ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ
---ꕥ✧ꕥ---

Sinh viên: Nguyễn Thị Bích Ngọc

***ĐỀ TÀI*:**

KIỂM THỬ THEO MÔ HÌNH FSM VÀ ỨNG DỤNG CỦA NÓ TRONG WEB

KHÓA LUẬN TỐT NGHIỆP ĐẠI HỌC CHÍNH QUY

Ngành: Công nghệ phần mềm

HÀ NỘI, 2010
ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ
---ꕥ✧ꕥ---

Sinh viên: Nguyễn Thị Bích Ngọc

ĐỀ TÀI:

KIỂM THỬ THEO MÔ HÌNH FSM VÀ ỨNG DỤNG CỦA NÓ TRONG WEB

KHÓA LUẬN TỐT NGHIỆP ĐẠI HỌC CHÍNH QUY

Ngành: Công nghệ phần mềm

Cán bộ hướng dẫn: Đặng Văn Hưng

HÀ NỘI, 2010

KHÁI QUÁT

Trong tài liệu này đề cập đế các vấn đề về kiểm thử máy trạng thái hữu hạn (Finite-state Machines Testing hay viết tắt là FSMs testing) và ứng dụng của nó trong web.

Chương 1là tìm hiểu về FSMs

Chương 2 là tìm hiểu về kiểm thử với mô hình FSMs,

Chương 3 là tìm hiểu về tìm hiểu về dòng điều khiển, phụ thuộc dữ liệu và kiểm thử tương tác.

Chương 4 là kiểm thử với mô hình FSMs trong web.

Giáo viên hướng dẫn: Thầy Đặng Văn Hưng

Học viên thực hiện: Nguyễn Thị Bích Ngọc

FSM sử dụng các cấp trung gian để mô hình các chương trình hoạt động hay xử lý tạo sự cân bằng giữa các phần đơn giản và phức tạp. FSM diễn tả sự xử lý và hoạt động của các chương trình liên hợp

Kiểm thử FSM được ứng dụng rất nhiều trong lĩnh vực phần mềm bằng bảng chọn, các hệ thống được thiết kế bằng phương pháp hướng đối tượng.

LỜI MỞ ĐẦU

Đảm bảo phần mềm là một nhiệm vụ vô cùng quan trọng trong phát triển phần mềm, nó liên quan mật thiết đến sự tồn tại và phát triển của các công ty phần mềm. Trong đó có sự kiểm thử chương trình, nó là sự kiểm tra thông qua việc thực hiện chương trình, được tiến hành sau khi đã phát triển chương trình (mã nguồn). Nó là kỹ thuật kiểm tra khá phổ biến ngày nay. Có rất nhiều kỹ thuật kiểm thử chương trình, song rất nhiều hạn chế với những kỹ thuật kiểm thử dựa trên các mô hình đơn giản, như là: kiểm tra danh sách, phân chia, mô hình cây.... Chi tiết hoạt động của chương trình, sự tương tác giữa các thành phần khác nhau của chương trình, cũng như là các thông tin về cách sử dụng chi tiết không thể được trình bày 1 cách đầy đủ trên những mô hình kiểm thử đơn giản. Trong đề tài này, tôi xin giới thiệu "Finite – state machines" (FSMs) như là cơ sở cho rất nhiều kỹ thuật kiểm thử.

MỤC LỤC

Chương 1. FINITE-STATE MACHINES

1.1.FSMs - Khái niệm cơ bản và ví dụ

FSMs là những mô hình bao gồm:

- Những yếu tố tĩnh: bao gồm trạng thái *(state)* và sự chuyển tiếp trạng thái *(state transition)*. Những sự chuyển tiếp trạng thái thường được gọi là các sự chuyển tiếp. Số lượng của những trạng thái là hữu hạn. Sự chuyển tiếp trực tiếp từ trạng thái A sang trạng thái B chỉ có thể theo 1 đường link duy nhất là A-B. Số lượng các cũng là giới hạn.
- Những yếu tố động: bao gồm đầu vào input được cung cấp cho FSMs và đầu ra output được lấy ra từ FSMs ở những sự thực hiện động. Nói chung, cả hai số lượng đầu vào và đầu ra đều là hữu hạn. Trong trường hợp mà số lượng đầu vào và đầu ra có thể chiếm một số lượng lớn hoặc một số lượng vô hạn các giá trị, thường thường chúng ta cần phải nhóm chúng vào các phân vùng.

Các FSMs và các yếu tố của chúng được biểu diễn bằng đồ thị. Các yếu tố chính trong đồ thị bao gồm:

- Mỗi trạng thái được mô tả như là một nút (node) trong đồ thị.
- Mỗi sự chuyển tiếp được diễn tả như một đường link được kết nối trực tiếp từ trạng thái này sang trạng thái khác.
- Input và output được nối với sự chuyển tiếp và được diễn tả như sự chú thích bởi các sự chuyển tiếp.

Thông thường một trạng thái thì tương ứng với vài trạng thái xử lý chương trình, hoặc là một khoảng thời gian cụ thể , hoặc là tương ứng với 1 trường hợp cá biệt giữa những hoạt động nào đó.Ví dụ, hãy xem xét trình tự thực hiện sau đây:

- Khi chương trình khởi động, chương trình ở trạng thái ban đầu.
- Sau khi thực hiện chức năng hướng người sử dụng (black-box view) hay thực hiện 1 câu lệnh hay một thủ tục bên trong (white-box view), sự hoạt động của chương trình được chuyển sang 1 trạng thái khác.
- Các bước trên được lặp lại một số lần, vài trạng thái cũng có thể được lặp lại.
- Trạng thái mà chương trình xử lý hoàn thành thì được gọi là trạng thái cuối cùng.
- Trong mỗi một sự chuyển tiếp, một vài thông tin đầu vào có thể cần thiết và một vài thông tin đầu ra có thể được đưa ra.

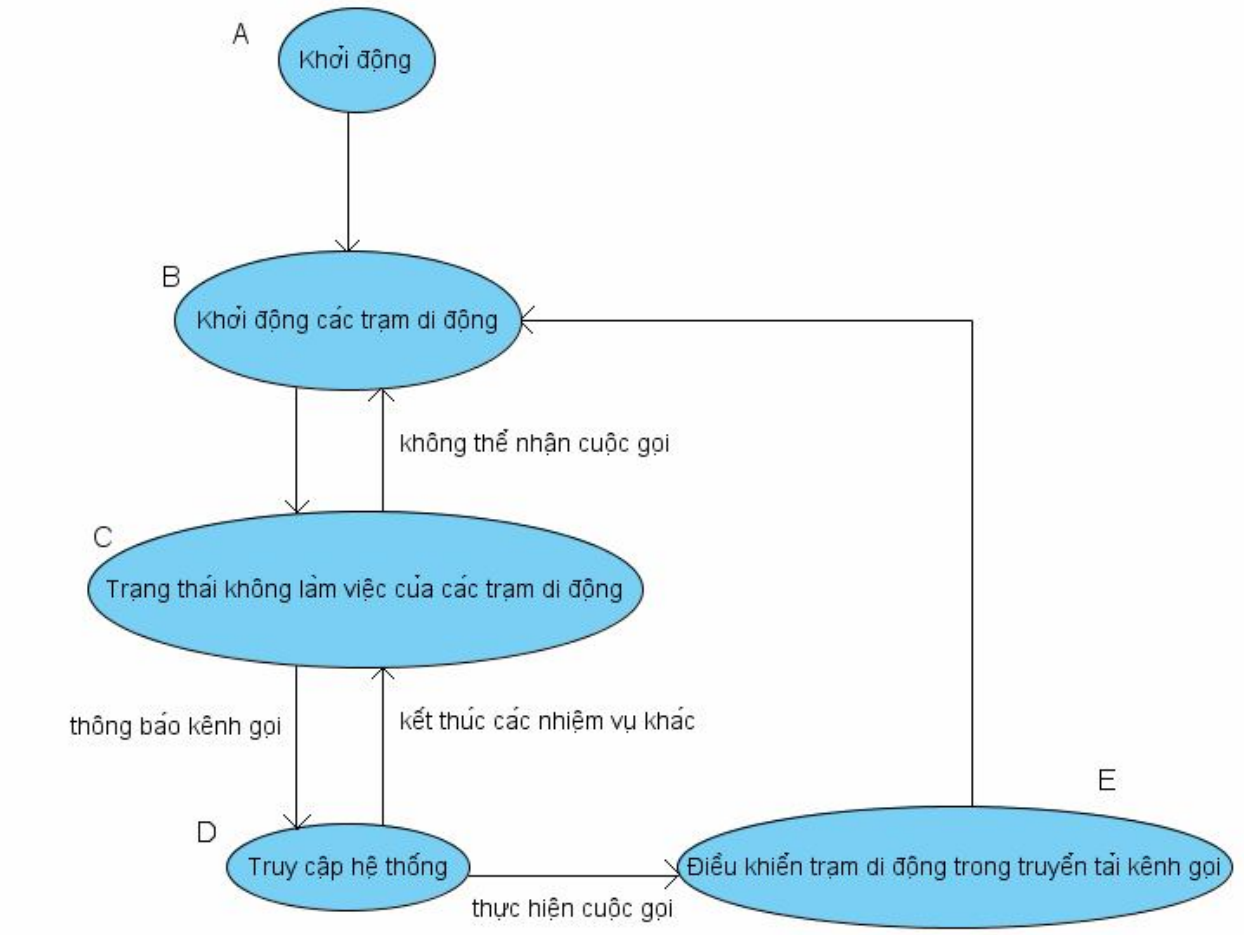
Trong ví dụ trên, những trạng thái đại diện cho 1 vài sự trừu tượng hóa của các tình trạng hoạt động hoặc là các trạng thái và hầu hết các hoạt động có liên quan đến các liên kết link hoặc trạng thái chuyển tiếp. Một ví dụ cụ thể rất quen thuộc với hầu hết mọi người trong xã hội hiện đại là việc sử dụng Web trên khắp thế giới. Sự lướt mỗi trang Web có thể coi là 1 trạng thái. Khi chúng ta khởi động Web Browser, trang khởi động mặc định hay trang khởi động do chúng ta tạo ra sẽ được tải về, điều đó tương ứng với trạng thái đầu tiên. Mỗi lần chúng ta làm theo 1 liên kết trong 1 trang hoặc lựa chọn 1 trang Web thông qua việc sử dụng lựa chọn Bookmark/favorite hay là bằng cách trực tiếp gõ lên URL (địa chỉ duy nhất cho mỗi trang Web riêng biệt), chúng ta đã khởi động đến 1 trang Web khác. Chúng ta có thể dừng lại bất kỳ lúc nào bằng cách tắt thanh Web Browser hoặc đơn giản là không tải Web nữa. Trang Web cuối cùng được xem được coi là trạng thái cuối cùng. Trong ví dụ ứng dụng của Web trên, tất cả những quy trình như là: yêu cầu và tải Web cũng như các lỗi liên quan hoặc là các thông báo khác đều được gắn liền với quá trình chuyển tiếp. Trạng thái FSM đại diện cho mục đích chính của việc sử dụng Web và người sử dụng có thấy điều đó 1 cách dễ dàng.

Trong nhiều ứng dụng, một hỗn hợp của hai loại trên đây của FSMS có thể được sử dụng miễn là không có sự nhầm lẫn. Một ví dụ cụ thể của FSMs cho trường hợp này miêu tả những cho sự xử lý cuộc gọi trong hệ thống mạng thông tin liên lạc. Những thông tin cụ thể bao gồm: Những trạng thái cụ thể liên quan đến các hoạt động khác nhau hay tình trạng của hệ thống đã được xác nhận, ví dụ: “Khởi động”-“Khởi đầu các trạm di động”,”Tình trạng nghỉ các trạm di động... được xác định bởi nhãn A, B, C, D, E, tương ứng.

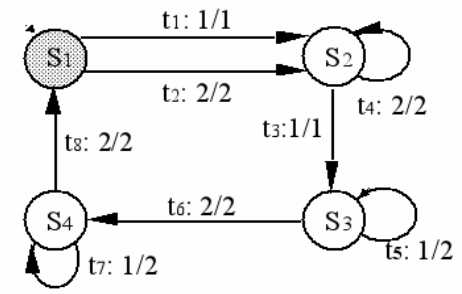
Vài sự chuyển tiếp không kết nối với bất kỳ input nào hoặc bất kỳ ouput nào. Chúng chỉ cần làm theo sau khi hoàn thành nhiệm vụ liên kết với các trạng thái hiện hành. Trong trường hợp đó, thường chỉ có 1 sự chuyển tiếp là có thể xảy ra, bởi vì nếu không, thì phải có những điều kiện và thông tin đầu vào rõ ràng để chỉ rõ sự chuyển tiếp nào được phép diễn ra.Ví dụ, sau khi trạng thái A, trạng thái tiếp theo sau luôn luôn là B. Tương tự, sau trạng thái B thì trạng thái tiếp theo luôn luôn là trạng thái C và trạng thái E thì trạng thái tiếp theo là trạng thái B. Nói chung, những quá trình chuyển tiếp đó không được kết nối với bất kỳ 1 quá trình xử lý nào mà chỉ kết nối với các mối quan hệ logic giữa các trạng thái.

Những quá trình chuyển tiếp khác được kết nối với những thông báo, điều kiện rõ ràng như thông tin đầu vào và một sô thông tin đầu ra có thể. Ví dụ, trạng thái sau

trạng thái C(Trạng thái không làm việc của các trạm di động) có thể là D (Truy cập hệ thống), được kết nối với sự trả lời thông báo kênh gọi nhận được. Cuộc gọi bắt đầu, hoặc đăng ký thực hiện cuộc gọi. Trạng thái B(Khởi động các trạm di động) cũng có thể theo sau trạng thái C trong điều kiện là trạm di động không có khả năng nhận kênh gọi. Tương tự trạng thái E cũng có thể sau trạng thái D nếu cuộc gọi được hình thành, hoặc trạng thái B sau trạng thái D trong trường hợp các nhiệm vụ truy cập hệ thống được hoàn thành.



Đồ thị 1.1 Ví dụ về finite-state machine (FSM) cho tiến trình gọi



Đồ thị 1.2 Ví dụ về mô hình hóa FSMs

Trong đó,

S1 là trạng thái ban đầu

là sự chuyển tiếp T1 từ trạng thái S1 đến trạng thái S2 có input là 1 và output là 1. Dấu "/" để phân biệt input và output.

