Kỷ yếu Hội nghị Quốc gia lần thứ VIII về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR); Hà Nội, ngày 9-10/7/2015
DOI: 10.15625/vap.2015.000202

SINH DỮ LIỆU THỬ CHO ỨNG DỤNG LUSTRE/SCADE SỬ DỤNG ĐIỀU KIỆN KÍCH HOẠT

Trịnh Công Duy[1], Nguyễn Thanh Bình[1], Ioannis Parissis[2]

[1] Trường Đại học Bách Khoa, Đại học Đà Nẵng, Đà Nẵng, Việt Nam
[2] Viện Đại học Bách khoa Grenoble, Cộng hòa Pháp.
tcduy@dut.udn.vn, ntbinh@dut.udn.vn, ioannis.parissis@grenoble-inp.fr

TÓM TẮT *- Kiểm thử là một trong những hoạt động quan trọng trong tiến trình phát triển phần mềm. Phần mềm phát triển ngày càng phức tạp, yêu cầu chất lượng ngày càng cao hơn, vai trò kiểm thử càng quan trọng hơn, đặc biệt là trong phát triển phần mềm cho các hệ thống phản ứng. Hệ thống phản ứng được phát triển phổ biến trong các lĩnh vực như hàng không, năng lượng, hạt nhân… Những hệ thống này thường yêu cầu rất cao về chất lượng và đòi hỏi các hoạt động kiểm thử nghiêm ngặt trước khi chúng được triển khai sử dụng. Lustre/SCADE là ngôn ngữ được sử dụng rộng rãi để phát triển phần mềm cho các hệ thống phản ứng, một lĩnh vực mà chất lượng phần mềm được yêu cầu hết sức nghiêm ngặt, gần như là không được phép xảy ra bất cứ một lỗi nhỏ nào. Điều này đòi hỏi việc kiểm thử phải được tiến hành thường xuyên và liên tục khi có bất cứ sự thay đổi nào trên ứng dụng. Bên cạnh đó, đối với những ứng dụng trong lĩnh vực này, việc kiểm thử thủ công sẽ rất khó thực hiện và không mang lại hiệu quả, do đó yêu cầu cần phải tự động hóa hoạt động kiểm thử cho các ứng dụng Lustre/SCADE. Trong bài báo này, chúng tôi tập trung nghiên cứu việc kiểm thử tự động cho các ứng dụng Lustre/SCADE. Chúng tôi đề xuất kỹ thuật sử dụng các điều kiện kích hoạt trên mạng lưới toán tử (operator network) để sinh ra các dữ liệu thử một cách tự động. Trong đó, kỹ thuật kiểm chứng mô hình (model checking) được sử dụng trong việc sinh tự động các dữ liệu thử này. Cuối cùng, chúng tôi thử nghiệm giải pháp này trên một chương trình Lustre/SCADE trong hệ thống điều khiển nhiệt độ môi trường.*

Từ khóa *- Hệ thống phản ứng; Kiểm chứng mô hình; Kiểm thử; Thuộc tính bẫy; Lustre/SCADE; Sinh dữ liệu thử.*

I. GIỚI THIỆU

Việc phát hiện và khắc phục lỗi cho các phần mềm là một công việc đòi hỏi nhiều nỗ lực và chi phí trong phát triển phần mềm. Hiện nay, phần mềm đang được ứng dụng ngày càng rộng rãi trong nhiều lĩnh vực, do đó chất lượng phần mềm ngày càng được quan tâm. Trong công nghệ phần mềm, kiểm thử là hoạt động cơ bản nhằm phát hiện các lỗi của phần mềm. Trong đó, kiểm thử tự động nhằm xử lý một cách tự động quy trình kiểm thử, sinh ca kiểm thử, thực thi kiểm thử và đánh giá kết quả kiểm thử.

Hệ thống phản ứng (reactive system) khi hoạt động đáp ứng (phản ứng) với các sự kiện bên ngoài [3]. Chẳng hạn, các hệ thống sinh học là những hệ thống phản ứng, bởi vì nó phản ứng với các sự kiện nhất định. Tuy nhiên, trong tin học và điều khiển, thuật ngữ này được sử dụng chủ yếu để mô tả các hệ thống nhân tạo. Hệ thống phản ứng liên tục tương tác với môi trường và thực thi theo yếu tố tác động bởi môi trường. Ở một khía cạnh trừu tượng thì nó được xem là một hộp đen, ta cung cấp các giá trị đầu vào thì hệ thống sẽ cho ra các giá trị đầu ra thích hợp. Các hệ thống phản ứng có thể được xem là đang ở một trạng thái nhất định và đợi thông tin đầu vào (input). Với mỗi đầu vào, chúng thực hiện tính toán và sinh đầu ra (output) và chuyển sang một trạng thái mới. Thông thường, các hệ thống nhúng và các hệ thống điều khiển cũng là các hệ thống phản ứng [2].

Lập trình đồng bộ (synchronous programming) [3] đã được giới thiệu vào cuối những năm 80 như là một phương pháp tiếp cận để thiết kế các hệ thống phản ứng. Kể từ đó, nó đã được sử dụng rộng rãi, chủ yếu trong lĩnh vực yêu cầu độ an toàn cao như hệ thống điện tử, vận tải, sản xuất năng lượng (các hệ thống phản ứng). Một lợi thế quan trọng của cách tiếp cận đồng bộ này là nhằm cung cấp một khuôn khổ chính thức, hiệu quả và đồng nhất cho các hoạt động phát triển - thiết kế, cài đặt và xác minh. Một số ngôn ngữ lập trình, đặc biệt đối với các ngôn ngữ sử dụng phương pháp tiếp cận đồng bộ như Esterel, Signal hoặc Lustre [4]. SCADE (Safety Critical Application Development Environment) là những công cụ thương mại dựa trên các mô hình đồng bộ và ngôn ngữ lập trình Lustre. SCADE đã được sử dụng trong các dự án quan trọng tại châu Âu (Airbus A340-600, A380, Eurocopter) và trở thành một tiêu chuẩn trong lĩnh vực này.

Lustre [4] là một ngôn ngữ đồng bộ luồng dữ liệu, được thiết kế năm 1984 bởi Viện IMAG tại Grenoble. Chương trình Lustre gồm một chuỗi có thứ tự các phương trình giúp xác định phương thức dòng đầu vào được chuyển thành các dòng đầu ra thông qua một tập hợp các toán tử. Do đó, cách biểu diễn phù hợp nhất cho các chương trình Lustre là một đồ thị có hướng, gọi là mạng lưới toán tử (trong thực tế, người sử dụng không viết chương trình Lustre mà sử dụng trình soạn thảo đồ họa trong công cụ SCADE để xây dựng các mạng lưới toán tử liên quan). Việc kết hợp của cả hai mô hình đồng bộ và dòng dữ liệu, cú pháp đồ họa đơn giản, áp dụng khái niệm thời gian rời rạc là một số trong những đặc điểm chính làm cho Lustre trở thành ngôn ngữ lý tưởng cho việc xây dựng các mô hình, các thiết kế hệ thống điều khiển trong một số lĩnh vực công nghiệp, chẳng hạn như hệ thống điện tử, ô tô và năng lượng, hạt nhân nói riêng và hệ thống phản ứng nói chung. Với các hệ thống này, yếu tố an toàn (safety) được quan tâm hàng đầu. Vì vậy việc hệ thống bị lỗi khi đang vận hành sẽ gây hậu quả rất nghiêm trọng. Hơn nữa, "lỗi hệ thống" có xu hướng được phát hiện muộn trong quá trình phát triển khi mà nó đã gây ra thiệt hại đáng kể, rất khó để gỡ lỗi và tốn nhiều chi phí. Trong tiến trình phát triển phần mềm cho các hệ thống phản ứng, giai đoạn kiểm thử đóng vai trò rất quan trọng. Phần

mềm càng lớn và càng phức tạp, thủ tục kiểm thử đòi hỏi tốn nhiều thời gian và công sức. Việc tìm lỗi càng sớm càng tốt, ngăn ngừa các khiếm khuyết trước khi thực hiện kiểm thử ở mức chi tiết hơn, phức tạp hơn và chi phí cao hơn [1].

