CHÀO MỪNG NGÀY THÀNH LẬP TRƯỜNG 01/4/2018

MỘT CÁCH TIẾP CẬN TRONG GIẢI QUYẾT BÀI TOÁN TỰ ĐỘNG HÓA SINH DỮ LIỆU KIỂM THỬ PHẦN MỀM

AN APPROACH TO AUTOMATED TEST DATA GENERATION

PHẠM ĐỨC TOÀN[1], PHAN NGUYÊN HẢI[2], PHẠM THỊ PHƯƠNG ANH[3]

[1]Phòng Tổ chức - Hành chính, Trường ĐHHH Việt Nam [2] Học viện Kỹ thuật Quân sự

[3] Viện Khoa học và Công nghệ Quân sự

Tóm tắt

Bài báo nghiên cứu lĩnh vực tự động hóa quy trình phát triển phần mềm, bài toán sinh dữ liệu kiểm thử tự động, hướng giải quyết cùng các khó khăn. Đề xuất cách tiếp cận biểu diễn thuật toán dựa trên khái niệm đồ thị, ứng dụng vào giải quyết bài toán sinh dữ liệu kiểm thử tự động.

Từ khóa**: Tự động hóa phát triển phần mềm, sinh dữ liệu kiểm thử, biểu diễn thuật toán.* ***Abstract

In this paper, the automated software development issue and test data generation problem are presented with solutions and challenges. Approach to represent algorithms based on graph concepts is proposed and applied to solving the problem of automated test data generation.

***Keywords**: Automated software development, test data generation, expressing algorithms.*

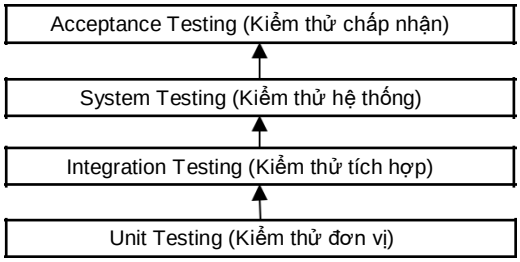
1. Bài toán tự động hóa phát triển phần mềm và tự động hóa kiểm thử phần mềm

Quy trình phát triển phần mềm thông thường gồm các giai đoạn: Xác định yêu cầu, thiết kế, lập trình, kiểm thử, tích hợp triển khai. Tùy thuộc vào từng dự án cụ thể mà các giai đoạn này có thể tuần tự hoặc lặp lại.

Hiện nay trong lĩnh vực tự động hóa phát triển phần mềm, các bài toán đang được quan tâm là sinh mã nguồn tự động và sinh dữ liệu kiểm thử tự động.

Với bài toán sinh mã nguồn tự động, đầu vào là bản thiết kế, đầu ra là biểu diễn của bản thiết kế bằng cú pháp của một ngôn ngữ lập trình nào đó (nói cách khác là mã nguồn).

Với bài toán sinh dữ liệu kiểm thử (test data generation), đầu vào là các bản đặc tả (yêu cầu, thiết kế), mã nguồn, đầu ra là dữ liệu đầu vào của phần mềm mà cần kiểm thử tại đó. Khi có đầu vào cần kiểm thử, thì có thể xác định đầu ra chuẩn của phần mềm tại đầu vào đó theo mô tả của phần mềm (test oracle), cặp giá trị đầu vào và đầu ra chuẩn hình thành lên một trường hợp kiểm thử (testcase) để người kiểm thử (tester) thực hiện kiểm thử. Việc kiểm thử thông thường có nhiều cấp độ, như Unit Testing để kiểm thử các đơn vị mã nguồn, Integration Testting để kiểm thử việc tích hợp các đơn vị mã nguồn, System Testing và Acceptance Testing để kiểm thử toàn bộ hệ thống phần mềm (Hình 1). Việc sinh dữ liệu tự động thường áp dụng cho mức Unit Testing và cách tiếp cận truyền thống trong bài toán sinh dữ liệu kiểm thử hiện nay là dựa vào mã nguồn (kiểm thử hộp trắng - whitebox testing). Trong khuôn khổ bài báo này, các tác giả tập trung vào bài toán tự động hóa sinh dữ liệu kiểm thử ở mức Unit Testing.