1.2. Mô tả FSMs

Cách hiệu quả nhất để mô tả FSMs là sự dụng biện pháp đồ thị như ví dụ trên. Các đồ thị như thế có thể chỉ rõ bằng 1 bộ các trạng thái cho phép, và các input/output được kết nối. Ví dụ, 1 tập trạng thái tương ứng với hình 1.1 là {A, B, C, D, E}, sự chuyển tiếp từ C→B được mô tả là {C, B, "Không thể nhận cuộc gọi", −}, đối với đầu vào được chỉ rõ bằng 3 thành phần và output không xác định(−). Tập sự chuyển đổi và input/output bao gồm chính những trạng thái đó và những thành phần giống nhau của chúng.

Mặc dầu sự mô tả đồ thị thì rất trực giác và dễ giải thích, nhưng nó trở nên không thực tế khi số lượng các trạng thái là lớn. Khi chúng ta có nhiều hơn 20 hoặc 30 trạng thái, thì bản vẽ sẽ trở nên lộn xộn và rất khó theo dõi. Vì thế dạng mô tả dạng bảng biểu (hay sự mô tả theo ma trận) được dùng 1 cách thường xuyên, điều đó cũng giúp máy tính xử lý dễ dàng hơn. Ví dụ đồ thị 1.1 có thể được mô tả bằng bảng 1.1, có thể được giải thích như sau:

Bảng 1.1: Ví dụ về finite-state machine (FSM) cho tiến trình gọi trong sự mô tả kiểu bảng ma trận

	A	**B**	**C**	**D**	**E**
A	sự chuyển tiếp tương ứng	-/-	sự chuyển tiếp tương ứng	sự chuyển tiếp tương ứng	sự chuyển tiếp tương ứng

	không được thực hiện		không được thực hiện	không được thực hiện	không được thực hiện
B	sự chuyển tiếp tương ứng không được thực hiện	sự chuyển tiếp tương ứng không được thực hiện	-/-	sự chuyển tiếp tương ứng không được thực hiện	sự chuyển tiếp tương ứng không được thực hiện
C	sự chuyển tiếp tương ứng không được thực hiện	không thể nhận kênh gọi/-	sự chuyển tiếp tương ứng không được thực hiện	thông báo kênh gọi /-	sự chuyển tiếp tương ứng không được thực hiện
D	sự chuyển tiếp tương ứng không được thực hiện	sự chuyển tiếp tương ứng không được thực hiện	hoàn thành các nhiệm vụ khác /-	sự chuyển tiếp tương ứng không được thực hiện	thực hiện cuộc gọi /-
E	sự chuyển tiếp tương ứng không được thực hiện	-/-	sự chuyển tiếp tương ứng không được thực hiện	na	sự chuyển tiếp tương ứng không được thực hiện

- Trạng thái được liệt kê theo cả hàng và cột.
- Hàng mô tả những trạng thái ban đầu và những cột mô tả trạng thái kết thúc cho sự chuyển tiếp xác định.

- Nếu sự chuyển tiếp từ trạng thái X(hàng X) sang trạng thái Y(cột Y) được cho phép, thì phần tử tương ứng(ở vị trí dòng X, cột Y) được đánh đấu bằng chính input/output của nó. Với những input/output không xác định thì được đánh dấu bởi(-).

Như chúng ta thấy, sự mô tả kiểu ma trận là rất hệ thống, thường thường là kiểu ma trận vuông (N x N ô) và không khó để mô tả. Vì thế nó được dùng 1 cách rộng rãi để mô tả FSMs. Sự cân đối giúp sự tính toán và phân tích dựa trên bảng FSMs dễ trình bày.

Tuy vậy, khi có rất nhiều các ô trống, chúng ta lãng phí rất nhiều không gian bộ nhớ để chứa đựng N x N ô. Vì thế chúng ta sử dụng cách mô tả thứ 3, đó là mô tả liệt kê. Về cơ bản, 1 tập trạng thái được mô tả bằng 1 danh sách và sự chuyển tiếp được cho phép thì được mô tả cũng bằng 1 danh sách, bao gồm các yếu tố, ví dụ như cấu trúc {C, B, "không thể nhận kênh gọi",- } nghĩa là sự chuyển tiếp từ trạng thái C→B biểu thị sự không thể nhận kênh gọi, được ký hiệu là -. Sự mô tả kiểu liệt kê danh sách thì chặt chẽ hơn nhưng cũng kém thông dụng hơn. Sự so sánh giữa sự mô tả kiểu ma trận và liệt kê danh sách cũng tương tự như sự so sánh giữa danh sách và cấu trúc dữ liệu dãy 2 chiều trong máy tính và xử lý thông tin.

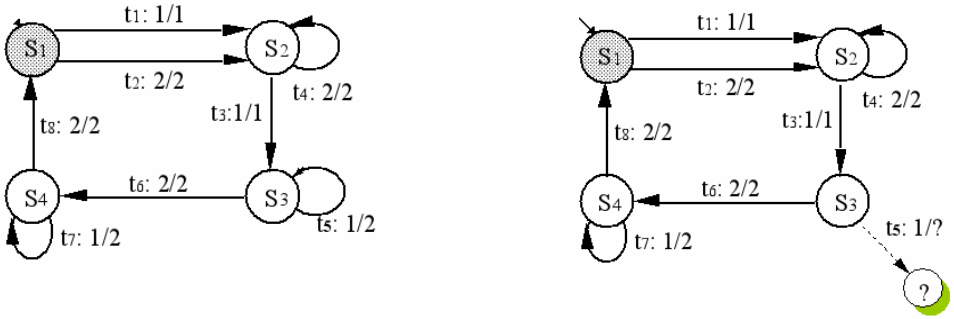
Cả 3 cách mô tả của FSMs : đồ thị, ma trận, danh sách đều được dùng rộng rãi trong tài liệu kiểm thử. Vì thế chúng ta nên làm quen với cả 3 loại, có thể qua sự diễn dải của các bài tập mở rộng.

Chương 2. KIỂM THỬ THEO MÔ HÌNH FSMs

2.1. Những rắc rối cơ bản đối với hệ thống được mô hình hóa bởi FSMs

Như đã đề cập ở trên, FSMs có thể được dùng để mô hình cả 2 trường hợp: Mô hình hành vi hệ thống bên ngoài (black-box view) và thực hiện chi tiết các cài đặt cụ thể (while-box view). Trong mỗi cách sử dụng chúng ta có thể xem xét 4 thành phần cơ bản: trạng thái, sự chuyển tiếp, input và output để kiểm tra những vấn đề có thể nảy sinh của hệ thống được mô hình hóa bởi FSMs như dưới đây:

- Vấn đề về trạng thái: thiếu, thừa hoặc lỗi của trạng thái.



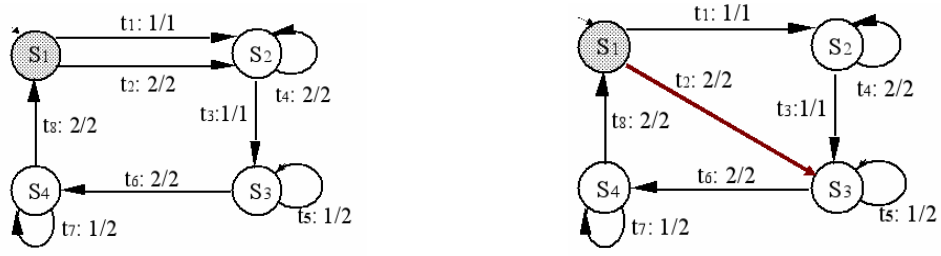
Hình 2.1 Ví dụ về thừa trạng thái

o Lỗi trạng thái là những trạng thái có hành vi khó xác định.

o Sự thiếu trạng thái: tương ứng với những trường hợp có trạng thái hiện tại nhưng trạng thái tiếp theo bị thiếu. Trường hợp đặc biệt của thiếu trạng thái là: hệ thống có trạng thái ban đầu là không xác định.

o Thừa trạng thái: có thể được hình dung là những trạng thái không được đưa ra hoặc là trạng thái chết, đó là những trạng thái không có sự kết nối với bất kỳ 1 trạng thái ban đầu nào thông qua 1 số sự chuyển tiếp. Nhiều sự chuyển tiếp cho cùng 1 input cũng có thể được kết nối với 1 vài trạng thái thêm. Trong trường hợp đó thì trạng thái hiện tại cũng là 1 trạng thái lỗi bởi vì hành vi của nó là khó xác định.

- Những vấn đề về sự chuyển tiếp: thiếu, thừa và lỗi chuyển tiếp.



Hình 2.2 Ví dụ về lỗi sự chuyển tiếp

o Thiếu sự chuyển tiếp: là 1 trường hợp tương ứng với 1 trạng thái hiện tại và đầu vào input hợp lệ nhưng trạng thái tiếp theo là thiếu hoặc không xác định.

o Thêm sự chuyển tiếp: được liên kết với nhiều sự chuyển tiếp cho cùng 1 trạng thái hiện tại và input.

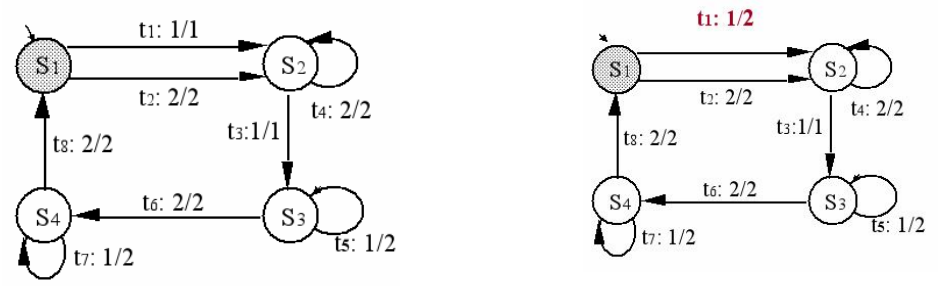
o Lỗi chuyển tiếp: là sự chuyển tiếp không mong đợi, hoặc là sự chuyển tiếp có output không mong đợi.

- Những vấn đề về input: Trong sự kiểm thử dựa trên FSMs, đặc biệt coi những vấn đề input như 1 phần của vấn đề trạng thái và vấn đề sự chuyển tiếp, giả định rằng tất cả cần phải được xử lý chính xác thông qua một số sự chuyển tiếp trạng thái của FSM này. Là một phần mở rộng nói chung, thậm chí input không hợp lệ được mong đợi là xử lý chính xác không gây treo hệ thống hoặc các vấn đề khác, chẳng hạn như những vấn đề sau đây:

  o Bỏ qua các đầu vào không hợp lệ: như là đứng yên ở cùng 1 trạng thái cho những trường hợp input không hợp lệ.

  o Xử lý trực tiếp các đầu vào không hợp lệ: chẳng hạn như đưa ra thông báo lỗi, đi qua một số xử lý ngoại lệ và sự chuyển tiếp liên quan.

- Những vấn đề về output:



Hình 2.3 Ví dụ về lỗi output

Chúng ta không giải quyết trực tiếp những vấn đề về output, đúng hơn là 1 phần của vấn đề phán xét kiểm thử trong những sự chuyển tiếp. Ví dụ, sự chuyển tiếp đưa ra những thông tin không mong đợi như thừa, thiếu, lỗi output thì sẽ xác định sự chuyển tiếp là sự chuyển tiếp lỗi.

Vì thế trong kiểm thử dựa trên FSMs, tập trung vào những vấn đề trạng thái, vấn đề về sự chuyển tiếp. Input được sử dụng chính cho sự cập nhật cập nhật và output được sử dụng chính cho sự kiểm tra kết quả.

2.2. Xây dựng mô hình và kiểm tra cho thiếu, thừa trạng thái và sự chuyển tiếp.

Chúng ta có thể xây dựng FSMs và xác nhận chúng trong các bước sau:

- Bước 1: Thu thập số liệu và xác nhận nguồn thông tin: dựa trên sự xử lý chức năng bên ngoài được mô phỏng (black-box view) hoặc là trạng thái hoạt động chương trình bên trong được mô phỏng (while-box view) trong FSMs, có thể xác định các nguồn thông tin khác nhau. Trong trường hợp trước đây, các nguồn thông tin bao gồm những thông số kỹ thuật sản phẩm bên ngoài hoặc cách sử dụng mong đợi. Chúng đại diện cho chức năng và quan hệ logic giữa các tập con khác nhau của

hoạt động và các giao diện. Trong trường hợp thứ 2, thông tin bên trong sản phẩm ,như là cấu trúc và sự kết nối của những thành phần cài đặt trong tài liệu thiết kế sản phẩm và trong mã hóa chương trình có thể được sử dụng cho quá trình xây dựng mô hình. Đối với nhiều sản phẩm hiện có, trường hợp kiểm thử và kiểm tra danh sách đã có, có thể được sử dụng như là 1 nguồn thông tin rất quan trọng. Những hoạt động con cần được rút ra từ những nguồn thông tin sẵn có và được kết nối với nhau để tạo thành FSMs.

- Bước 2: Xây dựng FSMs ban đầu dựa trên những nguồn thông tin được xác định trong Bước 1 ở trên: Chúng ta tiếp tục xem xét 4 yếu tố cơ bản: trạng thái, sự chuyển đổi, input và output để xây dựng hệ thống ban đầu của FSMs. Chúng ta có thể cùng 1 lúc xem xét các yếu tố theo những bước sau đây:

  - Bước 2.1: Liệt kê và xác nhận trạng thái :

Chúng ta cần giữ số lượng các trạng thái ở các mức có thể quản lý được từ 1 ít đến vài chục nhưng không phải hàng nghìn. Trong trường hợp hệ thống thật sự cần được mô tả bởi 1 số lượng trạng thái rất lớn, chúng ta có thể sử dụng lồng nhau hoặc hệ thống FSMs có trật tự, như sẽ mô tả chi tiết hơn trong tinh lọc mô hình dưới đây.

  - Bước 2.2: Xác nhận sự chuyển tiếp với sự giúp đỡ của giá trị đầu vào:

Với mỗi 1 trạng thái, chúng ta có thể xem xét tất cả những sự chuyển tiếp có thể có trong sự kết nối với tất cả các giá trị input có thể. Như đã đề cập ở phần 1.1, khi mà số lượng các giá trị input có thể rất lớn hoặc là vô hạn, chúng ta có thể sử dụng sự phân vùng input để giúp đỡ cho quá trình xác định sự chuyển tiếp cụ thể. Các phân vùng này mô tả những lớp tương đương với những đặc điểm của sự chuyển tiếp sẽ được thực thi. Một lợi ích khác của quá trình này là xác nhận những trạng thái thiếu từ Bước 2.1, nơi mà 1 số sự chuyển tiếp dẫn đến những trạng thái khác nhau đã được xác nhận ở trên.