Bắt đầu từ đầu những năm 90, lĩnh vực nghiên cứu về kiểm thử cho các hệ thống phản ứng được hình thành. Từ đó đến nay, nhiều công trình nghiên cứu đã và đang được thực hiện.

Nhóm tác giả Laya Madani ở Đại học Grenoble thực hiện kiểm thử tự động cho các ứng dụng phản ứng và áp dụng trên ngôn ngữ Lustre [10]. Nhóm nghiên cứu này đã xây dựng công cụ Lutes[11] để kiểm thử các ứng dụng phản ứng trên Lustre, tuy nhiên nghiên cứu này sử dụng phương thức xác định ngẫu nhiên các ca kiểm thử dựa trên các tiêu chí bao phủ. Năm 1999, nhóm tác giả Nicolas Halbwachs và Pascal Raymond đến từ Đại học Grenoble đã sử dụng công cụ LESAR để kiểm chứng mô hình các hệ thống phản ứng [9] [12]. Năm 2004, nhóm tác giả Pieter Koopman và Rinus Plasmeijer đến từ Đại học Nijmegen mở rộng công cụ kiểm thử tự động GAST để có thể kiểm thử các hệ thống phản ứng. Tiếp đến, nhóm tác giả Elizabeta Fourneret [13] ở đại học Franche-Comté đã thực hiện sinh ca kiểm thử và xây dựng tập ca kiểm thử cần thực hiện ở phiên bản mới, trong nghiên cứu này tác giả đề xuất kỹ thuật sinh ca kiểm thử cho các ứng dụng dựa trên mô hình UML.

Nhìn chung các nghiên cứu đến thời điểm hiện tại tập trung vào nghiên cứu các kỹ thuật kiểm chứng mô hình, kiểm thử, kiểm thử tự động các hệ thống phản ứng và chương trình Lustre/SCADE. Tuy nhiên, theo chúng tôi nhận thấy chưa có nhiều nghiên cứu thực hiện hướng tiếp cận sử dụng các điều kiện kích hoạt trên mạng lưới các toán tử trong việc kiểm thử các chương trình Lustre/SCADE.

Trong bài báo này, chúng tôi đề xuất kỹ thuật sinh dữ liệu thử tự động cho các chương trình Lustre/SCADE. Trong đó, sử dụng các điều kiện kích hoạt trên mạng lưới toán tử của chương trình Lustre, đồng thời sử dụng kiểm chứng mô hình để sinh tự động các dữ liệu thử dựa trên các điều kiện kích hoạt này. Chúng tôi sẽ sử dụng công cụ kiểm chứng mô hình LESAR để tiến hành kiểm chứng mô hình cho một chương trình Lustre cụ thể, từ đó sinh ra các dữ liệu thử dựa trên các kết quả sinh ra từ quá trình kiểm chứng này.

Nội dung của bài báo được tổ chức như sau: Mục I giới thiệu chung về bài báo và trình bày tổng quan về hệ thống phản ứng, ngôn ngữ lập trình Lustre và các nghiên cứu về kiểm thử cho các chương trình Lustre/SACDE trên thế giới. Mục II trình bày các cơ sở lý thuyết nền tảng sử dụng trong nghiên cứu này. Trong mục III, chúng tôi đề xuất giải pháp sử dụng điều kiện kích hoạt trên mạng lưới toán tử của một chương trình Lustre, kết hợp với việc sử dụng công cụ kiểm chứng mô hình LESAR để tạo dữ liệu thử cho các chương trình Lustre/SACDE. Phần IV trình bày việc ứng dụng giải pháp này cho một hệ thống cụ thể và các kết quả của việc thử nghiệm. Cuối cùng là phần kết luận và đề xuất hướng phát triển tiếp theo.

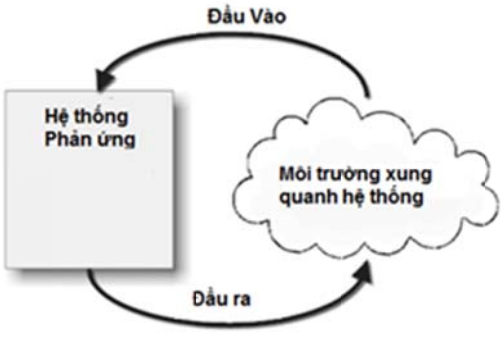
II. CƠ SỞ LÝ THUYẾT

Trong mục này chúng tôi trình bày cơ sở lý thuyết của nghiên cứu, bao gồm các nội dung về ngôn ngữ lập trình Lustre và môi trường SCADE. Đồng thời chúng tôi cũng trình bày kỹ thuật xây dựng mạng lưới toán tử, các lộ trình và các điều kiện kích hoạt cho một chương trình Lustre/SCADE. Và cuối cùng sẽ là những nội dung về ứng dụng kiểm chứng mô hình trong sinh dữ liệu thử.

A. Tổng quan về Lustre/SCADE

1. Ngôn ngữ lập trình Lustre trong các hệ thống phản ứng

Hệ thống phản ứng là một hệ thống thay đổi hành động của nó với đầu ra, điều kiện và trạng thái nhằm đáp ứng với các tác động từ bên ngoài nó. Hệ thống này có thể tự định hướng hoặc được điều khiển định hướng để phản ứng lại với các tác động bên ngoài. Hệ thống phản ứng liên tục phản ứng lại với các tác động từ môi trường, khi môi trường này không thể đồng bộ hóa một cách hợp lý với hệ thống. Nói cách khác, môi trường không thể chờ đợi và hệ thống phải tôn trọng nghiêm ngặt ràng buộc về thời gian thực cần đáp ứng kịp thời [12]. Việc thực thi của hệ thống phản ứng được xem là một chuỗi vô hạn các véc tơ đầu vào và đầu ra. Ở mỗi bước thì giá trị đầu ra được xác định bởi giá trị đầu vào trước đó và hiện tại.



Hình 1. Mô hình hệ thống phản ứng

Hình 1 minh họa mô hình của một hệ thống phản ứng. Với mỗi đầu vào, hệ thống thực hiện tính toán và sinh đầu ra và chuyển sang một trạng thái mới. Thông thường, các hệ thống nhúng và các hệ thống điều khiển cũng là các hệ thống phản ứng. Các hệ thống phản ứng thường được áp dụng chủ yếu để xây dựng các hệ thống điều khiển tự động trong các lĩnh vực như hàng không, năng lượng, hạt nhân, hệ thống liên quan đến giao diện người – máy…

Lustre [4] là một ngôn ngữ hướng dữ liệu đồng bộ, được xây dựng dành riêng cho việc lập trình nên các chương trình điều khiển trong các hệ thống phản ứng như: các hệ thống điều khiển tự động, các hệ thống giám sát trong các lĩnh vực năng lượng, hạt nhân... Ngôn ngữ Lustre rất thích hợp trong việc lập trình các thành phần quan trọng của các hệ thống thời gian thực. Khác với ngôn ngữ mệnh lệnh (imperative languages), mô tả dòng điều khiển của một chương trình, ngôn ngữ Lustre mô tả cách mà các đầu vào được chuyển thành kết quả đầu ra.

