tiến khi giải với các ràng buộc chuỗi. 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 khái niệm cơ bản

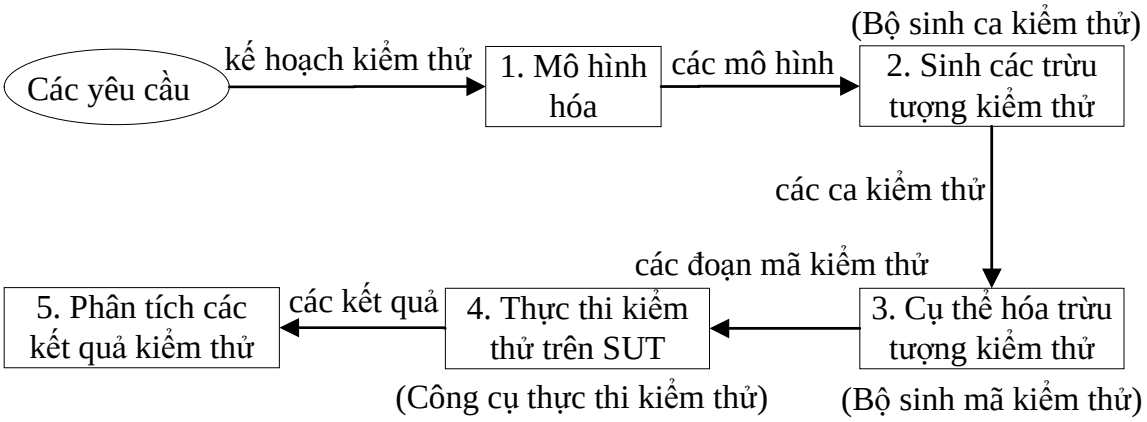
Kiểm thử phần mềm là quá trình kiểm tra đảm bảo sản phẩm phần mềm thực hiện đúng để thỏa mãn các yêu cầu của khách hàng. Mục đích của kiểm thử nhằm đánh giá chất lượng hoặc tính chấp nhận được của sản phẩm.

- Ca kiểm thử (Test case): gồm một tập các dữ liệu đầu vào, các bước thực hiện và các giá trị đầu ra mong đợi đối với phần mềm. Với từng ca kiểm thử, kết quả mong đợi được so sánh với kết quả thực tế của các kịch bản khi thực thi phần mềm.
- Dữ liệu kiểm thử (Test data): là tập các giá trị thực (thỏa mãn tiêu chuẩn 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.
- Tiêu chuẩn bao phủ dữ liệu kiểm thử (Test data coverage criteria): là một tập các quy tắc được sử dụng để xác định việc chọn các giá trị dữ liệu kiểm thử. Tiêu chuẩn này xác định độ phủ trong không gian dữ liệu đầu vào của chương trình hoặc mô hình.
- Kịch bản kiểm thử (Test scenario):là khái niệm ở mức cao những cái cần kiểm thử, và thường bao gồm nhiều ca kiểm thử liên quan nhau. Mục đích của kịch bản kiểm thử là kiểm tra việc thực hiện chức năng từ đầu đến cuối của một chức năng phần mềm và đảm bảo luồng logic đang hoạt động là đúng. Khi các kịch bản kiểm thử xác định, các trường hợp kiểm thử có thể được viết cho từng kịch bản với từng bộ dữ liệu đầu vào khác nhau. Các ca kiểm thử là các trường hợp miêu tả chi tiết ở mức thấp về các kịch bản kiểm thử.
- Tiêu chuẩn bao phủ (Coverage criteria): là một tập các quy tắc hoặc yêu cầu mà các bộ kiểm thử cần phải thoả mãn. Mục đích để đánh giá mức độ hiệu quả của các ca kiểm thử, đo phần trăm độ bao phủ của đặc tả hoặc chương trình của các ca kiểm thử so với yêu cầu phần mềm, thông qua kiểm thử để loại trừ sai sót và tăng chất lượng phần mềm.

2.2 Kiểm thử dựa trên mô hình

Kiểm thử dựa trên mô hình là một phương pháp kiểm thử mà các ca kiểm thử được sinh ra từ mô hình, đặc tả hành vi của hệ thống được kiểm thử (System Under Testing

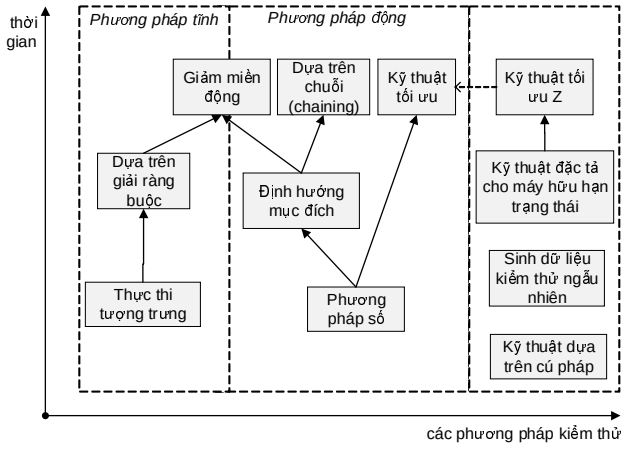
– SUT). Mô hình này được biểu diễn bằng máy hữu hạn trạng thái, ôtômát, đặc tả đại số, các biểu đồ UML, v.v. Kiểm thử dựa trên mô hình tự động hóa các thiết kế chi tiết của ca kiểm thử. Cụ thể, thay thế việc thiết kế hàng trăm ca kiểm thử thủ công thì người thiết kế kiểm thử xây dựng mô hình trừu tượng của SUT, sau đó công cụ kiểm thử dựa trên mô hình sinh ra một tập các ca kiểm thử từ mô hình đó. Toàn bộ thời gian thiết kế kiểm thử được giảm xuống, và ưu điểm là có thể phát sinh các tập các ca kiểm thử khác nhau từ cùng một mô hình bằng việc sử dụng các tiêu chuẩn bao phủ khác nhau. Hình 2.1 chỉ năm bước chính trong quy trình kiểm thử dựa trên mô hình.



Hình 2.1: Quá trình kiểm thử dựa trên mô hình.

2.3 Tổng quan về sinh dữ liệu kiểm thử tự động

Sự cần thiết trong việc sinh dữ liệu kiểm thử tự động xuất phát từ hai mục đích chính: để tăng chất lượng phần mềm và giảm chi phí phát triển. Các trường hợp kiểm thử ngoại lệ thì không có một tiêu chuẩn kiểm thử đầy đủ nào. Do đó, việc xem xét đầy đủ các tiêu chuẩn bao phủ kiểm thử cho các trường hợp cơ bản để hướng tới việc sinh kiểm thử tự động là phương pháp hiệu quả mang tính hệ thống.



Hình 2.2: Các hướng tiếp cận của Sinh dữ liệu kiểm thử tự động.

Các cách tiếp cận dựa trên cấu trúc có thể chia ra làm ba loại: phương pháp tĩnh, phương pháp động và kết hợp cả hai. Các phương pháp tiếp cận tĩnh sử dụng thực thi các tượng trưng để xác định tĩnh các đường đi và sau đó sử dụng các tiêu chuẩn khác nhau để sinh ra dữ liệu kiểm thử. Cách tiếp cận động thực thi các hệ thống cần kiểm

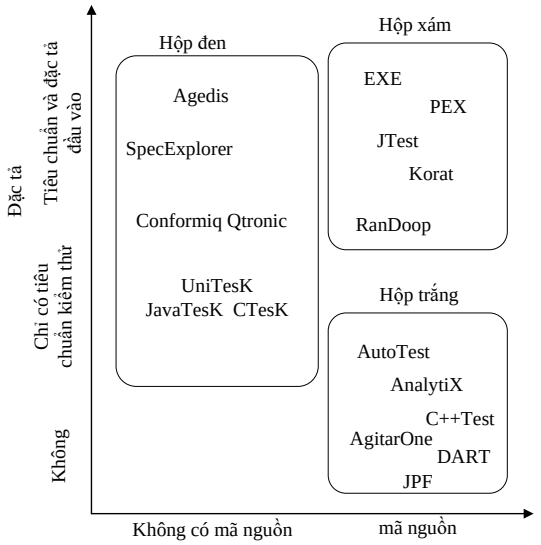
thử để tìm kiếm dữ liệu kiểm thử mong muốn. Việc sử dụng cả thực thi tượng tương và thực thi các hệ thống kiểm thử là phương pháp kết hợp cả hai. Hình 2.2 chỉ ra một cách tổng quan về các hướng tiếp cận sinh dữ liệu kiểm thử tự động. Trong đó, theo trục đứng hướng lên trên là chiều tăng dần về thời gian của các phương pháp kiểm thử được đưa ra và trục ngang là các phương pháp kiểm thử khác nhau.

Sinh dữ liệu kiểm thử từ các biểu đồ UML:

Các biểu đồ UML được sử dụng phổ biến trong thực tiễn của các công ty phát triển phần mềm. UML 2.0 chứa một tập các biểu đồ và ký hiệu được định nghĩa một cách linh hoạt và mở, biểu diễn các khía cạnh khác nhau của hệ thống bằng các biểu đồ khác nhau. Việc sử dụng các biểu đồ UML có thể kết hợp với OCL để biểu diễn các đặc tả ràng buộc mà nhiều khi biểu đồ không biểu diễn hết được. Vì vậy, kiểm thử từ các biểu đồ UML là cần thiết để chọn một tập con của các biểu đồ và phân loại ngữ nghĩa cho việc chọn đó. Mỗi công cụ kiểm thử dựa trên mô hình có cách tiếp cận khác nhau để hỗ trợ các biểu đồ khác nhau và định nghĩa một tập con có thể sử dụng trong các biểu đồ. Điều đó là cần thiết để định nghĩa cả phần dữ liệu của biểu đồ (biểu đồ lớp, biểu đồ đối tượng) và các khía cạnh hành vi động của biểu đồ. Biểu đồ tuần tự phù hợp cho việc mô tả các ca kiểm thử trừu tượng, được xem như là đầu ra của quá trình sinh kiểm thử. Thực hiện đưa ra các đường dẫn kiểm thử mong muốn thông qua một mô hình hành vi riêng biệt.

2.4 Các công cụ sinh dữ liệu kiểm thử hiện tại

Hiện nay, có rất nhiều công cụ hỗ trợ sinh các kịch bản kiểm thử tự động nói chung và dữ liệu kiểm thử nói riêng. Phần này đưa ra một cách tổng quan các công cụ có sẵn cả trong thương mại và nghiên cứu (trong Hình 2.3). Các công cụ được phân loại theo hai tiêu chí: sinh tự động từ mã nguồn và sinh tự động từ đặc tả và mô hình. Trong Hình 2.3 cũng phân biệt các công cụ không sử dụng đặc tả, sử dụng đặc tả cũng như các tiêu chuẩn kiểm thử. Các công cụ phân loại theo các kỹ thuật kiểm thử hộp trắng, kiểm thử hộp đen và kiểm thử hộp xám.