  - Bước 2.3: Xác nhận mối quan hệ của input và output: liên quan đến mỗi 1 sự chuyển tiếp riêng biệt. Output này sẽ được sử dụng như 1 phần của phán xét kiểm thử để kiểm tra kết quả của sự thử nghiệm.

- Bước 3: Sự làm có giá trị và tinh lọc mô hình

Bước này bao gồm 2 hoạt động kết nối. Trong quá trình làm FSMs ban đầu có giá trị, trạng thái hay là sự chuyển tiếp mới có thể được xác định, kết quả là sự tinh chế FSMs. Tuy nhiên, như đã đề cập ở trên, quy trình này không thể được tiến hành để thừa, tức là không để có quá nhiều trạng thái hay sự chuyển tiếp trong FSMs. Hậu quả là khi mà số lượng lớn trạng thái và sự chuyển tiếp cần được mô tả trong mô hình, chúng ta thường sử dụng lồng FSMs hoặc FSMs phân cấp. Với 1 số trạng thái xác định

ở FSMs cấp cao hơn có thể triển khai ở FSMs cấp thấp hơn. Chúng ta cũng có thể kiểm tra nguồn thông tin để xác định những trạng thái thừa, thiếu như 1 phần của hoạt động làm mô hình có giá trị.

Ý tưởng cơ bản để xác định trạng thái và sự chuyển tiếp thiếu thì tương tự như sự kiểm thử dựa trên kiểm tra danh sách và sự phân vùng. Ví dụ, sự kiểm tra danh sách dựa trên những chi tiết kỹ thuật chức năng của sản phẩm có thể được dùng để kiểm tra trực tiếp những trạng thái thiếu, hoặc những sự chuyển tiếp thiếu. Tuy nhiên, những chi tiết kỹ thuật chức năng đó thường tương ứng với những trạng thái và sự chuyển tiếp ở mức cao, những trạng thái đó cần được tinh chế đến cùng 1 mức của những trạng thái và sự chuyển tiếp đạt được bởi FSMs. Với những mức thấp hơn của FSMs, thông tin chi tiết sản phẩm hoặc mã hóa chương trình thường được dùng để giúp xác nhận những trạng thái và chuyển tiếp thiếu.

Kiểm tra những trạng thái và sự chuyển tiếp thừa có thể làm theo cùng 1 thủ tục được làm cho có giá trị chéo với các nguồn thông tin. Tuy nhiên, sự kiểm tra đó thường khó hơn rất nhiều so với sự xác nhận những trạng thái thiếu, tương tự như trường hợp yêu cầu khả năng truy vết cần thiết của sản phẩm. Nếu mọi trạng thái và mọi sự chuyển tiếp có thể được truy vết lại những nguồn thông tin tương ứng với sự tạo ra chúng, thì sự kiểm tra đó có thế được hoàn thành 1 cách dễ dàng. Tuy nhiên, 1 trạng thái không nên mong đợi hoàn thành tài liệu kết hợp với tất cả các trạng thái và sự chuyển đổi trong FSMs. Điều đó làm cho quá trình xác nhận những trạng thái và sự chuyển đổi thừa rất khó khăn nếu chúng ta không biết cái gì dẫn đến sự tạo thành chúng ở trạng thái ban đầu, vị trí ban đầu. Để thay thế cho thủ tục của sự kiểm tra những trạng thái và sự chuyển tiếp thừa này, chúng ta có thể thực hiện các phân tích đã được xác định, những chi tiết riêng biệt không thể xác định hoặc 1 số những trạng thái không thể xác định. Thường thì những trạng thái không thể xác định này mô tả những trạng thái thừa hoặc 1 vài rắc rối khác. Những thuật toán phân tích có thể thực hiện được từ những tác giả (Deo 1974, Knuth 1973) và những công cụ có liên quan có thể được sử dụng để thực hiện những phân tích đó.

Ngoài những phương pháp kiểm tra những trạng thái và sự chuyển tiếp thiếu, thừa, thì đôi khi nó cũng có thể được kiểm tra cùng với những trạng thái, sự chuyển tiếp không chính xác. Để thực sự kiểm tra những trạng thái và sự chuyển tiếp, chúng ta cần bắt đầu từ trạng thái ban đầu và bắt đầu từ trạng thái trung gian hiện tại được lưu giữ trong 1 vài cách thức, và sau đó là theo 1 loạt các sự chuyển tiếp để kiểm tra những trạng thái đúng mà chúng ta đã cố gắng đạt được và để kiểm tra những chuyển tiếp đúng mà chúng ta cố gắng làm theo.

2.3. Sự kiểm thử cho những trạng thái và sự chuyển tiếp

Các thử nghiệm nói chung dựa trên FSMs và sự kiểm tra riêng của những trạng thái và sự chuyển tiếp đúng có thể được xử lý như 2 vấn đề riêng biệt:

- Trạng thái hay là nút đưa ra thông tin

Chúng ta cần đảm bảo rằng mỗi 1 trạng thái đều có thể đạt tới và truy cập của 1 số trường hợp thử nghiệm. Đó là những trạng thái và vấn đề xuyên suốt trong học thuyết đồ thị (Deo 1974, Knuth 1974), kết quả là rất nhiều thuật toán có thể đi qua các nút đồ thị được dùng để giúp chúng ta phát triển các trường hợp thử nghiệm.

- Sự chuyển tiếp và kết nối thông tin đưa ra

Chúng ta cần đảm bảo rằng mỗi 1 sự kết nối hoặc sự chuyển tiếp thì đều được làm rõ, bao phủ bởi 1 số trường hợp thử nghiệm. Mặc dầu vấn đề này cũng có thể được xử lý như là 1 kết nối có thể vượt qua trong thuyết đồ thị, sự thử nghiệm thông tin đưa ra trạng thái trên đã giúp chúng ta đạt được những trạng thái có khả năng.

Trong nỗ lực đạt tới trạng thái cụ thể, mỗi trường hợp cụ thể về bản chất là 1 loạt các giá trị input giúp chúng ta có thể tạo ra các sự chuyển tiếp từ trạng thái ban đầu đến trạng thái muốn đạt tới bằng 1 loạt các bước nhảy qua các trạng thái trung gian. Có thể là mỗi 1 trường hợp thử nghiệm có khả năng giúp chúng ta thấy được tất cả các trạng thái. Do đó mà nhận được thông tin đưa ra trạng thái hoàn thành.

Tuy vậy, chúng ta cần nhiều hơn các trường hợp kiểm thử ở mọi tình huống bởi vì đó có thể là những trạng thái ban đầu, trạng thái kết thúc hay 1 chuỗi các sự chuyển tiếp phức tạp. Kết nối từ trạng thái ban đầu đến 1 trạng thái cụ thể. Trong hầu hết các hệ thống được mô hình hóa bởi FSMs, những trạng thái ban đầu là những trạng thái không có sự kết nối đầu vào và những trạng thái kết thúc là những trạng thái không có sự kết nối đầu ra. Trong tình huống như vậy, bất kể là trạng thái ban đầu hay trạng thái kết thúc, chúng ta cần càng nhiều các trường hợp kiểm thử càng tốt.

Từ trạng thái ban đầu, trạng thái tiếp theo thì được quyết định bởi input. Vì thế, sự chuyển tiếp trạng thái bước 1 có thể xem như là 1 sự phân loại đầu tiên các input vào các lớp tương đương và sau đó theo 1 trạng thái cụ thể từ sự phân loại đó.

Chương 3. DÒNG ĐIỀU KHIỂN, PHỤ THUỘC DỮ LIỆU, SỰ KIỂM THỬ TƯƠNG TÁC

Từ quan điểm về cấu trúc hay sự cài đặt, một hệ thống phần mềm được tạo thành từ các thành phần tương tác, các mô-đun, hoặc các hệ thống con. Nó được tạo ra để hướng tới đối tượng là khách hàng, hệ thống tổng thể bao gồm các chức năng liên kết

hoặc các hoạt động. Máy trạng thái hữu hạn (FSMs) và các mô hình sử dụng có liên quan mà tôi đã giới thiệu ở chương trước có thể được sử dụng để mô hình hóa và thử nghiệm các chức năng hệ thống có liên hệ với nhau, sự cài đặt, và cách sử dụng có liên quan. Tôi tập trung vào các trạng thái, sự chuyển tiếp, và cách sử dụng có liên quan chứ không chú ý nhiều đến sự ảnh hưởng qua lại ngoại trừ những sự kết nối đơn giản. Trong phần này, tôi giới thiệu các kỹ thuật thử nghiệm để giải quyết sự ảnh hưởng qua lại liên hợp vượt ra ngoài những sự kết nối bước 1 trong FSMS. Hai loại chính của sự ảnh hưởng qua lại này là:

- Sự tương tác dọc theo một đường dẫn thi hành, nơi mà các hoạt động sau bị ảnh hưởng bởi toàn bộ các hoạt động trước đó.
- Những tương tác cụ thể giữa các mục dữ liệu trong quy trình thực hiện.

Sự kiểm thử của sự tương tác ở trên thường được gọi lần lượt là: sự kiểm thử dòng điều khiển (CFT) và sự kiểm thử dòng dữ liêu (DFT). Những kỹ thuật này là những kỹ thuật white-box truyền thống áp dụng cho việc kiểm thử dòng chức năng các mức của hệ thống, sự phụ thuộc dữ liệu và sự tương tác có liên quan.

3.1. Sự kiểm thử dòng điều khiển cơ bản

Sự kiểm thử dòng điều khiển(CFT) là sự mở rộng một cách tự nhiên và trực tiếp của sự kiểm thử FSM với một loại chuyên dụng của FSMs gọi là đồ thị dòng điều khiển (CFGs) và tập trung vào hoàn thành đường dẫn thực thi thay vì trạng thái hoặc các liên kết .

3.1.1Khái niệm chung

FSMs phân biệt các dạng: xử lý thông tin có liên quan đến các sự chuyển tiếp và dạng xử lý thông tin liên quan tới các trạng thái. Đồ thị dòng điều khiển (CFGs) có thể được coi là trường hợp đặc biệt của loại thứ hai, với các yếu tố và các đặc điểm quy định như sau:

- Các điểm nút (Nodes): Mỗi nút trong CFG tương ứng với một đơn vị xử lý thông tin (white-box view) hoặc khối lượng công việc được xử lý bởi các phần mềm (black-box view). Các nút trong CFGs tương ứng với các trạng thái trong FSMs.
- Các liên kết (Links): Mỗi link trong một CFG chỉ đơn giản là đại diện cho mối quan hệ "được theo sau bởi": Nếu chúng ta có một link trực tiếp từ nút A tới nút B, nó được xem như là A được theo sau bởi B, hoặc B sau A. Các link trong CFGs tương ứng với các sự chuyển tiếp trong FSMs, nhưng ở CFGs thì sự xử lý hay khối lượng công việc được xử lý không được kết nối với các link. Các link trùng lặp thì không cần

thiết ở CFGs bởi vì không cần thiết để xác định rõ mối quan hệ đơn giản "được theo sau bởi" thêm một lần nữa.

- Các nút vào (initial/entry) và các nút ra (final/exit): Các nút vào là các nút mà tại đó bắt đầu sự thực thi của chương trình được gọi. Các nút ra là các nút mà tại đó kết thúc sự thực thi chương trình được gọi. Trong CFT, chủ yếu là xử lý với các chương trình thích hợp hoặc các chức năng mà chỉ có duy nhất một nút vào và một nút ra.

- Các liên kết ngoài *(*Outlinks): Một liên kết mà bắt nguồn từ một nút thì được gọi là một outlink đối với nút đó. Khi có nhiều outlink từ một nút, thì mỗi một outlink được dán nhãn bằng điều kiện cụ thể của nó. Sự thực thi thực tế sẽ chỉ làm theo một trong các outlink đó.

- Các liên kết trong (Inlinks): Một liên kết mà kết thúc tại một nút thì được gọi là một inlink đối với nút đó. Khi có nhiều inlink tới một nút, thì sự thực thi thực tế sẽ chỉ làm theo một trong các inlink bởi vì điều kiện của outlink phía trên đảm bảo rằng sự thực thi của chương trình sẽ chỉ làm theo một liên kết tại một thời điểm.