Một chương trình Lustre gồm các nốt (node). Một node là một tập hợp các phương trình xác định kết quả đầu ra và cũng chính là hàm đầu vào của nó. Mỗi biến có thể được định nghĩa chỉ một lần trong một node và thứ tự của chúng trong chương trình là không quan trọng [4]. Một khi một node được định nghĩa, nó có thể được sử dụng bên trong các node khác. Một node xác định luồng đầu ra như các chức năng của luồng đầu vào, thông qua một tập hợp có thứ tự các phương trình, trong node có thể chứa các biến cục bộ. Hình 2 là ví dụ một chương trình Lustre để thực hiện phép toán *never*, đoạn chương trình này khai báo một node *never* với đầu vào là *E* và đầu ra là *S*.

```
node never (E: bool) returns (S : bool);
let
   S = not(E) -> (not(E) and pre(S));
tel;
```

Đồng hồ		C_1	C_2	C_3	$C4$	...
Đầu vào	***A***	*false*	*false*	*true*	*false*	...
Đầu ra	***never(A)***	*true*	*true*	*false*	*false*	...

Hình 2. Ví dụ một đoạn chương trình Lustre

Lustre được dựa trên khái niệm về các luồng (flows) và đồng hồ (clocks). Luồng là một chuỗi các giá trị tương ứng với một chuỗi các tuần tự các đồng hồ. Ngôn ngữ Lustre gồm các toán tử cơ bản sau:

- Toán tử logic: ***and, or, not, xor,*** =>

- Toán tử số học: **+, -, *, /, div, mod**

- Toán tử so sánh: <>, <, <=, >, >=

Các kiểu dữ liệu cơ bản của ngôn ngữ Lustre:

- Kiểu luận lý (boolean): Gồm 2 giá trị ***true***, ***false***. Kiểu này được sử dụng với các toán tử logic***: and, or, not, xor,*** =>, kiểu này thường được dùng trong lập trình các cấu trúc điều khiển.

- Kiểu số nguyên, số thực (int, real): Các kiểu này được sử dụng cùng với các phép toán*:* **+, -, *, /, div, mod,** và các phép so sánh: <>, <, <=, >, >=.

Bên cạnh các toán tử trên, ngôn ngữ Lustre có các toán tử thời gian (temporal operator): Toán tử ưu tiên, toán tử khởi tạo và toán tử điều kiện. Các toán tử thời gian này có hoạt động đặc biệt trên luồng dữ liệu. Các toán tử có thể được sử dụng để tạo ra các toán tử mới và phức tạp hơn.

- Toán tử ưu tiên (precedence) được ký hiệu là ***pre()***: Các toán tử ***pre()*** ghi nhớ giá trị của các đối số từ chu kỳ đồng hồ trước đó. Các toán tử sẽ nhận giá trị nil ở chu kỳ đồng hồ đầu tiên. Nếu chuỗi *X(x1, x2, x3,x4,x5,x6 ...)* được sử dụng như là tham số của toán tử *pre(), pre(X)* cho kết quả chuỗi tuần tử sẽ là (*nil, x1, x2, x3, x4,x5,x6 ...*).

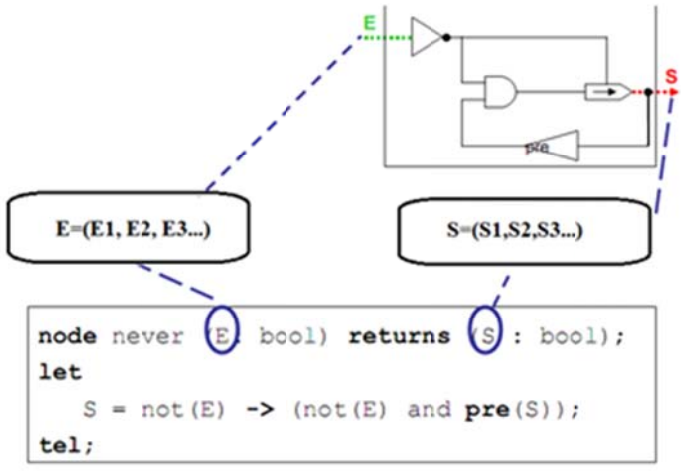
- Toán tử khởi tạo (initialization) được ký hiệu là → hoặc ***fby***: Các toán tử → được sử dụng để cung cấp cho dòng dữ liệu một giá trị khởi tạo ban đầu. Ví dụ, các chuỗi *X(x1, x2, x3,x4,x5,x6 ...)* và *Y(y1, y2, y3,y4,y5,y6 ...)* được sử dụng để tạo ra một chuỗi mới $Z = X \rightarrow Y$, chuỗi kết quả sẽ là *Z(x1, x2, x3,x4,x5,x6 ...)*. Toán tử này có thể được sử dụng cùng với các toán tử ***pre()*** vì giá trị chuỗi đầu tiên trong ***pre()*** là không xác định. Sử dụng các chuỗi trong ví dụ trước các biểu hiện $Z = X \rightarrow pre(Y)$ sẽ có chuỗi giá trị *Z= (x1, y2, y3,y4,y5,y6 ...)*.

- Toán tử điều kiện ***when***: Toán tử ***when*** được sử dụng để thay đổi tần suất đồng hồ. Toán tử ***when*** cũng được gọi là bộ lọc. Ví dụ ta có chuỗi *E* và *F* và *G = E when F*, sẽ tạo ra chuỗi mới *G* sẽ có giá trị tương tự như *E* khi và chỉ khi *F* là đúng. Khi *F* là sai thì chuỗi *G* sẽ không có giá trị gì cả.

Đặc biệt, ngôn ngữ Lustre không cung cấp các toán tử lặp ***(while, do while, for)*** hay thực hiện gọi đệ quy như những ngôn ngữ lập trình khác [4].

2. Môi trường SCADE

SCADE (Safety Critical Application Development Environment) của công ty Esterel Technologies [2] là một môi trường đồ họa dành riêng để phát triển các hệ thống nhúng quan trọng, các hệ thống phản ứng trong công nghiệp công nghệ cao. SCADE dựa trên ngôn ngữ Lustre và nó cho phép định nghĩa phân cấp của các thành phần hệ thống và sinh mã tự động (sinh mã chương trình C, Ada). SCADE cung cấp môi trường đồ họa cho phép thiết kế các mô hình và sinh ra các chương trình bằng ngôn ngữ Lustre. Thực tế, các nhà phát triển phần mềm thường không trực tiếp lập trình bằng ngôn ngữ Lustre mà thường sử dụng SCADE để thiết kế, sau đó chương trình Lustre sẽ được sinh ra tự động.



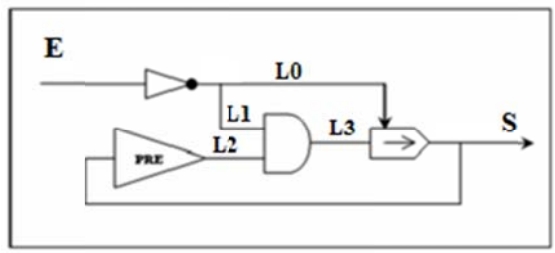
Hình 3. Mô hình SCADE và chương trình Lustre

Hình 3 minh họa mô hình được thiết kế bởi môi trường SCADE và chương trình Lustre được sinh ra tương ứng.

3. Mạng lưới toán tử

Lustre là một ngôn ngữ luồng dữ liệu: Luồng vào của một chương trình được biến đổi thành luồng ra bởi một tập hợp độc lập hoặc không độc lập các toán tử. Do đó một chương trình Lustre thường được biểu diễn bằng mạng lưới các toán tử. Trong môi trường SCADE, một chương trình Lustre được biểu diễn bằng một biểu đồ trực quan, được gọi là mạng lưới toán tử [6].