Hình 2.3: Phân loại các công cụ sinh kiểm thử tự động.

Chương 3

Sinh dữ liệu kiểm thử cho kiểu dữ liệu số và cấu trúc động

3.1 Giới thiệu

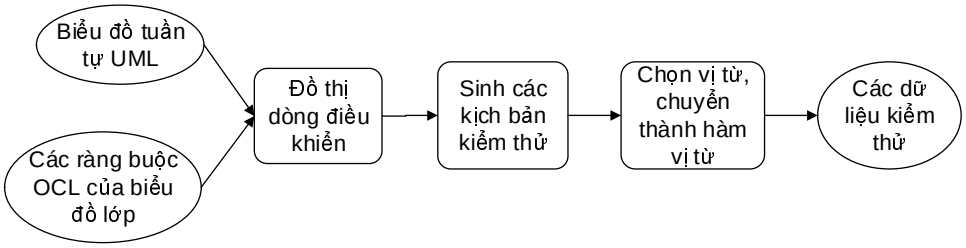
Trong hướng tiếp cận sinh các dữ liệu kiểm thử tự động từ các biểu đồ UML, bài toán đặt ra là làm thế nào sinh tự động dữ liệu kiểm thử từ các biểu đồ tuần tự UML 2.0 và các ràng buộc OCL để các kịch bản kiểm thử có độ bao phủ tốt và kiểm soát được số lượng kịch bản kiểm thử được sinh ra. Một số hướng không giải quyết triệt để tất cả các toán tử và các trường hợp lồng nhau trong các biểu đồ UML. Thông tin của mô hình trung gian được chuyển từ biểu đồ tuần tự không đầy đủ từ các biểu đồ gốc. Các phương pháp giải các ràng buộc trên các kịch bản đó cũng thường chỉ giải trong trường hợp chung, không chú ý giải trong các trường hợp đặc biệt với biến cấu trúc động (kiểu dữ liệu cấu trúc động bao gồm thành phần kiểu dữ liệu con trỏ và các thành phần kiểu cấu trúc, có thể là các kiểu dữ liệu cơ bản). Việc sinh dữ liệu kiểm thử đối với biến con trỏ từ các mô hình UML và thực thi các đoạn mã kiểm thử tự động là vấn đề mà theo hiểu biết của tác giả chưa thấy nghiên cứu nào đưa ra. Việc xem xét độ bao phủ, khả năng tìm lỗi của các kịch bản kiểm thử sinh ra cũng gặp nhiều thách thức. Vì vậy, luận án này đề xuất phương pháp sinh các dữ liệu kiểm thử tự động từ các biểu đồ tuần tự UML 2.0 và ràng buộc OCL với biến là kiểu dữ liệu số và cấu trúc động. Phương pháp đề xuất có thể áp dụng cho tất cả mười hai toán tử, các trường hợp lồng nhau trong biểu đồ tuần tự UML 2.0.

3.2 Phương pháp sinh dữ liệu kiểm thử cho biến kiểu dữ liệu số và cấu trúc động

3.2.1 Tổng quan về phương pháp đề xuất

Từ những vấn đề còn tồn tại được nêu ở trên, luận án cải tiến phương pháp sinh dữ liệu kiểm thử từ các biểu đồ UML 2.0 và ràng buộc OCL với các biến có kiểu dữ liệu số và cấu trúc động. Phương pháp được chia làm bốn bước sau:

- Bước đầu tiên chuyển đổi biểu đồ tuần tự, biểu đồ lớp thành CFG.
- Từ CFG sinh các kịch bản kiểm thử tương ứng với tiền và hậu điều kiện trong các kịch bản và tập các ràng buộc.



Hình 3.1: Các bước cơ bản sinh các dữ liệu kiểm thử.

- Trong các kịch bản kiểm thử, mỗi vị từ (từ ràng buộc) đã chọn được chuyển thành các hàm vị từ.
- Sinh dữ liệu kiểm thử trong các kịch bản kiểm thử từ các hàm vị từ đó.

Chuyển biểu đồ tuần tự thành đồ thị dòng điều khiển

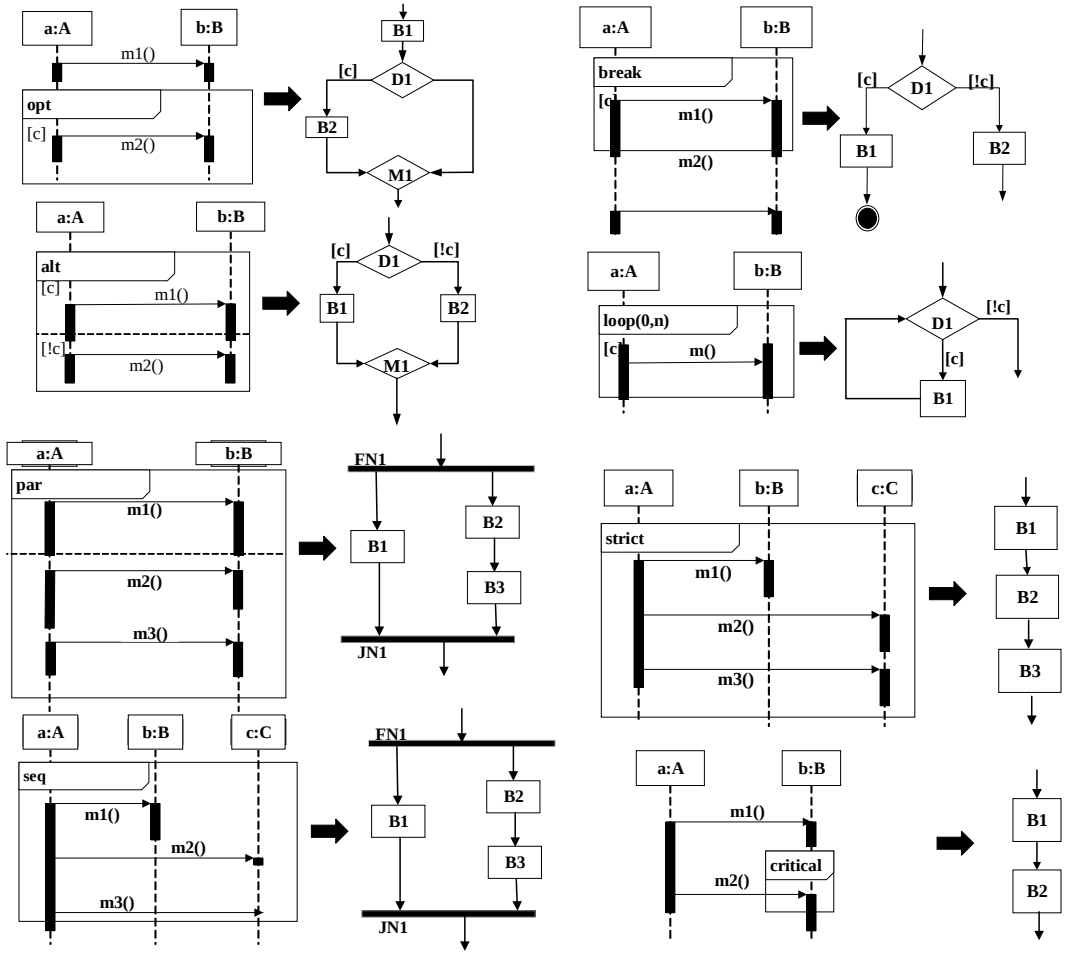
Đồ thị dòng điều khiển (Control Flow Graph – CFG) là đồ thị trung gian để sinh ra các kịch bản kiểm thử. Các thông tin trong CFG được chuyển từ biểu đồ tuần tự UML và các ràng buộc OCL tương ứng. Một CFG là một đồ thị được biểu diễn trực tiếp tương ứng với biểu đồ tuần tự đưa ra và các thông tin về ràng buộc được lấy từ biểu đồ lớp. Có năm loại nút (đỉnh) của đồ thị: block node (BN), decision node (DN), merge node (MN), fork node (FN) và join node (JN). Trong đó, BN là nút tương ứng với từng thông điệp; DN biểu diễn biểu thức điều kiện là các biểu thức logic thỏa mãn cho việc lựa chọn các toán hạng trong các toán tử; MN biểu diễn nút ra của các toán tử lựa chọn; FN biểu diễn đầu vào trong khi đó JN biểu diễn đầu ra của toán tử song song và tuần tự yếu.

Ý tưởng phương pháp sinh CFG: Đầu tiên, việc sinh cấu trúc dữ liệu của biểu đồ tuần tự tạo nên hàng đợi bao gồm: thông điệp (message), toán tử (fragment) và toán hạng (operand). Sử dụng hàm lặp với mục đích sinh ra các loại nút khác nhau từ hàng đợi. Mỗi bước lặp phân tích mỗi phần tử của hàng đợi để tạo nút ra tương ứng. Bởi vì các tham số của một thông điệp trong biểu đồ tuần tự thiếu các thông tin về ràng buộc và kiểu dữ liệu của chúng, do đó các ràng buộc này sẽ được lấy từ biểu đồ lớp và cập nhật thông tin vào từng nút tương ứng. Thuật toán phân tích các phần tử của biểu đồ tuần tự trong trường hợp các toán tử lồng nhau và đan xen của các toán tử. Thuật toán đưa ra bắt buộc chuyển biểu đồ tuần tự thành dạng file *xmi*. Tất cả các phần tử được sắp xếp theo đúng trình tự của biểu đồ tuần tự đưa vào. Hình 3.2 và Hình 3.3 là minh họa cho việc chuyển sang CFG của mười hai toán tử.