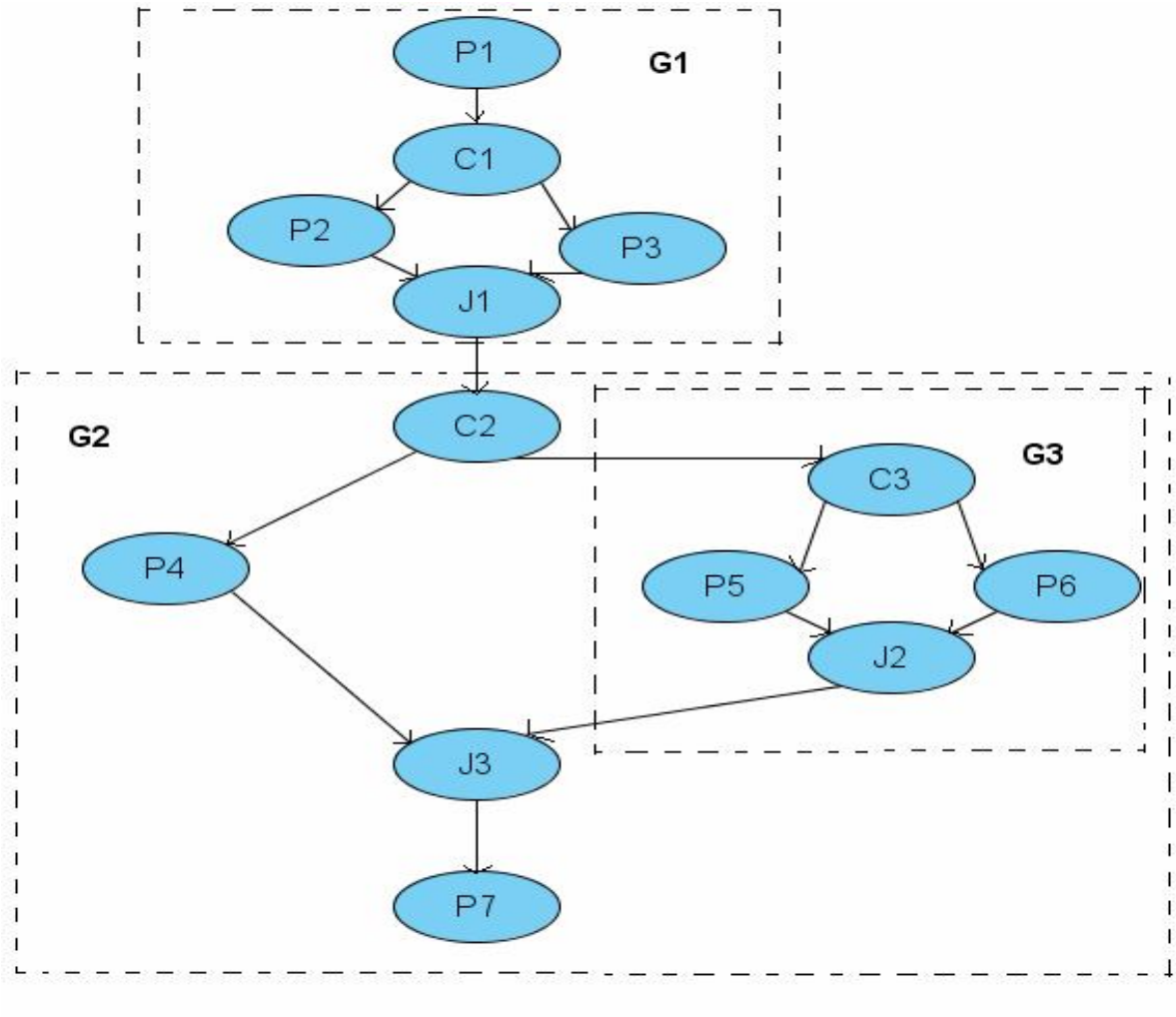
- Các nút quyết định (decision node), nút mối nối (junction node), nút xử lý (processing node): Một nút mà liên kết với nhiều outlink thì được gọi là một nút quyết định bởi vì tại nút đó hình thành quyết định lựa chọn một outlink để làm theo trong sự thực thi thực tế. Nó cũng được gọi là một nút nhánh. Tương tự, một nút mà liên kết với nhiều inlink thì được gọi là một nút mối nối. Một nút mà không phải là một nút quyết định và cũng không phải là một nút mối nối thì nó được gọi là một nút xử lý vì nó thường tương ứng với một số quá trình xử lý bên trong hay bên ngoài. Có hai trường hợp đặc biệt tại các nút vào: một là không có inlink và nút ra, hai là không có outlink. Tuy nhiên, chúng vẫn được xếp là các nút xử lý, bởi vì chúng được kết nối với một vài quá trình xử lý ban đầu hay kết thúc. Để rõ ràng hơn, chúng ta chia các nút thành 3 loại với sự xử lý thông tin được kết nối với các nút xử lý và với 1 nút mối nối tương ứng với mỗi nút nhánh.

- Đường dẫn Path: Một đường dẫn hoàn chỉnh, hoặc đơn giản là một đường dẫn bắt đầu từ một nút vào, theo sau là một loạt các link và duyệt qua một loạt các nút trung gian, cuối cùng kết thúc tại điểm ra. Vì không cho phép có link trùng lặp, chúng ta chỉ có thể xác định đường dẫn bởi một chuỗi các nút được duyệt qua.

- Phân đoạn Segment: Một path segment hay một segment là một phần của một path hoàn chỉnh, nơi nút đầu tiên có thể không phải là nút vào và nút cuối cùng có thể không phải là nút ra.

- Vòng lặp (Loop): Một path hay một segment chứa một loop nếu một số nút trong path hay segment được duyệt lại.

Hình 2.1 là một ví dụ về CFG với các nút xử lý P1, P2, P3, P4, P5, P6, P7; các nút quyết định C1, C2, C3 ; và các nút mối nối J1, J2, J3. CFG cũng chia thành ba phần khác nhau GI, G2,G3, với mỗi phần được hiển thị bên trong một hình chữ nhật. Ở sự biến đổi khác của CFG thường được sử dụng trong lý thuyết và thực hành, chúng ta có thể chập J1 và C2 thành một nút; J2, J3, P7 thành một nút.



Hình 2.1 Đồ thị dòng điều khiển CFT

Ý tưởng chủ đạo của kiểm thử dòng điều khiển (CFT) là lựa chọn đường dẫn và cập nhật chúng bằng cách gán những giá trị input tương ứng.

3.1.2. Xây dựng mô hình

Chú ý rằng Hình 2.1 tương tự như các biểu đồ dòng thường được sử dụng trong phát triển phần mềm. Trong thực tế, biểu đồ dòng như thế là một trong những nguồn

thông tin quan trọng cho chúng ta xây dựng CFGs và để thực hiện CFT. Một điểm khác biệt nhỏ giữa CFGs và biểu đồ dòng là: các loại khác nhau của các nút được biểu diễn bằng các ký hiệu khác nhau trong các biểu đồ dòng, nhưng chúng ta thường không sử dụng các ký hiệu khác nhau trong CFGs. Trong trường hợp không có các biểu đồ dòng, phần code hoặc phần thiết kế có thể là các nguồn thông tin cho sự xây dựng CFG. Các CFGs xây dựng theo cách này là mô hình kiểm thử white-box bởi vì thông tin cài đặt sản phẩm được sử dụng như sau:

- Các nút xử lý thường tương ứng với các nhiệm vụ, lời gọi hàm, hoặc các lời gọi thủ tục.
- Các nút quyết định hoặc các nút nhánh thường tương ứng câu lệnh nhánh như nhánh nhị phân "if-then-else" hoặc "if-then" (rỗng "else"), hoặc các nhánh khác như là "switch-case". Mỗi nhánh đi ra sẽ được đánh dấu bởi điều kiện cụ thể của nó. Ví dụ, đối với nhánh nhị phân, nó thường được đánh dấu bằng giá trị chân lý (T / F, hoặc True/False) của các điều kiện liên quan. Đối với phân nhánh đa chiều, điều kiện cụ thể hơn sẽ được đánh dấu.
- Câu lệnh Loop tương ứng với một loại đặc biệt của các nút nhánh.
- Các nút vào và các nút ra thường dễ xác định, tương ứng với câu lệnh đầu tiên và câu lệnh cuối cùng hoặc đơn vị xử lý trong phần code hoặc các đồ thị dòng tương ứng.

Một trong những vấn đề với thủ tục xây dựng CFG trên là rất nhiều các nút sẽ được sử dụng trong các CFG. Tuy nhiên, vì CFG được sử dụng để kiểm thử đường dẫn, chúng ta có thể nhóm một số các nút với nhau, chẳng hạn như một số nút xử lý tuần tự, hình thành những super-node nếu như nhóm sẽ không ảnh hưởng đến những đường dẫn thi hành.

Chú ý rằng việc sử dụng “goto” không được bao gồm trong thủ tục xây dựng CFG ở trên. Việc sử dụng tự do “goto” sẽ tạo ra những chương trình rất xấu tương ứng với trường hợp CFGs rất khó được kiểm thử. Đó là một trong những nguyên nhân chính khiến “goto” bị coi là có hại. May mắn là với những chương trình cấu trúc và những chương trình kế thừa nó, chương trình hướng đối tượng, được sử dụng rộng rãi trong phát triển hướng đối tượng hiện nay, chúng ta không gặp phải quá nhiều “goto”. Các cấu trúc được sử dụng rộng rãi là các chuỗi mắt xích liên tục, như là giữa các phần G1 và G2 trong Hinh 2.1; chuỗi lồng nhau như là G3 lồng trong G2 trong Hình 2.1. Tất nhiên nhiều chuỗi mắt xích hay nhiều cấp lồng nhau có thể được sử dụng trong các chương trình và phản ánh trong CFGs của nó.

Ở trong CFGs:

L1: input(a, b, c);

L2: d ← b*b – 4*a*c;

L3: if (d>0) then

L4: r ← 2

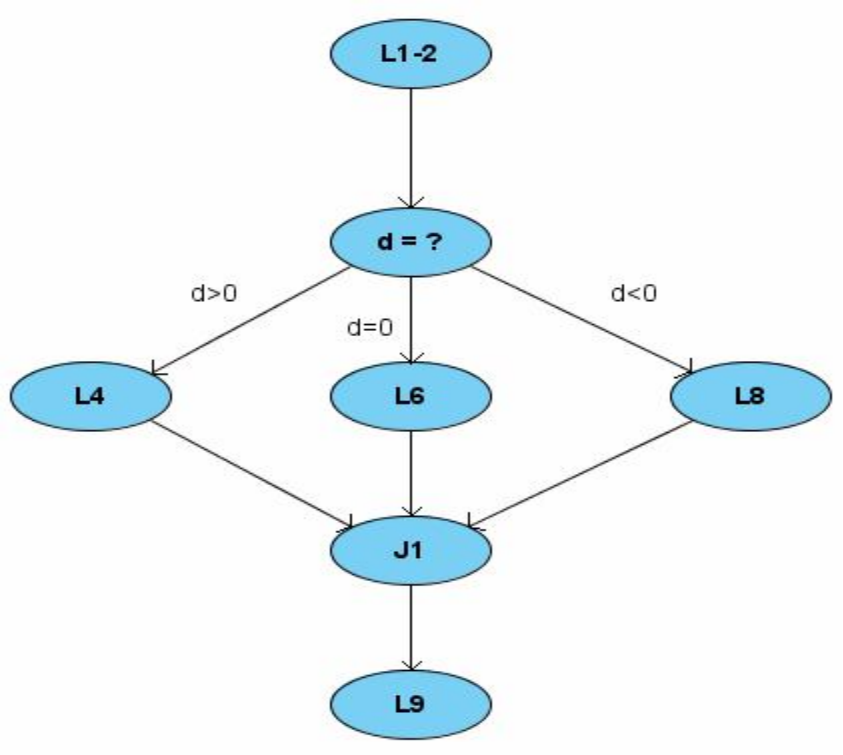
L5: else-if (d=0) then

L6: r ← 1

L7: else-if (d<0) then

L8: r ← 0

L9: output(r);



Hình 2.2 Ví dụ về chương trình và đồ thị dòng điều khiển của nó.

Hình 2.2 đưa ra một chương trình giải mã để xác định số lượng các nghiệm của phương trình $ax^2 + bx + c = 0$. Mỗi dòng được đánh số riêng. Chúng ra sử dụng 3 nhánh được thực hiện bởi **"if -else-if -else-if".** Chúng ta chập các đường L3, L5, L7 lại với nhau bởi vì chúng xác định một cách cơ bản 1 nhánh 3 đường, với nút nhánh được đánh dấu bằng điều kiện d=?. Đường L1 và L2 cũng được chập lại với nhau vì câu lệnh liên tục đó không ảnh hưởng đến dòng điều khiển. Ngoài ra, nút mối nối J1 được khai báo để đánh dấu sự kết nối.

CFGs cũng có thể nhận được từ các chi tiết kỹ thuật chức năng bên ngoài hoặc từ sự mô tả kịch bản sử dụng của khách hàng, do đó CFGs cũng có thể được coi như là

kiểm thử black-box. Chúng ta có thể trực tiếp điều chỉnh và sửa đổi biểu đồ dòng cho các chi tiết kỹ thuật sản phẩm hoặc các bản mô tả kịch bản sử dụng vào trong CFGs. Nếu biểu đồ dòng không có sẵn, chúng ta cần trích xuất thông tin từ các chi tiết kỹ thuật hay các bản mô tả đó bằng cách kiểm tra cấu trúc và các mối quan hệ của chúng như sau:

- Các nút xử lý thường tương ứng với một số hành động được mô tả, thường liên quan tới các cụm từ như "làm / nhập / tính toán" cái gì đó.
- Các nút nhánh thường liên quan tới các quyết định hay các điều kiện.
- Các nút vào và các nút ra thường tương ứng với các mục đầu tiên và các mục cuối cùng trong các chi tiết kỹ thuật hoặc các bản mô tả, mặc dù chúng cũng thường được chỉ rõ.

Ví dụ, CFG trong hình 2.2 có thể hình dung được sự mô tả sản phẩm:

- Để giải quyết phương trình bậc 2 : $ax^2 + bx + c = 0$, người sử dụng cần nhập các tham số.
- Nếu $b^2 - 4ac < 0$, không có nghiệm số thực và người sử dụng sẽ được thông báo
- Nếu $b^2 - 4ac = 0$, nghiệm sẽ là: r = -b/(2a) sẽ được tính ra kết quả.
- Nếu $b^2 - 4ac > 0$, nghiệm sẽ là: r = $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ sẽ được tính ra kết quả.

Chú ý rằng mặc dù các nghiệm được tính toán ở đây chỉ thay thế cho số nghiệm ở trong chương trình hình 2.2, các CFG kết quả sẽ giống nhau trong cấu trúc. Sự khác biệt duy nhất có thể là sự xử lý độc lập liên quan với các nút xử lý.

3.1.3. Sự lựa chọn đường dẫn

Chúng ta tiếp tục giới thiệu kỹ năng để lựa chọn 1 cách hệ thống các đường dẫn cho cấu trúc CFGs. Kỹ năng bao gồm 2 bước cơ bản:

1. Phân tích CFG.
2. Định nghĩa đường dẫn từ dưới lên.

Trong khi làm điều này, người ta tận dụng một số đặc tính quan trọng về cấu trúc CFGs từ lý thuyết đồ thị và lý thuyết ngôn ngữ lập trình. Cấu trúc CFG là một cấu trúc mà chỉ có chuỗi mắt xích liên tục và chuỗi lồng nhau, chỉ có 1 nút vào và chỉ có1 nút ra. Cấu trúc CFGs có thể được phân tích thành các đồ thị con sub-CFGs, và các sub-CFGs có thể được kết nối thông qua chuỗi mắt xích liên tục hoặc chuỗi lồng nhau.