Một mạng lưới toán tử là một đồ thị có nhãn trực tiếp nhiều đầu vào, gồm một tập hợp N toán tử và một tập hợp $E \subseteq N \times N$ các cạnh (edge) nối các toán tử. Mỗi toán tử biểu diễn bằng một biểu thức logic hoặc một phép toán [6]. Một toán tử được biểu thị bởi một tập hợp có trật tự các cặp <e_i,s> mà trạng thái e_i (i=1,2,…) của các cạnh đầu vào và s trạng thái cho các cạnh đầu ra. Với mỗi cặp <e_i,s> được liên kết bởi một thuộc tính bao gồm các điều kiện của luồng dữ liệu chuyển từ cạnh e_i sang cạnh s.



Hình 4. Mạng lưới toán tử của nốt never trong chương trình Lustre

Mạng lưới toán tử giúp xác định dòng dữ liệu dịch chuyển từ đầu vào đến đầu ra [6]. Cạnh xác định dòng dữ liệu giữa 2 toán tử. Có một quan hệ 1-1 (đơn ánh) giữa mạng lưới toán tử và toán tử Lustre. Một mạng lưới toán tử thường chứa các biểu thức: *AND, OR, NOT, + , -, /, *, LT (<), GT (>), LTE (<=), GTE (>=), EQ (==), NEQ (<>), ITE (if - then - else), PRE (pre), FBY (→).*

Có hai chức năng được liên kết với một toán tử ***op*** bất kỳ (*op=operator*): Khi đó ***in(op)*** trả về tập hợp toán tử nối với cạnh đầu vào và ***out(op)*** trả về tập hợp toán tử nối với cạnh đầu ra. Có 3 loại cạnh: cạnh đầu vào (input edge), cạnh đầu ra (out edge) và cạnh nội bộ (internal edge).

- Cạnh đầu vào tương ứng với các biến đầu vào của một chương trình Lustre;
- Cạnh đầu ra tương ứng với các biến đầu ra của chương trình Lustre;
- Cạnh nội bộ tương ứng với các biến cục bộ của chương trình Lustre.

Mỗi cạnh có một toán tử nguồn duy nhất và một toán tử đích duy nhất. Cạnh $\mathbf{e_2}$ là kế tiếp của cạnh $\mathbf{e_1}$ khi và chỉ khi có một toán tử mà $\mathbf{e_1}$ là đầu vào và $\mathbf{e_2}$ là đầu ra.

Hình 4 biểu diễn mạng lưới toán tử của một nốt ***never*** trong chương trình Lustre. Đây là một đồ thị có một đầu vào và một đầu ra. Nó bao gồm bốn cạnh nội bộ **(*L1*, *L2* và *L3*)** và bốn toán tử (***NOT***, ***AND***, ***PRE*** và ***FBY***).

4. Lộ trình trên mạng lưới toán tử

Trong một chương trình Lustre, mạng lưới toán tử được biểu diễn thành các lộ trình (paths) [6] từ đầu vào đến đầu ra. Lộ trình được tạo ra từ kết quả của các cạnh $p=<e_0,e_1,\ldots,e_n>$, là một dãy hữu hạn các cạnh nối liên tiếp nhau. Giả sử như tất cả các biến $i \in [0, n-1]$ thì cặp $<e_i, e_{i+1}>$ thuộc mạng lưới toán tử [6].

N-path là độ dài của lộ trình (là số lượng các cạnh) bằng với n. Lộ trình xuất phát (initial path) là lộ trình mà cạnh đầu tiên là giá trị vào. Vòng lặp là một phần đồ thị của mạng lưới toán tử được lặp đi lặp lại với các điều kiện khác nhau. Một lộ trình có chứa vòng lặp gọi là lộ trình lặp (*cyclic paths*), lộ trình lặp này có chứa một hoặc nhiều toán tử ***PRE***.

Lộ trình đơn vị (Unit path) là một cặp của hai cạnh có trình tự, có chiều dài n=2 (*2-paths*). Lộ trình hoàn chỉnh (*complete path*) là lộ trình đi từ cạnh khởi tạo cho đến cạnh đầu ra. Lộ trình cơ bản (*elementary path*) là lộ trình không có vòng lặp. Cạnh được chú thích với các nhãn chỉ rõ trình tự dịch chuyển của luồng dữ liệu.

Ngoài ra, lộ trình $p'=<e_0,e_1,\ldots,e_{n-1}>$ được gọi là tiền tố của $p=<e_0,e_1,\ldots,e_{n-1},e_n>$ với n>2.

Bảng 1 liệt kê các lộ trình trên mạng lưới toán tử của chương trình ***never*** ở Hình 4.

Bảng 1. Các lộ trình của chương trình **never**

#	**Lộ trình**	**Kiểu**	**Chiều dài**
1	(E ,L0 ,S)	*acyclic*	*3*
2	(E ,L1 ,L3 ,S)	*acyclic*	*4*
3	(E ,L0 ,S ,L2 ,L3 ,S)	*cyclic*	*6*
4	(E ,L1 ,L3 ,S ,L2 ,L3 ,S)	*cyclic*	*7*

5. Toán tử vị từ

Với *(e, s)* là một lộ trình đơn vị và op là một toán tử ta sẽ có $e \in in(op)$ và $s \in out(op)$.

Toán tử vị từ (operator predicate) [6] được cấu thành từ 2 thành phần là 2 toán tử thể hiện mối quan hệ giữa chúng được biểu diễn dưới dạng *OC(<toán tử 1>, <toán tử 2>)*. Như vậy để biểu diễn toán tử e là đầu vào và s là đầu ra, ta có thể biểu diễn theo dạng *OC(e,s)*.

Toán tử vị từ *OC(e,s)* với 2 thành phần là 2 toán tử *(e,s)* có giá trị thuộc kiểu *boolean,* ta có:

- *OC(e, s) = true* nếu *op* là một toán tử NOT hoặc là một toán tử quan hệ.
- *OC(e, s) = not(e) or e'* nếu *op* là một toán tử AND và *in(op) = {e, e'}*.
- *OC(e, s) = e or not(e')* nếu *op* là một toán tử OR và *in(op) = {e, e'}*.
- *OC(c, s) = true, OC(e, s) = c and OC(e', s) = not(c)* nếu *op* là một toán tử ITE, như vậy *in(op) = {c, e, e'}*.

6. Điều kiện kích hoạt

Điều kiện kích hoạt (Activation Condition, sau đây gọi là AC) [6] là điều kiện để luồng dữ liệu được chuyển từ cạnh vào sang cạnh ra của một toán tử. Mỗi một điều kiện kích hoạt được kết hợp với một lộ trình. Khi các điều kiện kích hoạt của một lộ trình là đúng, thì bất kỳ sự thay đổi các giá trị trên lộ trình sẽ tạo ra những thay đổi trong kết quả cuối cùng. Một lộ trình được kích hoạt nếu điều kiện kích hoạt của nó có ít nhất một lần có giá trị là đúng trong quá trình thực thi.

Cho *N* là mạng lưới toán tử và $p=<e_0,\ldots\ldots,e_n>$ là lộ trình n-path, $p' = (e_1, e_2, \ldots e_{n-1})$ là lộ trình trước của *p* và *op* là một toán tử trên lộ trình *p* (ví dụ: $e_{n-1} \in in(op)$ và $e_n \in out(op)$).Điều kiện kích hoạt của lộ trình $p= <e_0\ldots\ldots\ldots,e_n>$ là một biểu thức *Boolean*, *AC(p)*, được xác định theo quy tắc sau đây:

- Nếu $n = 1$ thì $AC(p) = true$: có nghĩa điều kiện kích hoạt của một cạnh đơn luôn có giá trị *true*.

