

ĐỘ TƯƠNG TỰ HÀNH VI CHƯƠNG TRÌNH VÀ THỰC NGHIỆM

Đỗ Đăng Khoa

....

Độ tương tự hành vi của chương trình
và thực nghiệm

Đỗ Đăng Khoa

Ngày 6 tháng 6 năm 2018

Todo list

Viết lại chương này cho cẩn thận	7
Thêm links, bib	7
Bỏ tất cả dấu xuống dòng	7
Đọc related works trong bài báo, trình bày lại, nhớ tham chiếu	11
Viết tổng kết mỗi chương	11
Giới thiệu về phép đo RS	15
Giới thiệu về phép đo SSE	15
Giới thiệu về phép đo PSE	16
Trích dẫn bài báo	21
scan quyet dinh sang pdf và include	22
Đưa một số đoạn code quan trọng	25

Lời cam đoan

Tôi xin cam đoan: Luận văn này là công trình nghiên cứu thực sự của cá nhân, được thực hiện dưới sự hướng dẫn khoa học của TS. Phạm Văn Việt.

Các số liệu, những kết luận nghiên cứu được trình bày trong luận văn này trung thực và chưa từng được công bố dưới bất cứ hình thức nào.

Tôi xin chịu trách nhiệm về nghiên cứu của mình.

Lời cảm ơn

Tôi xin chân thành cảm ơn sự hướng dẫn, chỉ dạy và giúp đỡ tận tình của các thầy cô giảng dạy sau đại học - Trường đại học Quy Nhơn.

Đặc biệt, tôi cảm ơn thầy TS.Phạm Văn Việt, giảng viên bộ môn Công nghệ phần mềm, khoa Công nghệ thông tin, Trường Đại học Quy Nhơn đã tận tình hướng dẫn truyền đạt những kiến thức và kinh nghiệm quý báu để giúp tôi có đầy đủ kiến thức và nghị lực hoàn thành luận văn này.

Và tôi xin cảm ơn bạn bè, đồng nghiệp và những người thân trong gia đình đã tin yêu, động viên giúp tôi thêm nghị lực trong quá trình học tập và nghiên cứu.

Mặc dù đã cố gắng rất nhiều trong việc thực hiện luận văn, song với thời gian có hạn, nên luận văn không thể tránh khỏi những thiếu sót và chưa hoàn chỉnh. Tôi rất mong nhận được ý kiến đóng góp của quý Thầy Cô và các bạn.

Một lần nữa, tôi xin chân thành cảm ơn!

HỌC VIÊN

**Đỗ Đăng
Khoa**

Mục lục

Mục lục	6
1 Giới thiệu	7
1.1 Lý do chọn đề tài	7
1.2 Đối tượng, phạm vi, phương pháp nghiên cứu	9
1.3 Những nghiên cứu có liên quan	11
2 Kiến thức cơ sở	12
2.1 Kiểm thử phần mềm	12
2.2 Sinh ngẫu nhiên dữ liệu thử	12
2.3 Kỹ thuật Dynamic symbolic execution	12
3 Đo độ tương tự về hành vi giữa các chương trình	13
3.1 Hành vi của chương trình	13
3.2 Một số phép đo độ tương tự hành vi	14
3.3 Tiêu chí đánh giá hiệu quả	16
4 Thực nghiệm, đánh giá, kết luận	17
4.1 Dữ liệu thực nghiệm	17
4.2 Công cụ dùng trong thực nghiệm	17
4.3 Đánh giá kết quả thực nghiệm	17
4.4 Khả năng ứng dụng, hướng phát triển	17
4.5 Kết luận	17
Tài liệu tham khảo	18
A Quyết định XXX	22
B Phụ lục XXX	24
C Một số mã lệnh quan trọng	25

Chương 1

Giới thiệu

Chương này trình bày ...

1.1 Lý do chọn đề tài

Giới thiệu chung

Hiện nay, ngành Công nghệ thông tin với các chương trình đào tạo lập trình Online, kỹ sư phần mềm... đang trở nên rất phổ biến. Trên thế giới cũng có nhiều chương trình đào tạo nổi tiếng như Massive Open Online Courses (MOOC), edX, Coursera, Udacity thu hút hàng ngàn sinh viên theo học. Một số chương trình học lập trình online như Pex4Fun hay Code Hunt là một nền tảng học lập trình online thông qua trò chơi.

