MỤC LỤC

DANH SÁCH HÌNH ẢNH:

LỜI CẢM ƠN

Trước tiên, em xin gửi lời cảm ơn chân thành đến trường Đại học Dân Lập Hải Phòng đã tạo điều kiện thuận lợi cho em trong suốt quá trình học vừa qua,em xin gửi lời cảm ơn chân thành đến qúy thầy cô trong khoa Công nghệ Thông tin đã nhiệt tình giảng dạy em trong thời gian qua, qua đó em đã có được những kiến thức rất bổ ích đểlàm đề tài này. Đặc biệt em gửi lời cảm ơn chân thành đến thầy Đỗ Văn Chiểu trực tiếp hướng dẫn tạo mọi điều kiện cho em hoàn thành đề tài.

Cuối cùng em xin gửi lời cảm ơn đến gia đình, bạn bè, người thân đã giúp đỡ động viên em rất nhiều trong quá trình học tập và làm đồ án tốt nghiệp.

Do thời gian thực hiện có hạn, kiến thức còn nhiều hạn chế nên đồ án thực hiện chắc chắn không tránh khỏi những thiếu sót nhất định. Em rất mong nhận được ý kiến đóng góp của thầy cô giáo và các bạn để em có thêm kinh nghiệm và tiếp tục hoàn thiện đồ án của mình.

Em xin chân thành cảm ơn!

Hải Phòng, ngàythángnăm 2012

Sinh viên

Vũ Đức Hậu

GIỚI THIỆU

ĐẶT VẤN ĐỀ

Đặc tả và kiểm chứng phần mềm là một trong những phương pháp kiểm tra các hệ thống liệu có thỏa thiết kế hay không? Việc kiểm tra hệ thống được thực hiện trên nhiều pha trong quy trình sản xuất phần mềm,từ thiết kế,đặc tả,viết mã,kiểm thử,kiểm chứng,kiểm tra có thỏa yêu cầu người dùng (*validation*).

Trong lập trình muốn có được một chương trình thì người lập trình không thể thành công trong lần chạy đầu tiên và chưa thể tốt được trong lần biên dịch đầu tiên.Một chương trình ban đầu trông có vẻ hoàn hảo và luôn đúng nhưng khi đưa vào chạy thật có thể chứa những lỗi ở đâu đó,khi đó nó sẽ gây ra những thiệt hại về thời gian và tiền bạc của chùng ta rất nhiều. Trong quá trình thiết kế và sản xuất phần cứng cũng như phần mềm,chúng ta tốn rất nhiều thời gian và công sức trong việc thiết kế,đôi khi nhiều hơn cả việc xây dựng chúng.

Có rất nhiều ứng dụng mà khi có lỗi dù là nhỏ nhất được đưa vào sử dụng có thể dẫn đến thiệt hại về người, tài sản cũng như tốn thật nặng nề đến môi trường.Việc thiết kế phần mềm dành cho các hệ thống này là vô cùng khó khăn.

Việc một lập trình viên phải làm để có một sản phẩm phần mềm luôn là phân tích,lập trình,kiểmtra lại,kiểm thử.Để việc kiểm chứng được nhẹ nhàng và nhanh chóng, tăngsự chính xác hơn thì chúng ta luôn tìm kiếm các công nghệ để giúp cho việc đó và *SPIN* chính là một trong các công cụ đó.

*SPIN*là một công cụ chung để kiểm chứng tính đúng đắn của mô hình phần mềm một cách chặt chẽ và hầu hết là tự động.Ban đầu nó đã được viết bởi *Gerard J.Holzmann* và cộng sự trong nhóm *Unix* của các ngành khoa học máy tính Trung tâm nghiên cứu tại *Bell Labs* vào những năm 80 của thế kỷ trước.*SPIN* miễn phí và tiếp tục phát triển để bắt kịp với sự phát triển mới trong lĩnh vực này.

Các hệ thống cần kiểm chứng đã được đặc tả bằng *Promela (Process Meta Language)*sau đó dùng (*SPIN-Simple Promela Interpreter*) để kiểm chứng.Các tính chất cần kiểm chứng được biểu diễn bằng công thức của *LTL*,lấy phủ định rồi chuyển sang *Büchi Automata*.Ngoài việc kiểm chứng mô hình,*SPIN* cũng có thể hoạt động như một bộ mô phỏng,sau khi thi hành một dãy các thực thi của hệ thống và hiển thị vết thi hành cho người dùng.