- Ngược lại nếu $n > 1$, điều kiện kích hoạt của lộ trình p_n là một hàm đệ quy của các toán tử trên lộ trình đó. Tùy theo loại toán tử, điều kiện kích hoạt AC(p) được xác định như sau:

- $AC(p) = AC(p')$ *and* $OC(e_{n-1}, e_n)$ khi *op* là một toán tử ***boolean***, toán tử quan hệ hoặc toán tử điều kiện, và $OC(e_{n-1}, e_n)$ là một vị từ của toán tử *op*.

- $AC(p) = false \rightarrow pre(AC(p'))$ khi *op* là toán tử ***pre***.

- Nếu *op* là toán tử *fby(init; nonInit)*, và nếu toán tử p_{init} và toán tử $p_{nonInit}$ tương ứng với 2 trường hợp khởi tạo *init* và không khởi tạo *nonInit* thì các điều kiện kích hoạt sẽ được định nghĩa bằng hai phương trình sau:

$AC(p) = AC(p_0) \rightarrow false$ (*)

$AC(p) = false \rightarrow AC(p_0)$ (**)

Phương trình đầu tiên *(*)* cho rằng lộ trình *p* sẽ được kích hoạt nếu lộ trình trình trước của nó được kích hoạt tại chu kỳ khởi tạo. Còn ở phương trình thứ hai *(**)* cho rằng lộ trình luôn được kích hoạt nhưng cho chu kỳ khởi tạo.

Ví dụ: Với lộ trình *(E ,L1 ,L3 ,S)* của chương trình never ở Hình 1, chúng ta có thể xác định được điều kiện kích hoạt tương ứng:

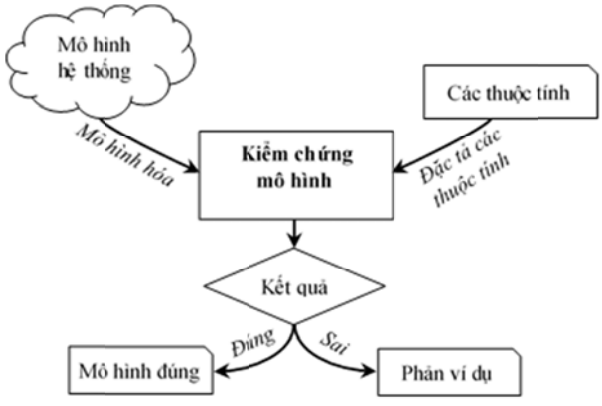
AC = (true and (not(L1) or L2) and false->true).

B. Kiểm chứng mô hình

Trong phát triển phần mềm, kỹ thuật kiểm chứng mô hình được sử dụng để chứng minh một cách tự động tính đúng đắn của phần mềm hoặc chỉ ra tại sao phần mềm không thực thi đúng thông qua phản ví dụ (counterexample). Các phản ví dụ sẽ được sử dụng để xây dựng nên các dữ liệu thử của hệ thống. Trong phần này chúng tôi trình bày kỹ thuật kiểm chứng mô hình và ứng dụng kỹ thuật này trong việc sinh dữ liệu thử.

1. Tổng quan về kiểm chứng mô hình

Kiểm chứng mô hình là kỹ thuật phân tích hệ thống để xác định tính hợp lệ của một hay nhiều tính chất mà người dùng quan tâm trong một mô hình cho trước [7]. Cụ thể hơn, với mô hình M và thuộc tính p cho trước, nó kiểm tra liệu mô hình M có thỏa mãn thuộc tính *p* hay không: M |= p [4]. Về mặt thực thi, kiểm chứng mô hình là kỹ thuật tĩnh, nó duyệt qua tất cả các trạng thái, các lộ trình thực thi có thể có trong mô hình M để xác định tính phủ định của *p*.



Hình 5. Sơ đồ tổng quát của kiểm chứng mô hình

Kiểm chứng mô hình bao gồm 3 bước: mô hình hóa, đặc tả và kiểm chứng (Hình 5).

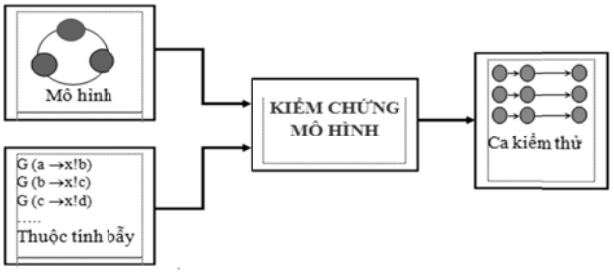
- Đầu tiên là bước mô hình hóa, đầu vào của bước này có thể là thiết kế hoặc là mã nguồn.

- Bước tiếp theo, định nghĩa các thuộc tính mà mô hình cần thỏa mãn được đặc tả. Các thuộc tính này thường được diễn đạt bằng các biểu thức logic. Kết quả của hai bước mô hình hóa và đặc tả là đầu vào của kiểm chứng mô hình.

- Trong bước cuối cùng, công cụ kiểm chứng sẽ tự động thực hiện và trả về kết quả là đúng nếu mô hình thỏa mãn các thuộc tính, hoặc đưa ra một phản ví dụ nếu mô hình không thỏa mãn.

2. Kiểm chứng mô hình trong kiểm thử phần mềm

Kiểm chứng mô hình là một giải pháp để kiểm tra tính đúng đắn của mô hình, tuy nhiên, kỹ thuật này cũng thường được áp dụng trong trong kiểm thử phần mềm. Ý tưởng cơ bản của việc này là: Trên cơ sở mô hình đã có, chúng ta định nghĩa các thuộc tính làm cho mô hình không thỏa mãn, gọi là các thuộc tính bẫy. Việc này nhằm mục đích tạo ra các phản ví dụ sau khi thực thi quá trình kiểm chứng mô hình. Từ các phản ví dụ này, chúng ta sẽ tạo được các dữ liệu thử mong muốn.



Hình 6. Quy trình tạo dữ liệu thử bằng kiểm chứng mô hình

Hình 6 minh họa việc sinh dữ liệu thử dựa vào kiểm chứng mô hình. Với một mô hình đầu vào, ta định nghĩa được các thuộc tính bẫy dựa trên các điều kiện kích hoạt. Thông qua quá trình kiểm chứng mô hình, chúng ta sẽ xác định được các ca kiểm thử tương ứng [8].

3. Công cụ kiểm chứng mô hình LESAR

LESAR [9] là công cụ kiểm chứng mô hình cho các chương trình Lustre/SCADE, được phát triển bởi trung tâm nghiên cứu Verimag. Đây là thành phần được tích hợp vào bộ công cụ phát triển ngôn ngữ Lustre, được phát triển với mục đích kiểm chứng các chương trình Lustre.

Đầu vào của công cụ LESAR là chương trình Lustre với đặc tả là mã nguồn viết bằng ngôn ngữ Lustre và các dòng lệnh mô tả thuộc tính cũng chính bằng ngôn ngữ Lustre.

Giả sử, để thực hiện kiểm chứng mô hình của chương trình Lustre *never* với các thuộc tính đảm bảo kích hoạt các lộ trình được trình bày ở bảng 1.