Những lớp học như thế này thường đặt ra một số thách thức, làm thế nào để kiểm soát được chất lượng học tập của người học. Trong khi các lớp học có hàng trăm người tham gia, nhưng thành viên tham giảng dạy chỉ vài người trong một lớp học. Hằng ngày, người giáo viên phải thường xuyên kiểm tra, nhắc nhở và phải đọc, nghiên cứu giúp đỡ cho học viên. Chỉ riêng việc đọc và hiểu code do học viên viết ra đã tốn quá nhiều thời gian của người dạy, nếu như bỏ qua thì khó có thể đánh giá được chất lượng của người học.

Để giảm bớt những vất vả, khó khăn của người dạy và người học. Một công cụ hỗ trợ, tự động đánh giá hành vi chương trình của người học sẽ giúp tiết kiệm được thời gian, giúp cho giáo viên việc quản lý chất lượng học tập của học viên được tốt hơn. Ví dụ, công cụ sẽ tự động đánh giá hành vi, so sánh hành vi chương trình của người học viết với hành vi chương trình của giáo viên. Nếu hành vi của hai chương trình có tỷ lệ giống nhau càng cao thì điểm số cho chương trình của người học càng cao. Ngược lại, nếu điểm số thấp người dạy và học sẽ tìm hiểu nguyên nhân vì sao và có hướng khác phục những hạn chế mà người học đang gặp phải. Đây cũng là một cách giúp cho người học không đi lệch khỏi kiến thức của chương trình đào tạo, tiết kiệm được thời gian cả người dạy và người học.

Ý tưởng trong việc đánh giá độ tương tự hành vi của hai chương trình đó là tính toán tìm ra một miền giá trị đầu vào chung cho cả hai chương trình.

Viết lại chương này cho cẩn thận

Thêm links, bib

Bỏ tất cả dấu xuống dòng

Đưa từng giá trị đầu vào chạy đồng thời trên cả hai chương trình và so sánh kết quả đầu ra của hai chương trình. Tuy nhiên, việc này sẽ không đạt, không khả thi, vì trong những trường hợp chương trình có miền giá trị đầu vào lớn hoặc vô hạn. Vì vậy, để đánh giá độ tương tự hành vi của hai chương trình khả thi hơn nếu chúng ta chỉ sử dụng một số giá trị đầu vào đại diện cho toàn bộ miền giá trị đầu vào chung của hai chương trình.

Kỹ thuật Dynamic Symbolic Execution (DSE) là một kỹ thuật thu thập các ràng buộc từ các nhánh của chương trình, phủ nhận lại có hệ thống một phần các ràng buộc để tạo ra giá trị đầu vào cho một chương trình. Hiện nay, đã có nhiều công cụ sử dụng kỹ thuật DSE để giải quyết các ràng buộc một cách mạnh mẽ, tạo ra các giá trị đầu vào tin cậy có độ phủ cao.

Tên Công cụ	Ngôn ngữ	Url
KLEE	LLVM	klee.github.io/
JPF	Java	babelfish.arc.nasa.gov/trac/jpf
jCUTE	Java	github.com/osl/jcute
janala2	Java	github.com/ksen007/janala2
JBSE	Java	github.com/pietrobraione/jbse
KeY	Java	www.key-project.org/
Mayhem	Binary	forallsecure.com/mayhem.html
Otter	C	bitbucket.org/khooy/otter/overview
Rubyx	Ruby	www.cs.umd.edu/~avik/papers/ssarorwa.pdf
Pex	.NET Framework	research.microsoft.com/en-us/projects/pex/
Jalangi2	JavaScript	github.com/Samsung/jalangi2
Kite	LLVM	www.cs.ubc.ca/labs/isd/Projects/Kite/
pysymemu	x86-64 / Native	github.com/feliam/pysymemu/
Triton	x86 and x86-64	triton.quarkslab.com
BE-PUM	x86	https://github.com/NMHi/BE-PUM

Trong những năm gần đây, xu hướng đào tạo lập trình viên nói riêng và công nghệ phần mềm nói chung đang ngày càng trở nên phổ biến. Các trường đại học và một số trung tâm đào tạo đã cho ra đời nhiều chương trình đào tạo phong phú về nội dung, đa dạng về hình thức và thu hút được nhiều sự quan tâm.