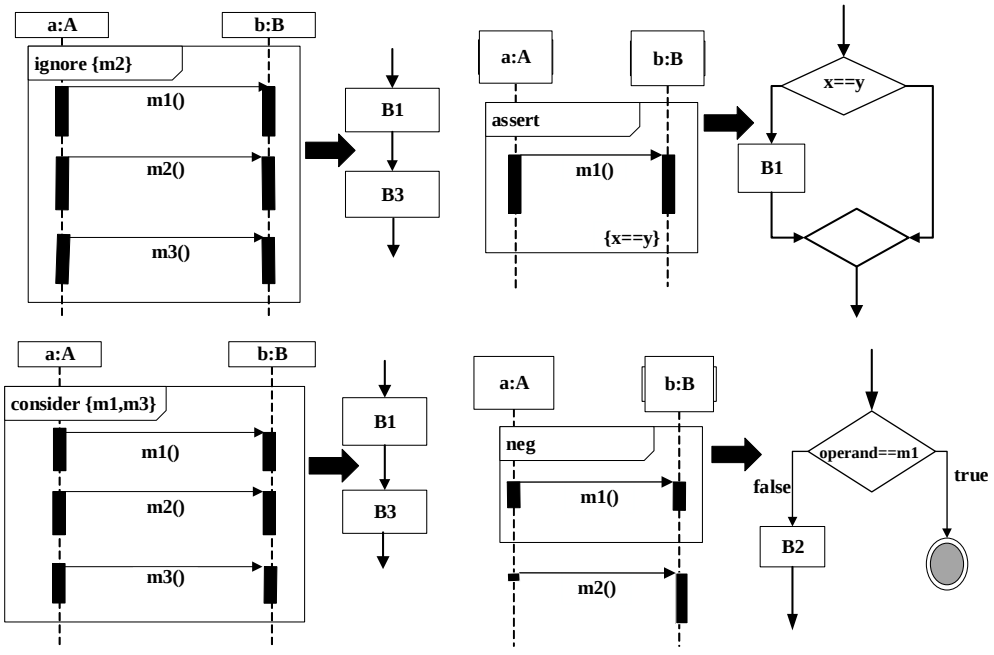
Sinh các kịch bản kiểm thử

Giả sử CFG gồm: (A, E, *in*, F). Mỗi đỉnh $A_i \in A$ là một bộ có cấu trúc như sau: <noId, noType, noDetails, outEdge>, trong đó: noId là nhãn duy nhất gắn trong từng nút; noType là một trong các loại nút *in*, BN, DN, MN, FN, JN và fn_i; $noDetails =< m_1, m_2, ..., m_q >$ với q là số các thông điệp trong $B_i \in BN$. Mỗi nút B_i biểu diễn một thông điệp m_i. Mỗi thông điệp m_i trong một đỉnh của đồ thị G đều có tiền và hậu điều kiện ($preC_i$ và $postC_i$). Thuật toán 1 miêu tả chi tiết với đầu vào bao gồm các thông tin của đồ thị G, đầu ra là tập các kịch bản kiểm thử T mà mỗi kịch bản thỏa

mãn các độ bao phủ từng thông điệp, dòng thông điệp và độ bao phủ tiền và hậu điều kiện.



Hình 3.2: Chuyển từ biểu đồ tuần tự sang CFG (của toán tử opt, alt, break, loop, parallel, weak sequencing, strict và critical region).



Hình 3.3: Chuyển từ biểu đồ tuần tự sang CFG (của toán tử ignore, consider, assert và negative).

Chọn các vị từ và chuyển thành các hàm vị từ

Tập các kịch bản kiểm thử T mà mỗi kịch bản T_j gồm tập các nút đi qua: $T_j = <n_{a1}, n_{a2}, ..., n_{an}>$. Mục đích là tìm dữ liệu kiểm thử $a_i \in D$ để kịch bản kiểm thử T_j duyệt qua các nút n_i mà thỏa mãn các ràng buộc (vị từ) trên đường dẫn kiểm thử T_j đó. Trước khi sinh dữ liệu kiểm thử, thực hiện chuyển đổi các vị từ thành các hàm vị từ F. Xem xét tập dữ liệu khởi tạo I_0 mà I_0 bao gồm tất cả các giá trị biến mà ảnh hưởng đến vị từ (gọi là vị từ q) trên luồng T_j trong CFG. Phương pháp chia hai điểm ON và OFF cho ranh giới đưa ra thỏa mãn tiêu chuẩn kiểm thử biên. Mục đích chuyển vị từ thành hàm F: để hàm phụ thuộc vào các biến (là các dữ liệu kiểm thử), phương pháp này thay đổi giá trị của các biến để tìm ra các bộ giá trị dữ liệu trên vùng biên và gần vùng biên nhất có thể (để thỏa mãn tiêu chuẩn kiểm thử biên). Nếu vị từ q có dạng: ($E1$ op $E2$), trong đó $E1$, $E2$ là các biểu thức toán học (với các phép toán +, -, *, / và mod) và op là toán tử quan hệ thì $f = (E1 - E2)$ hoặc $(E2 - E1)$ phụ thuộc vào liệu hàm F có giá trị dương khi thỏa mãn I_0.

Hàm vị từ từ các ràng buộc OCL

Các ràng buộc OCL được miêu tả trong các biểu đồ tuần tự UML, biểu đồ lớp: các tiền điều kiện và hậu điều kiện của các phương thức; các bất biến trong biểu đồ lớp. Giả sử một ràng buộc OCL được định nghĩa là một tập các mệnh đề $\{B_1, B_2, .., B_n\}$ kết hợp sử dụng các toán tử logic $\{and, or, xor, not\}$. Mỗi mệnh đề logic B_i được xác định một tập các biến $\{B_{i1}, B_{i2}, .., B_{im}\}$. Đường dẫn T được giải quyết bằng thuật toán tìm kiếm, được biểu diễn như sau: $T = \bigcup_{y=1}^{n} \bigcup_{z=1}^{m_y} \{B_{yz}\}$ trong đó, n là số các mệnh đề trong một ràng buộc và m_y là số lượng biến được bao gồm trong mệnh đề. Trong một tập các ràng buộc OCL bao gồm các mệnh đề khác nhau và giá trị trong các mệnh đề ngoài các giá trị là *true, false* thì các mệnh đề còn có thể nhận giá trị *undefined*. Các mệnh đề $\{B_i\}$ được biểu diễn: y op z trong đó, op là các toán tử: =, <>, <, ≤, >, ≥; chuyển đổi thành $t = abs(y - z)$. Hàm f được tính như sau:
if y.oclIsUndefined() or z.oclIsUndefined() **then** $f = k$
else (t=true) **then** $f = 0$;
else (t=false) **then** $f = k * nor(abs(y - z) + 1)$; trong đó, $nor(y) = y/y + 1$

Sinh dữ liệu kiểm thử từ các hàm vị từ

Giả sử tập các biến đầu vào là a_i trong một vector; hàm vị từ f đưa ra trong thuật toán tìm kiếm với giá trị a_i là điểm bắt đầu tìm kiếm. Bắt đầu từ a, hàm thực hiện đánh giá điểm a-1 và a+1 lần đầu tiên để xác định chiều. Trừ khi a là một tối ưu cục bộ sẽ thực hiện tìm kiếm mẫu, di chuyển theo hướng giảm giá trị f. Kích thước bước tăng gấp đôi theo từng bước, do đó mỗi lần di chuyển được tăng lên theo chỉ số hàm duyệt qua các điểm. Tìm kiếm mẫu dừng lại nếu điểm tiếp theo không cải tiến hàm f.

3.3 Thực nghiệm

Công cụ được xây dựng SequenceTesting sử dụng phát triển phương pháp đề xuất. Tất cả các thực nghiệm được thực thi với công cụ đó trên máy tính Intel Core i3-6100U CPU 2.30 GHz với RAM 2GB.

Thuật toán 1 Sinh các kịch bản kiểm thử từ đồ thị dòng điều khiển G

Input: Đồ thị dòng điều khiển G

Output: Tập các kịch bản kiểm thử T, mà mỗi kịch bản T_i thỏa mãn độ bao phủ mỗi thông điệp, độ bao phủ luồng thông điệp và độ bao phủ tiền và hậu điều kiện.

```
P= EmumerateAllPaths (G); // list all the paths from the starting node to ending nodes in G
for each path p_i ∈ P do
    n_j = in //current node start from in
    preC_i = FindPreCond(n_j); //preC_i is pre-condition at the n_j node of the traversed test path
    T_i ← ∅; //initialize test scenarios T_i empty
    for each node n_j of path p_i do
        e_j = (m, a, b, c) //the event e_j corresponds to the node n_j and is given the message m from the sending object
        a to the receiver b and the condition c
        if (c == ∅) then
            t = {preC, I(a_1, a_2, ..., a_n), O(d_1, d_2, ..., d_m), postC};
            preC is pre-condition of message m;
            I(a_1, a_2, ..., a_n) is set of input in message m from sender object
            O(d_1, d_2, ..., d_m) is set of output in message m from receiver object
            postC is post-condition of message m;
        end if
        if (c ≠ ∅) then
            c(v) = (c_1, c_2, ..., c_l) // set of constraints in the p_i path
            t = {preC, I(a_1, a_2, ..., a_n), O(d_1, d_2, ..., d_m), c(v)postC};
        end if
        T_i = T_i ∪ t
    end for
    T ← T ∪ T_i
end for
return (T);
```

3.3.1 Thực nghiệm và đánh giá phần sinh đồ thị trung gian

Áp dụng công cụ đã phát triển cho các biểu đồ tuần tự trong các trường hợp chỉ chứa duy nhất một phân đoạn hoặc chứa nhiều phân đoạn lồng ghép đan xen nhau, từ đó so sánh kết quả thu được với kết quả phương pháp của nhóm tác giả Nayak như Bảng 3.1. Như vậy, phương pháp đề xuất đã giải quyết được mười hai toán tử, các trường hợp lồng nhau trong biểu đồ tuần tự UML 2.0, trong khi phương pháp của Nayak chỉ giải quyết 7 toán tử.

Bảng 3.1: So sánh kết quả đề xuất và kết quả nghiên cứu của nhóm tác giả Nayak

Thứ tự	**Khối đơn**	**Phương pháp đưa ra của Nayak**	**Phương pháp đề xuất**
1	Không có khối nào	Yes	Yes
2	Alternative	Yes	Yes
3	Loop	Yes	Yes
4	Option	Yes	Yes
5	Break	Yes	Yes
6	Parallel	Yes	Yes
7	Critical	No	Yes
8	Strict	No	Yes
9	Consider	No	Yes
10	Ignore	No	Yes
11	Sequencing	No	Yes
12	Nagative	No	Yes
13	Assertion	No	Yes
14	Các khối lồng nhau	Yes	Yes

Bảng 3.2: Kết quả thực nghiệm so sánh phương pháp đề xuất với phương pháp của Trung Dinh–Trong

Loại lỗi	Số lỗi gieo	Phương pháp Trung Dinh–Trong		Phương pháp đề xuất	
		Số lỗi bởi bao phủ điều kiện	Số lỗi bởi bao phủ thông điệp	Số lỗi bởi bao phủ điều kiện	Số lỗi bởi bao phủ thông điệp
MM	5	5	5	5	5
FOA	5	4	4	4	4
SCS	2	2	2	2	2
BCS	1	0	0	1	1
FM	5	5	5	5	5
FP	5	4	4	4	5
MBA	5	5	5	5	5
FC	4	3	1	4	3
FAM	4	2	2	4	3
WIT	2	2	2	2	2
WRP	1	0	0	1	1
FOP	3	3	2	3	3
Tổng	42	35	32	40	39