Ví dụ: Với lộ trình 2 *(E ,L1 ,L3 ,S)*, chúng ta có thể xác định được điều kiện kích hoạt tương ứng:

AC = (true and (not(L1) or L2) and false->true);

Mã nguồn đầy đủ chứa mô hình và thuộc tính làm đầu vào để thực hiện kiểm chứng mô hình cho chương *never* sẽ như sau:

```
node never(E: bool) returns (AC, S: bool)
let
        S = not(E) -> (not (E) and pre(S));
        AC = ( true  and  (not(L1) or L2) and false->true);
tel;
```

Cú pháp thực thi LESAR.

lesar <đường dẫn chương trình Lustre> <Node cần kiểm chứng> <định dạng kết quả trả về>

Ví dụ: *lesar neverlesar.lus never –diag*

```
root@congduy-linux:/rd/nevernode# lesar neverlesar.lus never -diag
--Pollux Version 2.3a

DIAGNOSIS:

--- TRANSITION 1 ---
true
FALSE PROPERTY
root@congduy-linux:/rd/nevernode#
```

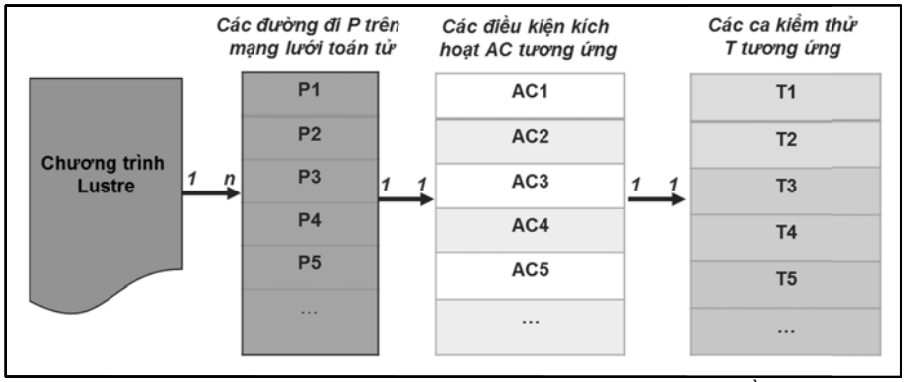
Hình 7. Kiểm chứng mô hình với LESAR

III. ỨNG DỤNG ĐIỀU KIỆN KÍCH HOẠT TRÊN MẠNG LƯỚI TOÁN TỬ TRONG KIỂM THỬ CÁC CHƯƠNG TRÌNH LUSTRE/SCADE

Trong phần này, chúng tôi trình bày giải pháp để kiểm thử các chương trình Lustre/SCADE sử dụng điều kiện kích hoạt trên mạng lưới toán tử tương ứng để mô hình hóa và định nghĩa các thuộc tính bẫy. Sau đó, kỹ thuật kiểm chứng mô hình sẽ được ứng dụng để tạo ra các dữ liệu thử. Trong bài báo này, chúng tôi sử dụng công cụ kiểm chứng mô hình LESAR để tiến hành công đoạn kiểm chứng mô hình.

A. Giải pháp sinh dữ liệu thử cho chương trình Lustre sử dụng điều kiện kích hoạt

Vấn đề chúng tôi tập trung giải quyết ở đây là: với các chương trình Lustre cho trước, ta cần sinh ra các dữ liệu thử một cách tự động dựa trên các điều kiện kích hoạt các lộ trình trên mạng lưới toán tử của chương trình Lustre.



Hình 8. Mô hình giải pháp sinh dữ liệu thử cho chương trình Lustre sử dụng điều kiện kích hoạt

Trong Hình 8, chúng tôi đề xuất mô hình giải pháp tổng thể để sinh dữ liệu thử cho chương trình Lustre, bao gồm các công việc sau:

- Từ một chương trình Lustre cho trước cùng với mạng lưới toán tử được minh họa dưới dạng đồ thị luồng dữ liệu, ta có thể xác định tự động được tất cả các lộ trình từ cạnh đầu vào đến cạnh đầu ra.

- Trên cơ sở các lộ trình được xác định ở bước 1, ta có thể xác định tự động các điều kiện kích hoạt tương ứng.

- Tiến hành kiểm chứng mô hình chương trình Lustre ban đầu với các thuộc tính đầu vào là các thuộc tính bẫy được tạo ra trên cơ sở phủ định các điều kiện kích hoạt được sinh ra từ bước 2.

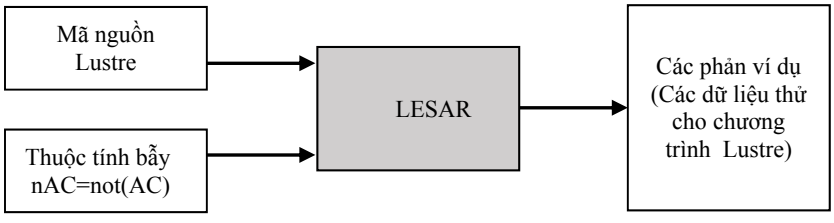
Việc xác định các lộ trình và các điều kiện kích hoạt tương ứng đóng vai trò quan trọng trong việc định nghĩa ra các thuộc tính bẫy cho quá trình kiểm chứng mô hình. Mục tiêu của quá trình kiểm chứng mô hình không phải để kiểm tra mô hình của hệ thống được thiết kế đúng hay không, mà ở đây mục đích chính của chúng ta là sinh ra các phản ví dụ, các phản ví dụ này chính là dữ liệu thử cần được tạo ra.

B. Sinh dữ liệu thử tự động với công cụ kiểm chứng mô hình LESAR

Để thực hiện việc sinh ra dữ liệu thử sau khi có điều kiện kích hoạt và các phản ví dụ tương ứng, chúng tôi sử dụng công cụ kiểm chứng mô hình LESAR cho các chương trình Lustre. Tuy nhiên, mục đích của chúng ta là cần sinh ra các phản ví dụ, do đó các thuộc tính đầu vào phải là các thuộc tính bẫy dựa trên các điều kiện kích hoạt, thuộc tính bẫy *nAC* được xác định theo công thức:

nAC=not (AC)

Hình 9 minh họa quy trình sử dụng công cụ LESAR để sinh ra dữ liệu thử dựa trên mã nguồn chương trình Lustre và các điều kiện kích hoạt.



Hình 9. Sử dụng LESAR tạo các dữ liệu thử

Việc sinh dữ liệu thử sử dụng kỹ thuật kiểm chứng mô hình bằng công cụ LESAR dựa trên các điều kiện kích hoạt được thực hiện trình tự qua các bước chi tiết như sau:

Bước 1 - Xác định các điều kiện kích hoạt: Từ chương trình Lustre chúng ta xác định các điều kiện kích hoạt tương ứng với các lộ trình trên mạng lưới toán tử;

Bước 2 – Định nghĩa các thuộc tính bẫy: Tương ứng với từng điều kiện kích hoạt bước 1, ta xác định được các thuộc tính AC của chương trình cần kiểm chứng. Sau đó định nghĩa các thuộc tính bẫy *nAC* dựa trên các thuộc tính AC:

nAC=not AC;

Bước 3 – Định nghĩa dữ liệu đầu vào cho việc kiểm chứng mô hình: Điểm đặc biệt của công cụ LESAR là mô hình đầu vào và các thuộc tính cần kiểm chứng đều được định nghĩa bằng ngôn ngữ Lustre chứa trong một tệp duy nhất. Do đó, dựa trên mã nguồn Lustre ban đầu và các thuộc tính bẫy nAC ở bước 2, chúng ta tạo ra được tệp đầu vào cho việc thực thi kiểm chứng mô hình.

Bước 4 – Thực thi công cụ LESAR: Quá trình kiểm chứng mô hình cho chương trình Lustre được công cụ LESAR đảm nhận. Tương ứng với mỗi đầu vào, quá trình thực thi công cụ LESAR sẽ tạo ra các phản ví dụ tương ứng.

