

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC QUY NHƠN

ĐỒ ĐĂNG KHOA

**ĐỘ TƯƠNG TỰ HÀNH VI CỦA
CHƯƠNG TRÌNH VÀ THỰC NGHIỆM**

Chuyên ngành: Khoa học máy tính

Mã số: 8480101

TÓM TẮT LUẬN VĂN THẠC SĨ KHOA HỌC MÁY TÍNH

Bình Định - Năm 2018

Công trình được hoàn thành tại
TRƯỜNG ĐẠI HỌC QUY NHƠN

Người hướng dẫn: TS. PHẠM VĂN VIỆT

Phản biện 1:

Phản biện 2:

Luận văn được bảo vệ tại Hội đồng đánh giá luận văn thạc sĩ chuyên ngành Khoa học máy tính, ngày 26 tháng 8 năm 2018 tại Trường Đại học Quy Nhơn.

Có thể tìm hiểu luận văn tại:

- Trung tâm Thông tin tư liệu, Trường Đại học Quy Nhơn
- Khoa Công nghệ thông tin

Chương 1

PHẦN MỞ ĐẦU

1.1 Lý do chọn đề tài

Hiện nay, ngành Công nghệ thông tin đang có xu hướng phát triển mạnh mẽ và trở nên phổ biến. Một số chương trình đào tạo nổi tiếng như Massive Open Online Courses (MOOC) [?], edX [?], Coursera [?], Udacity [?], chương trình học lập trình online như Pex4Fun [?] hay Code Hunt [?] thu hút nhiều sự quan tâm của các bậc phụ huynh và các em học sinh.

Những lớp học như vậy thường có nhiều sinh viên tham gia, các em sinh viên đến từ nhiều nơi trên thế giới với nhiều nền văn hóa và ngôn ngữ khác nhau. Để có thể quản lý được những lớp học như vậy là cả một khối lượng công việc lớn, yêu cầu nỗ lực của những người quản lý và giảng viên. Chỉ riêng việc đọc hiểu để đánh giá kết quả mã lệnh do sinh viên viết đã tốn nhiều thời gian, nếu như bỏ qua thì giảng viên sẽ không theo dõi được quá trình học tập của sinh viên. Những lúc khó khăn trong việc viết mã chương trình, sinh viên có thể nhờ bạn bè hoặc nhờ những người có kinh nghiệm giúp đỡ. Tuy nhiên, không phải lúc nào cũng có người bên cạnh để giúp đỡ cho họ, và kinh nghiệm cũng như kiến thức của những người này chưa chắc có thể đáp ứng được yêu cầu.

Để giảm bớt khó khăn cho giảng viên và sinh viên, một công cụ hỗ trợ tự động đánh giá kết quả chương trình của sinh viên với chương trình của giảng viên sẽ giúp tiết kiệm được thời gian, giúp cho giảng viên quản lý việc học tập của sinh viên được tốt hơn. Sinh viên có thể

nhanh chóng biết được chương trình của mình viết đúng hay sai.

Cách thức hoạt động của công cụ này là đánh giá độ tương tự về hành vi của hai chương trình. Công cụ sẽ tính toán để tìm ra các mẫu dữ liệu đầu vào chung cho cả hai chương trình, tiến hành lấy từng mẫu dữ liệu đầu vào chạy đồng thời trên cả hai chương trình và so sánh kết quả đầu ra của hai chương trình. Nếu kết quả đầu ra của hai chương trình có tỷ lệ giống nhau càng cao thì điểm số của sinh viên càng cao. Ngược lại, nếu tỷ lệ càng thấp thì tương ứng với điểm số của sinh viên càng thấp. Dựa trên kết quả này, giảng viên có thể nắm bắt được tình hình học tập của sinh viên và có hướng khắc phục những hạn chế mà sinh viên đang gặp phải. Đây cũng là một cách giúp cho sinh viên không đi lệch khỏi định hướng kiến thức, các kỹ thuật, kỹ năng lập trình và hạn chế được những nguy cơ tìm ẩn trong cách viết mã lệnh chương trình. Đồng thời giúp tiết kiệm được thời gian cho cả giảng viên và sinh viên.

1.2 Đối tượng, phạm vi, phương pháp nghiên cứu

Mục tiêu nghiên cứu

Mục tiêu nghiên cứu chính của luận văn là đánh giá độ tương tự về hành vi của các chương trình.

Mục tiêu nghiên cứu cụ thể

- Tìm hiểu sự tương tự hành vi của chương trình
- Tìm hiểu kỹ thuật, công cụ sinh Test Case tự động và áp dụng kỹ thuật sinh Test Case tự động trên các kỹ thuật đo độ tương tự
- Tìm cách kết hợp các kỹ thuật đo với nhau
- Đánh giá kết quả thực nghiệm

Đối tượng, phạm vi nghiên cứu

Đối tượng nghiên cứu

- Kỹ thuật sinh Test Case
- Các kỹ thuật đo độ tương tự hành vi

- Ứng dụng của các kỹ thuật đo độ tương tự hành vi

Phạm vi nghiên cứu

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

Phương pháp nghiên cứu, thực nghiệm

Nghiên cứu lý thuyết

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

Thực nghiệm

- Tiến hành cài đặt các kỹ thuật đo độ tương tự hành vi
- Thực nghiệm trên dữ liệu thực của CodeHunt, và một số dữ liệu thử khác
- Phân tích, đánh giá dựa trên kết quả thực nghiệm

Chương 2

PHẦN NỘI DUNG

2.1 Kiến thức cơ sở

Những kiến thức cơ sở để triển khai luận văn bao gồm kiến thức về kiểm thử phần mềm, kỹ thuật sinh ngẫu nhiên dữ liệu thử, và kỹ thuật Dynamic Symbolic Execution.

Kiểm thử phần mềm

Các phương pháp kiểm thử

Có nhiều phương pháp để kiểm thử phần mềm, trong đó hai phương pháp kiểm chính là kiểm thử thử tĩnh và kiểm thử động là hai phương pháp chính.

Kiểm thử tĩnh (Static testing): Là phương pháp kiểm thử phần mềm bằng cách duyệt lại các yêu cầu và các đặc tả bằng tay, thông qua việc sử dụng giấy, bút để kiểm tra tính logic từng chi tiết mà không cần chạy chương trình. Kiểu kiểm thử này thường được sử dụng bởi chuyên viên thiết kế, người viết mã lệnh chương trình. Kiểm thử tĩnh cũng có thể được tự động hóa bằng cách thực hiện kiểm tra toàn bộ hệ thống thông qua một trình thông dịch hoặc trình biên dịch, xác nhận tính hợp lệ về cú pháp của chương trình.

Kiểm thử động (Dynamic testing): Là phương pháp kiểm thử thông qua việc chạy chương trình để kiểm tra trạng thái tác động của chương trình, dựa trên các ca kiểm thử xác định các đối tượng kiểm thử của chương trình. Đồng thời, kiểm thử động sẽ tiến hành kiểm tra

cách thức hoạt động của mã lệnh, tức là kiểm tra phản ứng từ hệ thống với các biến thay đổi theo thời gian. Trong kiểm thử động, phần mềm phải được biên dịch và chạy, và bao gồm việc nhập các giá trị đầu vào và kiểm tra giá trị đầu ra có như mong muốn hay không.

