BÁO CÁO KIỂM THỬ PHẦN MỀM – ĐỀ XUẤT PHƯƠNG PHÁP KIỂM CHỨNG BẢN VÁ TỰ ĐỘNG

Họ và tên: Hoàng Quốc Bảo

MSSV: 23020012

1. Giới thiệu

Hệ thống Sửa lỗi Chương trình Tự động (APR) có khả năng tạo ra các bản vá hợp lệ về mặt kiểm thử (test-suite-adequate patches), nghĩa là các bản vá này thành công vượt qua tất cả các test cases đã có trong bộ kiểm thử (test suite).

Tuy nhiên, tính hợp lệ về mặt kiểm thử không đảm bảo tính đúng đắn (correctness) của bản vá. Một bản vá được gọi là đúng đắn khi nó không chỉ sửa lỗi (fail test case) mà còn phù hợp với ý đồ của lập trình viên và yêu cầu chức năng của hệ thống, không gây ra các hành vi sai lệch mới (overfitting hoặc side-effect).

Báo cáo này đề xuất các phương pháp kiểm chứng bổ sung nhằm nâng cao độ tin cậy và sự đúng đắn của các bản vá được tạo ra bởi hệ thống APR.

2. Các Vấn đề Tiềm ẩn với Bản vá Hợp lệ về Kiểm thử

Trước khi đề xuất phương pháp kiểm chứng, cần nhận diện các vấn đề chính:

- Overfitting (Quá khớp): Bản vá chỉ sửa đúng hành vi đối với các test cases bị lỗi, nhưng có thể thay đổi hành vi đúng đắn của chương trình đối với các trường hợp khác. Đây là các bản vá không đúng đắn (incorrect patches) hoặc bản vá vụng về (plausible-but-incorrect patches).
- Side Effects (Tác dụng phụ): Bản vá có thể vượt qua bộ kiểm thử hiện tại nhưng lại gây ra lỗi (regression) ở các chức năng hoặc mô-đun khác chưa được kiểm thử đầy đủ.
- Deviation from Intent (Lệch lạc khỏi ý đồ): Bản vá có thể vượt qua mọi kiểm thử nhưng lại không phải là giải pháp mà lập trình viên dự kiến (ví dụ: thay vì fix logic, nó chỉ thêm điều kiện `if (input == known_failing_value) return correct_output;`).

3. Các Phương pháp Đề xuất Kiểm chứng Tính Đúng đắn

3.1 Tăng cường và Sinh Test Cases (Test Case Augmentation and Generation)

Đây là phương pháp cơ bản nhất nhằm mục đích làm yếu đi tính hợp lệ về mặt kiểm thử của các bản vá không đúng đắn.

- Mục tiêu: Sinh ra các test cases mới có khả năng "bẻ gãy" các bản vá vụng về (overfitting patches).
- Phương pháp:
    - Phủ mã (Code Coverage): Sinh test cases để đạt được độ phủ cao hơn (ví dụ: độ phủ nhánh, độ phủ điều kiện) trên các đoạn mã bị ảnh hưởng bởi bản vá.
    - Fuzz Testing: Sử dụng các kỹ thuật sinh đầu vào ngẫu nhiên hoặc bán ngẫu nhiên để khám phá các hành vi bất thường.
    - Test Case Dựa trên Thuộc tính/Bất biến (Property/Invariant-Based Generation): Sử dụng các công cụ để tự động suy luận hoặc kiểm tra các thuộc tính bất biến của chương trình và sinh test case vi phạm các bất biến đó.
- Ví dụ minh họa:

    - Lỗi: Một hàm tính tổng bị lỗi ở giá trị đầu vào N=0.
    - Bản vá APR: Thay vì fix logic, nó sửa bằng cách thêm `if(N == 0) return 0;` (Bản vá overfitting).
    - Test Case Tăng cường: Sinh ra test case với N = -1 hoặc N = 1000. Nếu logic gốc có lỗi ở những giá trị này nhưng test suite chưa có, bản vá có thể vượt qua bộ cũ nhưng vẫn sai hoặc gây lỗi mới ở test case tăng cường.

3.2 Kiểm chứng theo Đặc tả (Specification Checking)

Sử dụng các mô tả chính thức về hành vi mong muốn của chương trình.

