XÂY DỰNG CÔNG CỤ SINH DỮ LIỆU THỬ CHO CHƯƠNG TRÌNH LUSTRE/SCADE DỰA TRÊN KIỂM CHỨNG MÔ HÌNH

CREATING TEST DATA GENERATION TOOL FOR LUSTRE/SCADE PROGRAMS USING MODEL CHECKING

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

[1]Đại học Đà Nẵng; tcduy@dut.udn.vn

[2]Trường Đại học Bách khoa, Đại học Đà Nẵng; ntbinh@dut.udn.vn

Tóm tắt - 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. 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, đề xuất sử dụng kỹ thuật kiểm chứng mô hình trên trên mạng lưới toán tử (operator network) để sinh ra các ca kiểm thử một cách tự động. Cuối cùng, chúng tôi ứng dụng giải pháp này để xây dựng công cụ sinh ca kiểm thử cho chương trình Lustre/SCADE và minh họa trên một chương trình cụ thể.

Từ khóa - hệ thống phản ứng; kiểm chứng mô hình; kiểm thử; thuộc tính bẫy; Lustre/SCADE; sinh ca kiểm thử.

Abstract - Lustre/SCADE is the language widely used for developing applications of reactive systems which have very strict requirements on software reliability and even the smallest of errors was unacceptable. Moreover, in such applications, the manual testing would be very difficult to implement and ineffective, so using automatic testing tool for Lustre /SCADE becomes necessary. In this paper, we concentrate on studying automated testing tool for Lustre/SCADE programs. We propose the use of model checking techniques on operator network for automatically generating test cases. Finally, we apply this technique to develop test data generation tool for Lustre/SCADE programs and illustrate it on a specific program.

Key words - reactive system; model checker; testing; trap propertive; Lustre/SCADE; generate test cases

1. Giới thiệu

Lustre [9] 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. 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 [7].

Hiện nay trên thế giới, chưa có nhiều nghiên cứu cũng như chưa có nhiều công cụ để sinh dữ liệu thử tự động cho các phần mềm trong các hệ thống phản ứng, mà nhất là các ứng dụng Lustre/SCADE. Nhóm tác giả Abdesselam Lakehal, Ioannis Parissis đề xuất công cụ Lustes để kiểm thử các ứng dụng Lustre/SCADE, nhưng sử dụng phương pháp sinh ngẫu nhiên dựa trên các thuộc tính [2]. Trong bài báo này, chúng tôi đề xuất kỹ thuật sinh d tự động dữ liệu kiểm thử cho các chương trình Lustre/SCADE. Trong đó, chúng tôi sử dụng kỹ thuật kiểm chứng mô hình trên mạng lưới toán tử để sinh dữ liệu thử. Chúng tôi sẽ sử dụng công cụ kiểm chứng mô hình LESAR để tiến hành kiểm chứng mô hình cho một chương trinh Lustre cụ thể, từ đó sinh ra các ca kiểm 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 1 giới thiệu chung về bài báo và trình bày tổng quan về hệ thống phản ứng, ngôn ngữ lập trình Lustre. Mục 2 trình bày các cơ sở lý thuyết nền tảng sử dụng trong nghiên cứu này. Trong mục 3, chúng tôi đề xuất giải pháp sử dụng sử dụng công cụ kiểm chứng mô hình LESAR dựa trên mạng lưới toán tử để tạo ca kiểm thử cho các chương trình Lustre/SACDE. Phần 4 trình bày về ứng dụng sinh ca kiểm thử tự động và các kết quả của việc thử nghiệm. Cuối cùng là phần kết luận và đề xuất hướng phát triển tiếp theo.

2. Cơ sở lý thuyết

Mục này, trình bày cơ sở lý thuyết của nghiên cứu, bao gồm các hệ thống phản ứng, 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. Cuối cùng sẽ là những nội dung về ứng dụng kiểm chứng mô hình trong sinh ca kiểm thử.

2.1. 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 [3].

Hệ thống phản ứng khác với hệ thống tương tác (interactive system) [8]. Hệ thống tương tác thường xuyên giao tiếp với môi trường nhưng với một mức độ riêng như: hệ điều hành, giao diện cơ sở dữ liệu, web server. Trong khi đó hệ thống phản ứng là phản ứng lại với môi trường, ví dụ như quá trình điều khiển trong công nghiệp, nhà máy điện, hệ thống nhúng trong xe lửa, máy bay ...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.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.