Nếu một sub-CFGs không thể phân tích tiếp thì nó được coi là prime CFG, tức là CFG hoàn hảo. Các CFG cấp bậc được sinh ra từ sự xử lý đó được gọi là sự phân tích CFG nguyên thủy. Ví dụ, CFG trong hình 2.2 có thể phân tích thành G = G1o G2 (-, G3), với G3 lồng trong G2, và G1 móc nối với G2. G2 (-, G3) chỉ ra rằng G3 lồng trong nhánh phải của G2 mà trong đồ thị thì là nhánh F, với việc sử dụng "-" chỉ ra rằng không có sub-CFGs nào lồng vào nhánh trái T của G2. Còn G2 (G3) chỉ ra rằng G3 được lồng vào trong G2 nhưng không biết lồng vào nhánh trái hay phải của G2. Đường biên cho G1, G2, G3 trong hình 2.1 được chỉ rõ bằng hình chữ nhật nét đứt.

Với sự phân tích CFG ở trên, tôi có thể thực hiện định nghĩa đường dẫn từ dưới lên. Khi 2 CFGs, GI với đường dẫn M và G2 với đường dẫn N, kết hợp thành một CFG cấp cao hơn, chúng ta có thể xác định đường dẫn như sau:

- Với chuỗi mắt xích tuần tự, G = G1o G2, thì G sẽ có M x N đường dẫn. Nghĩa là mỗi đường dẫn trong M đường luôn có thể nối với 1 đường dẫn trong N đường dẫn. Và tất cả các đường dẫn đó tạo thành đường dẫn trong G. Ví dụ chuỗi mắt xích của 2 FGs hoàn hảo dạng nhị phân (mỗi cái có 2 đường dẫn tương ứng với giá trị logic T hoặc F cho các điều kiện của nó) có thể cung cấp 4 đường dẫn: TT, TF, FT, FF.
- Với chuỗi lồng nhau, G = G1 (G2), thì G sẽ có M + N - 1 đường dẫn. Nghĩa là, một đường dẫn trong đường dẫn ở G1 sẽ được thay thế bởi N đường dẫn ở G2. Ví dụ, chuỗi lồng nhau trong CFGs hoàn hảo dạng nhị phân ở hình 2.2 là G2 (-, G3) có 3 đường dẫn là: T, FT, FF, đúng theo công thức 2+2-1=3 đường dẫn.

Sự xử lý đó có thể được tiến hành cho mỗi mức độ, bằng cách bắt đầu với các CFGs hoàn hảo và tiếp tục với sự kết hợp ở cấp cao hơn, cho đến khi chúng ta xác định đường dẫn đầy đủ cho toàn bộ CFG. Trong ví dụ trên của hình 2.1, chúng ta có thể làm theo các thủ tục ở trên để chọn đường dẫn. CFG Các đã được phân tích, với G = G1o G2 (-, G3), Chúng ta sau đó có thể tập trung vào bước thứ hai như sau:

- Đầu tiên chúng ta xác định 2 đường dẫn trong G3, tương ứng với C3=T và C3=F.
- Tiếp theo, đường dẫn lồng G3 trong G2 tạo ra 3 đường dẫn, tương ứng với C2=T; C2=F, C3=T; C2=F, C3=F. Chúng ta có thể biểu thị đường dẫn như T-, FT, FF.
- Cuối cùng chúng ta kết hợp G2 (G3) với G1 để tạo ra 6 đường dẫn: TT-, TFT, TFF, FT-, FFT, FFF.

3.1.4.Cập nhật đường dẫn

Chìa khóa để để làm cập nhật đường dẫn là các nút quyết định hay các nút nhánh hay các điều kiện kết nối giữa các nút. Nếu tất cả các điều kiện đó độc lập với nhau, mọi đường dẫn được xác định ở trên có thể được cập nhật bằng cách lựa chọn các giá trị thay đổi để thỏa mãn các điều kiện cụ thể cho mỗi đường dẫn. Ví dụ, nếu những biến logic được sử dụng cho CFG trong hình 2.1, thì 6 đường dẫn TT-, TFT, TFF, FT-, FFT, FFF được cập nhật một cách trực tiếp. Tương tự, nếu C1 ≡ (x>0), C2 ≡ (y<1000), C3 ≡ (z=10), sau đó chúng ta có thể chọn các giá trị cho x, y, z để cập nhật các điều kiện tương ứng. Ví dụ, cho đường dẫn TFT, chúng ta có thể cập nhật bằng cách cho x=1, y=1001,z=10.

Nếu các điều kiện liên quan đến nhau chúng ta cần phải phân tích sâu hơn để loại bỏ các đường dẫn không thể xảy ra. Ví dụ, với chuỗi mắt xích liên tục của hai sub-CFG dạng nhị phân với các điều kiện trái ngược nhau C1 = - C2, chúng ta có thể loại bỏ 2 trong 4 đường dẫn TT, TF, FT, FF cho chúng ta 2 đường dẫn là TF và FT, bởi vì TT và FF có thể không được cập nhật.

Một ví dụ khác, hãy xem xét chuỗi mắt xích liên tục gồm 2 sub-CFG với C1 ≡ (x>0) và C2 ≡ (x <100). Hai điều kiện được liên kết thông qua biến số chung x. Trong trường hợp này, đường dẫn chung FF có thể bị loại bỏ vì sự trái ngược dưới đây:

$$(C1 = F) \wedge (C2 = F)$$
$$\equiv \neg(x>0) \wedge \neg(x<100)$$
$$\equiv (x\leq 0) \wedge (x\geq 100)$$
$$\equiv \emptyset$$

Điều này có nghĩa là 1 bộ x thỏa mãn điều kiện trên là tập rỗng (Ø)

3.1.5. Kiểm tra vòng lặp, cách sử dụng CFT và các vấn đề khác

3.1.5.1. Các kiểu vòng lặp khác nhau và các CFG tương ứng

Các vòng lặp liên quan với các thủ tục lặp đi lặp lại của xử lý thông tin, hoặc tương ứng với cài đặt thực tế (white-box view) hoặc các chức năng định hướng sử dụng (black-box view). Như đã đề cập ở trên, nếu một đường dẫn xuyên suốt một CFG chứa đựng một hay nhiều nút đi qua nhiều hơn một lần, thì một vòng lặp được tạo thành. Ví dụ, nếu chúng ta có một đường dẫn ABCDBE, thì đường dẫn con BCDB sẽ

tạo thành một đường dẫn vòng. Các đường dẫn vòng cũng có thể được tạo thành dễ dàng thông qua các đặc điểm ngôn ngữ chương trình, chẳng hạn như đệ quy. Cũng có thể tạo thành các vòng ngầm thông qua và bước nhảy "goto". Một vòng lặp có thể được xác định như sau:

- Phải có thân của đường dẫn vòng, nơi mà hoàn thiện một số thứ và được nhắc lại một vài lần. Nó thường được mô tả bởi 1 nút hoặc 1 số CFG lồng nhau bên trong vòng lặp..
- Phải có một số sự kiểm soát vòng lặp để đưa ra các quyết định vòng lặp để thực hiện thân vòng lặp hoặc thoát khỏi vòng lặp. Những sự kiểm soát vòng lặp này có thể được sử dụng nhiều lần cho mỗi sự lặp lại của vòng lặp để đưa ra quyết định dưới môi trường động hiện thời. Nó thường được đại diện bởi 1 nút có liên quan đến vị ngữ được xác định bởi 1 vài biến điều khiển – là các biến động được sử dụng để đưa ra quyết định vòng lặp.
- Phải có 1 vài nút vào và nút ra vòng lặp. Những nút mà chúng ta thường giải quyết trong chương trình cấu trúc có 1 điểm vào đơn và 1 điểm ra đơn, ví dụ vòng lặp "while" và vòng lặp "for". Ngoài ra, trong rất nhiều loại vòng lặp đó thì các nút vào, các nút ra và các nút điều khiển vòng lặp là giống nhau. Ngoại trừ các vòng lặp "repeat-until" và các vòng lặp không cấu trúc sử dụng "goto" hoặc không sử dụng "goto".
- Hai hoặc nhiều vòng lặp có thể được kết hợp thông qua chuỗi lồng nhau và chuỗi mắt xích liên tục. Mặc dù sự kết hợp không có cấu trúc sử dụng "goto" là được dùng trong nhiều ngôn ngữ lập trình, song chúng không được khuyến khích sử dụng và thường bị hạn chế tối đa.

Đường dẫn vòng thông dụng nhất trong các ngôn ngữ lập trình là "while" và "for"

- "while (C) do { B }", với C là điều kiện của vòng lặp, B là thân vòng lặp. Điểm vào cũng là điểm ra.
- "for **(I** ; C ; **U)** do { B }", với I là giá trị khởi tạo sau khi bắt đầu vòng lặp, U là giá trị cập nhật vòng lặp sau mỗi lần lặp lại, C là điều kiện của vòng lặp, B là thân vòng lặp. Điểm vào cũng là điểm ra của vòng lặp.

Một trong những câu hỏi cơ bản và quan trọng để thử nghiệm là liệu chúng ta có thể xác định số lần lặp lại cho một vòng lặp trước khi các hoạt động kiểm tra diễn ra hay không. Nếu có, nó được gọi là một vòng lặp xác định (điển hình là vòng lặp "for"), nếu không nó là vòng lặp không xác định (điển hình là vòng lặp "while"). Các

vòng lặp xác định thường được dùng để xử lý một số dữ liệu hoặc các thực thể, chẳng hạn như thực hiện một vài xử lý cho mọi phần tử của một mảng có kích thước cố định.

3.1.5.2. Vấn đề của vòng lặp

Mỗi lần chúng ta đi qua một vòng lặp, với một số lượng lặp lại cụ thể, chúng ta lại có một đường dẫn riêng biệt. Khi chúng ta kết hợp hai vòng lặp thành 1 chuỗi mắt xích liên tục, số lượng các đường dẫn riêng biệt có thể nhận được bằng cách nhân các đường dẫn riêng biệt cho mỗi vòng lặp, trong cùng một cách chúng ta nối 2 CFGs có vòng lặp tự do. Tuy nhiên, số lượng có thể có của các lần lặp lại cho một vòng lặp thường lớn. Do đó, sự kết hợp của chúng tạo ra một số lượng lớn hơn các tổng đường dẫn.

Việc lồng 2 vòng lặp thì khác với việc lồng 2 CFG có vòng lặp tự do: Kết quả là số lượng các đường dẫn không còn là M + N -1, nhưng là số lượng lớn hơn nhiều do sự lặp lại. Và số lượng tổng cộng các đường dẫn CFG được xác định bởi công thức:

$$\sum_{i=0}^{M-1} N^i = \frac{N^M - 1}{N - 1}$$ (với N^i là đường dẫn được kết nối)

Nếu N, M là lớn thì tổng đường dẫn sẽ là rất lớn. Do đó, ta phải có các biện pháp thay thế. Ta có thể dựa trên kinh nghiệm và sự quan sát.

3.2.Kiểm thử dòng dữ liệu và phụ thuộc dữ liệu

Trong sự cập nhật các trường hợp kiểm thử CFT, chúng ta đã gặp phải những khó khăn khi dùng chung biến thay vì các hằng số có liên quan đến các điểm quyết định, sự phân tích các giá trị biến số đó đã được thực hiện để loại trừ các đường dẫn không thể xác định. Thực tế các quyết định có tương quan không nhất thiết bao gồm các biến số dùng chung.

3.2.1. Các khái niệm cơ bản. Sự hoạt động của dữ liệu phụ thuộc dữ liệu

Sự sử dụng của các biến số hay thư mục dữ liệu trong các quyết định CFG được gọi là P-use trong phân tích phụ thuộc dữ liệu để chỉ rõ cách sử dụng của nó trong các thuộc tính hoặc các điều kiện. Một loại sử dụng khác được gọi là C-use, hay là cách sử dụng có dùng máy tính. Một cách hiểu thông thường của các tình huống sử dụng trên là các biến số đó hay dữ liệu đó phải được xác định sớm hơn. Vì thế chúng ta có thể thể loại và nhận được các giá trị của nó, và sử dụng vào các mục đích khác nhau. Chúng ta có thể xác định sự hoạt động của các dữ liệu đó như sau:

- Sự xác định dữ liệu thông qua sự hình thành, giá trị ban đầu, nhiệm vụ một cách rõ ràng thông hay trong một số trường hợp thông qua chiều hướng tác động như là: Vị trí bộ nhớ được chia sẻ, hộp thư, các tham số đọc/viết....Và thường được viết tắt là D-operation hay D. Đặc tính cơ bản của D là sự phá hủy, điều đó nghĩa là bất kỳ cái gì được lưu trữ trong các thư mục dữ liệu đều bị xóa bỏ sau khi hoạt động và không thể khôi phục trừ khi sử dụng các kỹ thuật đặc biệt.

- Cách sử dụng dữ liệu trong máy tính thông thường hay trong tính chất, thông thường có liên quan đến C-use hay P-use. Cả 2 cách sử dụng trên đều được gọi chung là U-operation hay ngắn gọn là U. Đặc tính cơ bản của U là không phá hủy, nghĩa là giá trị của các thư mục dữ liệu sẽ vẫn còn tồn tại sau khi nó hoạt động. Tuy nhiên, cách sử dụng loại P của thư mục dữ liệu về tính chất có thể khiến các đường dẫn hoạt động được lựa chọn và theo sau. Cách sử dụng loại C của các thư mục dữ liệu thường được diến ra trên các mẫu biến số hay hằng số trong máy tính hay như các tham số trong các chức năng của chương trình. Cách sử dụng C đó thường ảnh hưởng đến các kết quả máy tính với một vài kết quả biến thiên được xác nhận.

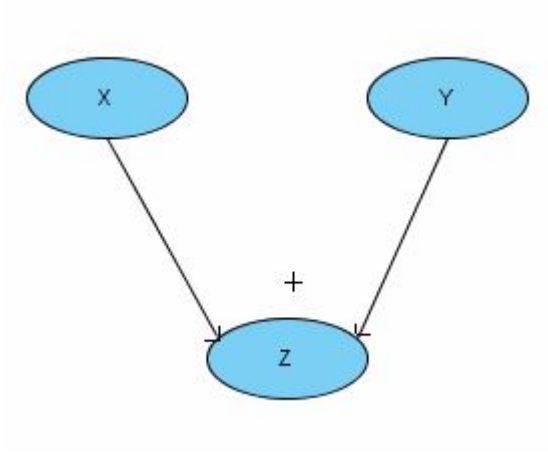
Với sự định nghĩa về 2 cách sử dụng của xử lý dữ liệu, chúng ta có thể xem xét tiếp các mối quan hệ như sau.