Sinh dữ liệu kiểm thử

Sinh ngẫu nhiên dữ liệu thử là một kỹ thuật kiểm thử phần mềm Black-Box, kỹ thuật này tạo ra ngẫu nhiên các giá trị đầu vào và thực thi từng giá trị đầu vào trên chương trình được kiểm thử. Kết quả đầu ra của chương trình được so sánh với các thông số kỹ thuật của phần mềm, xác định đầu ra thử nghiệm thành công hoặc không thành công [?].

Kỹ thuật sinh ngẫu nhiên dữ liệu thử không quan tâm đến hành vi và cấu trúc bên trong của chương trình, chỉ tập trung tìm kiếm những trường hợp chương trình không hoạt động theo đặc tả kỹ thuật của chương trình. Trong phương pháp này, dữ liệu thử nghiệm được tạo ngẫu nhiên từ các đặc tả kỹ thuật của phần mềm (tức là không liên quan tới hành vi và cấu trúc của chương trình).

Mã lệnh 2.1: Hàm sinh ngẫu nhiên dữ liệu thử

```
int myAbs(int x)
{
    if (x > 0) {
        return x;
    }
    else {
        return x;
    }
}

void testAbs(int n)
{
    for (int i = 0; i < n; i++)
    {
        int x = getRandomInput();
        int result = myAbs(x);
        assert(result >= 0);
    }
}
```

Mã lệnh 2.1 là một hàm sinh ngẫu nhiên dữ liệu thử, chúng ta thấy hàm *testAbs* chỉ thực hiện việc tạo giá trị đầu vào ngẫu nhiên *intx* theo đặc tả tham số đầu vào của chương trình *myAbs*, và kiểm tra kết quả đầu ra của chương trình *assert(result >= 0)*, không quan tâm hành vi và cấu trúc bên trong của hàm *myAbs*.

Ưu điểm, hạn chế, hướng khắc phục

Sinh ngẫu nhiên dữ liệu thử có một số ưu điểm và nhược điểm như sau:

* *Ưu điểm:*

- Đơn giản, dễ dàng sinh các đầu vào ngẫu nhiên
- Không tốn nhiều tài nguyên bộ nhớ lúc thực thi

* *Hạn chế:*

- Một nhánh hành vi của chương trình được kiểm thử nhiều lần với nhiều đầu vào khác nhau
- Có thể một số nhánh hành vi của chương trình bị bỏ qua
- Khó xác định khi nào việc kiểm thử nên dừng lại
- Không biết dữ liệu thử có duyệt được tất cả các nhánh trong chương trình hay không

* *Hướng khắc phục:* Để xác định khi nào việc kiểm thử dừng lại, hệ thống kiểm thử ngẫu nhiên có thể kết hợp với kỹ thuật Adequacy Criterion [?]. Kỹ thuật Adequacy Criterion là một kỹ thuật yêu cầu duyệt tất cả các nhánh của chương trình, bằng việc kết hợp này cho phép việc kiểm thử chỉ dừng lại khi tất cả các câu lệnh của chương trình được thực thi ít nhất một lần.

Kỹ thuật Dynamic symbolic execution

Dynamic symbolic execution (DSE) là một kỹ thuật duyệt tự động tất cả các đường đi có thể của chương trình bằng cách chạy chương trình với nhiều giá trị đầu vào khác nhau để tăng độ phủ của dữ liệu thử [?].

Dựa trên các tham số đầu vào của chương trình, DSE sẽ tạo ra các giá trị đầu vào cụ thể và thực thi chương trình với các giá trị cụ thể này.

Trong quá trình thực thi, DSE sẽ ghi nhận lại ràng buộc tại các nút, phủ định lại các ràng buộc này và sinh các giá trị đầu vào thỏa điều kiện ràng buộc tại các nút rẽ nhánh này. Với một giá trị đầu vào cụ thể, DSE sẽ thực thi chương trình và duyệt được một đường đi cụ thể, quá trình thực thi này sẽ lặp lại cho đến khi duyệt hết tất cả các đường đi của chương trình.

Algorithm 1 DSE

```

Set J :=  $\emptyset$       ▷ J: Tập hợp các đầu vào của chương trình phân tích
loop
    Chọn đầu vào  $i \notin J$     ▷ Dừng lại nếu không có  $i$  nào được
    tìm thấy
    Xuất ra  $i$ 
    Thực thi P( $i$ ); lưu lại điều kiện đường đi C( $i$ ); suy ra C'( $i$ )
    Đặt  $J := J \cup i$ 
end loop
  
```

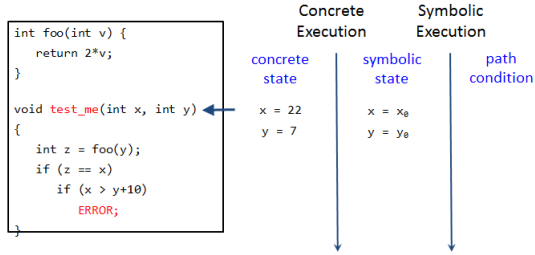
Mã lệnh 2.2: Minh họa kỹ thuật DSE

```

int foo(int v) {
    return 2*v;
}
void test_me(int x, int y) {
    int z = foo(y);
    if (z == x)
        if (x > y+10)
            ERROR;
}
  
```

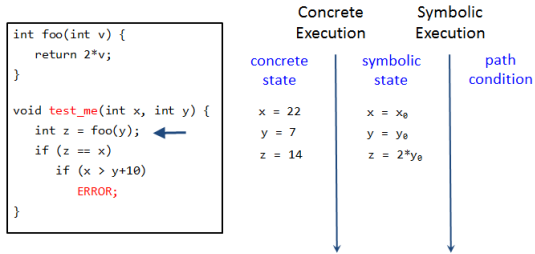
Trong mã lệnh 2.2, chúng ta xem xét hàm `test_me` với hai tham số đầu vào là `intx` và `inty`, và hàm này không có giá trị trả về. Cách thức làm việc của DSE trên hàm `test_me` như sau:

Đầu tiên, DSE tạo hai giá trị đầu vào thử nghiệm ngẫu nhiên x và y , giả sử $x = 22$ và $y = 7$. Ngoài ra, DSE sẽ theo dõi trạng thái các giá trị đầu vào thử nghiệm của chương trình với x bằng một số x_0 và y bằng một số y_0 .



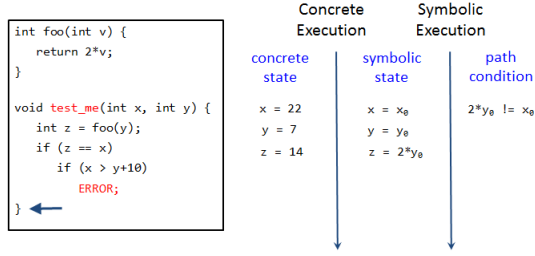
Hình 2.1: DSE khởi tạo các giá trị đầu vào

Ở dòng đầu tiên, số nguyên z được gán bằng hàm $foo(y)$. Điều này có nghĩa là $z = 14$, và ở trạng thái tượng trưng, biến $z = 2 * y_0$.