- Mục tiêu: Xác minh rằng bản vá không vi phạm các tiên quyết (preconditions), hậu điều kiện (postconditions) và bất biến (invariants) của hàm hoặc lớp bị thay đổi.
- Phương pháp:
    - Design by Contract (DbC): Sử dụng ngôn ngữ đặc tả để định nghĩa rõ ràng "hợp đồng" của hàm. Bản vá được kiểm tra xem có tuân thủ hợp đồng này không.
    - Temporal Logic (Logic Thời gian): Kiểm tra các thuộc tính liên quan đến chuỗi sự kiện hoặc trạng thái (ví dụ: "Lệnh A luôn phải được thực hiện trước lệnh B").
- Ví dụ minh họa:
    - Hàm `transfer(sender, receiver, amount)`.
    - Tiên quyết: `amount > 0` và `sender.balance >= amount`.
    - Hậu điều kiện: `sender.balance_new == sender.balance_old - amount`.
    - Nếu bản vá bỏ qua việc kiểm tra amount > 0 và chỉ tập trung vào lỗi cân bằng, nó sẽ bị phát hiện vì vi phạm Tiên quyết đã được đặc tả.

3.3 Phân tích Tĩnh (Static Analysis)

Phân tích mã nguồn mà không cần thực thi chương trình.

- Mục tiêu: Xác định các lỗi hoặc mẫu mã đáng ngờ được tạo ra bởi bản vá, hoặc kiểm tra xem bản vá có thay đổi hành vi ở các đoạn mã không liên quan hay không.
- Phương pháp:
    - Kiểm tra Mẫu mã (Code Pattern Checking): Sử dụng các công cụ như Linters hoặc Code Analyzers để phát hiện các lỗ hổng bảo mật, lỗi cú pháp hoặc vi phạm quy tắc lập trình.
    - Phân tích Luồng Dữ liệu (Data-Flow Analysis): Theo dõi cách bản vá thay đổi việc sử dụng hoặc định nghĩa của các biến, đảm bảo không có biến nào được sử dụng trước khi được khởi tạo.
- Ví dụ minh họa:
    - Bản vá APR vô tình thêm một biến cục bộ X nhưng không khởi tạo nó. Sau đó, nó sử dụng X trong một điều kiện rẽ nhánh.
    - Phân tích Tĩnh sẽ phát hiện lỗi tiềm ẩn "Sử dụng biến chưa được khởi tạo" (Uninitialized Variable Use) và đánh dấu bản vá này là không đáng tin cậy.

3.4 Phân tích Động (Dynamic Analysis)

Phân tích hành vi của chương trình khi nó đang chạy.

- Mục tiêu: Quan sát tác động thực tế của bản vá lên bộ nhớ, hiệu suất và luồng điều khiển trong môi trường thực thi.
- Phương pháp:
    - Lập hồ sơ (Profiling): Đo lường sự thay đổi về hiệu suất hoặc mức sử dụng bộ nhớ sau khi áp dụng bản vá. Một bản vá đúng đắn không nên làm giảm hiệu suất quá mức.
    - Kiểm tra An toàn Bộ nhớ (Memory Safety Checking): Sử dụng các công cụ như Address Sanitizer để phát hiện các lỗi truy cập bộ nhớ ngoài vùng (buffer overflow) do bản vá gây ra.
- Ví dụ minh họa:
    - Bản vá APR sửa một lỗi vòng lặp bằng cách thay đổi điều kiện, nhưng vô tình làm cho vòng lặp đó chạy thêm 1000 lần so với trước đây.
    - Phân tích Động (Profiling) sẽ cho thấy thời gian thực thi của hàm tăng lên đột ngột, báo hiệu rằng bản vá này có thể đúng về chức năng nhưng không hiệu quả về hiệu suất (performance-incorrect).

3.5 So sánh Tương đương Hành vi (Behavioral Equivalence Comparison)

So sánh hành vi của phiên bản chương trình đã vá với phiên bản gốc (hoặc phiên bản đúng đắn đã biết).