- Mối quan hệ D-U: đây là trường hợp sử dụng thông dụng. Khi 1 dữ liệu được sử dụng, chúng ta cần nhận được giá trị của nó được xác nhận trước đó. Hầu hết các phân tích phụ thuộc dữ liệu (DDA) và kiểm thử dòng dữ liệu (DFT) tập trung vào cách sử dụng này.

- Mối quan hệ D-D: mối quan hệ này được diễn tả các trường hợp quá tải. Khi các hoạt động kiểu D trước xóa bỏ hết những gì chứa đựng trước đó. Một trường hợp đặc biệt là khi quan hệ D-D tồn tại mà không có hoạt động U ở giữa, nghĩa là một thư mục dữ liệu được xác định lại cà không có sự xác định nào trước đó được sử dụng. Tình huống này diễn tả 1 vài lỗi phần mềm, hoặc ít ra là sự thiếu hiệu quả bởi vì sự xác định trước đó đều không được sử dụng.

- Mối quan hệ U-U: không có sự ảnh hưởng hay phụ thuộc dữ liệu vì tính tự nhiên của hoạt động kiểu U không phá hủy. Vì thế, những mối quan hệ này không được quan tâm trong DDA và DFT. Như chúng ta đã đề cập trước đây, những sự kết nối này có thể ảnh hưởng tới khả năng có thể thực hiện được các đường dẫn hoạt động khác nhau. Tuy nhiên, như chúng ta sẽ thấy, chúng ta có thể tập trung vào mối quan hệ D-U tương ứng cho mỗi một trường hợp sử dụng để nhận ra các đường dẫn khác nhau

trong CFT hay trong các phần khác nhau của DFT, các quan hệ đó hoàn toàn đảm nhiệm các điều kiện có tương quan với nhau.



VD 3.1 Đồ thị phụ thuộc dữ liệu (DDG): VD về sự định nghĩa dữ liệu thông qua nhiệm vụ.

Quan hệ U-D: được gọi là sự chống sử dụng. Tình huống thú vị với quan hệ này là các thư mục dữ liệu được sử dụng mà chưa từng được xác định trước đó (không có hoạt động dạng D đi trước hoạt động dạng U đầu tiên), điều đó chỉ xảy ra 1 lỗi của phần mềm.

Vì thế, với sự nhận dạng cơ bản đó và sự phân tích của sự kết hợp hoạt động của dữ liệu, một vài vấn đề có thể xảy ra có thể nhận ra ngay lập tức. Với sự phân tích phụ thuộc dữ liệu (DDA) được sử dụng trong kiểm thử dòng điều khiển dữ liệu (DFT), chúng ta tập trung vào quan hệ D-U và các vấn đề khác có liên quan.

3.2.2. Những vấn đề cơ bản của DFT va DDG

Ý tưởng chính của sự kiểm thử dòng điều khiển dữ liệu (DFT) là tiến hành kiểm thử sự chuyển giao chính xác của phụ thuộc dữ liệu trong suốt quá trình hoạt động của chương trình. Từ khi hoạt động của chương trình theo 1 mô hình hoạt động liên tiếp, chúng ta có thể thấy sự phụ thuộc dữ liệu như là 1 phần được gắn vào trong dòng dữ liệu, nơi mà dòng dữ liệu là 1 cơ cấu mà dữ liệu có thể được chuyền tải dọc theo sự hoạt động của chương trình. Các trường hợp kiểm thử có thể nhận được từ những phân tích phụ thuộc dữ liệu (DDA), với sự tập trung vào các mối quan hệ D-U, và mô hình có liên quan mà chúng ta gọi là đồ thị phụ thuộc dữ liệu (DDG). Trong hệ thống DDG, mỗi 1 điểm mô tả sự xác định của 1 thư mục dữ liệu, như là 1 biến số, 1 hằng số hay là 1 cấu trúc dữ liệu kép. Các đường kết nối trong hệ thống DDG mô tả mối quan hệ D-U, hay "được sử dụng bởi". Điều đó có nghĩa là, nếu chúng ta có 1 đường kết nối từ A

đến B, chúng ta phân tích nó như 1 dữ liệu được xác định ở trong A mà được sử dụng để xác nhận ở trong B. Ví dụ như sự thể hiện nhiệm vụ "Z ← X + Y" ở ví dụ trên có thể xác định cho X, Y (C-use) để nhằm mục đích xác định Z. Sự phân tích của 1 chuỗi các quan hệ D-U được sử dụng để xác định các thư mục dữ liệu muộn hơn, có thể được tiến hành để xác nhận sự xác định của thư mục sớm hơn.

Ở DFT, chúng ta tập trung trực tiếp vào phụ thuộc dữ liệu nhận được ở hệ thống DDG thay thế cho sự liên tiếp có sử dụng máy tính hay các dòng điều khiển ở CFT. Một điều chứng tỏ rằng DFT thì sát hơn trong việc kiểm thử tính chất của máy tính, bởi vì những sự phụ thuộc dữ liệu đó ảnh hưởng trực tiếp đến kết quả máy tính, trong khi đó, các dãy hoặc các dòng điều khiển ở CFTđược sử dụng chính do sự hạn chế của các dãy máy móc của chúng ta và các ngôn ngữ chương trình được sử dụng (nếu không, rất nhiều máy tính có thể được sử dụng tương tự nhau). Mặt khác, sự liên tục trong các sự thực hiện chương trình có thể được thay đổi mà không ảnh hưởng đến kết quả. Vì thế, trong sự thực hiện DDA và DFT, chúng ta tách riêng các sự phụ thuộc dữ liệu dễ thay đổi với dãy hoạt động liên tục của chương trình để tập trung vào sự chuyển giao chính xác của các thư mục dữ liệu và các sự phụ thuộc của nó, và không hạn chế các kết quả tính toán chính xác.

Tương tự các kỹ thuật kiểm thử mang tính hệ thống khác, chúng ta tập trung vào khâu chuẩn bị kiểm thử cho DFT, theo các bước:

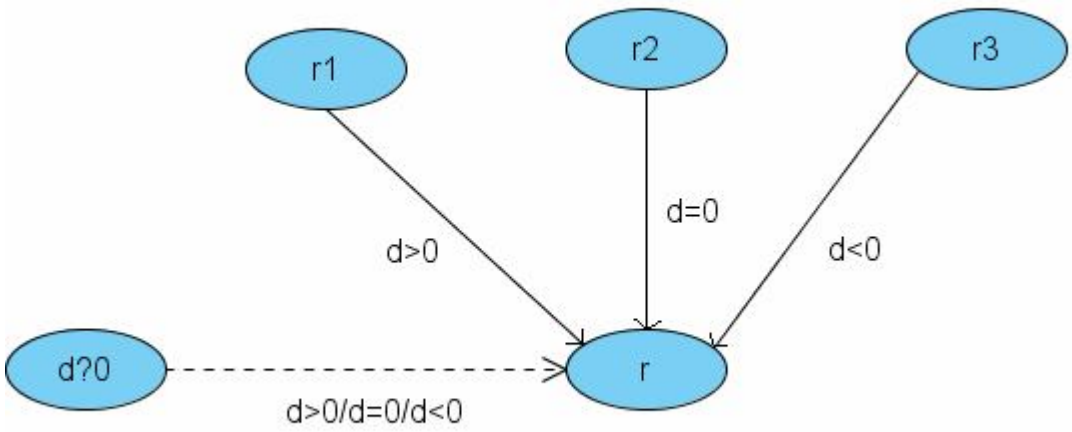
- Xây dựng và thẩm tra hệ thống DDG
- Xác định và lựa chọn các phần dữ liệu cho các trường hợp kiểm thử riêng lẻ dựa trên DDGs. Phần dữ liệu bao gồm cả các thư mục dữ liệu, thường là 1 biến số output, với sự xác định của nó trong các thư mục dữ liệu khác, có thể được lựa chọn từ nhiều sự xác định.
- Cập nhật các phần tử dữ liệu hay các trường hợp kiểm thử bằng cách gán các giá trị input.
- Kế hoạch kiểm tra kết quả.

3.2.3. Các thuộc tính và yếu tố của DDG

Chúng ta có thể mô tả đặc điểm của các yếu tố đồ thị khác nhau trong DDGs như sau:

Mỗi 1 điểm mô tả sự xác định của 1 thư mục dữ liệu x, được biểu thị là D(x) và được mô tả là x nằm trong 1 hình Oval trong DDG. Các điểm này có thể được phân loại thành 3 loại:

- Các điểm kết quả và output mà mô tả các kết quả tính toán cho chương trình đang kiểm thử.
- Các điểm input hay hằng số mô tả input được cung cấp từ người sử dụng hay các hằng số được xác định sẵn. Các điểm này mô tả các điểm cuối mà không cần được phân tích thêm nữa.
- Các điểm dự trữ hoặc trung gian là những điểm không là input hay output. Trong hầu hết các tính toán, các điểm này được bắt đầu để làm các thủ tục tính toán trở nên thuận tiện nhờ đó các kết quả từ input nhận được 1 cách đơn giản.



Hình 3.2 Đồ thị DDG: ví dụ của điểm bộ chọn lọc dữ liệu

Mối quan hệ được mô hình hóa trong DDG luôn là mối quan hệ D-U, hay là mối quan hệ "được sử dụng bởi".

Trường hợp đặc biệt đối với các cấu trúc DDG ở trên là những sự xác định có lựa chọn của các thư mục dữ liệu nào đó có sử dụng các điểm chọn lọc dữ liệu, Ví dụ trong sự xác định số lượng nghiệm thực cho phương trình bậc 2: $ax^2 + bx + c = 0$, kết quả ra phụ thuộc vào giá trị $d=b^2 – 4ac$ như sau:

(d>0) thì r ← 2;

(d=0) thì r ← 1;

(d<0) thì r ← 0;

Ta thấy 3 giá trị có thể có của ra được đánh dấu là r1, r2, r3. Kết quả cuối cùng r sẽ được lựa chọn giữa 3 giá trị dựa trên điều kiện d. Vì thế, chúng ta có thể đặt r ở điểm lựa chọn dữ liệu, kết nối r1, r2, r3 với r là các đường kết nối dữ liệu trong, và điểm điều kiện "d?0" kết nối với r qua đường kết nối điều khiển trong. Chúng ta có thể phân biệt đường dẫn điều khiển trong cà đường dẫn dữ liệu trong bằng cách sử dụng đường gạch (-). Chỉ có một r được lựa chọn từ các giá trị đề cử r1, r2, r3. Bằng cách nối các điều khiển dễ dàng trong hình 3.2.

Chú ý trong ví dụ đó, cả 2 P-use và C-use đều được giới thiệu: C-use được kết nối với tính toán theo biến số r1, r2, r3 trong mỗi nhánh của DDG, nơi mà các hằng số 0, 1, 2 được sử dụng. P-use được kết nối với biến số d và hằng số 0 cho sự dự đoán trước trong đường kết nối điều khiển trong.

Chúng ta có thể coi DDG như 1 hệ thống bao gồm các điểm input, output, các điểm trung gian, các điểm lựa chọn dữ liệu và các đường kết nối cùng thuộc tính của chúng. Một lần nữa, sự tập trung vào output hoặc kết quả và sự giải quyết của nó thông qua DDGs trong lĩnh vực các hằng số và biến số input. Bởi vì các quy trình cho sự xử lý dữ liệu thông qua chuỗi phía sau sử dụng các quan hệ D-U, hệ thông DDG thường chỉ ra các tính chất sau:

- Thường chỉ có 1 thư mục dữ liệu output hay biến số, hoặc cùng lắm là 1 vài thư mục đó.
- Có nhiều hơn các biến số và hằng số input.
- Thường có nhiều đường dẫn kết nối trong (kết nối nội bộ).

3.2.4. Quy trình chung cho sự xây dựng đồ thị DDG

Khi sự mã hóa thực tế (hoặc thiết kế chi tiết) có giá trị, thì xu hướng tự nhiên của chúng ta là đi theo từ khi bắt đầy đến khi kết thúc để xác định đồ thị DDG. Trong quá trình đó, chúng ta có thể xác nhận các biến số và hằng số input trước, sau đó đi theo sự mã hóa để nhận dạng các biến số trung gian khác, kết nối chung và cuối cùng bao gồm cả các biến số output.

Trong sự tập trung vào các kết quả tính toán, sự xây dựng DFT được xếp thành hành theo sự giải quyết dữ liệu bậc thang từ output đến input.

3.2.5. Xử lý các đường vòng

Cách sử dụng của các đường vòng sẽ làm phức tạp thêm cho đồ thị DDG bởi vì nhiều sự phụ thuộc dữ liệu có liên quan. Dữ liệu được sử dụng trong mỗi sự lặp lại sẽ phụ thuộc vào toàn bộ sự lặp lại trước đó, tương ứng với sự kết nối liên tiếp n mức. Phân tích phụ thuộc dữ liệu thậm chí cho 1 đường vòng với số lần lặp lại vừa phải sẽ là không thực tế. Trường hợp này tệ hơn nhiều so với CFT, nơi mà chỉ đường dẫn hoạt động là được phân tích, còn các tính toán chi tiết và các sự xác định dữ liệu có liên quan trong mỗi lần lặp lại nhận được ở DDG thì không.

Tuy nhiên, rất nhiều đường vòng trong quy trình thực hiện thực tế có thể không tương ứng với mô hình lý thuyết hay các chi tiết kỹ thuật chức năng.

Đối với trường hợp đó, hệ thống làm theo 2 kỹ thuật, trước hết kiểm thử đường vòng bằng CFT, sau đó phá vỡ đường vòng và dồn vào 1điểm xử lý.

Chương 4. KIỂM THỬ DỰA TRÊN FSM CỦA ỨNG DỤNG WEB

Với sự lan khắp nơi của world wide web (*WWW*), kiểm thử và đảm bảo chất lượng cho web ngày càng trở nên quan trọng.

4.1. Các đặc điểm của các ứng dụng web

Ứng dụng web có nhiều đặc điểm độc đáo mà ảnh hưởng đến sự lựa chọn những kỹ thuật thích hợp cho sự kiểm thử web. Một trong những khác biệt cơ bản là tài liệu (*document*)và sự tập trung thông tin(*information focus*) cho trang web so với sự tập trung tính toán (*computational focus*) cho phần lớn phần mềm truyền thống. Mặc dù một số khả năng tính toán được phát triển trong các ứng dụng web, tài liệu mới hơn và tìm kiếm cũng như sự truy tìm thông tin retrieval vẫn còn sự dụng có ưu thế cho phần lớn người sử dụng web. Ngoài ra, những thế mạnh của điều hướng, định vi(*navigational facility*) là 1 phần quan trọng của các ứng dụng dựa trên web, với phần lớn tài liệu HTML (*hyper-text markup language*) sử dụng thông đóng một vai trò trung tâm trong việc cung cấp cả thông tin và liên kết định vị, điều hướng (*navigational links*). Về mặt này, các ứng dụng dựa trên web giống như nhiều sản phẩm phần mềm điều khiển bằng menu(*menu-driven software products*). Tuy nhiên, cũng có một số sự khác biệt đáng kể, như sau:

- Phần mềm điều khiển bằng menu truyền thống vẫn còn tập trung vào một số tính toán, trong khi các ứng dụng webbased thì tập trung vào thông tin và tài liệu.