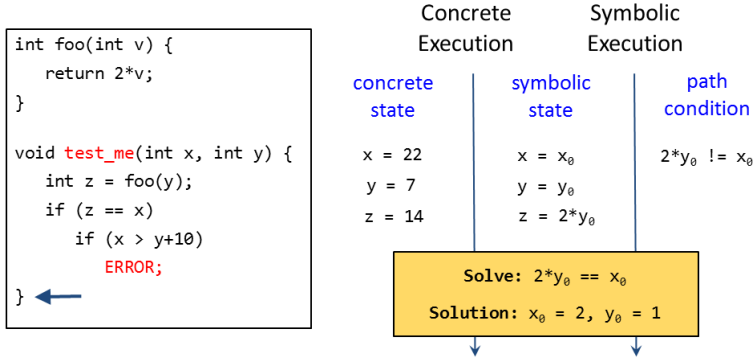
Hình 2.2: Số nguyên Z được gán bằng hàm foo(y)

Tại nhánh $z == x$, DSE nhận biết giá trị của z không bằng giá trị của x và lưu trữ ràng buộc này là $z \neq x$, và giá trị tượng trưng của đường này là: $2 * y_0 \neq x_0$. DSE đi theo nhánh *false* dẫn đến kết thúc chương trình.



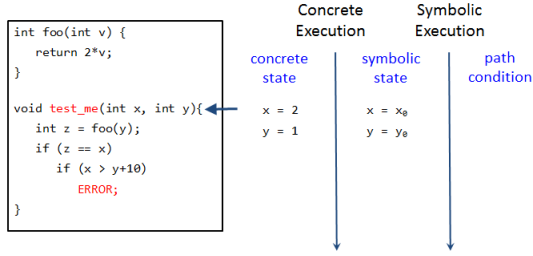
Hình 2.3: Với điều kiện $z == x$, giá trị của $z \neq x$ nên DSE chạy theo nhánh *false*

Sau khi kết thúc chương trình, DSE sẽ quay trở lại điểm nhánh gần nhất và chọn nhánh *true*. Với mục đích này, nó phủ định ràng buộc được thêm gần nhất trong điều kiện đường dẫn $2 * y_0 \neq x_0$ thành $2 * y_0 = x_0$. Để thỏa mãn ràng buộc $2 * y_0 = x_0$ thì hai số nguyên này sẽ là $x_0 = 2$ và $y_0 = 1$.



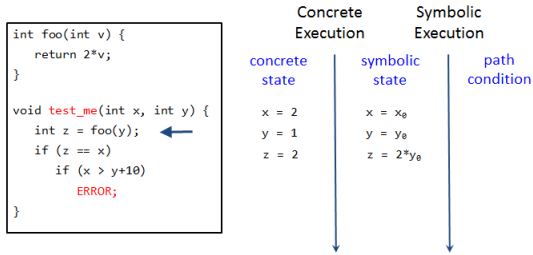
Hình 2.4: Các giá trị DSE sinh ra sau khi thực thi chương trình lần 1

Sau đó, DSE khởi động lại hàm `test_me`, lần này nó gọi các giá trị đầu vào cụ thể với giá trị: $x = 2$ và $y = 1$ được tạo ra bởi quá trình giải quyết ràng buộc trước đó. DSE tiếp tục theo dõi trạng thái các biến với $x = x_0$ và $y = y_0$.



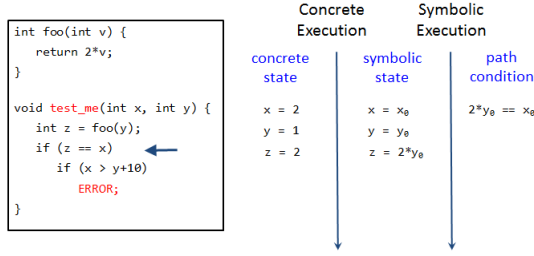
Hình 2.5: DSE khởi động lại hàm test_me

Sau khi thực hiện dòng đầu tiên, z có giá trị cụ thể 2 và giá trị biểu tượng $2 * y_0$.



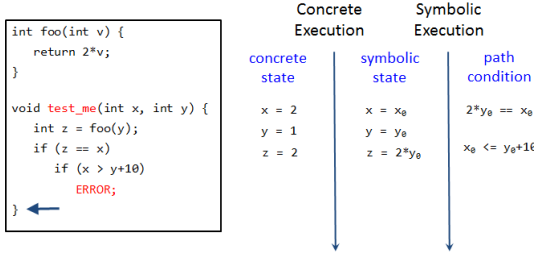
Hình 2.6: Thực hiện dòng đầu tiên $int z = foo(y)$

Ở dòng kế tiếp, chúng ta kiểm tra tình trạng nhánh $z == x$. Trong trường hợp này, điều kiện là đúng vì vậy điều kiện đường dẫn của chúng ta trở thành $2 * y_0 == x_0$. Sau đó DSE kiểm tra dòng tiếp theo của nhánh *true*.



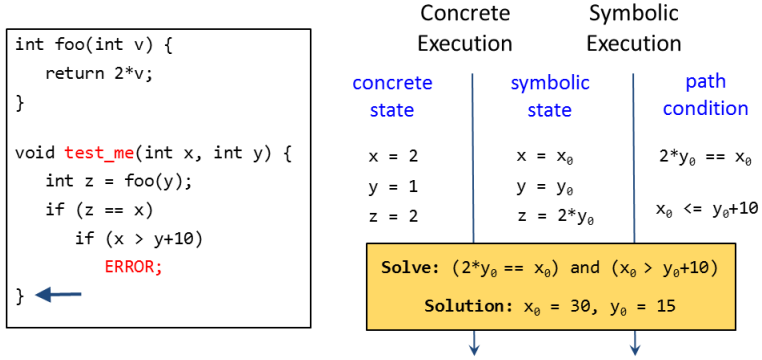
Hình 2.7: DSE thực hiện điều kiện đường dẫn *true*

Tại điểm nhánh tiếp theo, x có giá trị cụ thể là 2 và $y + 10$ có giá trị cụ thể là 11, vì vậy DSE lấy nhánh *false*, kết thúc chương trình. Thêm ràng buộc tượng trưng $x_0 \leq y_0 + 10$ vào điều kiện đường dẫn, đây là sự phủ định của điều kiện nhánh mà DSE phát hiện là *false*.



Hình 2.8: DSE lấy nhánh *false* kết thúc chương trình

Vì DSE đã đến cuối chương trình, nó phủ nhận ràng buộc vừa được thêm gần nhất trong điều kiện đường dẫn để có được $x_0 > y_0 + 10$, và sau đó nó vượt qua các ràng buộc $2 * y_0 == x_0$ và $x_0 > y_0 + 10$. Để thỏa những ràng buộc này, DSE trả về $x_0 = 30$ và $y_0 = 15$.