2.2. Giới thiệu về Lustre/SCADE

2.2.1. Ngôn ngữ Lustre

Lustre [9] 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 Lustremô tả cách mà các đầu vào được chuyển thành kết quả đầu ra.

Một chương trình Lustre gồm các 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 [9]. Hình 1 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 1. *Ví dụ một chương trình Lustre*

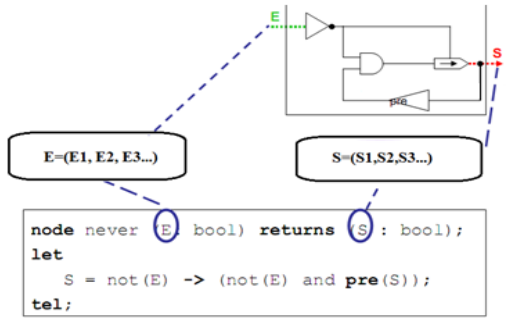
Lustre được dựa trên khái niệm về các luồng (flows) và đồng hồ (clocks). Luồng là một chuỗi các giá trị tương ứng với một chuỗi các tuần tự các đồng hồ. Ngôn ngữ Lustre gồm các toán tử cơ bản: Toán tử logic: ***and, or, not, xor,*** *=>;* Toán tử số học: ***+, -, *, /, div, mod;*** Toán tử so sánh: ***<>, <, <=, >, >=.***

Các kiểu dữ liệu cơ bản của ngôn ngữ Lustre bao gồm: Kiểu luận lý (boolean) gồm 2 giá trị ***true***, ***false***. Kiểu số nguyên, số thực (int, real). Bên cạnh các toán tử trên, ngôn ngữ Lustre có các toán tử thời gian (temporal operator): Toán tử ưu tiên ***pre()***, toán tử khởi tạo ***fby*** (hay →) và toán tử điều kiện ***when***. 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.

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 [9].

2.2.2. Môi trường SCADE

SCADE (Safety Critical Application Development Environment) của công ty Esterel Technologies [4] 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. Hình 2 minh họa mô hình được thiết kế bởi môi trường SCADE và chương trình Lustre được sinh ra tương ứng.



Hình 2. *Mô hình SCADE và chương trình Lustre*

2.2.3. Mạng lưới các 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 [1].

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 [1]. 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.

Mạng lưới toán tử giúp xác định dòng dữ liệu dịch chuyển từ đầu vào đến đầu ra [1]. 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*** (*operator*) bất kỳ: **in(op)** và **out(op)**. Trong 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.

2.2.4. Lộ trình trên mạng lưới toán tử

Trong một chương trình Lustre, mạng lưới toán tử được biểu diễn thành các lộ trình (paths) [1] từ đầu vào đến đầu ra. Lộ trình được tạo ra từ kết quả của các cạnh $\boldsymbol{p=<e_0,e_1,\ldots,e_n>}$, 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 $\boldsymbol{<e_i, e_{i+1}>}$ thuộc mạng lưới toán tử.

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 đường đi có chứa vòng lặp gọi là đường đi lặp (*cyclic paths*), trong đường đi lặp này có chứa một hoặc nhiều toán tử ***PRE***.

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

2.2.5. 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) [1] 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 toán tử NOT hoặc là một toán tử quan hệ.

- *OC(e, s) = not(e) or e'* nếu *op* là toán tử AND và *in(op) = {e, e'}*.

- *OC(e, s) = e or not(e')* nếu *op* là 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 toán tử ITE, như vậy *in(op) = {c, e, e'}*.

2.2.6. Điều kiện kích hoạt

Điều kiện kích hoạt (Activation Condition - AC) [1] 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.