3.3.2 Thực nghiệm và đánh giá phần sinh kịch bản kiểm thử tự động

Trong thực nghiệm này, tác giả sử dụng cùng một tập các chức năng và dữ liệu được dùng trong cách tiếp cận của Trung Dinh–Trong: hệ thống bán hàng online (OSHOP) và hệ thống các thành phần mô hình UML (COMP). Một tập hợp các loại lỗi được đưa ra bởi việc xác định các lỗi phổ biến trong các hệ thống: Thiếu thông điệp (MM); Thứ tự gọi các thông điệp sai (FOM); Sai thông điệp (FM); Giới hạn phạm vi của cấu trúc điều kiện (SCS); Mở rộng phạm vi của cấu trúc điều kiện (BCS); Các tham số sai (FP); Thiếu nhánh thay thế (MBA); Điều kiện sai (FC); Bản số trong quan hệ giữa các lớp sai (FAM); Cây thừa kế sai (WIT); Tham chiếu đến lớp cha sai (WRP); Các ràng buộc OCL sai (FOP). Kết quả thực nghiệm trong Bảng 3.2 chỉ ra số lượng các lỗi theo từng tiêu chuẩn.Theo phương pháp của Trung Dinh–Trong có 42 lỗi được cấy vào thì 35 lỗi (khoảng 83%) được tìm thấy bởi đầu vào kiểm thử thỏa mãn tiêu chuẩn bao phủ điều kiện. Theo tiêu chuẩn bao phủ thông điệp thì tìm được 32 lỗi. Trong khi đó theo phương pháp đề xuất thì tìm được 40 lỗi của tiêu chuẩn bao phủ điều kiện (khoảng 95%) và tiêu chuẩn bao phủ thông điệp tìm được 39 lỗi (khoảng 92%).

3.4 Kết luận

Chương này đã trình bày một phương pháp đề xuất sinh dữ liệu kiểm thử từ các biểu đồ tuần tự UML 2.0 và biểu đồ lớp với các ràng buộc số và cấu trúc động. Phương pháp áp dụng cho tất cả mười hai toán tử và các khối lồng ghép trong biểu đồ tuần tự. Các kết quả thực nghiệm chỉ ra rằng phương pháp đề xuất có thể sinh ra các kịch bản kiểm thử có độ bao phủ và có khả năng tìm lỗi tốt hơn so với một số phương pháp hiện có. Với giải pháp này, một số vấn đề khó áp dụng các phương pháp kiểm thử dựa trên mô hình do thiếu các phương pháp thiết kế kiểm thử được giải quyết. Một phần kết quả đã được chúng tôi công bố tại tại Chuyên san các công trình Nghiên cứu phát triển và ứng dụng công nghệ thông tin và truyền thông và Hội nghị quốc tế APLAS 2016. Tác giả đang nghiên cứu với các hệ thống thực và lớn hơn để chứng minh tính hiệu quả của nó. Đồng thời, tiếp tục phát triển sinh các kịch bản kiểm thử từ các loại biểu đồ UML khác hoặc kết hợp các loại biểu đồ khác để chúng có độ bao phủ tốt hơn.

Chương 4

Sinh dữ liệu kiểm thử cho vòng lặp và các ứng dụng tương tranh

4.1 Giới thiệu

Trong thực tế rất nhiều hệ thống phức tạp, quá trình sinh kịch bản kiểm thử tự động từ biểu đồ tuần tự UML 2.0 và các ràng buộc trong biểu đồ lớp là thực sự cần thiết, nhưng cách tiếp cận của quá trình sinh kiểm thử tự động đối mặt với một số vấn đề. Thứ nhất, các chương trình tương tranh có thể không đơn định khi thực hiện cho các đầu ra khác nhau trong khi với cùng một đầu vào trong các lần thực thi khác nhau. Tính tương tranh trong các biểu đồ tuần tự được miêu tả bằng toán tử song song hoặc tuần tự yếu. Do đó, quá trình sinh các kịch bản kiểm thử trong các chương trình tương tranh có những thách thức nhất định mà các chương trình tuần tự không có. Thứ hai, các đặc trưng nhánh trong biểu đồ tuần tự dẫn đến việc sinh ra một số lượng lớn các kịch bản kiểm thử và trong số đó có cả các kịch bản không khả thi.

Chương này của luận án đề xuất để giải quyết các vấn đề trên. Phương pháp cải tiến sinh các kịch bản kiểm thử từ biểu đồ tuần tự UML 2.0 và các ràng buộc trong biểu đồ lớp theo các tiêu chuẩn bao phủ tương tranh. Ưu điểm của phương pháp đề xuất là sử dụng thuật toán chọn lọc các kịch bản kiểm thử khả thi được sinh ra, tránh được việc bùng nổ số lượng kịch bản trong các trường hợp của toán tử song song và tuần tự yếu. Dữ liệu kiểm thử cũng được đưa ra từ việc giải quyết các ràng buộc bởi việc sử dụng các vị từ và giảm miền của các biến từng bước một. Vì vậy, có thể sinh ra dữ liệu kiểm thử trong trường hợp bao phủ vòng lặp.

4.2 Phương pháp đề xuất

4.2.1 Tổng quan về phương pháp đề xuất

Từ những vấn đề còn tồn tại được nêu ở trên, luận án đề xuất phương pháp sinh các kịch bản kiểm thử từ biểu đồ tuần tự UML 2.0 và các ràng buộc trong biểu đồ lớp theo các tiêu chuẩn bao phủ tương tranh khác nhau. Hơn nữa, tác giả cải tiến phương pháp giảm miền động để sinh dữ liệu kiểm thử trong trường hợp các độ bao phủ lặp khác nhau.

Phương pháp đề xuất bao gồm các bước thực hiện như sau:

- Đầu tiên, chuyển đổi biểu đồ tuần tự UML 2.0 và các ràng buộc trong biểu đồ lớp thành CFG.

- Tiếp theo, từ CFG sinh ra các kịch bản kiểm thử theo các độ bao phủ tương tranh khác nhau.
- Sau đó, sinh các dữ liệu kiểm thử từ các ràng buộc trong từng kịch bản kiểm thử theo độ bao phủ lặp.

4.2.2 Sinh các kịch bản kiểm thử

Đầu vào của quá trình sinh các kịch bản kiểm thử là CFG đã đưa ra ở Chương 3. Đầu ra của quá trình này là một tập các kịch bản hữu hạn mà mỗi kịch bản là một đường dẫn hoàn thành từ nút bắt đầu cho đến nút kết thúc. Không thể khả thi nếu chúng ta kiểm thử tất cả các đường dẫn có thể vì nguồn lực kiểm thử có giới hạn. Vì vậy, phương pháp mà luận án đề xuất sử dụng ba tiêu chuẩn kiểm thử tương tranh khác nhau khi sinh các kịch bản kiểm thử. Tiêu chuẩn bao phủ yếu là một trường hợp của tiêu chuẩn bao phủ trung bình, trong khi tiêu chuẩn bao phủ yếu chỉ đưa ra ít nhất một tuần tự của một toán hạng còn tiêu chuẩn bao phủ trung bình đưa ra tất cả các tuần tự trong các toán hạng. Thuật toán 2 được đưa ra để sinh các kịch bản kiểm thử theo tiêu chuẩn tương tranh trung bình.

Thuật toán 2 Sinh các kịch bản kiểm thử theo độ bao phủ tương tranh trung bình

Input: Đồ thị dòng điều khiển G với nút khởi tạo *in* và nút kết thúc fn_i
Output: T là tập hợp các kịch bản kiểm thử, t là một đường dẫn kiểm thử

```
1: T = ∅; t = ∅;
2: curNode = in; //Nút hiện tại bắt đầu từ nút in
3: repeat
4:    move to next node;
5:    if curNode == BN then
6:       t.append(BN);
7:    end if
8:    if curNode == DN then
9:       Append true part of BN up to MN in t
10:      Append false part of BN up to MN in t
11:   end if
12:   if (curNode == FN) then
13:      create sub path t_i for each fork out flow;
14:      append BN of each fork flow up to JN in respective sub path t_i;
15:   end if
16:   if (curNode == fn_i) then
17:      T = T ∪ {t};
18:   end if
19: until Graph end
```

Mục đích của Thuật toán 3 đề xuất sử dụng tiêu chuẩn bao phủ mạnh có chọn lọc để giải quyết vấn đề chia sẻ dữ liệu và đồng bộ nhưng không bị bùng nổ số lượng kịch bản kiểm thử sinh ra. Trong toán tử song song hoặc tuần tự yếu, việc chọn lựa của các điểm hoán đổi tương ứng của các thông điệp đan xen trong các toán hạng là quan trọng. Nếu không có điểm hoán đổi cho từng luồng đồng thời thì thông điệp sẽ được thực thi sau thông điệp khác theo một cách tuần tự. Các tuần tự này không kiểm tra được tính đồng thời giữa các thông điệp. Nếu điểm hoán đổi sau từng thông điệp thì số lượng các đường dẫn được sinh ra sẽ tăng rất nhanh, có thể dẫn đến bùng nổ. Tuy nhiên, nếu các thông điệp của các luồng đồng thời có chia sẽ dữ liệu hoặc cần thứ tự của các thông điệp trong các luồng khác nhau mà thứ tự này cần sự chen ngang của các thông điệp một cách chặt chẽ thì các luồng đồng thời cần lựa chọn cẩn thận các

Thuật toán 3 Sinh các kịch bản kiểm thử theo độ bao phủ tương tranh mạnh

Input: Đồ thị dòng điều khiển G với nút khởi tạo *in* và nút kết thúc fn_i
Output: T là tập hợp các kịch bản kiểm thử, t là một đường dẫn kiểm thử

```
1: T = ∅; t = ∅;
2: curNode = in; //Nút hiện tại bắt đầu từ nút in
3: repeat
4:    move to next node;
5:    if (curNode == BN) then
6:       t.append(BN);
7:    end if
8:    if (curNode==DN and Branch is TRUE) then
9:       Append t with true part BN to MN;
10:   else
11:      Append t with false part BN to MN;
12:   end if
13:   if (curNode == FN) then
14:      active all sub paths of FN;
15:      repeat
16:         Select random sub path;
17:         Append t with node up to before or after node having isAsyn //Thông điệp có thuộc tính isAsyn (true) là
            điểm hoán đổi
18:      until all sub paths are empty
19:   end if
20:   if (curNode == fn_i) then
21:      T = T ∪ {t};
22:   end if
23: until Graph end
```

điểm hoán đổi để sinh ra các kịch bản kiểm thử tương ứng. Một chọn lựa đúng của điểm hoán đổi là các điểm chia sẻ dữ liệu giữa các luồng. Những điểm hoán đổi này đưa ra với mục đích tìm các lỗi về thứ tự và các lỗi an toàn dữ liệu trong thực thi các luồng đồng thời. Các kịch bản kiểm thử được sinh ra bởi Thuật toán 3 có khả năng phát hiện các lỗi an toàn dữ liệu, thực thi đồng thời tìm các trạng thái không đồng bộ của các điểm chia sẻ dữ liệu vì sự chen ngang được chỉ rõ bởi các luồng thực thi. Do đó, Thuật toán 3 đề xuất áp dụng cho các hệ thống để sinh các kịch bản kiểm thử theo độ bao phủ tương tranh mạnh và tránh được vấn đề bùng nổ các kịch bản kiểm thử.