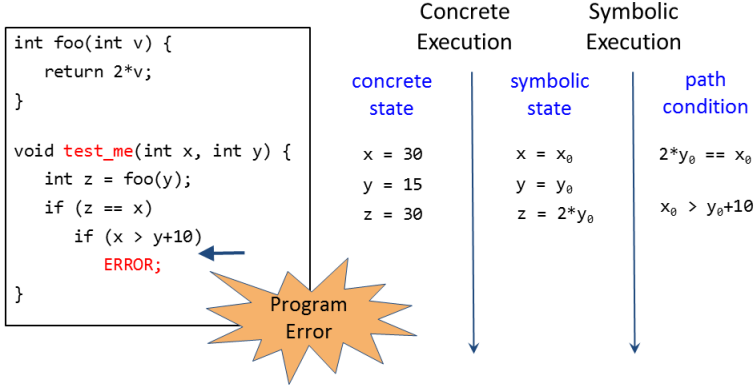


Hình 2.9: Các giá trị DSE sinh ra sau khi thực thi chương trình lần 2

Bây giờ, DSE chạy hàm `test_me` một lần nữa, lần này với các đầu vào $x = 30$ và $y = 15$. Trạng thái biểu tượng các biến bắt đầu $x = x_0$ và $y = y_0$. z được gán giá trị cụ thể là 30, trong khi giá trị tượng trưng của nó là $2 * y_0$ như những lần chạy trước.

Khi tới điều kiện rẽ nhánh $z == x$, DSE nhận thấy đây là điều kiện *true*, vì vậy DSE thêm điều kiện tượng trưng $2 * y_0 == x_0$.

Sau đó tại điểm nhánh tiếp theo, với $x > y + 10$ vì vậy DSE thêm ràng buộc tượng trưng mới $x_0 > y_0 + 10$. Nhánh này dẫn đến *ERROR*, tại thời điểm đó chúng ta đã xác định được đầu vào cụ thể làm cho chương trình dẫn đến *ERROR* là: $x = 30$ và $y = 15$.



Hình 2.10: Các giá trị DSE sinh ra sau khi thực thi chương trình lần 3

Kết quả, sau 3 lần chạy chương trình, DSE tạo ra được các cặp giá trị đầu vào có thể duyệt hết các nhánh của chương trình `test_me` đó là: $[22, 7]$, $[2, 1]$, $[30, 15]$

2.2 Hành vi của chương trình

Để định lượng hai chương trình tương tự nhau, chúng ta nghiên cứu một số định nghĩa liên quan đến hành vi của hai chương trình như sau:

Thực thi chương trình

Định nghĩa 1 Cho P là một chương trình, I là tập hợp các trị đầu vào của P và O là tập hợp các giá trị đầu ra của P . Thực thi chương trình P là ánh xạ $exec : P \times I \rightarrow O$. Với giá trị đầu vào $i \in I$, sau khi thực thi P trên i ta có giá trị đầu ra tương ứng $o \in O$ và ký hiệu $o = exec(P, i)$.

Độ tương đương về hành vi

Dựa trên định nghĩa về thực thi chương trình, chúng ta tìm hiểu thế nào là độ tương đương về hành vi giữa hai chương trình thông qua hai chương trình minh họa sau:

Mã lệnh 2.3: 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 2.4: 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;
}
```

Mã lệnh 2.3 và 2.4 có tham số đầu vào cùng kiểu giá trị *int*. Mã lệnh 2.3 sử dụng cấu trúc **switch...case**, mã lệnh 2.4 sử dụng cấu trúc **if...else** để kiểm tra giá trị đầu vào *x*. Mặc dù cú pháp sử dụng trong hai chương trình là khác nhau nhưng cách thức xử lý trả về kết quả *y* là như nhau. Từ đó, chúng ta có thể định nghĩa thế nào là độ tương đương hành vi giữa hai chương trình như sau:

Định nghĩa 2 (Độ tương đương về hành vi) Cho P_1 và P_2 là hai chương trình có cùng miền các giá trị đầu vào I . Hai chương trình này được gọi là tương đương khi và chỉ khi thực thi của chúng giống nhau trên mọi giá trị đầu vào trên I , ký hiệu là $exec(P_1, I) = exec(P_2, I)$.

Sự khác biệt về hành vi

Để tìm hiểu sự khác biệt về hành vi của hai chương trình, chúng ta tìm hiểu hai mã lệnh sau:

Mã lệnh 2.5: Chương trình P_1

```
using System;
public class Program
{
    public static int P1(int x)
    {
        return x - 10;
    }
}
```

Mã lệnh 2.6: Chương trình P_2

```
using System;
public class Program
{
    public static int P2(int x)
    {
        return x + 10;
    }
}
```

Mã lệnh 2.5 và Mã lệnh 2.6 của Chương trình P_1 và P_2 , cả hai Chương trình có miền giá trị đầu vào cùng kiểu *int*, giá trị trả về của Chương trình P_1 là $x - 10$, giá trị trả về của Chương trình P_2 là $x + 10$. Với mọi giá trị của x được thực thi trên cả hai chương trình P_1 và P_2 kết quả

trả về sẽ không giống nhau. Mặc dù cả hai Chương trình P_1 và P_2 có miền giá trị đầu vào như nhau, nhưng hành vi của hai Chương trình hoàn toàn khác nhau. Qua đó, chúng ta có thể định nghĩa sự khác biệt về hành vi như sau:

Định nghĩa 3 (Sự khác biệt hành vi) Cho P_1 và P_2 là hai chương trình có cùng một miền các giá trị đầu vào I . Hai chương trình này được xem là có sự khác biệt về hành vi khi và chỉ khi thực thi của chúng khác nhau trên mọi giá trị đầu vào I , ký hiệu là $exec(P_1, I) \neq exec(P_2, I)$.

Độ tương tự hành vi

Để hiểu thế nào là tương tự hành vi, chúng ta phân tích hai Mã lệnh sau:

Mã lệnh 2.7: Chương trình P_1

```
using System;
public class Program
{
    public static int P1(int x)
    {
        if (x >= 0 && x <= 100)
            return x + 10;
        else
            return x;
    }
}
```

Mã lệnh 2.8: Chương trình P_2