Hình 4.1 Các ứng dụng web đa tầng

- Phần mềm điều khiển bằng menu truyền thống thường tách các khung điều hướng của nó từ sự tính toán của mình; trong khi sự tập trung thông tin là chặt chẽ lẫn cho các ứng dụng dựa trên web.
- Trong phần mềm điều khiển bằng menu truyền thống, thường có một trình đơn đầu trang top menu duy nhất phục vụ như điểm đầu vào *entry point*; trong khi với các ứng dụng dựa trên web, có khả năng bất kỳ trang web hoặc nội dung web có thể là điểm khởi đầu *starting point*. Những *entry piont* hay những *starting point* này thường tương ứng với trạng thái ban đầu *initial states* trong một FSM. Những sự khác nhau tương tự dành cho các điểm kết thúc *end points* hoặc trạng thái cuối cùng *final states*, với phần mềm điều khiển menu truyền thống có lối ra hạn chế trong khi các ứng dụng dựa trên web thường có thể kết thúc ở bất kỳ điểm nào khi người dùng chọn để thoát khỏi trình duyệt web hoặc ngừng hoạt động của trình duyệt web.
- Một khác biệt đáng kể là sự khác biệt về chất lượng trong số lượng lớn các trang điều hướng của các ứng dụng dựa trên web ngay cả đối với các trang web có kích thước vừa phải và số lượng hạn chế các thực đơn cho tất cả các ứng dụng điều khiển menu truyền thống.
- Các ứng dụng trên nền Web thông thường liên quan nhiều đến phương tiện hỗ trợ đa dạng hơn phần mềm điều khiển bằng menu truyền thống. Chức năng của web thường được phân phối thành nhiều lớp và hệ thống con như minh họa trong hình 4.1. Chúng tôi cần chắc chắn rằng tất cả các chức năng và các thành phần liên quan làm việc tốt với nhau, để loại trừ nguồn gốc rủi ro hoặc để giảm nguy cơ rủi ro.

Tương tự như kiểm thử tổng quát, kiểm thử cho các ứng dụng web tập trung vào công tác phòng chống rủi ro web, giảm các cơ hội cho những rủi ro đó. Vì vậy, chúng ta cần phải xem xét các vấn đề phổ biến và khái niệm liên quan như rủi ro web, thiếu sót, và lỗi, trước khi chúng tôi có thể tiến hành với việc lựa chọn các kỹ thuật kiểm thử thích hợp để xác định và loại bỏ những vấn đề này và nguồn gốc vấn đề.

4.2.Kiểm tra đặc điểm của các vấn đề web

Chúng ta có thể xem xét những nguồn rủi ro sau:

- Rủi ro mạng network hoặc máy chủ host: Rủi ro phần cứng, rủi ro hệ thống tại máy chủ đích hoặc máy chủ nguồn, cũng như rủi ro mạng, có thể dẫn đến rủi ro web. Những rủi ro này phần lớn liên quan tới tầng middleware và tầng web server trong hình 4.1. Tuy nhiên, sự rủi ro như vậy không khác với rủi ro hệ thống hay rủi ro mạng, và có thể được phân tích bằng kỹ thuật hiện có.
- Rủi ro trình duyệt Browser: rủi ro của trình duyệt liên quan đến các vấn đề ở các tầng cao nhất trong Hình 4.1 về phía client. Những rủi ro có thể được xử lý cùng một cách như rủi ro sản phẩm phần mềm, do đó kỹ thuật hiện có để kiểm thử các phần mềm có thể được sử dụng.
- Rủi ro nội dung Content hay rủi ro tài nguyên Source: Rủi ro Web cũng có thể được gây ra bởi các nguồn thông tin của chính nó ở phía máy chủ, liên quan đến tầng thấp nhất trong Hình 4.1.

Ngoài ra, các lỗi người dùng cũng có thể gây ra các vấn đề mà có thể được giải quyết thông qua sự hướng dẫn người sử dụng, thiết kế khả năng sử dụng tốt hơn, v v... Các rủi ro máy chủ, mạng, trình duyệt đã đề cập ở trên có thể được giải quyết bởi cộng đồng web toàn cầu bằng cách sử dụng các kỹ thuật hiện có. Tuy nhiên, rủi ro nội dung và tài nguyên web thường liên quan trực tiếp đến các dịch vụ hoặc chức năng mà ứng dụng dựa trên web đang cố gắng cung cấp. Ngoài ra, khả năng sử dụng là một trong những mối quan tâm chính cho người sử dụng web mới làm quen, còn độ tin cậy ngày càng trở thành một chính mối quan tâm cho người sử dụng web phức tạp. Vì vậy, chúng tôi sẽ tập trung về rủi ro tài nguyên web và cố gắng để đảm bảo độ tin cậy của ứng dụng dựa trên web từ quan điểm của người dùng trong trường hợp nghiên cứu này. Các thành phần web liên quan bao gồm :

- Tài liệu HTML, vẫn là hình thức phổ biến nhất cho các tài liệu trên web.
- Java, JavaScript, và ActiveX thường sử dụng để hỗ trợ những sự thực thi độc lập nền tảng.

- Cgi-Bin Scripts sử dụng để truyền dữ liệu hoặc thực hiện một số hoạt động khác.
- Cơ sở dữ liệu, phần chính của các phụ trợ backend.
- Các thành phần đa phương tiện được sử dụng để đưa ra và xử lý thông tin đa phương tiện.

4.3. FSMs trong kiểm thử web

Từ những quan điểm của người sử dụng web, mỗi ứng dụng dựa trên web hoặc chức năng bao gồm nhiều thành phần, giai đoạn, hoặc các bước, nhìn thấy được cho người sử dụng web, và thường bắt đầu bằng chúng. Do đó, sự chuyển tiếp dựa trên mô hình FSMS là thích hợp cho các ứng dụng loại này. Chúng tôi tiếp tục xem xét bốn yếu tố cơ bản của FSMS và sắp xếp chúng lên các ứng dụng dựa trên web:

- Mỗi trang web tương ứng với 1 trạng thái trong FSMs. Khả năg gây bất kỳ trang nào cũng có thể là trạng thái ban đầu và bất kỳ trang nào cũng có thể là trạng thái cuối cùng.
- Sự chuyển tiếp tương ứng với các điều hướng web (*web navigations*) sau liên kết siêu văn bản *hypertext links* được nhúng trong tài liệu HTML và các nội dung web khác. Một trường hợp đặc biệt là người dùng có thể chọn theo một link đã lưu trước đó (trang yêu thích đã đánh dấu *bookmarked favorites*), hoặc trực tiếp gõ một URL . Việc sử dụng các công cụ điều hướng cách 2 làm cho sự chuyển tiếp khó đoán trước hơn. Tuy nhiên, cũng có là hai nhân tố đáng chú ý trong sự điều hướng web như sự chuyển tiếp trong mô hình FSMs:
  - Từ quan điểm nhà cung cấp dịch vụ trên web và Internet, thì việc đảm bảo nội dung chính thức là chính xác luôn quan trọng hơn việc đảm bảo những trang đã đánh dấu (*bookmarks*) bởi người sử dụng hoặc tự gõ URL là cập nhật hoặc chính xác.
  - Có bằng chứng thực nghiệm cho thấy đại đa số các điều hướng web theo sau các liên kết siêu văn bản nhúng thay vì sử dụng đánh dấu hoặc đánh địa chỉ URL. Ví dụ, với trang web www.**seas.**smu.**edu** đã nghiên cứu (Ma và Tian, 2003), 75,84% của các điều hướng có nguồn gốc từ các liên kết nhúng trong cùng một trang web, chỉ có 12,42% là có nguồn gốc từ người sử dụng, và phần còn lại từ bên ngoài và các liên kết khác.

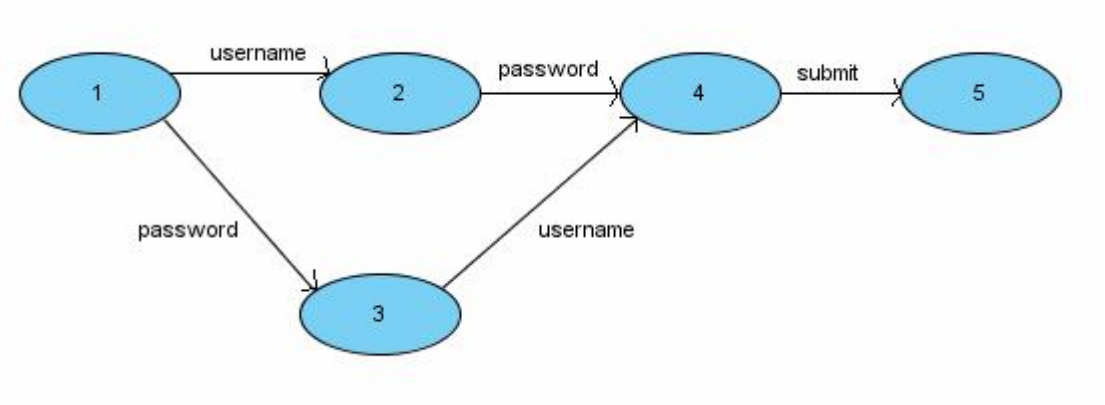
Bởi vậy, chúng tôi chọn tập trung vào các liên kết điều hướng nhúng và chụp chúng trong FSMS để kiểm thử web.

- Các đầu vào và đầu ra liên kết với các điều hướng như vậy là khá đơn giản và dễ hiểu: đầu vào là nhấn vào liên kết nhúng được hiển thị như nội dung nổi bật - sáng

nhất (*highlighted content*); và đầu ra tương ứng là tải về các trang yêu cầu hoặc nội dung với những thông điệp đi kèm cho thấy tình trạng HTML, lỗi hoặc các thông báo khác, v v....

Có một nhược điểm rõ ràng để kiểm thử web bằng cách sử dụng FSMS như trên là: số lượng các trang web cho ngay cả một trang web có kích thước vừa phải có thể là hàng nghìn hoặc nhiều hơn nữa. Do đó, sẽ có số lượng đáng kể của các trạng thái trong các FSMS. Ta có thể làm theo phương pháp thống kê.

Ví dụ website môn học của trường Đại học Công Nghệ, thì ban đầu có trang login bắt nhập username, password đúng và sau đó submit thì mới vào được trang nội dung. Như vậy, tại trang login có các trạng thái: trạng thái ứng với input là tài khoản đúng, trạng thái ứng với input là mật khẩu đúng, trạng thái ứng với input là click vào nút submit.



Hình 4.2 Ví dụ về trang login

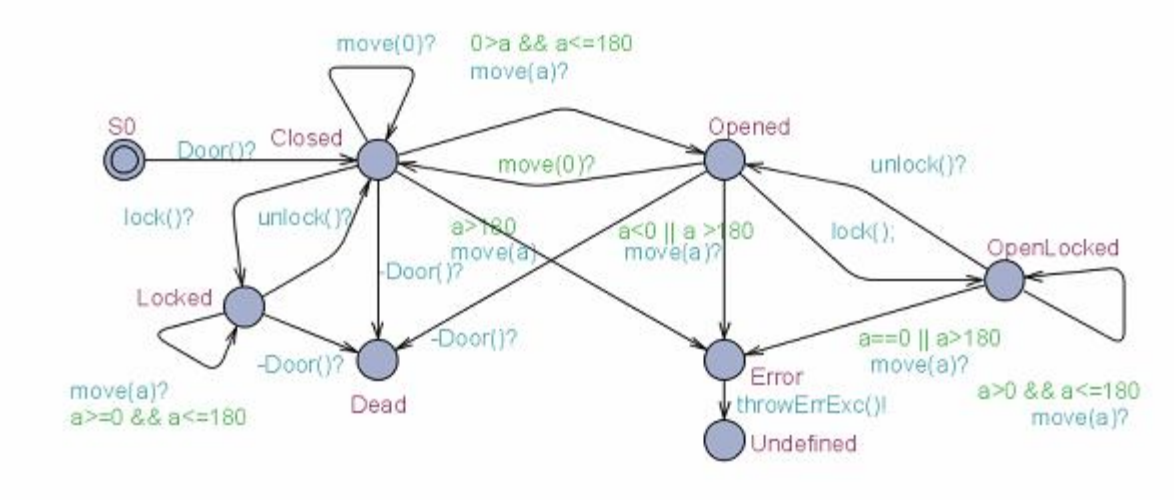
Ví dụ về đối tượng:

Có 1 lớp Door, Door door = new Door();

Kiểm tra xem door được mở hay đóng?

door.OpenLocked();

qua con đường: Door()?.move(1)?.lock()?



Hình 4.3 Ví dụ về Đối tượng của lớp Door

KẾT LUẬN

Tổng quát lại chúng ta có được những điều như sau:

FSMs là những mô hình tiêu chuẩn trong nghiên cứu cơ bản của khoa học máy tính. FSMs bao gồm 4 yếu tố, được chia làm 2 nhóm: là yếu tố tĩnh và yếu tố động. Yếu tố tĩnh bao gồm trạng thái và sự chuyển tiếp. Số lượng của 2 loại trạng thái này là giới hạn. Sự chuyển tiếp từ trạng thái này sang trạng thái khác là duy nhất. Còn yếu tố động bao gồm input được cung cấp cho FSMs, và output được rút ra từ FSMs thông qua những quá trình xử lý động của FSMs. Số lượng các output và input là giới hạn, nếu số lượng các input và output là quá lớn thì chúng ta thường nhóm chúng thành các nhóm phân chia nhỏ.