Trong mỗi khóa học, số lượng học viên thường có hàng trăm, hàng ngàn người tham gia, nhưng chỉ một vài giáo viên giảng dạy. Trong đó, chất lượng việc quản lý, truyền đạt nội dung của giáo viên và mức độ hiểu biết, nắm bắt nội dung chương trình học của học viên là yêu cầu tất cả các khóa học cần đạt được. Để đạt được điều đó, một yêu cầu bắt buộc đó là giáo viên phải đọc và hiểu tất cả các đoạn code của học sinh, nhưng công việc này lại tốn quá nhiều thời gian. Nếu bỏ qua các công việc như vậy thì người dạy không thể theo dõi được quá trình học tập của người học. Về phía người học, yêu cầu đặt ra là phải tiến bộ theo thời gian, nắm vững lý thuyết và thành thạo kỹ năng lập trình. Những không phải lúc nào gặp khó khăn người học đều có sự hỗ trợ kịp thời từ giảng viên. Họ có thể nhờ sự giúp đỡ từ đồng nghiệp, bạn bè, nhưng những người được nhờ giúp đỡ chưa chắc đã đủ trình độ, kinh nghiệm hoặc

thời gian để ngồi bên cạnh giúp đỡ người học khi cần.

Để giảm bớt những khó khăn nêu trên, một công cụ hỗ trợ giáo viên và học sinh hiệu quả hơn, tiết kiệm thời gian hơn đó là một công cụ tự động hóa (có thể một phần) việc đánh giá kết quả lập trình của học sinh, cũng như hỗ trợ theo dõi sự tiến bộ của học sinh. Công cụ tự động hóa này sẽ tính toán, định lượng tỷ lệ chính xác sự tương tự về hành vi giữa chương trình của người học và chương trình của người dạy đưa ra trước đó. Dựa trên kết quả, giáo viên sẽ đánh giá được kỹ năng lập trình của học sinh, sự giống nhau giữa hai chương trình càng cao thì tỷ lệ độ tương tự càng cao, điểm số càng cao. Nếu điểm số thấp, người học có thể quay lại kiểm tra để viết mã chương trình đúng hơn, hạn chế được nguy cơ tìm ẩn trong cách viết chương trình của người học.

Những đánh giá này có thể thực hiện được nếu ta đo được độ tương tự giữa các chương trình có độ chính xác cao. Đề tài “Độ tương tự về hành vi của các chương trình và làm thực nghiệm” với mục đích sẽ giải quyết các vấn đề nêu trên.

1.2 Đối tượng, phạm vi, phương pháp nghiên cứu

Mục tiêu nghiên cứu

Mục tiêu nghiên cứu chính

Đánh giá độ tương tự về hành vi của các chương trình

Mục tiêu nghiên cứu cụ thể

- Tìm hiểu sự tương tự ngữ nghĩa của chương trình
 - Tìm hiểu kỹ thuật, công cụ sinh Test Case tự động
 - Phân tích các độ đo và áp dụng kỹ thuật sinh Test Case tự động trên các độ đo
- Tìm cách kết hợp các độ đo với nhau
- Tìm một số ứng dụng của độ đo, chọn một ứng dụng để làm thực nghiệm, đánh giá kết quả thực nghiệm

Đối tượng, phạm vi nghiên cứu

Đối tượng nghiên cứu

- Kỹ thuật sinh Test Case
 - Độ đo tương tự hành vi
 - Một số ứng dụng của độ đo

Phạm vi nghiên cứu

- Đo độ tương tự hành vi dựa vào Test Case
 - Thực nghiệm, đánh giá trên các chương trình C Sharp

Phương pháp nghiên cứu, thực nghiệm

Nghiên cứu lý thuyết

- Độ tương tự hành vi
- Một số kỹ thuật sinh Test Case tự động
- Độ đo tương tự hành vi dựa trên Test Case
- So sánh, kết hợp các độ đo

Thực nghiệm

- Tiến hành cài kỹ thuật đo độ tương tự hành vi
- Thực nghiệm trên dữ liệu thực của CodeHunt
- Phân tích, đánh giá dựa trên kết quả thực nghiệm

Lý do chọn đề tài, ngữ cảnh bài toán

Trong những năm gần đây, xu hướng đào tạo lập trình viên nói riêng và công nghệ phần mềm nói chung đang ngày càng trở nên phổ biến. Các trường đại học và một số trung tâm đào tạo đã cho ra đời nhiều chương trình đào tạo phong phú về nội dung, đa dạng về hình thức và thu hút được nhiều sự quan tâm.