```
using System;
public class Program
{
    public static int P2(int x)
    {
        if (x >= 0 && x <= 100)
            return x + 10;
        else
            return -1;
    }
}
```

Với Mã lệnh 2.7 và Mã lệnh 2.8 của hai Chương trình P_1 và P_2 , chúng ta thấy cả hai Chương trình có giá trị đầu vào cùng kiểu dữ liệu là *int*, nếu giá trị đầu vào của biến x nằm trong khoảng 0 đến 100 thì giá trị trả về của cả hai Chương trình đều bằng nhau là $x + 10$. Ngược lại, giá trị đầu vào của biến x nằm ngoài khoảng 0 đến 100, thì giá trị trả về của Chương trình P_1 là x và giá trị trả về của Chương trình P_2 là -1 . Hai Chương trình P_1 và P_2 tuy có kiểu dữ liệu đầu vào như nhau nhưng kết quả đầu ra có thể giống nhau hoặc khác nhau tùy theo giá trị đầu vào của biến x . Dựa trên kết quả phân tích, chúng ta định nghĩa độ tương tự hành vi của Chương trình như sau:

Định nghĩa 4 (Độ tương tự hành vi) Cho P_1 và P_2 là hai chương trình có cùng miền giá trị đầu vào I , và I_s là tập con của I . Hai Chương trình được xem là tương tự hành vi khi thực thi chúng giống nhau trên

mọi giá trị đầu vào I_s , ký hiệu $exec(P_1, I_s) = exec(P_2, I_s)$ và khác nhau $\forall j \in I \setminus I_s$, ký hiệu $exec(P_1, j) \neq exec(P_2, j)$

2.3 Một số phép đo độ tương tự hành vi

Để đo độ tương tự về hành vi giữa hai chương trình, chúng ta có thể chạy từng giá trị đầu vào trong miền giá trị đầu vào của hai chương trình. Tỷ lệ giữa số lượng đầu vào thử nghiệm khi thực thi trên cả hai chương trình cho kết quả đầu ra giống nhau trên tổng số lượng đầu vào được thử nghiệm là độ tương tự hành vi giữa hai chương trình. Dựa trên cách tính tỷ lệ kết quả đầu ra của hai chương trình, chúng ta có một số phép đo độ tương tự hành vi như sau:

Phép đo lấy mẫu ngẫu nhiên (RS)

Kỹ thuật của phép đo **RS** là thực hiện lấy mẫu ngẫu nhiên giá trị đầu vào trên miền giá trị đầu vào của hai chương trình. Thực thi cả hai chương trình trên từng giá trị đầu vào, tiến hành so sánh giá trị kết quả đầu ra của cả hai chương trình. Tỷ lệ giữa tổng số mẫu đầu vào khi thực thi những mẫu này hai chương trình cho kết quả đầu ra có giá trị giống nhau, trên tổng số mẫu đầu vào được thử nghiệm là kết quả cho phép đo RS. Từ đó, chúng ta có định nghĩa phép đo **RS** như sau:

Định nghĩa 5 (Phép đo RS) Cho P_1 và P_2 là hai chương trình có cùng miền giá trị đầu vào I , I_s là tập con ngẫu nhiên của I , I_a là tập con I_s . Hai chương trình thực thi giống nhau với $\forall i \in I_a$, ký hiệu $exec(P_1, i) = exec(P_2, i)$ và hai chương trình thực thi khác nhau với $\forall j \in I_s \setminus I_a$, ký hiệu $exec(P_1, j) \neq exec(P_2, j)$. Chỉ số phép đo **RS** được định nghĩa là $M_{RS}(P_1, P_2) = |I_a| / |I_s|$.

Algorithm 2 Phép đo RS

P_1, P_2 : Là hai chương trình cần đo độ tương tự
 I : Miền giá trị đầu vào của P_1, P_2
 Set $I_s = \text{Random}(I)$ $\triangleright I_s$: Tập con ngẫu nhiên của I
 Set $I_a = \emptyset$
while $i \in I_s$ **do** \triangleright Vòng lặp dừng lại khi $\forall i \in I_s$ đã được thực thi
 if ($\text{exec}(P_1, i) = \text{exec}(P_2, i)$) **then**
 $I_a = I_a \cup i$
 $M_{RS}(P_1, P_2) = |I_a| / |I_s|$.

Phép đo **RS** là một phép đo đơn giản và hiệu quả để tính độ tương tự của hành vi. Khi miền giá trị của tham số đầu vào rất lớn hoặc vô hạn, phép đo **RS** thực hiện lấy mẫu ngẫu nhiên để tính toán độ tương tự của hành vi, kết quả đầu ra tương đối tốt và hợp lý so với hành vi thực tế của chương trình. Phép đo **RS** xử lý, tính toán độ tương tự hành vi dưới dạng hộp đen và không phân tích chương trình để tạo thử nghiệm, nên tốc độ xử lý nhanh và chiếm ít tài nguyên. Mặc khác, phép đo **RS** không phân tích chương trình để tạo đầu vào thử nghiệm nên phép đo **RS** có thể bỏ qua một vài tham số đầu vào thử nghiệm có thể được sử dụng để thực thi một số nhánh khác nhau giữa hai chương trình. Vì vậy, phép đo **RS** không phân biệt được các chương trình có một số hành vi khác nhau. Chúng ta phân tích Mã lệnh 3.7 và 3.8 sau để thấy được hạn chế của phép đo **RS**:

Mã lệnh 2.9: Chương trình P_1

```

public static int Y(string x)
{
    if (x == "XYZ")
        return 0;
    if (x == "ABC")
        return 1;
    return -1;
}
  
```

Mã lệnh 2.10: Chương trình P_2

```

public static int Y(string x)
{
    if (x == "ABC")
        return 1;
    return -1;
}
  
```

Chúng ta thấy đoạn Mã lệnh 2.9 và 2.10 của hai Chương trình P_1 và P_2 có cùng miền giá trị đầu vào là *string* x , cấu trúc mã lệnh hai chương trình gần như nhau. Nhưng Chương trình P_1 khác với Chương trình P_2 đó là sẽ trả kết quả về 0 nếu tham số đầu vào có giá trị là XYZ. Tỷ lệ phép đo **RS** lấy ngẫu nhiên giá trị đầu vào x trên miền giá trị đầu vào của hai chương trình có giá trị bằng XYZ là rất thấp, vì vậy khả

năng câu lệnh $if(x == "XYZ")return0$; của Chương trình P_1 có thể sẽ không được thực thi nên kết quả của phép đo **RS** sẽ ở mức tương đối so với hành vi thực tế của chương trình.

Phép đo tương trưng trên một chương trình (SSE)

Phép đo **SSE** là một phép đo dựa trên số lượng các nhánh đường đi của Chương trình mẫu, mỗi nhánh đường đi của Chương trình mẫu được xem là một hành vi của chương trình. Nếu chọn một giá trị đầu vào thử nghiệm cho một nhánh đường đi trong Chương trình thì các giá trị đầu vào thử nghiệm này sẽ khám phá hết các hành vi trong Chương trình mẫu. Do vậy, số phần tử trong tập các giá trị đầu vào thử nghiệm của phép đo **SSE** sẽ nhỏ hơn tập các giá trị đầu vào thử nghiệm được chọn theo phương pháp lấy ngẫu nhiên giá trị đầu vào.

Để tính độ tương tự hành vi của hai chương trình với phép đo **SSE**, chúng ta chọn Chương trình mẫu làm Chương trình tham chiếu và áp dụng kỹ thuật **DSE** để tạo ra các đầu vào thử nghiệm dựa trên Chương trình tham chiếu. Sau đó thực thi cả hai chương trình dựa trên các giá trị đầu vào thử nghiệm. Tỷ lệ số lượng các kết quả đầu ra giống nhau của cả hai chương trình trên tổng số các giá trị đầu vào thử nghiệm của Chương trình tham chiếu là kết quả của phép đo **SSE**. Qua đó, chúng ta có định nghĩa phép đo **SSE** như sau:

Định nghĩa 6 Cho P_1 và P_2 là hai chương trình có cùng miền giá trị đầu vào I , Chương trình P_1 là Chương trình tham chiếu, I_s là tập các giá trị đầu vào được tạo bởi DSE trên chương trình P_1 , và I_a là tập con I_s . Hai Chương trình thực thi giống nhau với $\forall i \in I_a$, ký hiệu $exec(P_1, i) = exec(P_2, i)$ và hai Chương trình thực thi khác nhau với $\forall j \in I_s \setminus I_a$, ký hiệu $exec(P_1, j) \neq exec(P_2, j)$. Chỉ số phép đo **SSE** được định nghĩa là $M_{SSE}(P_1, P_2) = |I_a| / |I_s|$.

Algorithm 3 Phép đo SSE

P_1, P_2 : Là hai chương trình cần đo tương tự
 I : Miền giá trị đầu vào của P_1, P_2
 P_1 : Là Chương trình tham chiếu
 Set $I_s = DSE(P_1)$ $\triangleright I_s$: Tập đầu vào của P_1 theo DSE
 Set $I_a = \emptyset$
while ($i \in I_s$) **do** \triangleright Vòng lặp dừng lại khi $\forall i \in I_s$ đã được thực thi
 if ($exec(P_1, i) = exec(P_2, i)$) **then**
 $I_a = I_a \cup i$
 $M_{SSE}(P_1, P_2) = |I_a| / |I_s|$.

Ngược lại với phép đo RS, phép đo SSE khám phá những đường đi khả thi khác nhau trong chương trình tham chiếu để tạo dữ liệu đầu vào của chương trình. Do đó, các đầu vào thử nghiệm này sẽ thực thi hết các đường đi của chương trình tham chiếu và có khả năng phát hiện được những chương trình cần tính có những hành vi khác so với Chương trình tham chiếu. Những phép đo SSE vẫn còn hạn chế, đó là phép đo SSE không xem xét đường đi của Chương trình cần phân tích để tạo các giá trị đầu vào thử nghiệm mà chỉ dựa vào các đầu vào thử nghiệm được phân tích từ Chương trình tham chiếu. Các đầu vào thử nghiệm này không nắm bắt được hết các hành vi của Chương trình cần phân tích, Chương trình cần phân tích có thể sẽ có những hành vi khác so với Chương trình tham chiếu. Một số chương trình có thể có những vòng lặp vô hạn phụ thuộc vào giá trị đầu vào nên SSE không thể liệt kê được tất cả các đường dẫn của chương trình. Chúng ta xem xét và phân tích 2 đoạn Mã lệnh 3.9 và 3.10 để thấy được hạn chế của phép đo SSE như sau:

Mã lệnh 2.11: Chương trình P_1