Hình 1. Các mức độ kiểm thử

Với bài toán sinh dữ liệu kiểm thử với phương pháp hộp trắng, cách thực hiện có thể gồm các bước chính như sau [1]:

- Xây dựng đồ thị chu trình từ mã nguồn (control flow graph - CFG), mỗi đỉnh của đồ thị là một dòng mã thực hiện việc tính toán hoặc xử lý điều kiện;
- Từ đồ thị xác định các đường đi từ điểm bắt đầu đến điểm kết thúc chương trình;

- Với mỗi đường đi, xác định các điều kiện đối với đầu vào để chương trình thực thi theo đường đi đó. Chọn một đại diện thỏa mãn điều kiện.

Với phương pháp này, chương trình có bao nhiêu đường thực thi thì có thể có bấy nhiêu testcase. Tuy nhiên, nếu chương trình có rẽ nhánh, vòng lặp thì số lượng đường thực thi có thể rất lớn. Trong trường hợp này, có thể chỉ cần tìm các đường cơ sở và từ các đường cơ sở đó xác định các dữ liệu kiểm thử.

Trong ba bước nêu trên, thì việc xây dựng đồ thị CFG từ mã nguồn và việc xác định các đường đi trên đồ thị là hoàn toàn có thể thực hiện tự động bằng lập chương trình. Tuy nhiên bước xác định điều kiện đối với đầu vào để chương trình thực thi theo một đường nhất định là một bài toán rất khó. Đây thực chất chính là bài toán giải quyết thỏa mãn ràng buộc (Constraint satisfaction problems - CSPs) [2, 3, 4]. Bài toán CSP được đặc trưng bởi bộ ba (*X, D, C*), với:

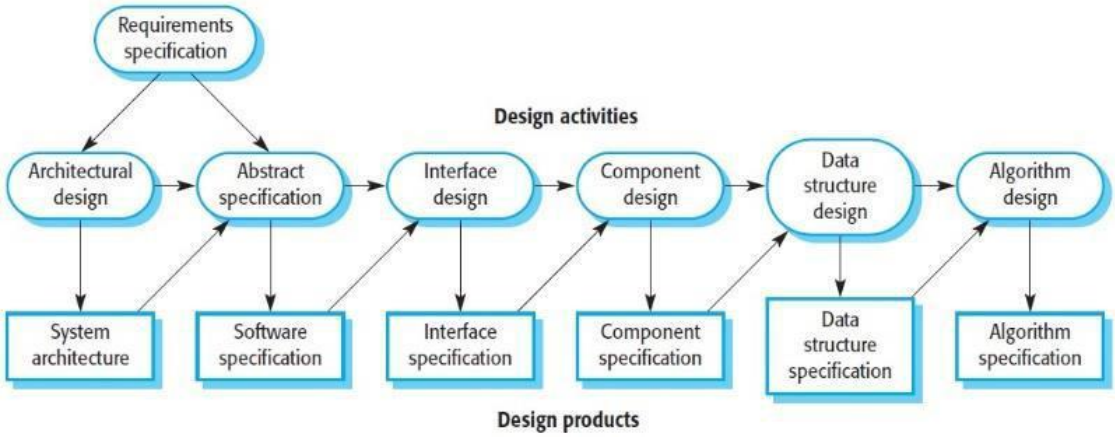
$X = \{X_1, X_2, \ldots, X_n\}$ là tập hợp các biến;

$D = \{D_1, D_2, \ldots, D_n\}$ tương ứng là các tập hợp miền giá trị của các biến *X*;

$C = \{C_1, C_2, \ldots, C_m\}$ là tập hợp các ràng buộc. *Ci* là các hàm của *X*, thường có dạng bất phương trình.