Trong mỗi khóa học, số lượng học viên thường có hàng trăm, hàng ngàn người tham gia, nhưng chỉ một vài giáo viên giảng dạy. Trong đó, chất lượng việc quản lý, truyền đạt nội dung của giáo viên và mức độ hiểu biết, nắm bắt nội dung chương trình học của học viên là yêu cầu tất cả các khóa học cần đạt được. Để đạt được điều đó, một yêu cầu bắt buộc đó là giáo viên phải đọc và hiểu tất cả các đoạn code của học sinh, nhưng công việc này lại tốn quá nhiều thời gian. Nếu bỏ qua hoặc trì hoãn các công việc như vậy thì người dạy không thể theo dõi được quá trình học tập của người học. Về phía người học, yêu cầu đặt ra là phải tiến bộ theo thời gian, nắm vững lý thuyết và thành thạo kỹ năng lập trình. Những không phải lúc nào gặp khó khăn người học đều có sự hỗ trợ kịp thời từ giảng viên. Họ có thể nhờ sự giúp đỡ từ đồng nghiệp, bạn bè, nhưng những người được nhờ hiups đỡ chưa chắc đã đủ trình độ, kinh nghiệm hoặc thời gian để ngồi bên cạnh giúp đỡ người học khi cần.

Để giảm bớt những khó khăn nêu trên, một công cụ hỗ trợ quá trình giảng dạy và học tập của giáo viên và học sinh hiệu quả hơn, tiết kiệm thời gian hơn đó là một công cụ tự động hóa (có thể một phần) việc đánh giá kết quả lập trình của học sinh, cũng như hỗ trợ theo dõi sự tiến bộ của học sinh. Công cụ tự động hóa này sẽ tính toán, định lượng tỷ lệ chính xác sự tương tự về hành vi giữa chương trình của người học và chương trình của người dạy đưa ra trước đó. Dựa trên kết quả, giáo viên sẽ đánh giá được kỹ năng lập trình của học sinh, sự giống nhau giữa hai chương trình càng cao thì tỷ lệ độ tương tự càng cao, điểm số càng cao. Nếu điểm số thấp, người học có thể quay lại kiểm tra để viết mã chương trình đúng hơn, hạn chế được nguy cơ tìm ẩn trong cách viết chương trình của người học.

Những đánh giá này có thể thực hiện được nếu ta đo được độ tương tự giữa các chương trình có độ chính xác cao. Đề tài “Độ tương tự về hành vi của các

chương trình và làm thực nghiệm” với mục đích sẽ giải quyết các vấn đề nêu trên.

1.3 Những nghiên cứu có liên quan

Tổng kết chương

Dọc re-
lated works
trong bài
báo, trình
bày lại, nhớ
tham chiếu

Viết tổng
kết mỗi
chương

Chương 2

Kiến thức cơ sở

Chương này trình bày những kiến thức cơ sở để triển khai luận văn này, gồm:

- Kiến thức 1
- Kiến thức 2
- Kiến thức 3.

2.1 Kiểm thử phần mềm

Phần này trình bày sơ lược về đảm bảo chất lượng phần mềm nói chung và kiểm thử phần mềm nói riêng cùng với việc đánh giá bộ dữ liệu kiểm thử.

2.2 Sinh ngẫu nhiên dữ liệu thử

Phần này trình bày cơ bản về sinh dữ liệu thử ngẫu nhiên, những ưu và nhược điểm cùng những cải tiến để nâng cao hiệu quả.

2.3 Kỹ thuật Dynamic symbolic execution

Là một kỹ thuật thu thập các ràng buộc từ các nhánh của chương trình và phủ nhận một phần các ràng buộc để tạo ra dữ liệu đầu vào của chương trình và có độ phủ cao

Tổng kết chương

Tổng kết chương viết ở đây.

Chương 3

Đo độ tương tự về hành vi giữa các chương trình

Chương này trình bày ...

3.1 Hành vi của chương trình

Để định lượng hai chương trình tương tự nhau, chúng ta định nghĩa các khái niệm về hành vi chương trình và các định nghĩa liên quan đến sự tương tự của hai chương trình, và ví dụ minh họa cho các định nghĩa.