```
public static int sol(int x)
{
    int y = 0;
    switch (x)
    {
        case 1:
            y += 4;
            break;
        default:
            y = x - 100;
            break;
    }
    return y;
}
```

Mã lệnh 2.12: Chương trình P_2

```
public static int sub(int x)
{
    int y = 0;
    if (x == 1)
        return y += 4;
    if (x == 2)
        return y *= 4;
    else
        return y = x - 100;
}
```

Hai đoạn Mã lệnh 2.11 và Mã lệnh 2.12 của hai Chương trình P_1 và P_2 , chọn Chương trình P_1 làm Chương trình tham chiếu, sử dụng kỹ thuật **DSE** để phân tích chương trình P_1 ta được tập các giá trị đầu vào thử nghiệm là $(0, 1)$. Trong khi đó, phân tích Chương trình P_2 chúng ta được tập các giá trị đầu vào thử nghiệm của Chương trình P_2 là $(0, 1, 2)$. Do đó, chúng ta thấy tập giá trị đầu vào thử nghiệm do phép đo **SSE** tạo ra thiếu giá trị đầu vào thử nghiệm 2 để có thể thực thi hết các đường đi của Chương trình P_2 .

Kỹ thuật thực thi chương trình kết hợp (PSE)

Để giải quyết giới hạn của phép đo **SSE** khi tạo ra tập các giá trị đầu vào thử nghiệm không thực thi hết các các đi của Chương trình cần phân tích. Phép đo **PSE** giải quyết giới hạn của phép đo **SSE** bằng cách tạo một Chương trình kết hợp giữa Chương trình cần phân tích với Chương trình tham chiếu. Dựa trên Chương trình kết hợp sử dụng kỹ thuật **DSE** để tạo ra đầu vào thử nghiệm cho cả hai chương trình, các đầu vào thử nghiệm này bao gồm các đầu vào thử nghiệm đúng và không đúng. Các đầu vào thử nghiệm đúng là những giá trị khi thực thi trên cả hai chương trình sẽ cho kết quả đầu ra như nhau, ngược lại các đầu vào thử nghiệm không đúng là những giá trị khi thực thi trên cả hai chương trình sẽ cho kết quả khác nhau. Do đó, phép đo **PSE** được tính bằng tỷ lệ các giá trị đầu vào thử nghiệm đúng trên tổng số các giá trị đầu vào được thử nghiệm.

Mã lệnh 2.13: Chương trình kết hợp PSE

```
public int P_3 (int number)
{
    if (Program1(args) == Program2(args))
        return 1;
    return 0;
}
```

Dựa trên cách thức hoạt động của phép đo **PSE**, chúng ta có định nghĩa phép đo **PSE** như sau:

Định nghĩa 7 (Phép đo PSE) Cho P_1 và P_2 là hai Chương trình có cùng miền giá trị đầu vào I . P_3 là Chương trình kết hợp của P_1 và P_2 , ký hiệu ($exec(P_1, I) = exec(P_2, I)$). I_s là tập các giá trị đầu vào được tạo bởi **DSE** từ Chương trình P_3 , I_a là tập con I_s . Hai Chương trình P_1 và P_2 thực thi giống nhau khi Chương trình P_3 thực thi cho giá trị đúng với $\forall i \in I_a$, ký hiệu $exec(P_3, i) = T$, và $\nexists j \in I_s \setminus I_a$ thực thi Chương trình P_3 cho giá trị đúng. Chỉ số phép đo **PSE** được định nghĩa là $M_{PSE}(P_1, P_2) = |I_a| / |I_s|$.

Algorithm 4 Phép đo PSE

P_1, P_2 : Là hai chương trình cần đo tương tự

I : Miền giá trị đầu vào của P_1, P_2

P_3 : Là Chương trình kết hợp của P_1, P_2 , ký hiệu ($exec(P_1, I) = exec(P_2, I)$)

Set $I_s = DSE(P_3)$ $\triangleright I_s$: Tập đầu vào của P_3 theo DSE

Set $I_a = \emptyset$

while $i \in I_s$ **do** \triangleright Vòng lặp dừng lại khi $\forall i \in I_s$ đã được thực thi

if ($exec(P_1, i) = exec(P_2, i)$) **then**

$I_a = I_a \cup i$

$M_{PSE}(P_1, P_2) = |I_a| / |I_s|$.

Phép đo **PSE** đã cải thiện được hạn chế của phép đo **SSE** khi dữ liệu thử nghiệm được tạo ra từ trên Chương trình kết hợp, tập dữ liệu thử nghiệm có khả năng thực thi hết các nhánh đường đi của Chương trình tham chiếu và Chương trình cần tính. Tuy nhiên, phép đo **PSE** cũng có hạn chế trong quá trình xử lý các vòng lặp lớn hoặc vô hạn. Để giảm bớt hạn chế này, chúng ta có thể giới hạn miền đầu vào hoặc đếm

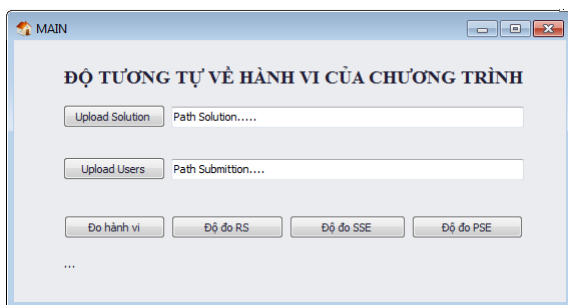
số vòng lặp của các Chương trình. Ngoài ra, phép đo **PSE** khám phá đường dẫn của Chương trình kết hợp nên quá trình xử lý sẽ tốn thời gian và tài nguyên hơn so với phép đo **SSE** khi chỉ khám phá đường dẫn của Chương trình tham chiếu.

Chương 3

PHẦN KẾT LUẬN

3.1 Đánh giá kết quả thực nghiệm

Trong nội dung Chương 2 của đề tài, tôi đã trình bày một số lý thuyết về kiểm thử phần mềm, các kỹ thuật sinh dữ liệu thử nghiệm, và các kỹ thuật đo độ tương tự hành vi chương trình **RS**, **SSE**, **PSE**. Áp dụng những lý thuyết trên, tôi thực nghiệm bằng cách xây dựng một ứng dụng, mô tả quá trình hoạt động của các kỹ thuật đo, và đánh giá kết quả các kỹ thuật đo, kết quả thực nghiệm đạt được như sau:



Hình 3.1: Giao diện màn hình chính

Hai nút đầu tiên, cho phép chúng ta chọn file chương trình tham chiếu và files cần tính độ tương tự. Bên dưới là các nút chức năng đo hành

vi, và tính toán độ tương tự hành vi của chương trình theo các độ đo khác nhau RS, SSE và PSE.

Một số files mẫu chương trình tham chiếu và chương trình cần tính độ tương tự (chương trình của sinh viên) có nội dung như sau:

Mã lệnh 3.1: Chương trình tham chiếu