Bài toán đòi hỏi cần tìm các giá trị của *X* trong miền *D* để thỏa mãn các ràng buộc *C*. Đây là bài toán thuộc lớp NP-đầy đủ (NP-complete), bài toán này không có lời giải trong trường hợp chung. Để giải quyết bài toán này phục vụ cho mục đích sinh dữ liệu kiểm thử tự động, đã có một cộng đồng nghiên cứu đông đảo với rất nhiều đề xuất tiếp cận như quay lui (backtracking), lan truyền ràng buộc (contraint propagation), tìm kiếm cục bộ (local search), giải thuật di truyền, tiến hóa [5, 6],...
Tuy nhiên các đề xuất chỉ thích hợp cho một lớp bài toán nhất định, kết quả sinh dữ liệu kiểm thử phụ thuộc vào rất nhiều tham số, thời gian chạy lâu, và thường không có sự bảo đảm về mặt toán học. Nguyên nhân chung có thể kết luận là do quá ít thông tin thu được từ mã nguồn chương trình khi xây dựng đồ thị CFG cùng các ràng buộc, dẫn đến quá trình tìm kiếm lời giải cho bài toán CSP gặp khó khăn. Xuất phát từ thực tế này, trong bài báo đề xuất một cách tiếp cận cho việc sinh dữ liệu kiểm thử được dễ dàng hơn.

Để tiếp tục, trước hết chúng ta xem xét giai đoạn thiết kế phần mềm trong quy trình phát triển phần mềm. Giai đoạn thiết kế phần mềm gồm các bước được mô tả trên Hình 2 [7].



Hình 2. Các bước của giai thoạn thiết kế phần mềm

Trong sơ đồ trên Hình 2, hoạt động thiết kế cốt lõi chính là thiết kế thuật toán (algorithm design). Trong phương pháp thiết kế hướng cấu trúc, thiết kế thuật toán thể hiện qua việc thiết kế thuật toán cho các hàm (chức năng) cơ sở. Trong phương pháp hướng đối tượng, thiết kế thuật toán thể hiện qua thiết kế thuật toán cho các phương thức của các lớp.

Bài toán sinh mã nguồn tự động có thể quy về sinh mã nguồn tự động từ thuật toán, từ đó có thể suy ra bài toán sinh dữ liệu kiểm thử tự động cũng có thể quy về các bài toán sinh mã nguồn tự động từ thuật toán (thực chất, mã nguồn chương trình chính là việc biểu diễn thuật toán bằng cú pháp một ngôn ngữ lập trình nào đó). Vấn đề tiếp theo cần quan tâm đến, đó là xem xét việc biểu diễn thuật toán trong bản thiết kế.

2. Vấn đề biểu diễn thuật toán

Trong lý thuyết thuật toán, thuật toán có thể biểu diễn bằng các cách truyền thống là ngôn ngữ tự nhiên, sơ đồ khối, giả mã (pseudo code),... Với các cách biểu diễn này thì chỉ có con người

mới hiểu được thuật toán, và việc xây dựng chương trình, xác định dữ liệu kiểm thử cũng phải do con người thực hiện. Nói một cách khác là với các cách biểu diễn thuật toán truyền thống thì việc tự động sinh mã nguồn, tự động sinh dữ liệu kiểm thử bằng máy tính là không thể thực hiện được. Để tự động sinh mã nguồn, tự động sinh dữ liệu kiểm thử cần phải biểu diễn thuật toán theo các cách mới để máy tính có thể hiểu được.