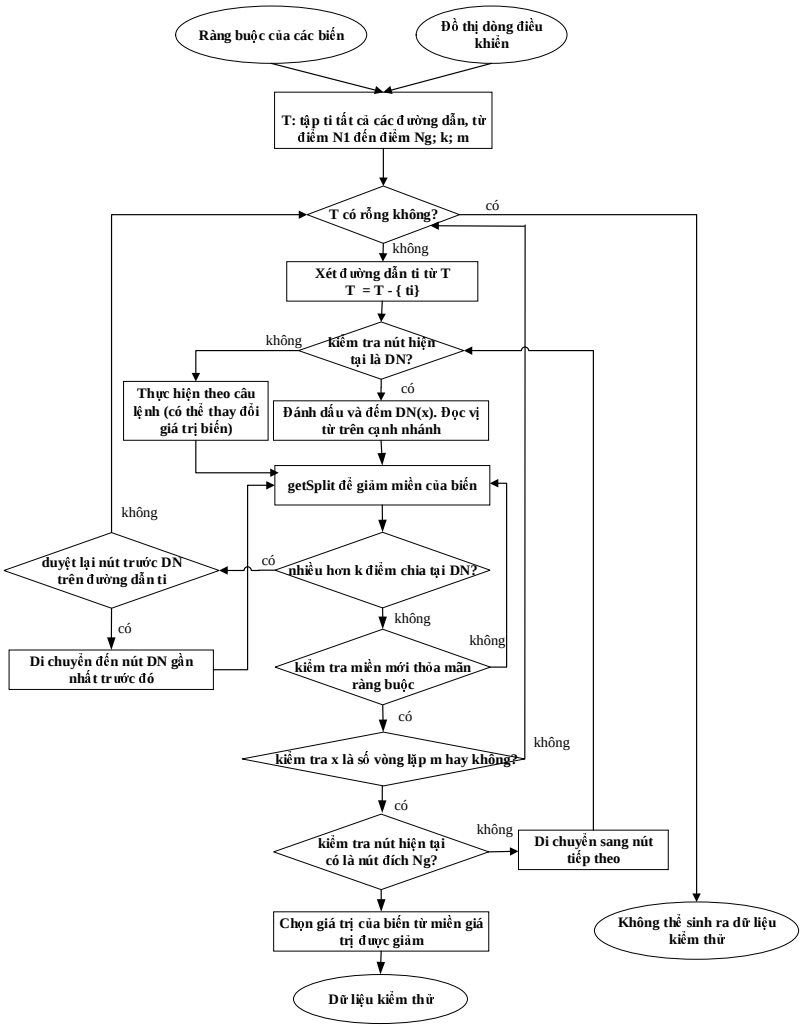
4.2.3 Sinh dữ liệu kiểm thử

Các kịch bản kiểm thử được sinh ra ở trên chính là một tuần tự các thông điệp thực hiện và các ràng buộc. Tuần tự này là tuần tự khả thi nếu tìm thấy dữ liệu kiểm thử để thỏa mãn tất cả các ràng buộc đi kèm với kịch bản đó. Phương pháp đưa ra giải quyết vấn đề bằng tìm giá trị trong các kịch bản kiểm thử của CFG, sử dụng một vị từ tại một thời điểm và giảm miền của các biến từng bước một. Chúng tôi cải tiến phương pháp giảm miền động (dynamic domain reduction — DDR), đặc biệt trong trường hợp vòng lặp. Cách tiếp cận sử dụng phương pháp giảm miền trực tiếp cho các miền giá trị biến, tạo ra một tập các giá trị biểu diễn các điều kiện trên đường dẫn sẽ được thực thi. Phương pháp được cải tiến trong trường hợp bao phủ vòng lặp. Theo độ bao phủ trong kiểm thử vòng lặp, tùy từng trường hợp tham số trong toán tử loop có hoặc không bao gồm các tham số về số vòng lặp tối đa (max) và số vòng lặp tối thiểu (min) thì thuật toán sẽ sinh ra các trường hợp kiểm thử tương ứng theo độ bao phủ lặp. Thuật toán 4 đề xuất để sinh ra số vòng lặp m tương ứng trong các trường hợp khác nhau của độ bao phủ trong kiểm thử vòng lặp.

Thuật toán 4 Sinh các trường hợp kiểm thử tương ứng theo độ bao phủ lặp

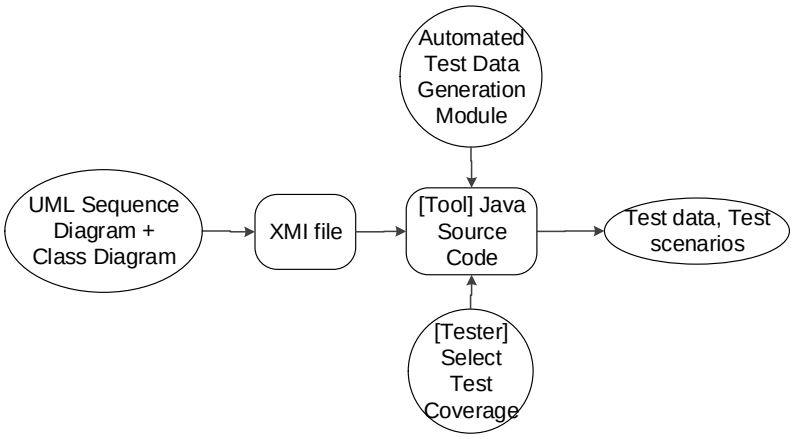
Input: t là một đường dẫn kiểm thử bao gồm nút DN của toán tử lặp và tập ràng buộc tương ứng
Output: Các đường dẫn kiểm thử với m vòng lặp

if (fragment=='loop' && guard ==true && parameters have max and min) **then**
  $k := random(max - 1)$; //k is random number with $2 < k < max - 1$;
  $m := \{0; min; min + 1; k; max - 1; max\}$;
else if (parameters only have max) **then**
  $m := random(max - 1)$; //k is random number with $2 < k < max - 1$;
  $m := \{0; 1; 2; k; max - 1; max\}$;
else if (parameters only have min) **then**
  k is random number >2;
  $m := \{0; min; min + 1; k\}$;
else
  k is random number >2;
  $m := \{0; 1; 2; k\}$;
end if



Hình 4.1: Quá trình sinh dữ liệu kiểm thử.

Hình 4.1 thể hiện quy trình sinh dữ liệu kiểm thử từ tập các đường dẫn T và các ràng buộc. Một đường dẫn có thể được biểu diễn bởi danh sách các vị từ với một vị từ cho từng DN. Từ các ràng buộc của các biến và các vị từ theo từng đường dẫn t_i, *GetSplit* để tạo ra điểm chia mới nhằm giảm miền giới hạn của các biến. Đường dẫn t_i được duyệt, một quá trình tìm kiếm được sử dụng để chia các miền của biến với mục đích nỗ lực để tìm một tập hợp các giá trị mà vẫn thỏa mãn các ràng buộc. *GetSplit* là thuật toán thay đổi miền các biến trong một ràng buộc thỏa mãn: các miền mới vẫn



Hình 4.2: Kiến trúc phát triển của công cụ SequenceConcur.

Bảng 4.1: Kết quả MS sử dụng cho từng kịch bản kiểm thử được sinh ra trong cách tiếp cận của nhóm tác giả Sun và Phương pháp đề xuất

		Phương pháp của Sun				**Phương pháp đề xuất**			
Mức	**Số ca kiểm thử**	**Kịch bản 1**	**Kịch bản 2**	**Kịch bản 3**	**Kịch bản 4**	**Kịch bản 1**	**Kịch bản 2**	**Kịch bản 3**	**Kịch bản 4**
Phương thức	2	41.2%	38.5%	37.5%	40.2%	51.2%	40.5%	45.7%	50.2%
	10/20/30	42.5%	45.7%	43.7%	46.4%	56.5%	55.7%	47.7%	60.4%
Lớp	2/10/20/30	64.5%	64.5%	68.5%	68.5%	74.5%	74.5%	81.5%	81.5%

thỏa mãn các ràng buộc và kích thước của hai miền được chia là bằng nhau. Với các kịch bản kiểm thử được duyệt, các biến được kiểm tra tự động, đối với toán tử vòng lặp khi DN được đánh dấu và đếm với biến x, kiểm tra số lần duyệt một DN có là m hay không. Trong trường hợp thỏa mãn thì dữ liệu kiểm thử được sinh ra với m vòng lặp. Do đó, sinh dữ liệu kiểm thử thỏa mãn bao phủ vòng lặp.

4.3 Thực nghiệm

Chúng tôi xây dựng công cụ SequenceConcur sử dụng phương pháp đề xuất. Hình 4.2 chỉ ra kiến trúc của công cụ. Tất cả các thực nghiệm được thực thi với công cụ đó trên máy tính Intel Core i3- 6100U CPU 2.30 GHz với RAM 2GB.

4.3.1 Thực nghiệm 1

Áp dụng cho biểu đồ tuần tự UML của chức năng Chuyển tiền trong hệ thống Ngân hàng, sau đó so sánh độ phân tích đột biến (Mutation Score – MS) của các kịch bản kiểm thử sinh ra trong cách tiếp cận đề xuất và cách tiếp cận của nhóm tác giả Sun. Bộ gieo lỗi: sau khi có các kịch bản kiểm thử trong hai cách tiếp cận trên thì sử dụng hệ thống đột biến mã nguồn mở *muJava* gieo ngẫu nhiên các lỗi cho hệ thống trên. Sử dụng hệ thống muJava, các toán tử đột biến có 9 loại mức phương thức và 5 loại mức lớp được áp dụng. Trong những loại toán tử áp dụng này, có tổng số 351 mức phương thức và 26 mức lớp đột biến được sinh ra.