Bước 5 – Xác định dữ liệu thử: Lần thực thi kiểm chứng mô hình với từng điều kiện kích hoạt các lộ trình trên mạng lưới toán tử, chúng ta sẽ nhận được kết quả là các phản ví dụ. Tương ứng với từng phản ví dụ được sinh sau khi thực thi kiểm chứng mô hình bằng công cụ LESAR ta có được dữ liệu thử. Chính những phản ví dụ này chính là các dữ liệu thử cho chương trình Lustre.

Ví dụ: Đối với chương trình *never* trong Hình 4, dữ liệu thử được sinh ra dựa trên điều kiện kích hoạt lộ trình *(E ,L0 ,S ,L2 ,L3 ,S),* được trình bày trong Bảng 2.

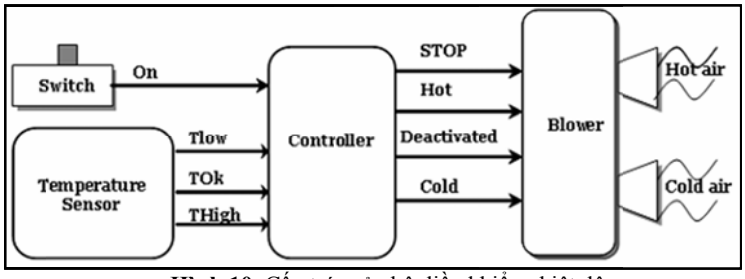
Bảng 2. Sinh dữ liệu thử dựa trên điều kiện kích hoạt

Lộ trình	(E ,L0 ,S ,L2 ,L3 ,S)
Điều kiện kích hoạt	*AC=(false->pre(true and true->false)) and (not(L2) or L1) and false->true)*
Thuộc tính bẫy	nAC=not(AC)
Phản ví dụ được tạo ra	--- TRANSITION 1 --- true
Dữ liệu thử	{1} Tương ứng với vector (E)

IV. ỨNG DỤNG

Trong phần này, chúng tôi ứng dụng kỹ thuật được trình bày ở phần trước để thực hiện việc sinh dữ liệu thử tự động cho hệ thống điều khiển nhiệt độ dựa trên điều kiện kích hoạt các lộ trình trên mạng lưới toán tử.

Trước tiên, chúng ta hãy xem xét một thiết bị điều khiển của máy điều khiển nhiệt độ trong Hình 10, hệ thống điều khiển nhiệt độ này được cấu tạo bởi 3 phần: công tắc, cảm biến nhiệt độ và quạt gió. Thiết bị cảm biến cho ta 3 tín hiệu kiểu *boolean*: *Tlow, Tok , Thigh*, tương ứng 3 mức: nhiệt độ thấp, bình thường, và nhiệt độ cao. Thiết bị điều khiển dùng công tắc và tín hiệu cảm biến để có được 4 chế độ điều khiển tín hiệu *(STOP, Deactivated, Hot và Cold)* đến quạt gió: *STOP* nghĩa là quạt đang ngừng hoạt động, *Deactivated* tức là quạt gió ở chế độ chờ (quạt gió không làm việc), *Hot* và *Cold* tương ứng với không khí nóng hay không khí lạnh được đưa ra.



Hình 10. Cấu trúc của bộ điều khiển nhiệt độ

Hệ thống này hoạt động như sau:

– Nút Switch sử dụng để điều khiển bật và tắt hệ thống. Nếu hệ thống đang ở trạng thái STOP (bị tắt) thì các điều khiển khác không có tác dụng.

– Nếu hệ thống đang ở trạng thái On (mở) và người dùng chọn Thight thì quạt gió sẽ hoạt động mạnh làm cho nhiệt độ của môi trường sẽ chuyển sang trạng thái Cold (lạnh).

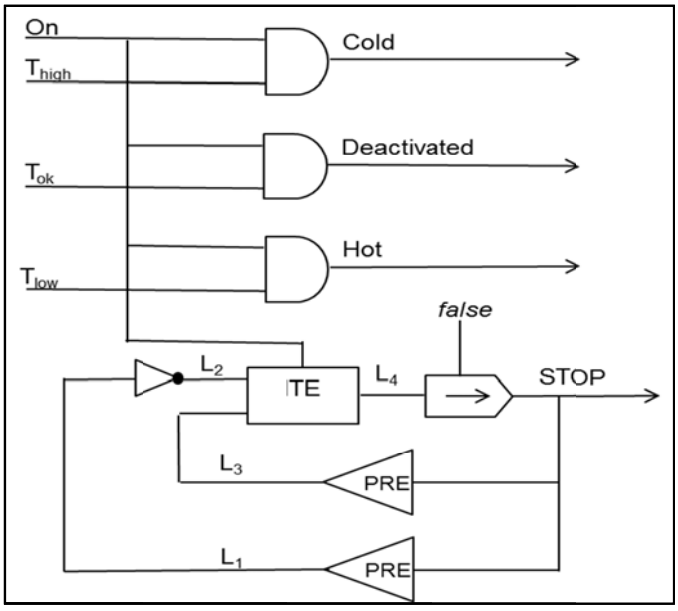
– Tương tự, nếu hệ thống đang ở trạng thái On và người dùng chọn Tlow thì quạt gió sẽ hoạt động chậm lại làm cho nhiệt độ của môi trường sẽ chuyển sang trạng thái Hot (nóng).

– Nếu hệ thống đang ở trạng thái On (mở) và nhưng người dùng không tác động, thì hoạt động của quạt gió sẽ không thay đổi và không ảnh hưởng đến nhiệt độ hiện tại.

Chương trình Lustre của hệ thống điều khiển nhiệt độ trên được trình bày trong Hình 11.

Hình 11. Chương trình LUSTRE hệ thống điều khiển nhiệt độ

Trong Hình 12, chương trình Lustre trên được biểu diễn bằng một mạng lưới toán tử dưới dạng một sơ đồ luồng dữ liệu.



Hình 12. Mạng lưới toán tử tương ứng của hệ thống điều khiển nhiệt độ

Trong mạng lưới toán tử này, có 4 cạnh đầu vào kiểu *boolean* (*On, Tlow, Tok và Thigh*), 4 cạnh đầu ra kiểu ***boolean*** (*STOP, Deactivated, Hot và Cold*) và những cạnh nội bộ khác.

Dựa trên mạng lưới toán tử này, ta xác định được các lộ trình và các điều kiện kích hoạt tương ứng của hệ thống điều khiển nhiệt độ, chi tiết trong Bảng 3.

Bảng 3. Các lộ trình và điều kiện kích hoạt của hệ thống điều khiển nhiệt độ

#	**Lộ trình**	**Điều kiện kích hoạt**
1	(On, Cold)	not(On) or T_{High}
2	(On, Deactivated)	not(On) or T_{Ok}
3	(On, Hot)	not(On) or T_{low}
4	(T_{low},Hot)	not(T_{low}) or On
5	(T_{hight}, Cold)	not(T_{hight}) or On
6	(T_{ok}, Deactivated)	not(T_{ok}) or On
7	(On, _L4, STOP)	true and false → true
8	(On, _L4, STOP, _L3, _L4, STOP)	false → pre(false→true) **and** not(On) **and** false → true
9	(On, _L4, STOP, _L1, _L2, STOP)	false → pre(false→true) **and** On **and** false → true

Tiếp theo, chúng ta áp dụng công cụ kiểm chứng LESAR để sinh ra dữ liệu thử. Việc kiểm chứng mô hình được thực hiện lần lượt cho từng điều kiện kích hoạt, mỗi lần thực hiện chúng ta nhận kết quả trả về là các phản ví dụ, cũng chính là các dữ liệu thử.