Cho *N* là mạng lưới toán tử và $p=<e_0,\ldots,e_n>$ 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 $p= <e_0\ldots\ldots,e_n>$ 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:

• $AC(p) = AC(p')andOC(e_{n-1}, e_n)$ 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:

$$AC(p) = AC(p') \rightarrow false \quad (*)$$

$$AC(p) = false \rightarrow AC(p') \quad (**)$$

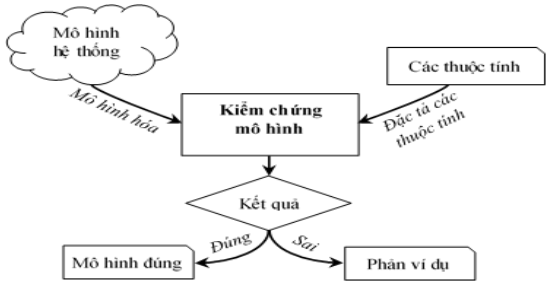
Phương trình đầu tiên (*) cho rằng lộ trình p sẽ được kích hoạt, nếu lộ trình trước của nó được kích hoạt tại chu kỳ khởi tạo. Còn ở phương trình thứ hai (**) cho rằng lộ trình luôn được kích hoạt, nhưng cho chu kỳ khởi tạo.

2.3. 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 ca kiểm 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 ca kiểm thử.

2.3.1. 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 [5]. 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. 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 3. *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 3) [6].

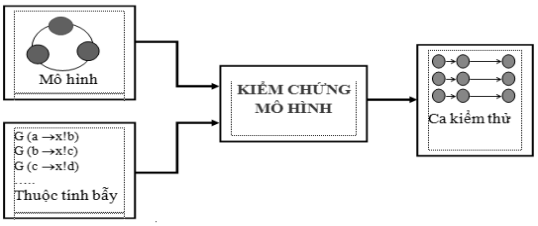
- Đầu tiên là bước mô hình hóa, đầu vào của bước này có thể là thiết kế hoặc là mã nguồn.

- Bước tiếp theo, định nghĩa các thuộc tính 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 cho 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.

2.3.2. 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 để tạo ca kiểm thử.



Hình 4. *Tạo ca kiểm thử với kiểm chứng mô hình*

Việc sinh dữ liệu thử dựa vào kiểm chứng mô hình được thực hiện như sau: 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 ca kiểm thử mong muốn.

Hình 4 minh họa việc sinh ca kiểm thử dựa vào kiểm chứng mô hình. Từ một mô hình đầu vào, chúng 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 [10], [11].

3. Ứng dụng điều kiện kích hoạt trên mạng lưới toán tử trong kiểm thử các chương trình Lustre/SCADE

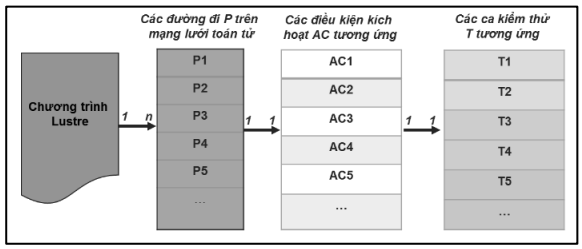
Trong phần này, chúng tôi trình bày giải pháp để kiểm thử các chương trình Lustre/SCADE sử dụng điều kiện kích hoạt trên mạng lưới toán tử tương ứng để mô hình hóa và định nghĩa các thuộc tính bẫy. Sau đó, chúng tôi sẽ dùng kỹ thuật kiểm chứng mô hình để tạo ra các ca kiểm thử. Trong bài báo này, chúng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

3.1. Giải pháp sinh ca kiểm thử cho chương trình Lustre sử dụng điều kiện kích hoạt

Vấn đề cần giải quyết là từ các chương trình Lustre cho trước, cần sinh ra các ca kiểm thử một cách tự động dựa trên các điều kiện kích hoạt các lộ trình trên mạng lưới toán tử của chương trình Lustre tương ứng.

Trong Hình 5, chúng tôi đề xuất giải pháp tổng thể để sinh ca kiểm thử cho chương trình Lustre, bao gồm 3 bước cơ bản sau:

- Bước 1: Từ một chương trình Lustre cho trước cùng với mạng lưới toán tử được minh họa dưới dạng đồ thị luồng dữ liệu, chúng ta có thể xác định được tất cả các đường đi từ cạnh đầu vào đến cạnh đầu ra.



Hình 5. *Mô hình giải pháp sinh ca kiểm thử cho chương trình Lustre sử dụng điều kiện kích hoạt*

- Bước 2: Trên cơ sở các đường đi được xác định ở bước 1, các điều kiện kích hoạt tương ứng được xác định.