FSMs được mô tả bằng phương pháp đồ họa, tuy nhiên phương pháp này không thực hiện được khi mà số lượng trạng thái là lớn. Khi chúng ta có nhiều hơn 20- 30 trạng thái thì bản vẽ đồ họa sẽ trở lên rối loạn và rất khó theo dõi. Khi đó chúng ta dùng cách mô tả ma trận để thực hiện FSMs. Ngoài 2 cách thể hiện trên, FSMs còn được mô tả bằng phương pháp danh sách thống kê. Tuy vậy, phương pháp đồ họa được sử dụng nhiều nhất.

FSMs được sử dụng để mô hình hóa cả 2 trường hợp: xử lý hệ thống bên trong (black box view) và hoạt động chi tiết của những hoạt động thực thi rõ ràng (white box view). Ứng dụng rộng rãi nhất của kiểm thử dựa trên FSMs là trong lĩnh vực phần mềm điều khiển bằng bảng chọn, trong đó mỗi một thực đơn yêu cầu một số input và cung cấp một số output thường được cung cấp thêm bởi một thực đơn mới. Trường hợp đặc biệt của phần mềm bằng bảng chọn là cách sử dụng của trang web mà chúng ta đã tìm hiểu trong đề tài này. Sự kiểm thử dựa trên FSMs thường thích hợp cho những hệ thống với những trạng thái và sự chuyển tiếp được xác nhận rõ ràng. Nó còn có ứng dụng to lớn trong phần mềm định hướng đối tượng.

Hạn chế lớn nhất của kiểm thử dựa trên FSMs là không có khả năng xử lý số lượng trạng thái lớn. Mặc dù sự phân tầng FSM có thể giúp làm giảm bớt vấn đề, nhưng vẫn có hạn chế trong việc phân chia các tầng rõ ràng.

Chúng ta cần phải đảm bảo các chức năng, hiệu suất, độ tin cậy, tiện ích, ... của các thành phần web này và các ứng dụng của chúng. Để làm điều này, nhiều loại kiểm thử web hiện nay có thể được thực bao gồm: kiểm thử chức năng, kiểm thử load, kiểm thử stress, dựng hình trình duyệt browser rendering, và kiểm thử tiện ích. Tuy nhiên, kiểm thử như vậy thường tập trung vào một khu vực nhỏ hoặc một khía cạnh cụ thể của vấn đề chất lượng trang web. Việc sử dụng FSMs trong kiểm thử web sẽ đảm bảo tổng thể hiệu suất thỏa đáng theo quan điểm của người dùng cho các trình tự và các kịch bản sử dụng. Tuy nhiên nó cũng có hạn chế lớn là số lượng các trang web là lớn, do đó ta phải dùng phương pháp thống kê.

Từ quan điểm về cấu trúc hay sự cài đặt, một hệ thống phần mềm được tạo thành từ các thành phần tương tác, các mô-đun, hoặc các hệ thống con. Nó được tạo ra để hướng tới đối tượng là khách hàng, hệ thống tổng thể bao gồm các chức năng liên kết hoặc các hoạt động. Máy trạng thái hữu hạn (FSMs) và các mô hình sử dụng có liên quan mà tôi đã giới thiệu ở chương trước có thể được sử dụng để mô hình hóa và thử nghiệm các chức năng hệ thống có liên hệ với nhau, sự cài đặt, và cách sử dụng có liên quan. Tôi tập trung vào các trạng thái, sự chuyển tiếp, và cách sử dụng có liên quan chứ không chú ý nhiều đến sự ảnh hưởng qua lại ngoại trừ những sự kết nối đơn giản. Trong phần này, tôi giới thiệu các kỹ thuật thử nghiệm để giải quyết sự ảnh hưởng qua lại liên hợp vượt ra ngoài những sự kết nối bước 1 trong FSMS. Hai loại chính của sự ảnh hưởng qua lại này là:

- Sự tương tác dọc theo một đường dẫn thi hành, nơi mà các hoạt động sau bị ảnh hưởng bởi toàn bộ các hoạt động trước đó.
- Những tương tác cụ thể giữa các mục dữ liệu trong quy trình thực hiện.

Sự kiểm thử của sự tương tác ở trên thường được gọi lần lượt là: sự kiểm thử dòng điều khiển (CFT) và sự kiểm thử dòng dữ liêu (DFT). Những kỹ thuật này là những kỹ thuật white-box truyền thống áp dụng cho việc kiểm thử dòng chức năng các mức của hệ thống, sự phụ thuộc dữ liệu và sự tương tác có liên quan.

Sự kiểm thử dòng điều khiển(CFT) là sự mở rộng một cách tự nhiên và trực tiếp của sự kiểm thử FSM với một loại chuyên dụng của FSMs gọi là đồ thị dòng điều khiển (CFGs) và tập trung vào hoàn thành đường dẫn thực thi thay vì trạng thái hoặc các liên kết .

So với thử nghiệm dựa trên FSMs, CFT dựa trên CFGs tập trung vào các đường dẫn hoàn chỉnh và các quyết định cũng sự tương tác dọc theo đường dẫn thi hành. Bởi vì tập trung vào các đường dẫn đó nên số lượng các trường hợp thử nghiệm cũng tăng

hơn nhiều so với thử nghiệm bằng FSMs xét về sự tương tự về cấu trúc (trạng thái và các liên kết), đặc biệt khi đường dẫn vòng được đề cập. Do đó, năng suất tăng lên ở các quyết định động và các vấn đề tương tác được đi kèm chi phí tăng lên do tăng lên các trường hợp thử nghiệm. Lợi ích cần phải được cân bằng với chi phí để đạt tới những giải pháp tối ưu cho môi trường ứng dụng cụ thể. Trong hầu hết các chương trình tập trung tính toán, bao gồm hầu hết hệ thông phần mềm truyền thống, chỉ trạng thái và liên kết sẽ là không đủ bởi vì các quyết định động được kết nối với nhau dọc theo đường dẫn thi hành. Vì vậy, CFT thường là một bước cần thiết trong các kỹ thuật kiểm thử khác nhau cho các hệ thống đó. Chúng ta cần tiến xa hơn CFT để kiểm tra những tương tác chi tiết có được trong phân tích phụ thuộc dữ liệu và kiểm thử dòng dữ liệu có liên quan.

Do sự phức tạp với số lượng lớn các đường dẫn khi chúng ta sử dụng các cấu trúc điều khiển phức tạp, đặc biệt là khi các vòng lặp lồng nhau được sử dụng, CFT thường được áp dụng như kỹ thuật kiểm thử white-box cho những chương trình nhỏ, hoặc cho những đơn vị chương trình nhỏ trong chương trình kiểm thử đơn vị. Nếu chúng ta muốn sử dụng nó cho các hệ thống phần mềm lớn hơn, chúng ta phải giảm thiểu số lượng đường dẫn đến 1 mức có thể quản lý được, ví dụ mỗi nút đại diện cho một chức năng chính (black-box view) hoặc một thành phần chính (white-box view).

CFT cũng có thể được tăng cường để hỗ trợ việc sử dụng kiểm thử thống kê dựa trên sự sử dụng.

Trong sự cập nhật các trường hợp kiểm thử CFT, chúng ta đã gặp phải những khó khăn khi dùng chung biến thay vì các hằng số có liên quan đến các điểm quyết định, sự phân tích các giá trị biến số đó đã được thực hiện để loại trừ các đường dẫn không thể xác định. Thực tế các quyết định có tương quan không nhất thiết bao gồm các biến số dùng chung.

Giữa các kỹ thuật kiểm thử, DFT gần CFT nhất, cả 2 đều cố gắng kiểm thử sự xử lý chính xác của sự hoạt động chung. Vì thế, sự ứng dụng của CFT và DFT thì tương tự nhau. Cả 2 kỹ thuật đó đều thích hợp cho các nhiệm vụ tính toán truyền thống, với CFT thì tập trung hơn vào các quyết định được bao gồm và các đường dẫn được xác lập. Còn DFT thì tập trung vào các kết quả tính toán và sự phụ thuộc dữ liệu có liên quan. Theo nghĩa đó, CFT thì định hướng về xử lý hơn, được minh họa bằng các đường dẫn từng bước một, còn DFT thì được định hướng về kết quả nhiều hơn, được minh họa bởi các lớp được sắp xếp theo hình quạt.

Mặc dù điểm khởi đầu cho cả CFT và DFT là hệ thống FSMs, chúng ta đã phát triển các mô hình cho chúng để tập trung vào các vấn đề khác nhau, CFT và DFT đều có những quan hệ mật thiết về ứng dụng của chúng, như sau:

- CFT dựa vào đồ thị CFG, một dạng đặc trưng của đồ thị FSM, trong khi DFT dựa vào đồ thị DDG, điều đó xa rời với đồ thị FSM.
- Đồ thị CFG gần giống với sự mã hóa chương trình hay dòng hoạt động chung thường được kết nối với các mô hình tính toán liên tục. Đồ thị DDG nhận được nhiều chi tiết về sự ảnh hưởng qua lại và sự phụ thuộc chủ yếu hơn trong khi bỏ sót những thông tin phụ liên tục nhận được ở đồ thị CFG.
- Đồ thị DDG nói chung là phức tạp hơn đồ thị CFG
- Khả năng xử lý đường vòng ở DFT thì hạn chế hơn rất nhiều so với CFT.
- Cả 2 CFT và DFT đều được ứng dụng cho những chương trình nhỏ, những phần nhỏ của hệ thống phần mềm lớn.
- Trong rất nhiều hệ thống, CFT và DFT có thể cùng được sử dụng để đảm bảo chất lượng của sản phẩm, thường thường CFT được sử dụng trước DFT vì những tính đơn giản có quan hệ của nó và các mối ràng buộc gần với sự mã hóa chương trình.
- Tương tự với CFT, DFT chứa đựng nhiều chi tiết hơn trong các mô hình của nó và sử dụng những chi tiết đó trong quy trình kiểm thử. Thông thường, những thông tin chi tiết đó chỉ có thể được nhận được dựa trên sự mã hóa chương trình và thiết kế chi tiết, điều đó làm chúng có thể được sử dụng như là các kỹ thuật kiểm thử white-box. Tuy nhiên, DFT thì gần các chi tiết kỹ thuật hơn là CFT, trong nó tập trung vào kết quả thay vì tập trung vào quy trình xử lý đường dẫn được tiến hành để có được các kết quả.Theo khía cạnh này, DFT có thể được sử dụng như kỹ thuật kiểm thử black-box cho rất nhiều trường hợp, bao gồm cả kiểm thử mức độ đối tượng cho các hệ thống được định hướng theo đối tượng.
- DFT có thể được nâng cấp để trợ giúp cho sự kiểm thử mang tính thống kê dựa trên cách sử dụng. Ví dụ, khi chúng ta sử dụng các mô hình có thứ bậc để thực hiện DFT cho các hệ thống lớn, thì các lớp dữ liệu quan trọng hay các lớp dữ liệu được liên kết với khả năng sử dụng có thể xảy ra có thể được mở rộng ra đồ thị DDG mức độ thấp hơn.
- Mở rộng ứng dụng của nó cho tính toán và các chức năng hệ thống được định hướng dựa trên kết quả và cho cả sự thực hiện của nó, DFT có thể được ứng dụng trong nhiều trường hợp ứng dụng khác, nhờ những thông tin nhận được từ đồ thị DDG của nó. Điều quan trọng nhất trong các ứng dụng khác của nó là các cách sử dụng của

các phân tích phụ thuộc dữ liệu trong các hệ thống được phân bổ và các hệ thống song song. Nếu chúng ta khái quát hóa đồ thị DDG để nhận được sự phụ thuộc cần thiết giữa các nhiệm vụ hệ thống khác nhau thay thế cho các thư mục dữ liệu, chúng ta có thể sử dụng đồ thị DDG đó để giúp những sự thực hiện hệ thống chung tăng đến tột đỉnh: Bất kỳ khi nào không có sự phụ thuộc được mô tả trong đồ thị DDG giữa các nhiệm vụ tính toán khác, chúng có thể được chạy song song. Các kết quả thực hiện sóng song có đồng bộ hóa để có được các kết quả chung. Thực tế, sự đồng bộ hóa là 1 cơ cấu có sẵn trong DFT, nếu chúng ta phân tích input thành 1 điểm trong DDG như những nhiệm vụ. Vì thế DFT ứng dụng trong kiểm thử đồng bộ hóa.

Từ đó ta rút ra những lưu ý:

- Nói chung, có 2 thành phần cơ bản ở bất ký 1 sự tính toán hay 1 nhiệm vụ xử lý thông tin nào; thành phần dữ liệu và thành phần điều khiển được tổ chức lại với nhau thông qua vài kỹ thuật thực hiện.
- Các phân tích cơ bản dựa trên đồ thị của FSM được phát triển để phân tích và kiểm thử những đường dẫn dòng điều khiển chung và những ảnh hưởng qua lại giữa các thư mục dữ liệu khác nhau thông qua kiểm thử dòng điều khiển (CFT) va kiểm thử dong dữ liệu (DFT). Cơ sở của DFT là sự phân tích phụ thuộc dữ liệu (DDA) có sử dụng các đồ thị phụ thuộc dữ liệu (DDG). Kiểm thử CFT cho các quyết định cơ bản hoặc các vấn đề dòng điều khiển và hạn chế sự đồng bộ. Cả 2 mô hình CFT và DFT có thể hoặc là dùng trong white-box, hoặc là dùng trong black-box, song vẫn thiên về white-box.
- Bên cạnh sự sử dụng đơn lẻ của CFT và DFT, chúng cũng có thể được kết hợp với nhau.
- Một đặc thù thông thường để chạy tốt những kỹ thuật thử nghiệm này là sự tập trung của chúng vào các chi tiết và cách sử dụng thông tin cụ thể trong việc vận hành kiểm thử thực sự. Vì thế, những kỹ thuật này thông thường phù hợp cho những kiểm thử nhỏ, đơn vị kiểm thử của những hệ thống phần mềm lớn và sự kiểm soát ở mức độ cao, dữ liệu và sự hòa hợp các chuyển tiếp cho những hệ thống lớn. Thông thường trong hệ thống phần mềm lớn, con người muốn sử dụng 1 sự kết hợp của những kỹ thuật kiểm thử khác nhau cho những mục đích khác nhau và dưới những môi trường khác nhau để làm tăng hiệu quả cao nhất những kỹ thuật kiểm thử khác nha mà vẫn giữ chi phí ở mức hợp lý.

TÀI LIỆU THAM KHẢO

1. Brunel University Research Archive

2.Wiley - Software Quality Engineering - Testing, Quality Assurance and Quantifiable Improvement - 2005 ! - (By Laxxuss)