Ví dụ với điều kiện kích hoạt *AC1= not(On) or THigh*, ta xây dựng mô hình cần kiểm chứng tương ứng như sau:

```
                                                    bool);

                                                    lse  pre
```

Hình 13. Mã nguồn Lustre của mô hình tương ứng vớ AC1= not(On) or THigh

Thực thi LESAR chúng ta có kết quả như Hình 14.

Hình 14. Kết quả thực thi LESAR

Quá trình kiểm chứng sẽ tạo ra phản ví dụ là một trường hợp lỗi:

TRANSITION 1

not On or THigh

Nghĩa là:

On=0 hoặc *THigh=1*

Khi đó dữ liệu thử tương ứng là *(On, Thight,T ok, Tlow)={0,1,0,0}*

Lần lượt thực hiện kiểm chứng chương trình Lustre của hệ thống điều khiển nhiệt độ với các điều kiện kích hoạt tương ứng, ta có thể sinh ra tất cả các dữ liệu thử cho chương trình như Bảng 4.

Bảng 4. Các dữ liệu thử của hệ thống điều khiển nhiệt độ

#	**Điều kiện kích hoạt**	**Dữ liệu thử** (*Dạng vector (On, T_{hight}, T_{ok}, T_{low})*)
1	not(On) or T_{hight}	{0,1,0,0}
2	not(On) or T_{Ok}	{0,0,1,0}
3	not(On) or T_{low}	{0,0,0,1}
4	not(T_{low}) or On	{1,0,0,0}
5	not(T_{hight}) or On	{1,0,0,0}
6	not(T_{ok}) or On	{1,0,0,0}
7	false → true	Giá trị bất kỳ
8	false → pre(false→true) **and** not(On) **and** false → true	Giá trị bất kỳ
9	false → pre(false→true) **and** On **and** false → true	Giá trị bất kỳ

Hệ thống điều khiển nhiệt độ là một ví dụ điển hình cho các hệ thống phản ứng được xây dựng bằng Lustre/SCADE. Sau khi ứng dụng kỹ thuật kiểm chứng mô hình trên cơ sở các điều kiện kích hoạt trên mạng lưới toán tử, dựa vào các phản ví dụ được sinh ra, chúng ta sẽ có được các dữ liệu thử cần thiết. Với những hệ thống lớn và phức tạp hơn, việc sinh dữ liệu thử tự động sẽ giúp tiết kiệm được rất nhiều chi phí cho quá trình kiểm thử.

V. KẾT LUẬN

Các hệ thống phản ứng ngày càng được sử dụng phổ biến trong ngành công nghiệp quan trọng như hạt nhân, năng lượng, hàng không. Đặc điểm chung quan trọng nhất của những hệ thống này là tính an toàn, do đó trong quá trình phát triển, việc kiểm thử luôn được quan tâm hàng đầu. Chương trình Lustre/SCADE được sử dụng để phát triển các hệ thống phản ứng, nên việc kiểm thử các chương trình Lustre/SCADE càng trở nên quan trọng.

Trong bài báo này, chúng tôi đã đề xuất giải pháp sử dụng điều kiện kích hoạt trên mạng lưới toán tử của các chương trình Lustre, kết hợp với công cụ kiểm chứng mô hình LESAR để sinh dữ liệu thử cho các chương trình Lustre/SCADE. Chúng tôi đã định nghĩa được quy trình xây dựng tập dữ liệu thử cho kiểm thử các chương trình này. Giải pháp này hoàn toàn có thể được tự động hóa nhằm giảm chi phí và tăng hiệu quả của quy trình kiểm thử các chương trình Lustre/SCADE.

Chúng tôi sẽ tiếp tục áp dụng phương pháp này cho việc sinh dữ liệu thử tự động trong kiểm thử hồi quy cho các ứng dụng Lustre/SCADE. Chúng tôi cũng hướng đến khả năng tự động hóa quá trình sinh dữ liệu thử cũng như kiểm thử hồi quy tự động cho các hệ thống phản ứng nói chung và các ứng dụng Lustre/SCADE nói riêng.

VI. TÀI LIỆU THAM KHẢO

[1] Trinh Cong Duy, Nguyen Thanh Binh, Ioannis Parissis, "Automatic Generation of Test Cases in Regression Testing for Lustre/SCADE", Journal of Software Engineering and Applications, Vol. 6, 2013.

[2] Trinh Cong Duy, Nguyen Thanh Binh, Ioannis Parissis, "Automatic generation of test cases in regression testing for Reactive Systems", FAIR 2013, Hue, Vietnam, 2013.

[3] Albert Benveniste, Gerard Berry, "The Synchronous Approach to Reactive and Real-Time Systems", Proceedings of the IEEE, 1991.

[4] P. Caspi, D. Pilaud, N. Halbwachs, and J. Plaice, "LUSTRE: A Declarative Language for Programming Synchronous Systems", ACM Symposium on Principles of Programming Languages, Munich, Germany, 1987.

[5] Esterel Technologies website: http://esterel-technologies.com.

[6] A. Lakehal and I. Parissis, "Lustructu: A Tool for the Automatic Coverage Assessment of Lustre Programs", IEEE International Symposium on Software Reliability Engineering, Chicago, Illinois, USA, 2005.

[7] Gordon Fraser, FranzWotawa, Paul E. Ammann, "Testing with model checkers: A survey", Competence Network Softnet Austria, Austria, 2007.

[8] Karolina Zurowska and Juergen Di, "Model-based generation of test cases for reactive systems", Applied Formal Methods Group School of Computing Queen's University Kingston, Ontario, Canada, 2010.

[9] Nicolas Halbwachs, "Lustre program verification: the tool Lesar", Synchronous Programming of Reactive Systems Volume 215 of the series The Springer International Series in Engineering and Computer Science pp 139-147, 1993.

[10] Laya Madani, Virginia Papailiopoulou, Ioannis Parissis, "Towards a testing methodology for reactive system: a case study of a landing gear controller", Laboratoire d'Informatique de Grenoble, 2006.

[11] L. duBousquet, F. Ouabdesselam, J.- L.Richier, and N. Zuanon, "Lutess: testing environment for synchronous software", Advances in Computing Science, Springer, 1998.

[12] Nicolas Halbwachs, Pascal Raymond, "Validation of Synchronous Reactive Systems: from Formal Verication to Automatic Testing", Grenoble, France, 1999.

[13] Elizabeta Fourneret, Fabrice Bouquet, Frédéric Dadeau and Stéphane Debricon, "Selective Test Generation Method for Evolving Critical Systems", Université de Franche-Comté, LIFC - INRIA CASSIS Project, 2010.

TEST DATA GENERATION FOR LUSTRE/SCADE PROGRAMS USING THE ACTIVATION CONDITIONS

Trinh Cong Duy, Nguyen Thanh Binh, Ioannis Parissis

ABSTRACT*-Testing is important activities in the software development process. Software development increasingly complex, requiring increasingly higher quality, it is even more important in developing software for reactive systems. Reactive systems are popularly developed in many indutrial fields such as aviation, energy, nuclear … These systems usually require very high quality and rigorous testing activities before they are deployed for use. Lustre/SCADE is a formal synchronous declarative language widely used for modeling and specifying safety critical applications in the fields of avionics, transportation, and energy production. In such applications, the testing activity to ensure correctness of the system plays a crucial role. So, testing work need to be conducted regularly and continuously when there are any changes in the application. Besides, for those applications in this field, the test manually would be very difficult to implement and not effective, so the need to automate testing activities for Lustre/SCADE applications. In this paper, we focus on the test automation for applications Lustre / SCADE. We propose to use activation condition of operator network and usenmodel checking techniques to automatically generated test data. Finally, we apply this approach on one program Lustre for Heater Controller System.*