- Bước 3: Tiến hành kiểm chứng mô hình chương trình Lustre ban đầu với các thuộc 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 đường đi và các điều kiện kích hoạt tương ứng ở bước 1 và bước 2 đóng vai trò quan trọng trong việc định nghĩa ra các thuộc tính bẫy cho quá trình kiểm chứng mô hình ở bước 3. 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 là sinh ra các phản ví dụ. Các phản ví dụ này chính là ca kiểm thử cần được tạo ra.

3.2. Sinh dữ liệu thử với công cụ kiểm chứng mô hình LESAR

Để thực hiện việc sinh ra ca kiểm 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 [2] cho các chương trình Lustre.

LESAR 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 [18], được phát triển với mục đích kiểm chứng các chương trình Lustre.

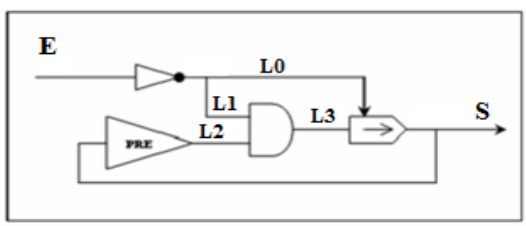
Đầu vào của công cụ LESAR là chương trình Lustre với đặc tả là mã nguồn viết bằng ngôn ngữ Lustre và các thuộc tính là các điều kiện kích hoạt. Tuy nhiên, mục đích của chúng ta là cần sinh ra các phản ví dụ, do đó các thuộc tính đầu vào phải là các thuộc tính bẫy dựa trên các điều kiện kích hoạt, thuộc tính bẫy *nAC* được xác định theo công thức:

$$nAC = not\ (AC)$$

4. Xây dựng công cụ sinh ca kiểm thử tự động

4.1. Ứng dụng minh họa

Trong phần này, chúng tôi ứng dụng kỹ thuật được trình bày ở phần trước để thực hiện việc sinh ca kiểm thử tự động cho chương trình Lustre ***never*** được giới thiệu ở Mục 1. Dựa trên mã nguồn Lustre của chương trình ***never,*** chúng ta có được mạng lưới toán tử trong Hình 6:



Hình 6. *Mạng lưới toán tử của chương trình never*

Hình 6 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***).

Dựa trên mạng lưới toán tử này, ta xác định được các lộ trình và các điều kiện kích hoạt tương ứng tương ứng, chi tiết trong Bảng 1.

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

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

Trên cơ sở các lộ trình trên mạng lưới toán tử, ta xác định được các điều kiện kích hoạt tương ứng.

Với lộ trình 2 *(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).

Tiếp theo, chúng ta áp dụng công cụ kiểm chứng LESAR để sinh ra ca kiểm 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 ca kiểm thử.

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*

Đối với chương trình never trong Hình 7, ca kiểm thử được sinh ra dựa trên điều kiện kích hoạt lộ trình *(E , L0 , S , L2 , L3 , S)*, được minh họa trong Bảng 2.

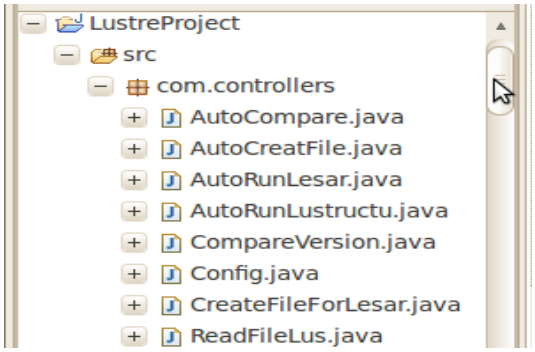
Bảng 2. *Sinh ca kiểm thử dựa trên điều kiện kích hoạt*

Đường đi	(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	--- TRANSITION 1 --- true
Ca kiểm thử	{1}

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 ***never***.

4.2. Công cụ sinh ca kiểm thử tự động

Trên cơ sở giải pháp được đề xuất, chúng tôi tiến hành cài đặt xây dựng công cụ giúp tự động sinh ca kiểm thử cho chương trình Lustre.