```
using System;
public class Program
{
    public static int Puzzle(int x)
    {
        int y = 0;
        switch (x)
        {
            case 1: y += 4; break;
            default: y = x - 100; break;
        }
        return y;
    }
}
```

Mã lệnh 3.2: Chương trình của sinh viên thứ nhất

```
using System;
public class Program
{
    public static int Puzzle(int x)
    {
        int y = 0;
        if (x == 1) return y += 4;
        if (x == 2) return y *= 4;
        else return y = x - 100;
    }
}
```

Mã lệnh 3.3: Chương trình của sinh viên thứ hai

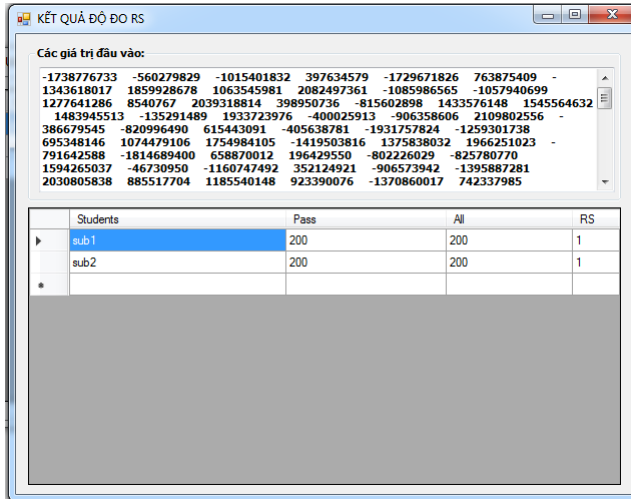
```
using System;
public class Program
{
    public static int Puzzle(int x)
```