Khả năng dò tìm lỗi: sử dụng độ đo MS để đo chất lượng của các bộ kiểm thử được sinh ra trong hai phương pháp đó. Các kích thước được thay đổi là 2, 10, 20 và 30 ca kiểm thử trên một kịch bản. Từ Bảng 4.1, Tác giả có một số nhận xét: Các kịch bản kiểm thử sinh ra của phương pháp đề xuất có độ đo MS cao hơn trong cả hai mức so

Bảng 4.2: Kết quả MS của phương pháp đề xuất và phương pháp kiểm thử ngẫu nhiên

				Phương pháp đề xuất		**Phương pháp ngẫu nhiên**	
ID	Số lượng ca kiểm thử	Mức	Tổng số đột biến	Tổng số đột biến bị giết	MS	Tổng số đột biến bị giết	MS
1	5	Phương thức	351	261	74.3%	139	39.6%
		Lớp	26	26	100%	23	88.4%
		Tổng số	377	287	76.2%	162	42.9%
2	10/20/30	Phương thức	351	266	75.7%	153	43.5%
		Lớp	26	26	100%	26	100%
		Tổng số	377	292	77.5%	179	47.4%
3	15/25/35	Phương thức	425	336	79.1%	173	40.7%
		Lớp	29	29	100%	28	96.5%
		Tổng số	454	365	80.3%	201	44.2%

với phương pháp của nhóm tác giả Sun; Các bộ kiểm thử được sinh ra chỉ ra hiệu quả tìm lỗi tốt: có khả năng tìm hơn 40.5% lỗi ở mức độ phương thức và có khả năng tìm hơn 74.5% lỗi ở mức độ lớp.

4.3.2 Thực nghiệm 2

Thực nghiệm thứ hai áp dụng cho ba biểu đồ tuần tự của ba chức năng điển hình thể hiện tính tương tranh và lặp. Sau đó, so sánh khả năng tìm lỗi của các kịch bản kiểm thử được sinh ra của phương pháp đề xuất và phương pháp kiểm thử ngẫu nhiên. Từ Bảng 4.2 tác giả có một số nhận xét: với cùng kích thước, các bộ kiểm thử được sinh ra bởi phương pháp đề xuất đạt độ đo MS cao hơn so với phương pháp kiểm thử ngẫu nhiên; Không phụ thuộc vào số các bộ kiểm thử, các bộ kiểm thử bằng phương pháp đề xuất có thể đạt được 100% mức độ lớp trong khi đó phương pháp ngẫu nhiên chỉ có thể đạt được 88.4%; Kết quả thực nghiệm chỉ ra rằng, bộ kiểm thử được sinh ra có thể dò tìm được đến 76% lỗi gieo với kích thước nhỏ của bộ kiểm thử (một ca kiểm thử trên một kịch bản).

4.4 Kết luận

Chương này đã trình bày một phương pháp đề xuất sinh dữ liệu kiểm thử tự động từ biểu đồ tuần tự UML 2.0 và biểu đồ lớp trong các ứng dụng tương tranh và bao phủ lặp. Chìa khóa của phương pháp là đề xuất thuật toán chọn các kịch bản kiểm thử khả thi trong trường hợp khai thác sự đan xen giữa các thông điệp tuần tự của toán tử song song và tuần tự yếu. Phương pháp sinh ra các kịch bản kiểm thử cũng có thể tránh được sự bùng nổ các ca kiểm thử bởi việc lựa chọn các điểm hoán đổi. Do đó, các lỗi tương tranh của hệ thống có thể được tìm thấy. Hơn nữa, phương pháp cải tiến việc sinh dữ liệu kiểm thử trong trường hợp bao phủ vòng lặp. Phương pháp này hỗ trợ các tiêu chuẩn bao phủ khác nhau và có thể kiểm thử tính tương tranh trong các ứng dụng khá hiệu quả. Thực nghiệm chỉ ra rằng khả năng tìm lỗi của các kịch bản kiểm thử đó tốt hơn so với phương pháp kiểm thử khác, chứng tỏ tính hiệu quả và độ tin cậy của phương pháp đưa ra về mặt thực nghiệm. Các đóng góp chính của chương này đã được công bố tại Kỷ yếu Hội thảo quốc tế SoICT 2015 và Tạp chí Khoa học Đại học Quốc Gia Hà Nội (VNU Journal of Science 2016).

Chương 5

Sinh dữ liệu kiểm thử cho kiểu dữ liệu chuỗi

5.1 Giới thiệu

Phương pháp sinh dữ liệu kiểm thử hiện tại từ các biểu đồ UML tập trung giải quyết các ràng buộc mà các loại biến có kiểu dữ liệu số. Tuy nhiên, có rất nhiều ứng dụng bị lỗi trong quá trình xử lý chuỗi. Vì vậy, nghiên cứu và phát triển các phương pháp sinh các dữ liệu kiểm thử từ các biểu đồ UML với giải các ràng buộc chuỗi là cần thiết, nhất là áp dụng trong các ứng dụng web có rất nhiều biến dữ liệu chuỗi được sử dụng.

Chương này của luận án cải tiến một phương pháp để sinh các kịch bản kiểm thử tự động từ các biểu đồ tuần tự UML 2.0 và biểu đồ lớp với các ràng buộc chuỗi. Thuật toán sinh các kịch bản kiểm thử để tránh bùng nổ các đường dẫn kiểm thử trong trường hợp không có điểm chia sẻ dữ liệu trong các luồng song song. Việc giải với các ràng buộc chuỗi được sử dụng và cải tiến bộ giải Z3–str. Điểm mới của quá trình sinh dữ liệu kiểm thử đó là đưa ra quy tắc giảm và phương pháp đệ quy cho các toán tử search và replaceAll; mở rộng các quy tắc tiền xử lý cho các toán tử khác như charAt, lastindexOf, trim, startsWith và endsWith.

5.2 Phương pháp đề xuất

5.2.1 Tổng quan về phương pháp đề xuất

Từ những vấn đề còn tồn tại được nêu ở trên, tác giả đề xuất phương pháp sinh các dữ liệu kiểm thử với giải các ràng buộc chuỗi từ các biểu đồ tuần tự và biểu đồ lớp UML. Phương pháp đề xuất bao gồm các bước thực hiện như sau:

- Đầu tiên, chuyển đổi biểu đồ tuần tự UML 2.0 và các ràng buộc trong biểu đồ lớp thành CFG (trong Chương 3).
- Tiếp theo, từ CFG sinh ra các kịch bản kiểm thử trong trường hợp không có sự chia sẻ dữ liệu trong các luồng song song.
- Sau đó, sinh các dữ liệu kiểm thử với các ràng buộc chuỗi trong từng kịch bản kiểm thử.

5.2.2 Sinh các kịch bản kiểm thử

Chương 4 đã đưa ra phương pháp giải quyết vấn đề này để tránh bùng nổ các đường dẫn kiểm thử bởi việc chọn lựa các điểm chia sẻ dữ liệu trong các luồng song song của

hai toán tử song song và tuần tự yếu. Tuy nhiên, có rất nhiều ứng dụng thực tế không có các điểm chia sẻ dữ liệu trong các luồng song song này. Trong trường hợp đó, luận án phát triển Thuật toán 5 (GenerateTestScenario) để duyệt CFG sử dụng cả thuật toán tìm kiếm theo chiều sâu (DFS) và tìm kiếm theo chiều rộng (BFS). Thuật toán BFS được sử dụng nếu nút hiện tại là FN. Việc sử dụng BFS với mục đích tránh việc duyệt và tìm các điểm chia sẻ dữ liệu trong các luồng song song. Hơn nữa, thuật toán BFS khi gặp FN sẽ tránh được vấn đề khóa chết và đồng bộ trong các luồng đó (vì nếu sử dụng DFS mà duyệt luồng này lại đợi luồng khác xử lý thì có thể xảy ra trường hợp khóa chết). Phần còn lại của thuật toán là việc duyệt CFG sử dụng DFS với các tuần tự thông điệp. Như vậy, tránh lãng phí thời gian trong việc tìm các điểm chia sẻ dữ liệu và tránh bùng nổ các đường dẫn kiểm thử cũng như các kịch bản kiểm thử.

Thuật toán 5 (GenerateTestScenario) Sinh các kịch bản kiểm thử từ CFG

Input: Đồ thị dòng điều khiển G với nút khởi tạo *in* và các nút kết thúc fn_i
Output: T là tập các kịch bản kiểm thử, t là một đường dẫn kiểm thử

```
1: T = ∅; t = ∅; queue = ∅;
2: curNode = in; // current node starts from in
3: repeat
4:    t.append(curNode);
5:    move to next node;
6:    if (curNode == DN && decision==TRUE) then
7:       Append true part of BN up to MN in t
8:    else
9:       Append false part of BN up to MN in t;
10:   end if
11:   if (curNode == FN) then
12:      active all nodes of threads; nodes = ready;
13:      put a beginning node x of queue; x= waiting;
14:      repeat
15:         remove front node y of queue; y=processed;
16:         t.append(y);
17:         The neighbour(z) of y having ready status add to the end of queue;
18:         z = waiting;
19:      until (queue is empty)
20:   end if
21:   if (curNode == fn_i) then
22:      T = T ∪ {t};
23:   end if
24: until Graph end
```

5.2.3 Giải các ràng buộc chuỗi

Đầu vào là các đường dẫn kiểm thử và hệ các ràng buộc bao gồm các ràng buộc chuỗi từ CFG. Mục đích sinh ra tập các dữ liệu kiểm thử để thỏa mãn các ràng buộc chuỗi trong từng kịch bản kiểm thử tương ứng. Phương pháp cải tiến bộ giải Z3–str, có hai ưu điểm chính khi sử dụng thuật toán trong Z3–Str. Thứ nhất, Z3–Str hỗ trợ kiểu dữ liệu nguyên thủy chuỗi vì vậy không cần chuyển các ràng buộc chuỗi này sang một biểu diễn khác chẳng hạn như bit–vectơ. Do đó, việc xử lý tránh được phải xác định độ dài của chuỗi trước khi giải quyết các ràng buộc. Thứ hai, bộ giải này sử dụng sức mạnh của Z3 để giải và khả năng giải quyết đồng thời các ràng buộc chuỗi và phi chuỗi một cách hiệu quả. Trong Z3–str sử dụng ba toán tử nguyên thủy: phương trình chuỗi, phép nối chuỗi và độ dài chuỗi. Phương pháp này sử dụng thuật toán giảm để giảm các toán tử chuỗi khác thành các công thức tương đương dựa trên toán tử nguyên

thủy. Z3 có thể mô hình hóa trực tiếp quan hệ tương đương giữa các đối tượng như các biến, các hằng và đặt chúng trong cùng một lớp tương đương. Do đó, chiến lược tổng quan là giảm các toán tử chuỗi khác nhau để đơn giản hóa các quan hệ tương đương. *Cải tiến quy tắc giảm:* Trong ngữ pháp ràng buộc của Z3–str ngoài các toán tử nguyên thủy trên thì hỗ trợ các toán tử chuỗi sau: substring, contains, indexof, replace và split. Toán tử replace chỉ thay thế chuỗi con đầu tiên được tìm thấy trong chuỗi gốc, không thay thế được tất cả các lần xuất hiện của chuỗi con đó. Chương này của luận án đề xuất quy tắc xử lý cho toán tử replaceAll và search, replaceAll định nghĩa dựa vào search và đưa ra việc sử dụng hàm định nghĩa đệ quy. Bảng 5.1 chỉ ra quy tắc giảm của cho các toán tử search và replaceAll.