Hình 8. *Cài đặt các mô đun của công cụ*

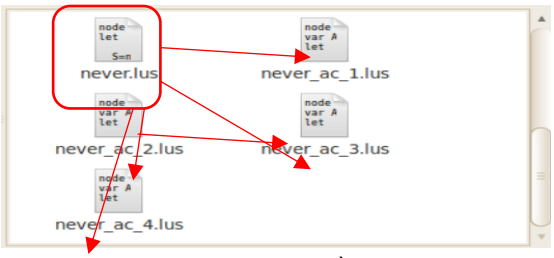
Công cụ bao gồm các mô đun chính sau (Hình 8):

- Mô đun tự động sinh ra các lộ trình và điều kiện kích hoạt lộ trình trên mạng lưới toán tử.

- Mô đun tự động sinh ra các tệp Lustre đầu vào cho quá trình kiểm chứng mô hình tự động với công cụ LESAR (Hình 9).

- Mô đun thực thi kiểm chứng mô hình tự động để sinh ra các ca kiểm thử và lưu vào cơ sở dữ liệu.

Công cụ được phát triển trên ngôn ngữ lập trình JAVA, cơ sở dữ liệu MySQL và vận hành trên hệ điều hành Linux.



Hình 9. *Tự động tạo các tệp đầu vào cho LESAR*

Chúng tôi tiến hành sử dụng công cụ này để ra các phản ví dụ được lưu trữ vào cơ sở dữ liệu MySQL như Bảng 3.

Bảng 3. *Các phản ví dụ được tạo ra*

#	AC	Phản ví dụ
1	(true and true->false)	--- TRANSITION 1 --- true
2	(true and (not(L1) or L2) and false->true)	--- TRANSITION 1 --- true --- TRANSITION 2 --- true
3	((false->pre(true and true->false)) and (not(L2) or L1) and false->true)	--- TRANSITION 1 --- true --- TRANSITION 2 --- true
4	((false->pre(true and (not(L1) or L2) and false->true)) and	--- TRANSITION 1 --- true

	(not(L2) or L1) and false->true)	--- TRANSITION 2 --- true

Dựa trên các phản ví dụ, ta xác định được các ca kiểm thử tương ứng trong Bảng 4.

Bảng 4. *Các phản ví dụ được tạo ra*

Lộ trình	Điều kiện kích hoạt	Ca kiểm thử
1	(true and true->false)	{true}
2	(true and (not(L1) or L2) and false->true)	{(true),(true)}
3	((false->pre(true and true->false)) and (not(L2) or L1) and false->true)	{(true),(true)}
4	((false->pre(true and (not(L1) or L2) and false->true)) and (not(L2) or L1) and false->true)	{(true),(true)}

Chương trình ***never*** là một ví dụ đơn giản của chương trình Lustre. 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 ca kiểm thử cần thiết. Với những hệ thống lớn và phức tạp hơn, việc sinh ca kiểm thử tự động sẽ giúp tiết kiệm được rất nhiều chi phí cho quá trình kiểm thử. Kết quả của quá trình thử nghiệm đã tạo ra các dữ liệu cần thiết cho quá trình kiểm thử các ứng dụng Lustre/SCADE. Công cụ tuy giao diện còn đơn giản, nhưng đã giải quyết được vấn đề sinh dữ liệu thử từ đầu vào là một chương trình viết bằng ngôn ngữ Lustre, đáp ứng được yêu cầu đặt ra của vấn đề cần nghiên cứu.

5. Kết luận

Bài báo này, đã đề xuất giải pháp sử dụng kỹ thuật kiểm chứng mô hình trên mạng lưới toán tử của các chương trình Lustre để sinh ra các ca kiểm thử mong muốn. Định nghĩa được quy trình xây dựng tập ca kiểm thử cho kiểm thử các chương trình này. Đồng thời, bài báo đã xây dựng công cụ tự động hóa việc sinh ca kiểm thử các chương trình Lustre/SCADE trên cơ sở giải pháp đã đề xuất.

Chúng tôi sẽ tiếp tục áp dụng phương pháp này cho việc sinh ca kiểm thử tự động trong kiểm thử hồi quy cho các ứng dụng Lustre/SCADE. Hướng đến khả năng tự động hóa quá trình sinh ca kiểm 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.

TÀI LIỆU THAM KHẢO

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

[2] Abdesselam Lakehal, Ioannis Parissis. Structural Coverage Criteria for Lustre/SCADE Programs. Software Testing, Verification & Reliability, John Wiley and Sons Ltd (2009).

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

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

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

[6] 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, June (2010).

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

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

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

[10] 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).

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

(BBT nhận bài: 01/08/2015, phản biện xong: 04/08/2015)