```

{
    int y = 0;
    if (x == 1) return y += 4;
    if (x == 2) return y *= 4;
    if (x == 3) return y *= 6;
    else return y = x - 100;
}

```

Kết quả độ đo RS



The screenshot shows a window titled "KẾT QUẢ ĐỘ ĐO RS" (RS Measurement Results). It contains a list of input values and a table of results.

Các giá trị đầu vào:

-1738776733	-560279829	-1015401832	397634579	-1729671826	763875409
1343618017	1859928678	1063545981	2082497361	-1085986565	-1057940699
1277641286	8540767	2039318814	398950736	-815602898	1433576148
1483945513	-135291489	1933723976	-400025913	-906358606	2109802556
386679545	-820996490	615443091	-405638781	-1931757824	-1259301738
695348146	1074479106	1754984105	-1419503816	1375838032	1966251023
791642588	-1814689400	658870012	196429550	-802226029	-825780770
1594265037	-46730950	-1160747492	352124921	-906573942	-1395887281
2030805838	885517704	1185540148	923390076	-1370860017	742337985

	Students	Pass	All	RS
▶	sub 1	200	200	1
	sub2	200	200	1
*				

Hình 3.2: Kết quả độ đo RS

Dựa vào kết quả độ đo RS ở trên, chúng ta có danh sách các giá trị đầu vào được sinh ngẫu nhiên từ miền giá trị đầu vào của các chương trình. Hai chương trình của sinh viên được đánh giá là tương đương với chương trình tham chiếu và đều có kết quả là 1. Trong khi đó, cấu trúc điều kiện của Chương trình tham chiếu *case 1*: $y += 4$; *break*;; cấu trúc điều kiện của sinh viên thứ nhất *if* ($x == 2$) *return* $y *= 4$; và cấu trúc điều khiển của sinh viên thứ hai *if* ($x == 2$) *return* $y *= 4$; và *if* ($x == 3$) *return* $y *= 6$; mã lệnh trong chương trình của hai sinh viên đều có hành vi khác biệt so với mã lệnh trong Chương trình tham chiếu. Vì

vậy, kết quả độ đo **RS** cho kết quả ở mức tương đối, giá trị thử nghiệm không phủ hết các đường đi trong chương trình của hai sinh viên.

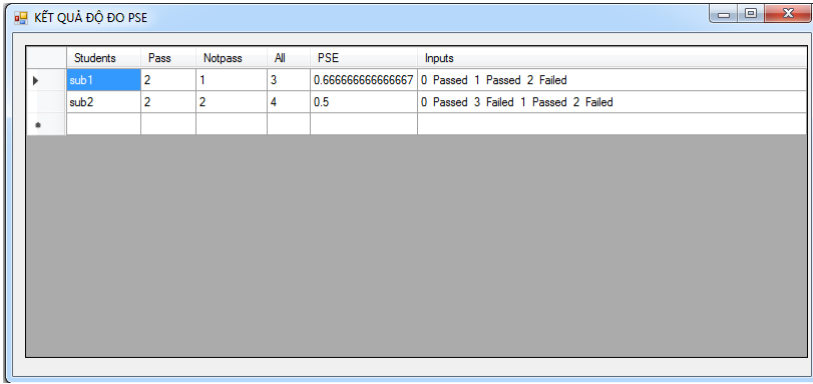
Kết quả độ đo SSE

Students	Pass	All	SSE
sub1	2	3	0.6666666666666667
sub2	2	3	0.6666666666666667

Hình 3.3: Kết quả độ đo SSE

Phân tích Chương trình tham chiếu với kỹ thuật DSE chúng ta có các giá trị đầu vào thử nghiệm lần lượt là 0, 1, 2. Các giá trị đầu vào thử nghiệm này khi thực thi trên chương trình của hai sinh viên cho 2 kết quả đầu giống nhau trên tổng số 3 kết quả so với Chương trình tham chiếu, đạt tỷ lệ 0,66. Kết quả độ đo SSE cho chúng ta một kết quả chính xác hơn kết quả độ đo RS. Những phép đo SSE lại khi không xem xét các hành vi trong chương trình của sinh viên, không tạo ra được các giá trị đầu vào thử nghiệm có khả năng phủ hết các nhánh đường đi trong Chương trình của sinh viên. Trong khi Chương trình của sinh viên hiện có hành vi khác biệt so với Chương trình tham chiếu.

Kết quả độ đo PSE



Students	Pass	Notpass	All	PSE	Inputs
sub1	2	1	3	0.666666666666667	0 Passed 1 Passed 2 Failed
sub2	2	2	4	0.5	0 Passed 3 Failed 1 Passed 2 Failed
*					

Hình 3.4: Kết quả độ đo PSE

Kết quả độ đo PSE đã thay đổi so với kết quả độ đo SSE. PSE tạo ra 4 giá trị đầu vào thử nghiệm trên Chương trình kết hợp giữa Chương trình tham chiếu và Chương trình của sinh viên thứ 2, lần lượt là 0, 1, 2, 3. Kết quả phép đo PSE cho sinh viên thứ nhất đạt 0.66, và kết quả phép đo PSE của sinh viên thứ hai đạt 0,5, kết quả này chính xác nếu chúng ta so sánh với việc phân tích thủ công. Các giá trị thử nghiệm đáp ứng được mục tiêu phủ hết các nhánh trong chương trình của hai sinh viên và Chương trình tham chiếu.

3.2 Khả năng ứng dụng

Các kỹ thuật đo độ tương tự trình bày trong đề tài này có thể áp dụng được trong nhiều lĩnh vực, nhưng chủ yếu tập trung vào giáo dục, chương trình đào tạo lập trình viên, hay đào tạo kỹ sư phần mềm... Một số ứng dụng thực tế có thể phát triển trong tương lai như:

Đánh giá tiến bộ trong lập trình: Theo dõi sự tiến bộ trong học tập là một việc quan trọng, mà ngay cả với giảng viên và sinh viên. Có nhiều tiêu chí đánh giá sự tiến bộ trong học tập của sinh viên, trong đó tiêu chí về điểm số là một trong những tiêu chí cơ bản nhất. Một bảng điểm thống kê điểm số, thành tích học tập của sinh viên sẽ thể hiện được sự tiến bộ của sinh viên trong học tập. Một ứng dụng hỗ trợ chấm điểm, lưu trữ, thống kê và đánh giá điểm số của sinh viên là sẽ

là một công cụ hỗ trợ đắc lực cho giảng viên trong công tác quản lý của mình. Nếu số liệu thống kê kết quả các bài kiểm tra của sinh viên ngày càng cao, chứng tỏ sinh viên nắm được nội dung và kiến thức của chương trình đào tạo, và kết quả tốt sẽ là một động lực giúp cho sinh viên thêm tự tin, đam mê công việc học tập của mình. Ngược lại, nếu một sinh viên có điểm số ngày càng thấp đi, chứng tỏ sinh viên đang có vấn đề trong kiến thức của mình, lúc này tốt nhất sinh viên nên dừng lại và kiểm tra xem vấn đề mình đang gặp phải.

Xếp hạng tự động: Công việc chấm điểm, phân loại và xếp hạng các bài kiểm tra của sinh viên cũng là một công việc tốn không ít công sức của giảng viên. Để giảm bớt gánh nặng cho giảng viên, chúng ta có thể sử dụng kết quả các phép đo trên từng bài tập của sinh viên như một phương pháp hỗ trợ công việc chấm điểm của từng sinh viên. Sự giống nhau về hành vi giữa chương trình của sinh viên và Chương trình tham chiếu có thể là một yếu tố để phân loại sinh viên. Độ tương tự càng cao thì điểm số càng cao, các chỉ số này dựa hoàn toàn trên ngữ nghĩa của chương trình. Cách tiếp cận này giải quyết được các giới hạn trong trường hợp Chương trình của sinh viên giống với Chương trình tham chiếu, nhưng khác nhau về ngữ nghĩa. Các kết quả trong việc xếp hạng tự động sẽ giúp tiết kiệm được thời gian và giảng viên có thể đưa ra giải pháp giúp những sinh viên có điểm số thấp khắc phục được hạn chế đang gặp phải.

Gợi ý giải pháp lập trình: Thông thường, sinh viên thường viết code mới thực hiện chạy chương trình, lúc này sinh viên mới biết được kết quả đoạn code vừa thực hiện. Để hỗ trợ sinh viên viết code được tốt hơn, nếu như có một công cụ hỗ trợ kiểm tra theo thời gian thực và gửi thông báo lỗi nếu sinh viên viết code sai cú pháp hoặc chương trình bị lỗi không thể thực thi được. Ngoài ra, công cụ sẽ gợi ý giải pháp lập trình cho sinh viên bằng hình thức tự động tính toán thông báo kết quả các tham số đầu vào và đầu ra của chương trình so với chương trình được tham chiếu, đưa ra các số liệu về độ tương tự hành vi của chương trình.

3.3 Kết luận

Qua quá trình nghiên cứu và triển khai thực nghiệm, trong tương lai đề tài có thể phát triển thành một ứng dụng hoàn chỉnh với việc bổ sung và hoàn thiện thêm một số chức năng như: Cải tiến các kỹ thuật đo, kết hợp với kỹ thuật DSE để có kết quả các phép được chính xác

hơn, nhạy hơn. Tiếp tục nghiên cứu, phát triển trên các ngôn ngữ khác như Java, C++ ... và chạy được trên nhiều nền tảng PC, Web. Phát triển thêm một số tính năng mới như đánh giá, xếp loại tự động hay gợi ý giải pháp lập trình...

Các kỹ thuật đo độ tương tự hành vi của chương trình không chỉ giúp quá trình giảng dạy của giảng viên được thuận lợi hơn, tiết kiệm được thời gian cũng như công sức trong công tác quản lý. Ngoài ra, sinh viên có được một môi trường tốt để tự rèn luyện, nâng cao các kỹ năng lập trình của bản thân. Việc tạo động lực giúp sinh viên có sự hứng thú và đam mê lập trình là rất cần thiết. Một khi sinh viên có tư duy và kỹ năng lập trình tốt, sinh viên sẽ tự tin vào năng lực của bản thân để tiếp tục phát phát triển sự nghiệp sau khi ra trường.

Để thực hiện đề tài, tôi đã thực hiện nghiên cứu nhiều vấn đề, như nghiên cứu kiểm thử phần mềm, sinh ngẫu nhiên dữ liệu thử, hay kỹ thuật DSE một kỹ thuật được ứng dụng trong công cụ PEX của Microsoft để giải quyết các ràng buộc sinh ra các tham số đầu vào thử nghiệm có độ phủ cao. Nghiên cứu các phép đo RS, SSE, PSE để đo độ tương tự hành vi của chương trình. Trên cơ sở lý thuyết các phép đo độ tương tự hành vi của chương trình, xây dựng một công cụ để minh họa cho các phép đo. Kết quả đạt được từ các phép đo là tương đối tốt, tạo ra nhiều hướng có thể phát triển thêm trong tương lai.

Trong quá trình thực hiện đề tài, bản thân tôi cũng gặp rất nhiều khó khăn như: Lượng kiến thức cơ sở cần phải nghiên cứu tương đối nhiều; bản thân thiếu kinh nghiệm trong việc thực hiện các đề tài... Tuy nhiên, nhờ sự động viên và giúp đỡ của bạn bè, gia đình, cùng với sự tận tình hướng dẫn của thầy TS. Phạm Văn Việt tôi đã hoàn thành luận văn, đáp ứng được yêu cầu đã đề ra.