Bảng 5.1: Quy tắc giảm sử dụng gọi đệ quy

Toán tử	**Quy tắc giảm**
$i = search(S, x)$	$(i = -1 \wedge \neg(contain(S, x))) \vee (i \geq 0 \wedge S = x_{s1} \cdot x_{s2} \cdot x_{s3} \wedge length(x_{s1}) = i \wedge x_{s2} = x \wedge \neg(contain(x_{s1}, x)))$
$R = replaceAll(S, x, T)$	$i = search(S, x) \wedge ((i = -1 \wedge R = S) \vee (i \geq 0 \wedge S = x_{s1} \cdot x_{s2} \cdot x_{s3} \wedge R = R_T \cdot R_R \wedge x_{s2} = x \wedge length(x_{s1}) = i \wedge R_T = x_{s1} \cdot T \cdot x_{s3} \wedge R_R = replaceAll(x_{s3}, x, T)))$

search: tìm chuỗi con x trong chuỗi S, nếu không tìm thấy chuỗi con x trong S thì giá trị chỉ số i trả về là -1. Ngược lại, nếu tìm thấy chuỗi x trong chuỗi gốc S thì giá trị i trả về chỉ vị trí của chuỗi x đầu tiên xuất hiện trong S. Giả sử chuỗi gốc S là phép nối của ba chuỗi con x_{s1}, x_{s2} và x_{s3}. Chuỗi đầu tiên x_{s1} không tìm thấy chuỗi con x, chuỗi thứ hai trong phép nối x_{s2} chính là chuỗi con x và độ dài của x_{s1} chính là i.

replaceAll: tìm chuỗi con x trong chuỗi gốc S, tất cả vị trí tìm thấy x sẽ thay bằng T được chuỗi R. Việc xử lý với replaceAll sử dụng dựa trên toán tử search và phương pháp đệ quy. Đầu tiên, tìm x trong chuỗi gốc S kết quả trả về chỉ số i. Nếu không có chuỗi con x trong S thì chuỗi thay thế R chính là S. Trường hợp trong chuỗi gốc S có tìm thấy chuỗi x thì chuỗi S được tách bằng phép nối: $S = x_{s1} \cdot x_{s2} \cdot x_{s3}$ và chuỗi thay thế R gồm phép nối của chuỗi R_T và R_R. Trong chuỗi x_{s1} không có chuỗi con x, độ dài của x_{s1} là i và $x_{s2} = x$ thì trong chuỗi thay thế đầu tiên R_T chuỗi con x_{s2} thay thế bằng T. Việc thay thế đệ quy được áp dụng cho chuỗi còn lại x_{s3} sẽ thay thế T bằng x để được chuỗi con R_R.

Bảng 5.2: Các quy tắc tiền xử lý cho các toán tử chuỗi

Toán tử	**Công thức đưa ra**
c=x.charAt(i)	$charAt(x, i) = c \rightarrow x = x_1.t.x_2 \wedge t = c \wedge length(x_1) = i$
i= $x_1.lastIndexOf(x_2)$	$lastIndexOf(x_1, x_2) = i \rightarrow (x_1 = x_{s1}.x_{s2}.x_{s3}) \wedge (i = -1 \vee i \geq 0) \wedge ((i = -1) \leftrightarrow (\neg contains(x_1, x_2)) \wedge ((i \geq 0) \leftrightarrow (i = length(x_{s1}) \wedge x_{s2} = x_2 \wedge (\neg contains(x_{s3}, x_2)))$
$x_2 = x_1.trim$	$trim(x_1, x_2) \rightarrow (x_1 = x_{s1}.x_{s2}.x_{s3}) \wedge (x_{s2} = x_2) \wedge ((x_{s1} = \text{“ ”}) \vee (x_{s3} = \text{“ ”}))$
j= $x.startsWith(x_t, i)$	$startsWith(x, x_t, i, j) \rightarrow (x = x_1.x_2.x_3) \wedge (j = 1 \vee j = 0) \wedge ((j = 1) \wedge x_2 = x_t \wedge length(x_1) = i) \wedge ((j = 0) \wedge (\neg contains(x, x_t))$
i= $x.endsWith(x_t)$	$endsWith(x, x_t, i) \rightarrow (x = x_1.x_2.x_3) \wedge (i = 1 \vee i = 0) \wedge ((i = 1) \wedge x_2 = x_t \wedge \neg contains(x_3, x_t)) \wedge ((i = 0) \wedge \neg contains(x, x_t))$

charAt: đưa ra hai tham số là x và i. Toán tử này trả về kí tự c, được định vị tại vị trí i được chỉ rõ trong chuỗi x, việc đánh chỉ số của x bắt đầu từ vị trí 0. Trong chuỗi x được tách thành ba phần: chuỗi x_1, kí tự t và chuỗi x_2; kí tự t ở giữa chính là kí tự trả về c và độ dài của chuỗi x_1 bằng vị trí i tương ứng của kí tự c trong x đưa ra.

lastIndexOf: Nếu chuỗi x_2 là chuỗi con của chuỗi x_1 thì toán tử sẽ trả về chỉ số của kí tự đầu tiên trong chuỗi x_2 xuất hiện cuối cùng. Nếu chuỗi x_2 không là chuỗi con của chuỗi x_1 thì toán tử trả về -1. Ngược lại, nếu và chỉ nếu $(i \geq 0)$ và x_1 tách thành ba phần x_{s1}, x_{s2} và x_{s3} thì chuỗi x_1 gồm chuỗi con x_2, x_2 chính là x_{s2}, i chính là độ dài của chuỗi x_{s1} và chuỗi x_{s3} không bao gồm x_2.

trim: $x_2 = x_1$.trim hàm trả về chuỗi x_2 chính là bản sao của chuỗi x_1 đã bỏ qua khoảng trắng ở phần đầu và phần sau của chuỗi x_1.

startsWith: toán tử kiểm tra liệu chuỗi x_t là chuỗi con của chuỗi x và x_t được xác định bắt đầu từ vị trí i trong chuỗi x. Nếu giá trị trả về $j = 1$ thì chuỗi con x_t là chuỗi con đầu tiên được xuất hiện trong x; ngược lại $j = 0$ thì x_t không là chuỗi con của x.

endsWith: toán tử trả về true $(i = 1)$ nếu chuỗi x_t là chuỗi con sau cùng của chuỗi x, ngược lại trả về false $(i = 0)$. Công thức đưa ra biểu diễn x thành ba chuỗi: x_1, x_2 và x_3. Giá trị biến i có hai lựa chọn: nếu và chỉ nếu chuỗi x không bao gồm chuỗi x_t thì i là 0. Ngược lại, nếu và chỉ nếu giá trị j là 1 thì x_t chính là x_2 và chuỗi x_3 không bao gồm chuỗi con x_t.

5.3 Thực nghiệm

Chúng tôi xây dựng công cụ SeqString để cài đặt phương pháp đưa ra. Tất cả các thực nghiệm được chạy trên Intel Core i3- 6100U CPU 2.30 GHz với RAM 4 GB.

5.3.1 Thực nghiệm 1

Ba ứng dụng có các chức năng với các ràng buộc chuỗi và các toán tử liên quan. Chúng tôi so sánh phần trăm tìm lỗi của các hệ thống khi sử dụng phương pháp đề xuất và sử dụng phương pháp sinh dữ liệu kiểm thử ngẫu nhiên.

Bảng 5.3: So sánh khả năng tìm lỗi của các chức năng trong các ứng dụng

Ứng dụng	Miêu tả	Đầu vào	Lỗi tìm được của phương pháp đề xuất (%)	Lỗi tìm được của kiểm thử ngẫu nhiên (%)
A	Checking information of user registration	3(strings)	100	42.5
B	Business ordering	5(strings)	100	36.5
C	Insurance registration	4(strings)	90	35.6

Giả sử lỗi được đưa vào các chức năng trên tại các điểm biên của dữ liệu kiểm thử. Trong chức năng kiểm tra thông tin người dùng đăng kí (A) có ba biến chuỗi và hai toán tử quan hệ; chức năng đặt đơn hàng (B) có năm biến chuỗi và ba toán tử quan hệ; chức năng đăng kí bảo hiểm (C) có bốn biến chuỗi và không có toán tử quan hệ. Từ Bảng 5.3, với ứng dụng C thì phần trăm tìm lỗi chỉ đạt 90% vì không có toán tử quan hệ nên không có biểu thức tại vùng biên của dữ liệu, do đó dữ liệu kiểm thử không đạt độ bao phủ biên. Như vậy, khả năng tìm lỗi của các kịch bản kiểm thử theo phương pháp đề xuất tốt hơn theo phương pháp kiểm thử ngẫu nhiên.

5.3.2 Thực nghiệm 2

Chúng tôi sử dụng dữ liệu thực nghiệm của nhóm tác giả Shoichiro, bổ sung các ràng buộc bao gồm các toán tử chuỗi đã được tiền xử lý. Thực nghiệm với các biểu đồ

UML và các ràng buộc giống nhau, sau đó so sánh việc sinh dữ liệu kiểm thử và thời gian thực hiện của phương pháp đưa ra và phương pháp của nhóm tác giả Shoichiro. Vì vậy, giả sử tiến hành thực nghiệm cho các ràng buộc chuỗi như nhau trong các biểu đồ tuần tự và biểu đồ lớp. Sau đó, tiến hành thực thi 30 lần trên từng trường hợp và lấy trung bình thời gian thực thi của công cụ đưa ra và so sánh với phương pháp của Shoichiro. Bảng 5.4 cho thấy năm trường hợp mà phương pháp giải một phần không chính xác để có giá trị dữ liệu kiểm thử. Phân tích chỉ ra rằng thời gian giải các ràng buộc chuỗi khi sử dụng các quy tắc tiền xử lý cho các toán tử (charAt, lastindexOf, trim, startsWith và endsWith) thì công cụ đưa ra SeqString có thời gian xử lý nhanh hơn trong cùng một kịch bản kiểm thử của một ứng dụng. Như vậy, phương pháp cải tiến giải quyết khá tốt với nhiều toán tử chuỗi trong hầu hết các trường hợp so với phương pháp của Shoichiro; Phương pháp sinh ra các kịch bản kiểm thử có độ bao phủ tốt hơn và tìm được nhiều lỗi hơn so với một số phương pháp hiện tại.