Thực thi chương trình

Định nghĩa 1 *Hành vi chương trình P là thực hiện hàm: $P \times I \rightarrow O$. Với giá trị đầu vào $i \in I$, giá trị đầu ra $o \in O$. Trong đó I là miền các giá trị đầu vào của chương trình P và O là tập hợp các giá trị đầu ra của chương trình P .*

Tương đương hành vi

Định nghĩa 2 *Hai chương trình P_1 và P_2 có cùng một miền các giá trị đầu vào I và tương đương về hành vi nếu $exec(P_1; I) = exec(P_2; I)$, với $\forall i \in I$ $exec(P_1; i) = exec(P_2; i)$.*

Ví dụ:

Mã lệnh 3.1: Tính y, sử dụng hàm switch...case

```
public static int TinhY(int x)
{
    y = 0;
    switch (x) {
        case 1: y += 4; break;
        case 2: y *= 2; break;
        default: y = y * y;
    }
    return y;
}
```

Mã lệnh 3.2: Tính y, sử dụng hàm If...else

```
public static int TinhY(int x)
{
    y = 0;
    if (x == 1)
        y += 4;
    else if (x == 2)
        y *= 2;
    else
        y = y * y;
    return y;
}
```

Ví dụ trên cho chúng ta thấy 2 chương trình là tương đương nhau về hành vi. Hai chương trình có giá trị đầu vào là như nhau (cùng kiểu **Int**). Chương trình đầu tiên sử dụng hàm **switch...case**, chương trình tiếp theo sử dụng hàm **if...else** để kiểm tra giá trị đầu vào, tuy cú pháp khác nhau nhưng cách xử lý và trả về kết quả **y** là như nhau.

Sự khác biệt về hành vi (Behavioral Difference)

Định nghĩa 3 Hai chương trình P_1 và P_2 có cùng một miền các giá trị đầu vào I và khác biệt về hành vi nếu $exec(P_1, I) \neq exec(P_2, I)$, $\exists i \in I$ $exec(P_1, i) \neq exec(P_2, i)$.

Tương tự hành vi (Behavioral Similarity)

Định nghĩa 4 Tương tự hành vi giữa hai chương trình P_1 và P_2 khi hai chương trình cùng miền giá trị đầu vào là tập hợp các giá trị $|I_s|/|I|$, trong đó $I_s \subseteq I$, nếu $exec(P_1, I_s) = exec(P_2, I_s)$, và $\forall j \in I \setminus I_s$, $exec(P_1; j) \neq exec(P_2; j)$.

Định nghĩa 5 Hai chương trình P_1 và P_2 có cùng một miền các giá trị đầu vào I và khác biệt về hành vi nếu $exec(P_1; I) \neq exec(P_2; I)$, $\exists i \in I$ $exec(P_1; i) \neq exec(P_2; i)$.

Tương tự hành vi (Behavioral Similarity)

Định nghĩa 6 Hai chương trình P_1 và P_2 được gọi là tương tự hành vi khai là hai chương trình có cùng miền giá trị đầu vào I và $|I_s|/|I|$, trong đó $I_s \subseteq I$, nếu $exec(P_1, I_s) = exec(P_2, I_s)$, và $\forall j \in I \setminus I_s$, $exec(P_1; j) \neq exec(P_2; j)$.

3.2 Một số phép đo độ tương tự hành vi

Để tính sự tương đồng về hành vi giữa hai chương trình, chúng ta có thể đo bằng cách tính tỷ lệ đầu vào và đầu ra trên cả hai chương trình trên cùng toàn bộ miền giá trị đầu vào. Một phương pháp tiếp cận đó là liệt kê tất cả các dữ liệu trong miền đầu vào và chạy từng đầu vào trên cả hai chương trình để so sánh các kết quả đầu ra. Nhưng việc này sẽ không thực tế hoặc không liên quan đến các chương trình với một miền đầu vào lớn hoặc vô hạn.

Để đo độ tương tự hành vi được chính xác hơn, chúng tôi chạy các dữ liệu đầu vào đại diện thay vì tất cả các dữ liệu đầu vào cho các chương trình. Bằng

cách thống nhất lấy mẫu một phần dữ liệu đầu vào từ miền đầu vào, độ tương tự về hành vi sẽ được tính dựa trên tỷ lệ các mẫu đầu vào trên các đầu ra.