Hiện nay, một trong những cách tiếp cận trong biểu diễn thiết kế là dùng ngôn ngữ XML [8]. Để có thể đưa bản thiết kế (thuật toán) vào máy tính, bản thiết kế được chuyển từ dạng biểu diễn truyền thống sang biểu diễn bằng XML, với biểu diễn XML thì có thể xây dựng được các phương pháp sinh mã nguồn và sinh dữ liệu kiểm thử tự động. Tuy nhiên, XML là ngôn ngữ quá mạnh mẽ, quá đa dạng nên việc đưa ra những bộ quy tắc chung trong việc chuyển đổi từ cách biểu diễn truyền thống sang XML cũng chưa có sự thống nhất.

Trong bài báo này, các tác giả đề xuất sử dụng khái niệm đồ thị để biểu diễn thuật toán. Việc biểu diễn thuật toán bằng đồ thị là hoàn toàn khả thi, thực chất sơ đồ khối cũng có thể coi là dạng biểu diễn trực quan của đồ thị. Ngoài ra, với đồ thị đã có rất nhiều lý thuyết đã được xây dựng về việc biểu diễn, lưu trữ và xử lý trên máy tính, nên hoàn toàn có hy vọng về khả năng trích rút các thông tin cần thiết từ thuật toán để phục vụ cho bài toán sinh tự động mã nguồn hay dữ liệu kiểm thử từ thuật toán.

Ý tưởng biểu diễn thuật toán bằng đồ thị không phải là mới. Tuy nhiên, cũng giống như XML, chuyển đổi từ biểu diễn truyền thống sang biểu diễn bằng đồ thị cũng có nhiều cách khác nhau.

3. Cách tiếp cận dựa trên đồ thị

Trong phạm vi bài báo, chúng ta sẽ xem xét bài toán sinh dữ liệu kiểm thử bằng biểu diễn đồ thị của thuật toán. Với lý thuyết kiểm thử cơ bản thì cũng phải cần đến biểu diễn chương trình bằng đồ thị CFG, đồ thị CFG thực chất chính là biểu diễn của thuật toán. Vì vậy, việc dùng đồ thị thuật toán để sinh dữ liệu kiểm thử là hoàn toàn khả thi, ngoài ra khi biểu diễn thuật toán bằng đồ thị trong giai đoạn thiết kế thì chúng ta sẽ nhận được nhiều thông tin hơn cho việc sinh dữ liệu kiểm thử.

Việc xây dựng đồ thị thuật toán do người thiết kế thực hiện, có thể tiến hành theo hai bước, bước một là xây dựng sơ đồ khối thuật toán, bước hai là chuyển sơ đồ khối về dạng biểu diễn đồ thị. Về cấu trúc tổng quát, thuật toán có thể biểu diễn bằng đồ thị có hướng G, mỗi đỉnh của đồ thị có thể là một phép toán hoặc một khối rẽ nhánh. Với các đỉnh phép toán, khi đi qua đỉnh này, có thể coi dữ liệu đầu vào được đưa qua một hàm biến đổi nào đó. Với các đỉnh rẽ nhánh, khi đi qua, dữ liệu đầu vào không bị biến đổi, nhưng được kiểm tra bằng một hàm mệnh đề nào đó. Tùy thuộc vào hàm mệnh đề mà có thể rẽ nhánh nọ hoặc nhánh kia. Và khi xác định thuật toán, đi kèm với mỗi đỉnh là thông tin về dữ liệu vào ra tại đỉnh đó, thông tin về phép toán, về biểu thức điều kiện tại các khối rẽ nhánh. Những thông tin này người thiết kế hoàn toàn có thể cung cấp. Một số ví dụ về các thông tin:

- Miền dữ liệu vào/ra tại các đỉnh;
- Phân loại hàm xử lý, hàm mệnh đề (liên tục, rời rạc, tuyến tính, phi tuyến, biến nào là chính, biến nào là phụ thuộc;
- Ví dụ về các giá trị thỏa mãn hàm mệnh đề tại các đỉnh;
- Khả năng tuyến tính hóa, đơn giản hóa các hàm xử lý, hàm mệnh đề, hàm ngược của các hàm xử lý,...