Bảng 5.4: So sánh xử lý các toán tử chuỗi và hiệu năng của SeqString với phương pháp của nhóm tác giả Shoichiro

Toán tử	**Đầu vào**	**Giải toán tử chính xác?**		**Thời gian(s)**	
		Shoichiro	**SeqString**	**Shoichiro**	**SeqString**
concat	8	x	✓	0.117	0.035
indexOf	12	✓	✓	0.292	0.055
charAt	12	✓	✓	0.142	0.041
match	13	x	✓	0.108	0.036
replace	15	✓	✓	0.152	0.045
substring-charAt	10	✓	✓	0.196	0.045
split - startsWith	9	x	✓	0.245	0.061
lastIndexOf	12	✓	✓	0.205	0.427
charAt-replace	10	x	✓	0.172	0.038
replaceAll	11	x	✓	-	0.055

5.4 Kết luận

Chương 5 của luận án đã trình bày một phương pháp cải tiến việc sinh dữ liệu kiểm thử tự động từ các biểu đồ tuần tự UML 2.0 và biểu đồ lớp với giải các ràng buộc chuỗi. Thuật toán đưa ra sinh các kịch bản kiểm thử để tránh bùng nổ các đường dẫn kiểm thử trong trường hợp không có điểm chia sẻ dữ liệu giữa luồng song song trong các ứng dụng. Với các kịch bản kiểm thử sinh ra đó và một tập các ràng buộc chuỗi cùng các biểu thức tại vùng biên của các biến được chuyển đổi thành định dạng đầu vào của bộ giải Z3–str. So sánh với tiếp cận của Z3–str, phương pháp đề xuất quy tắc giảm và phương pháp đệ quy cho toán tử search, replaceAll và mở rộng các quy tắc tiền xử lý cho các toán tử charAt, lastindexOf, trim, startsWith và endsWith. Các kết quả thực nghiệm chỉ ra rằng phương pháp mà luận án cải tiến có thể giải quyết nhiều toán tử chuỗi chính xác hơn so với một số phương pháp hiện tại. Hơn nữa, các kịch bản kiểm thử được sinh ra thỏa mãn độ bao phủ biên nên đa số các lỗi tại các điểm biên dữ liệu đều có thể tìm thấy trong các ứng dụng. Các đóng góp chính của chương này đã được công bố tại Hội nghị quốc tế Hệ thống thông tin và Cơ sở dữ liệu thông minh (ACIIDS 2017).

Chương 6

Kết luận

6.1 Các kết quả đạt được

Trong quá trình phát triển phần mềm, việc áp dụng kiểm thử dựa trên mô hình có ý nghĩa quan trọng, giúp giảm giá thành phát triển, tiết kiệm thời gian, nâng cao chất lượng và tăng độ tin cậy của phần mềm. Vì vậy, luận án đã nghiên cứu một số giải pháp hỗ trợ sinh dữ liệu kiểm thử tự động từ các biểu đồ UML 2.0. Luận án đã đạt được các kết quả chính như sau:

Thứ nhất, luận án đề xuất một quy trình sinh dữ liệu kiểm thử từ biểu đồ tuần tự UML 2.0 và biểu đồ lớp. Biểu đồ tuần tự UML 2.0 áp dụng cho tất cả mười hai toán tử, có cấu trúc phức tạp và các khối lồng ghép. Trong quá trình sinh và thực thi các kịch bản và dữ liệu kiểm thử tự động áp dụng cho các biến số và cấu trúc động trong các ràng buộc. Chúng có thể chuyển thủ công thành các đoạn mã kiểm thử và thực hiện tự động dựa trên công cụ kiểm thử. Các kết quả thực nghiệm chỉ ra rằng phương pháp đề xuất sinh ra các kịch bản kiểm thử có độ bao phủ và khả năng tìm lỗi tốt hơn so với phương pháp hiện tại.

Thứ hai, luận án đề xuất một phương pháp sinh dữ liệu kiểm thử tự động từ các biểu đồ tuần tự UML 2.0 và biểu đồ lớp trong trường hợp vòng lặp và các ứng dụng tương tranh. Phương pháp sinh ra các kịch bản kiểm thử có thể tránh được sự bùng nổ các kịch bản kiểm thử và các lỗi tương tranh của hệ thống có thể được tìm thấy. Hơn nữa, điểm mới của phương pháp là sinh dữ liệu kiểm thử trong bao phủ vòng lặp. Phương pháp này hỗ trợ các tiêu chuẩn bao phủ tương tranh khác nhau và có thể kiểm thử các lỗi liên quan đến khóa chết và đồng bộ. Thực nghiệm mà chúng tôi đã chứng minh khả năng tìm lỗi của các kịch bản kiểm thử đó tốt hơn so với một số phương pháp khác, chứng tỏ tính hiệu quả và độ tin cậy của phương pháp đưa ra về mặt thực nghiệm.

Thứ ba, luận án đã đưa ra một phương pháp cải tiến việc sinh dữ liệu kiểm thử tự động từ các biểu đồ tuần tự UML 2.0 và biểu đồ lớp với các ràng buộc chuỗi. Thuật toán đưa ra sinh các kịch bản kiểm thử để tránh bùng nổ các đường dẫn kiểm thử trong trường hợp không có điểm chia sẻ dữ liệu giữa luồng song song trong các ứng dụng. Với các kịch bản kiểm thử sinh ra đó và một tập các ràng buộc chuỗi cùng các biểu thức tại vùng biên của các biến được chuyển đổi thành định dạng đầu vào của bộ giải Z3–str. So sánh với các tiếp cận của Z3–str, luận án đề xuất quy tắc giảm, phương pháp đệ quy cho toán tử search, replaceAll và mở rộng các quy tắc tiền xử lý cho các toán tử charAt, lastindexOf, trim, startsWith và endsWith. Các kết quả thực nghiệm

chỉ ra rằng phương pháp mà luận án cải tiến có thể giải quyết nhiều toán tử chuỗi chính xác và hiệu năng tốt hơn trong nhiều trường hợp khi so với một số phương pháp hiện tại. Hơn nữa, các kịch bản kiểm thử được sinh ra thỏa mãn độ bao phủ biên nên đa số các lỗi tại các điểm biên dữ liệu đều có thể tìm thấy trong các ứng dụng.

Như vậy, các đóng góp mà luận án đã đề xuất giải quyết một số vấn đề trong giai đoạn thiết kế kiểm thử từ các biểu đồ UML 2.0, góp một phần giải quyết bài toán lớn về kiểm thử dựa trên mô hình. Kết quả của luận án có khả năng phát triển và ứng dụng thực tiễn, nhằm giải quyết một số vấn đề về chất lượng và kiểm soát chất lượng phần mềm đang được quan tâm hiện nay.

6.2 Hướng nghiên cứu tiếp theo

Trong quá trình nghiên cứu và thực nghiệm, tác giả nhận thấy một số hạn chế và hướng phát triển của luận án. Về thực nghiệm đưa ra trong các chương nói chung còn đơn giản, chưa thể hiện được sự phức tạp trong các thiết kế thực tế. Đối với từng phương pháp đề xuất thì các hạn chế và hướng nghiên cứu tiếp theo, cụ thể như sau:

Chương 3 chưa chứng minh được tính đúng đắn của phương pháp sinh CFG về mặt lý thuyết mà chỉ dựa trên cơ sở thực nghiệm với một số hệ thống điển hình. Trong các nghiên cứu tiếp theo, tác giả sẽ tiến hành chứng minh về mặt lý thuyết tính đúng đắn của phương pháp này. Đồng thời, tiếp tục phát triển sinh các kịch bản kiểm thử từ các loại biểu đồ UML khác hoặc kết hợp các loại biểu đồ khác để các kịch bản kiểm thử có độ bao phủ tốt hơn. Thuật toán sinh dữ liệu kiểm thử là hàm tìm kiếm cục bộ, chưa phải tìm kiếm toàn cục.

Chương 4 tiếp tục nghiên cứu việc xác định các kịch bản kiểm thử khả thi hoặc không khả thi mà không cần dữ liệu kiểm thử đầu vào là bài toán chưa được giải quyết. Trong việc đánh giá tính hiệu quả của khả năng dò tìm lỗi của các kịch bản kiểm thử, tác giả sử dụng độ đo đột biến, các loại độ bao phủ khác nhau trong biểu đồ. Hướng nghiên cứu tiếp phát triển đi sâu vào các độ đo và đặc điểm của từng loại độ đo khác nhau phù hợp với từng ứng dụng và các kịch bản kiểm thử được sinh ra.

Chương 5 đưa ra thuật toán sinh kịch bản kiểm thử sẽ khó kiểm soát theo các tiêu chuẩn bao phủ tương tranh khác nhau. Hơn nữa, việc chuyển đổi tập các ràng buộc chuỗi của các đường dẫn kiểm thử vào bộ giải Z3–Str được thực hiện thủ công. Tác giả đang nghiên cứu để áp dụng chuyển đổi các ràng buộc chuỗi có thể thực hiện tự động. Hướng nghiên cứu việc chuẩn hóa các kịch bản kiểm thử được sinh ra để có thể thực hiện tự động hóa trong các hệ thống thực cũng được quan tâm.

DANH MỤC CÁC CÔNG TRÌNH KHOA HỌC CỦA TÁC GIẢ LIÊN QUAN ĐẾN LUẬN ÁN

[1] Vũ Thị Đào, Tô Văn Khánh và Nguyễn Việt Hà (2014), "Phương pháp sinh ca kiểm thử tự động từ các mô hình thiết kế UML và ngôn ngữ ràng buộc đối tương OCL", *Chuyên san Các công trình nghiên cứu, phát triển và ứng dụng CNTT-TT*, ISSN: 1859-3526, Tập V-1, số 11(31), pp. 70-82.

[2] Thi Dao Vu, Pham Ngoc Hung, Viet Ha Nguyen (2015), "*A Method for Automated Test Data Generation from Sequence Diagrams and Object Constraint Language",* In *Proc. Of the Sixth International Symposium Information and Communication Technology (SoICT 2015),* pp. 335–441. ACM International Publishing.

[3] Thi Dao Vu, Pham Ngoc Hung, Viet Ha Nguyen (2016), "*Automated Testing for Java Programs with Test Cases Generation from Sequence Diagrams and Object Constraint Language"* (poster), *The 14th Asian Symposium on Programming Languages and Systems - APLAS 2016.* Springer International Publishing.

[4] Thi Dao Vu, Pham Ngoc Hung, Viet Ha Nguyen (2016), " A Method for Automated Test Cases Generation from Sequence Diagrams and Object Constraint Language for Concurrent Programs", *VNU Journal of Science: Computer Science and Communication Engineering*, ISSN: 0866-8612, Vol. 32, No.3(2016), pp. 54–71.

[5] Thi Dao Vu, Pham Ngoc Hung, Viet Ha Nguyen (2017), "*A Method for Au*tomated Test Cases Generation from UML Models with String Constraints", In *Proc. Of the 9th Asian Conference on Intelligent Information and Database Systems (ACIIDS 2017),* pp. 525–536. Springer International Publishing.