Dựa trên kỹ thuật Dynamic Symbolic Execution (**DSE**), chúng ta có thể tạo ra được dữ liệu đầu vào thử nghiệm có độ bao phủ cao, và sử dụng trong các kỹ thuật đo độ tương tự.

Phép đo Random Sampling (RS)

Giới thiệu về phép đo RS

Định nghĩa 7 Hai chương trình P_1 và P_2 là hai chương trình có cùng miền giá trị đầu vào I và I_s là tập con các giá trị đầu vào của tập I , và I_a là tập con các giá trị đầu vào của tập I_s , trong đó $\forall i \in I_a$, sao cho $exec(P_1, i) = exec(P_2, i)$ và $\forall j \in I_s \setminus I_a$, $exec(P_1, j) \neq exec(P_2, j)$. Độ đo của kỹ thuật của RS sẽ là $M_{RS}(P_1, P_2) = |I_a|/|I_s|$.

Phép đo **RS** là một phương pháp đo tương đối đơn giản và hiệu quả để tính độ tương tự hành vi. Nhưng phép đo **RS** cũng có hạn chế nhất định trong những trường hợp có những hành vi nhỏ giữa các chương trình và phép đo **RS** không thể phân biệt các hành vi hơi khác nhau này.

Mã lệnh 3.3: Chương trình P_1

```
public static string Puzzle(string x) {
    if (x == "Hello") return "NOT OK";
    if (x == "Funny") return "NOT OK";
    return "OK";
}
```

Mã lệnh 3.4: Chương trình P_2

```
public static string Puzzle(string x) {
    return "OK";
}
```

Trong ví dụ trên, chương trình P_1 và P_2 có hành vi khác biệt nhỏ nhưng độ đo **RS** không thể phân biệt được sự khác biệt này và trả về độ đo bằng 1.

Phép đo Single Program Symbol Execution (SSE)

Giới thiệu về phép đo SSE

Định nghĩa 8 Hai chương trình P_1 và P_2 là hai chương trình có cùng miền giá trị đầu vào I và P_1 là chương trình tham chiếu. Trong đó, I_s là tập các giá trị đầu vào được tạo bởi DSE trên P_1 , và I_a là tập con các giá trị đầu vào của tập I_s , sao cho $\forall i \in I_a$, $exec(P_1, i) = exec(P_2, i)$ và $\forall j \in I_s \setminus I_a$, $exec(P_1, j) \neq exec(P_2, j)$. Độ đo của kỹ thuật của SSE sẽ là $M_{SSE}(P_1, P_2) = |I_a|/|I_s|$.

Giới thiệu
về phép đo
PSE

Định nghĩa 9 Hai chương trình P_1 và P_2 là hai chương trình có cùng miền giá trị đầu vào I . Chúng ta có chương trình P_3 là chương trình kết hợp của P_1 và P_2 có dạng $(exec(P_1, I) = exec(P_2, I))$, thỏa điều kiện đầu vào và khẳng định điều kiện là đúng, $exec(P_3, i) = T$ với giá trị đầu vào i trên P_3 là thỏa điều kiện. Trong đó, I_s là tập các giá trị đầu vào được tạo bởi DSE trên P_3 , và I_a là tập con các giá trị đầu vào của tập I_s , sao cho $\forall i \in I_a, exec(P_3, i) = T$ và $\nexists j \in I_s \setminus I_a, exec(P_3, j) = T$. Độ đo của kỹ thuật của PSE sẽ là $M_{PSE}(P_1, P_2) = |I_a|/|I_s|$.

3.3 Tiêu chí đánh giá hiệu quả

Tổng kết chương

Tổng kết chương viết ở đây.

Chương 4

Thực nghiệm, đánh giá, kết luận

Chương này trình bày ...

4.1 Dữ liệu thực nghiệm

Phần này trình bày về kho dữ liệu dùng để thực nghiệm, Code Hunt [REF].

4.2 Công cụ dùng trong thực nghiệm

Phần này trình bày những công cụ được sử dụng để triển khai thực nghiệm như: công cụ sinh dữ liệu kiểm thử, môi trường lập trình, ...