- Mục tiêu: Chứng minh rằng bản vá chỉ thay đổi hành vi đối với các trường hợp gây lỗi, và giữ nguyên hành vi đối với tất cả các trường hợp khác.
- Phương pháp:
    - Differential Testing (Kiểm thử Dẫn xuất): Chạy cả hai phiên bản (gốc và đã vá) với cùng một tập dữ liệu đầu vào lớn (có thể là dữ liệu mới được sinh) và so sánh kết quả đầu ra.
    - Symbolic Execution (Thực thi Ký hiệu): Sử dụng các bộ giải SMT để cố gắng chứng minh một cách hình thức rằng hai phiên bản là tương đương về mặt toán học trên một tập hợp đầu vào xác định.
- Ví dụ minh họa:
    - Lỗi: Hàm `calculateFee(price, tax)` chỉ trả về kết quả sai khi price là số âm.
    - So sánh Hành vi: Chạy hàng ngàn test case với price >= 0. Cả hai phiên bản phải cho ra kết quả hoàn toàn giống nhau. Nếu bản vá tạo ra kết quả khác, dù chỉ một lần, nó được coi là không đúng đắn (introduces a side effect).

3.6 Kiểm chứng bằng Học máy (Machine Learning-based Verification)

Sử dụng các mô hình học máy để hỗ trợ quá trình kiểm chứng.

- Mục tiêu: Phân loại các bản vá được sinh ra là đúng đắn hay vụng về (incorrect).
- Phương pháp:
    - Feature Engineering: Trích xuất các đặc trưng từ bản vá (ví dụ: loại thay đổi mã, số lượng dòng bị thay đổi, các API được sử dụng).

    - Mô hình Phân loại: Huấn luyện mô hình Học máy (ví dụ: Random Forest, Neural Networks) trên một tập dữ liệu lớn các bản vá đã được con người dán nhãn (đúng/sai) để dự đoán tính đúng đắn của bản vá mới.
- Ví dụ minh họa:
    - Bản vá APR thường tạo ra các bản vá chỉ là thêm câu lệnh `return;` hoặc thêm điều kiện `if (x == known_failing_value) return correct_output;`.
    - Hệ thống Học máy có thể được huấn luyện để nhận dạng các mẫu mã "đáng ngờ" này, sau đó gán một điểm số rủi ro cao cho bản vá, giúp lập trình viên ưu tiên kiểm tra thủ công.

3.7 Đánh giá Thủ công bởi Lập trình viên (Manual Review by Developers)

Bước kiểm chứng cuối cùng và quan trọng nhất, không thể thay thế.

- Mục tiêu: Đảm bảo bản vá phù hợp với ý đồ thiết kế, phong cách mã hóa, và kiến trúc hệ thống.
- Phương pháp:
    - Code Review (Đánh giá mã): Bản vá phải trải qua quy trình đánh giá mã thông thường. Lập trình viên sẽ xem xét bối cảnh lỗi và đảm bảo rằng giải pháp tự động là giải pháp hợp lý nhất.
    - Kiểm tra Tính dễ đọc và Dễ bảo trì: Bản vá phải không làm giảm chất lượng mã nguồn (code quality) hoặc gây khó khăn cho việc bảo trì trong tương lai.
- Ví dụ minh họa:
    - APR sửa một lỗi trong mô-đun tài chính bằng cách sử dụng một thư viện tiện ích (utility library) không được phê duyệt trong dự án.
    - Đánh giá Thủ công sẽ bác bỏ bản vá này vì nó vi phạm quy tắc kiến trúc và không phù hợp với tiêu chuẩn dự án, dù nó vượt qua tất cả các test cases kỹ thuật.

4. Kết luận

Kiểm chứng bản vá tự động là một quy trình đa chiều, yêu cầu sự kết hợp giữa:

1. Phủ lỗ hổng kiểm thử: Bằng cách Tăng cường Test Cases và So sánh Hành vi (3.1 & 3.5).
2. Xác nhận ý đồ: Bằng cách Kiểm chứng theo Đặc tả và Đánh giá Thủ công (3.2 & 3.7).
3. Phân tích kỹ thuật sâu: Bằng cách Phân tích Tĩnh và Phân tích Động (3.3 & 3.4).
4. Hỗ trợ tự động: Bằng cách sử dụng Kiểm chứng bằng Học máy (3.6).

Sự kết hợp này tạo ra một quy trình kiểm soát chất lượng chặt chẽ, cho phép các tổ chức khai thác lợi ích của sửa lỗi tự động mà vẫn đảm bảo được độ tin cậy của phần mềm.

Link Github: https://github.com/bao2811/testing.git