Những thông tin này, trong những trường hợp nào đó sẽ giúp việc giải quyết bài toán thỏa mãn ràng buộc trong sinh dữ liệu kiểm thử được dễ dàng hơn. Một cách ngắn gọn, cách tiếp cận dựa trên đồ thị thuật toán này khác với cách tiếp cận truyền thống như sau:

- Cách truyền thống: Thiết kế (thuật toán) → Mã nguồn → Đồ thị → Sinh dữ liệu kiểm thử
- Cách đề xuất: Thiết kế (thuật toán) → Đồ thị + Thông tin → Sinh dữ liệu kiểm thử hoặc sinh mã nguồn.

4. Sinh dữ liệu kiểm thử tự động với biểu diễn dạng đồ thị của thuật toán

Chúng ta vận dụng ý tưởng của phương pháp hộp trắng để xác định các dữ liệu kiểm thử. Khi thuật toán được biểu diễn bằng đồ thị thì việc tự động xác định các đường đi từ điểm bắt đầu đến điểm kết thúc là bài toán kinh điển trong lý thuyết đồ thị và đã được giải quyết trọn vẹn ngay cả trong trường hợp đồ thị có chu trình (vòng lặp). Nghĩa là việc tìm đường đi bằng chương trình máy tính là hoàn toàn khả thi.

Tiếp theo với mỗi đường đi tìm được, cần tự động tìm dữ liệu đầu vào của thuật toán để thuật toán thực thi theo con đường đó. Ở đây, chúng ta chỉ xem xét ví dụ một trường hợp, giả sử thông

tin đi kèm tại các đỉnh đồ thị cho biết các hàm xử lý, hàm mệnh đề là các hàm tuyến tính và dữ liệu đầu vào của chương trình có dạng số. Ký hiệu:

Các giá trị đầu vào của chương trình là $X = \{X1, X2, ..., Xn\} \in R^n$;

Đường đi tìm được trên đồ thị gồm các hàm xử lý $X = F_i(X)$ và các hàm mệnh đề $G_j(X)>0$.

Khi đó việc tìm điều kiện của đầu vào X để chương trình đi theo đường đi này trở thành việc giải hệ bất phương trình tuyến tính:

(1)

Ở đây thứ tự các hàm xử lý, hàm mệnh đề trong hệ bất phương trình trùng với thứ tự của các đỉnh tương ứng trên đường đi của đồ thị.

Hệ bất phương trình có thể giải tự động được, thuật toán giải gồm các bước chính như sau:

Bước 1. Chuyển hệ về dạng hệ phương trình (2):

ở đây a_j là những số nào đó lớn hơn 0.

(2)

Bước 2. Tìm hạng *r* của ma trận biểu diễn hệ phương trình (2).

Bước 3. Cố định *n-r* biến (cho các giá trị cụ thể), giải hệ *r* phương trình để tìm *r* biến còn lại.

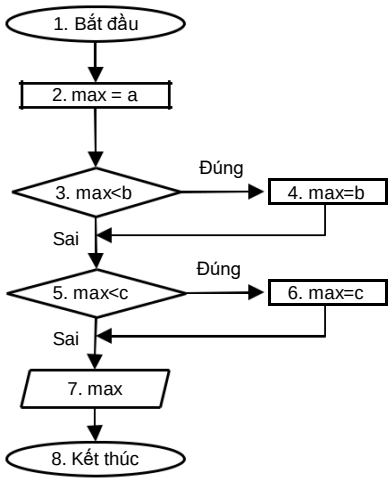
Nghĩa là, trong trường hợp này chúng ta hoàn toàn có thể tự động xác định được dữ liệu đầu vào để chương trình thực thi theo con đường nào đó.

Xét ví dụ sinh dữ liệu kiểm thử từ một thuật toán đơn giản, thuật toán tìm giá trị lớn nhất trong 3 số thực *a, b, c*. Thuật toán có thể như sau:

Bước 1. Đặt *max* = *a*;

Bước 2. So sánh *max* < *b*. Nếu đúng, gán *max* = *b*.