4.3 Đánh giá kết quả thực nghiệm

Phần này trình bày những kết quả đo được trên bộ dữ liệu thực nghiệm đã nêu trong Phần 4.1.

4.4 Khả năng ứng dụng, hướng phát triển

Phần này trình bày một số ứng dụng có thể của việc đo độ tương tự về hành vi của chương trình cùng hướng phát triển trong tương lai.

Đánh giá tiến bộ trong lập trình

Xếp hạng tự động

Gợi ý giải pháp lập trình

Hướng phát triển

4.5 Kết luận

Tổng kết chương

Tổng kết chương viết ở đây.

Tài liệu tham khảo

- [1] Kalle Aaltonen, Petri Ihantola, and Otto Seppälä. Mutation analysis vs. code coverage in automated assessment of students' testing skills. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 153–160. ACM, 2010.
- [2] Rajeev Alur, Loris D'Antoni, Sumit Gulwani, Dileep Kini, and Mahesh Viswanathan. Automated grading of dfa constructions. In *IJCAI*, volume 13, pages 1976–1982, 2013.
- [3] Samuel Bates and Susan Horwitz. Incremental program testing using program dependence graphs. In *Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 384–396. ACM, 1993.
- [4] David Binkley. Using semantic differencing to reduce the cost of regression testing. In *Software Maintenance, 1992. Proceedings., Conference on*, pages 41–50. IEEE, 1992.
- [5] Stefan Bucur, Vlad Ureche, Cristian Zamfir, and George Candea. Parallel symbolic execution for automated real-world software testing. In *Proceedings of the sixth conference on Computer systems*, pages 183–198. ACM, 2011.
- [6] Cristian Cadar, Patrice Godefroid, Sarfraz Khurshid, Corina S Păsăreanu, Koushik Sen, Nikolai Tillmann, and Willem Visser. Symbolic execution for software testing in practice: preliminary assessment. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 1066–1071. ACM, 2011.
- [7] Cristian Cadar and Koushik Sen. Symbolic execution for software testing: three decades later. *Communications of the ACM*, 56(2):82–90, 2013.
- [8] Tsong Yueh Chen, Fei-Ching Kuo, Robert G Merkel, and TH Tse. Adaptive random testing: The art of test case diversity. *Journal of Systems and Software*, 83(1):60–66, 2010.
- [9] John A Darringer and James C King. Applications of symbolic execution to program testing. *Computer*, 11(4):51–60, 1978.
- [10] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, 2008.

- [11] Christopher Douce, David Livingstone, and James Orwell. Automatic test-based assessment of programming: A review. *Journal on Educational Resources in Computing (JERIC)*, 5(3):4, 2005.
- [12] Jaco Geldenhuys, Matthew B Dwyer, and Willem Visser. Probabilistic symbolic execution. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, pages 166–176. ACM, 2012.
- [13] Patrice Godefroid, Nils Klarlund, and Koushik Sen. Dart: directed automated random testing. In *ACM Sigplan Notices*, volume 40, pages 213–223. ACM, 2005.
- [14] Patrice Godefroid, Michael Y Levin, David A Molnar, et al. Automated whitebox fuzz testing. In *NDSS*, volume 8, pages 151–166, 2008.
- [15] Jan B Hext and JW Winings. An automatic grading scheme for simple programming exercises. *Communications of the ACM*, 12(5):272–275, 1969.
- [16] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, pages 86–93. ACM, 2010.
- [17] Julia Isong. Developing an automated program checkers. In *Journal of Computing Sciences in Colleges*, volume 16, pages 218–224. Consortium for Computing Sciences in Colleges, 2001.
- [18] Daniel Jackson, David A Ladd, et al. Semantic diff: A tool for summarizing the effects of modifications. In *ICSM*, volume 94, pages 243–252, 1994.
- [19] Lingxiao Jiang and Zhendong Su. Automatic mining of functionally equivalent code fragments via random testing. In *Proceedings of the eighteenth international symposium on Software testing and analysis*, pages 81–92. ACM, 2009.
- [20] Edward L Jones. Grading student programs-a software testing approach. *Journal of Computing Sciences in Colleges*, 16(2):185–192, 2001.
- [21] Sarfraz Khurshid, Corina S Păsăreanu, and Willem Visser. Generalized symbolic execution for model checking and testing. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 553–568. Springer, 2003.
- [22] James C King. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, 1976.
- [23] Shuvendu K Lahiri, Chris Hawblitzel, Ming Kawaguchi, and Henrique Rebêlo. Symdiff: A language-agnostic semantic diff tool for imperative programs. In *CAV*, volume 12, pages 712–717. Springer, 2012.
- [24] Sihan Li, Xusheng Xiao, Blake Bassett, Tao Xie, and Nikolai Tillmann. Measuring code behavioral similarity for programming and software engineering education. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 501–510. ACM, 2016.

- [25] Christophe Meudec. Atgen: automatic test data generation using constraint logic programming and symbolic execution. *Software Testing, Verification and Reliability*, 11(2):81–96, 2001.
- [26] Carlos Pacheco, Shuvendu K Lahiri, Michael D Ernst, and Thomas Ball. Feedback-directed random test generation. In *Proceedings of the 29th international conference on Software Engineering*, pages 75–84. IEEE Computer Society, 2007.
- [27] Laura Pappano. The year of the mooc. *The New York Times*, 2(12):2012, 2012.
- [28] Corina S Păsăreanu and Neha Rungta. Symbolic pathfinder: symbolic execution of java bytecode. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, pages 179–180. ACM, 2010.
- [29] Corina S Păsăreanu and Willem Visser. Verification of java programs using symbolic execution and invariant generation. In *International SPIN Workshop on Model Checking of Software*, pages 164–181. Springer, 2004.
- [30] Corina S Păsăreanu and Willem Visser. A survey of new trends in symbolic execution for software testing and analysis. *International Journal on Software Tools for Technology Transfer (STTT)*, 11(4):339–353, 2009.
- [31] Suzette Person, Matthew B Dwyer, Sebastian Elbaum, and Corina S Păsăreanu. Differential symbolic execution. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 226–237. ACM, 2008.
- [32] Graham HB Roberts and Janet LM Verbyla. An online programming assessment tool. In *Proceedings of the fifth Australasian conference on Computing education-Volume 20*, pages 69–75. Australian Computer Society, Inc., 2003.
- [33] Koushik Sen, Darko Marinov, and Gul Agha. Cute: a concolic unit testing engine for c. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 263–272. ACM, 2005.
- [34] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. Automated feedback generation for introductory programming assignments. *ACM SIGPLAN Notices*, 48(6):15–26, 2013.
- [35] Fang-Hsiang Su, Jonathan Bell, Kenneth Harvey, Simha Sethumadhavan, Gail Kaiser, and Tony Jebara. Code relatives: Detecting similarly behaving software. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 702–714. ACM, 2016.
- [36] Fang-Hsiang Su, Jonathan Bell, Gail Kaiser, and Simha Sethumadhavan. Identifying functionally similar code in complex codebases. In *Program Comprehension (ICPC), 2016 IEEE 24th International Conference on*, pages 1–10. IEEE, 2016.

- [37] Kunal Taneja and Tao Xie. Diffgen: Automated regression unit-test generation. In *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pages 407–410. IEEE Computer Society, 2008.
- [38] Nikolai Tillmann and Jonathan De Halleux. Pex–white box test generation for. net. *Tests and Proofs*, pages 134–153, 2008.
- [39] Nikolai Tillmann, Jonathan De Halleux, and Tao Xie. Transferring an automated test generation tool to practice: From pex to fakes and code digger. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pages 385–396. ACM, 2014.
- [40] Nikolai Tillmann, Jonathan De Halleux, Tao Xie, Sumit Gulwani, and Judith Bishop. Teaching and learning programming and software engineering via interactive gaming. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 1117–1126. IEEE Press, 2013.
- [41] Andrew Walenstein, Mohammad El-Ramly, James R Cordy, William S Evans, Kiarash Mahdavi, Markus Pizka, Ganesan Ramalingam, and Jürgen Wolff von Gudenberg. Similarity in programs. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007.
- [42] Xusheng Xiao, Sihan Li, Tao Xie, and Nikolai Tillmann. Characteristic studies of loop problems for structural test generation via symbolic execution. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 246–256. IEEE, 2013.

Phụ lục A

Quyết định XXX

scan quyết
định sang
pdf và in-
clude

Quyết định


Phụ lục B

Phụ lục XXX

Nội dung phụ lục viết ở đây.

Phụ lục C

Một số mã lệnh quan trọng



Đưa một số
đoạn code
quan trọng