Bước 3. So sánh *max* < *c*. Nếu đúng, gán *max* = *c*. Đưa ra *max*. Kết thúc.



Hình 3. Đồ thị thuật toán

Đồ thị biểu diễn thuật toán có dạng như trên Hình 3. Với đồ thị như trên Hình 3, có thể có các đường thực thi từ đỉnh bắt đầu (*1. Bắt đầu*) đến đỉnh kết thúc (*8. Kết thúc*) như sau:

Đường 1: 1 → 2 → 3 → 5 → 7 → 8

Đường 2: 1 → 2 → 3 → 4 → 5 → 7 → 8

Đường 3: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8

Đường 4: 1 → 2 → 3 → 5 → 6 → 7 → 8

Chúng ta xem xét việc xác định đầu vào để chương trình thực thi theo đường 3. Với các đường khác việc xác định là tương tự.

Thông tin trên đồ thị cho thấy các hàm xử lý và hàm mệnh đề tại các đỉnh là hàm tuyến tính, hệ bất phương trình đối với đường 3 như sau:

$$max = a;\ max - b < 0;\ max = b;\ max - c < 0;\ max = c \quad (3)$$

Hệ này có thể chuyển thành hệ (4), bằng cách tính biến *max* lần lượt theo các biến đầu vào và thay các bất phương trình thành phương trình:

$$a - b = -1;\ b - c = -2 \quad (4)$$

Hạng của ma trận hệ (4) bằng 2, ta có thể cố định *a = 2*, suy ra *b = 3, c = 5* và nhận được một dữ liệu đầu vào để kiểm thử là (2,3,5), với đầu vào này thì đầu ra chuẩn là 5, nghĩa là nhận được testcase ((2,3,5),5). Tương tự đối với những đường thực thi khác. Nghĩa là chúng ta đã xác định được dữ liệu kiểm thử một cách tự động hoàn toàn theo tiêu chí bao phủ tất cả các đường thực thi.

5. Kết luận

Với cách biểu diễn thuật toán bằng đồ thị ngay trong giai đoạn thiết kế, chúng ta có thể thu được nhiều thông tin hơn về đồ thị nhận được, từ đó có thể giải quyết bài toán sinh dữ liệu kiểm thử tự động một cách dễ dàng hơn. Hướng phát triển tiếp theo là xem xét việc giải quyết bài toán sinh dữ liệu kiểm thử trong các trường hợp phức tạp và xem xét việc sinh mã nguồn tự động dựa trên đồ thị biểu diễn thuật toán.

TÀI LIỆU THAM KHẢO

[1] Arthur H. Watson and Thomas J. McCabe. "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric". NIST Special Publication 500-235, 1996.

[2] Dechter, Rina. Constraint processing. Morgan Kaufmann, 2003.

[3] Apt, Krzysztof. Principles of constraint programming. Cambridge University Press, 2003.

[4] Lecoutre, Christophe. Constraint Networks: Techniques and Algorithms. ISTE/Wiley, 2009.

[5] Phil McMinn. Search-based Software Test Data Generation: A Survey. Software Testing Verification and Reliability, Wiley. 2004.

[6] Chayanika Sharma, Sangeeta Sabharwal, Ritu Sibal. A Survey on Software Testing Techniques using Genetic Algorithm. IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 1, No 1, January 2013.

[7] Ian Sommerville. Software Engineering (9th Edition). Pearson Education, Addison-Wesley, 2010.

[8] Anshul, Sompal, Vikas Sheoran. Automatic Code Generation Using Uml To Xml Schema Transformation. International Journal of Advancement in Engineering Technology, Management and Applied Science, Volume 1, Issue 2, 2014.

Ngày nhận bài: 19/03/2018
Ngày nhận bản sửa: 05/04/2018
Ngày duyệt đăng: 09/04/2018