Nội dung của khóa luận:

Khóa luận tìm hiểu về bộ kiểm chứng mô hình *SPIN*,các mô hình hệ thống viết bằng ngôn ngữ promela mà *SPIN* có thể hiểu được và các kiểm chứng mô hình bằng *SPIN*.

CẤU TRÚC KHÓA LUẬN

Các phần còn lại của khóa luận có cấu trúc như sau:

Chương 1: Trình bày cơ sở lý thuyết của kiểm thử mô hình, bao gồm các khái niệm cơ bản, các bước thực hiện, lợi ích của kiểm thử mô hình và cách thức xây dựng mô hình.

Chương 2: Trình bày các khái niệm về ngôn ngữ mô hình *Promela,* bao gồm các định nghĩa cơ bản về khai báo biến và kiểu, định danh, hằng số, biểu thức, tiến trình.

Chương 3: Trình bày về bộ kiểm chứng, bao gồm giới thiệu về công cụ *ISPIN* và *JSPIN*, dùng *SPIN* để kiểm chứng, giới thiệu về *LTL*.

Chương 4: Trình bày về các kết quả thực nghiệm của quá trình mô tả hệ thống đèn, thiết kế mô hình hệ thống đènbằng Promela.

Kết luân: Kết quả của khóa luận đã đạt được và hướng nghiên cứu tiếp theo.

CHƯƠNG 1:KIỂM CHỨNG MÔ HÌNH

1. 1 KIỂM CHỨNG MÔ HÌNH.

Việc kiểm tra chương trình có thỏa mãn với thiết kế hay không là một trong những lĩnh vực nghiên cứu chính của ngành Công nghệ Phần mềm.Đã có nhiều hướng tiếp cận nhằm mục đích đưa ra phương pháp tốt nhất để giải quyết bài toán này.Kiểm thử phần mềm là phương pháp phổ biến nhất nhưng chúng ta chỉ có thể tìm ra lỗi trong chương trình chứ không chứng minh được chương trình không còn lỗi.Một hướng tiếp cận nữa đó là thay vì kiểm tra mức mã nguồn thì người ta kiểm tra ngay từ mức thiết kế nhưng khi viết mã cho chương trình vẫn không đảm bảo được là chương trình không còn lỗi.Trong thời gian qua đã có nhiều thành tựu trong lĩnh vực này, đã có nhiều ngôn ngữ và công cụ dùng để đặc tả và kiểm chứng chương trình.Các ngôn ngữ dùng trong đặc tả thuộc tính hệ thống như *LTL,CTL,CTL*,Promela,Alloy* cùng với các công cụtương ứng thìSPINđã trợ giúp rất tốt để giải quyết bài toán này.

Các phương pháp kiểm chứng phần mềm đã bổ trợ cho một số hạn chế của kiểm thử.Trong kiểm chứng phần mềm truyền thống chương trình được mô hình hóa bằng một đặc tả *S* thể hiện bằng ngôn ngữ *L(S)*.Bản thiết kế hệ thống được đặc tả bằng ngôn ngữ *L(A)*.Mã nguồn chương trình *P*được thể hiện bằng một ngôn ngữ lập trình*L(P)* nào đó.Khi kiểm chứng, mã nguồn chương trình được chuyển sang một mô hình*M* thể hiện bằng ngôn ngữ *L(M)* vậy:

$$L(M) \subseteq L(A)$$

Để kiểm tra mã nguồn chương trình có đúng với thiết kế hay không? Người ta kiểm tra xem *L(M)* có thỏa mãn *L(A)* hay không? Điều này khó chứng minh.Do vậy người ta sẽ tìm ¬L(M) rồi kiểm tra:

$$L(A) \cap \neg L(M) \equiv \varnothing$$

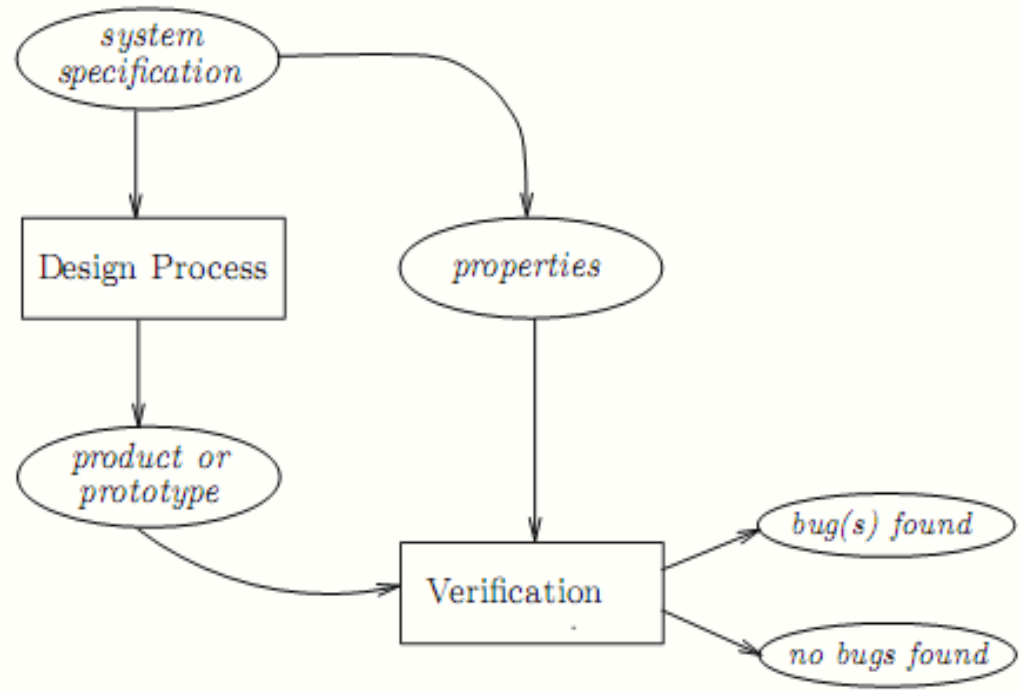
Nếu $L(A) \cap \neg L(M) \equiv \varnothing$ khẳng định mã nguồn chương trình không có lỗi trường hợp ngược lại khẳng định có lỗi và có thể cô lập đoạn mã có lỗi để khắc phục.

Nhưng điều quan trọng ở đây là:$L(M) \equiv L(P)$?Làmcách nào để khẳng định được $L(P) \equiv L(M)$?Điều này rất khó vì quá trình này có sự tham gia của con người (quy trình nào có người tham gia thì sẽ có lỗi).

Như đã đề cập ở trên,bản thiết kế chương trình được thể hiện bằng ngôn ngữ *L(A)* và trong thực tế người ta dùng *logic* để biểu diễn và *LTL* được chọn để biểu diễn thiết kế này.Để kiểm chứng chương trình người ta chuyển *L(A)* và *L(P)* về ngôn ngữ mà ôtômát Büchi đoán nhận đượcbằng thuật toán trong.

Kỹ thuật xác minh hệ thống đang được áp dụng cho việc thiết kế các hệ thống công nghệ thông tin một cách đáng tin cậy. Kiểm chứng mô hình chính là xem sét phần mềm sản xuất ra có đúng yêu cầu,hợp lý và đúng các trường hợp mà phần mềm phải đáp ứng không. Để thực hiên điều đó phần mềm sẽ được chạy trên một số hữu hạn đầu vào và được thiết kế sẵn,phần mềm có lỗi hay không sẽ được đem so sanh với dữ liệu đầu ra mong muốn.

Trong giai đoạn kiểm thử việc chạy hết các trường hợp có thể cócủa dữ liệu đầu vào là rất khó và thường không thực hiện được dẫn đến do rất có thể chứa lỗi.Không những thế trong giai đoạn kiểm thử lỗi phát hiện thường là muộn dẫn đến rất khó khắc phục,tiêu tốn về thời gian,tiền của.



Hình1.1 Sơ đồ về việc kiểm chứng hệ thống

Quá trình kiểm chứng rất quan trọng,nó giúp chúng ta biết được một thiết kế hay một sản phẩm phần mềm có đúng,đảm bảo những tính chất yểu cầu mà được quá trình đặc tả hệ thống đưa ra không.

Mọi công việc của quá trình kiểm thử đều được dựa trên việc đặc tả hệ thống. Một lỗi sẽ được phát hiện khihệ thống không thỏa mãn những tính chất đặc tả của hệ thống và ngược lại hệ thống sẽ đúng khinhững tính chất đó được thỏa mãn.Vì vậy việc kiểm chứng giúp phát hiện lỗi sớm.

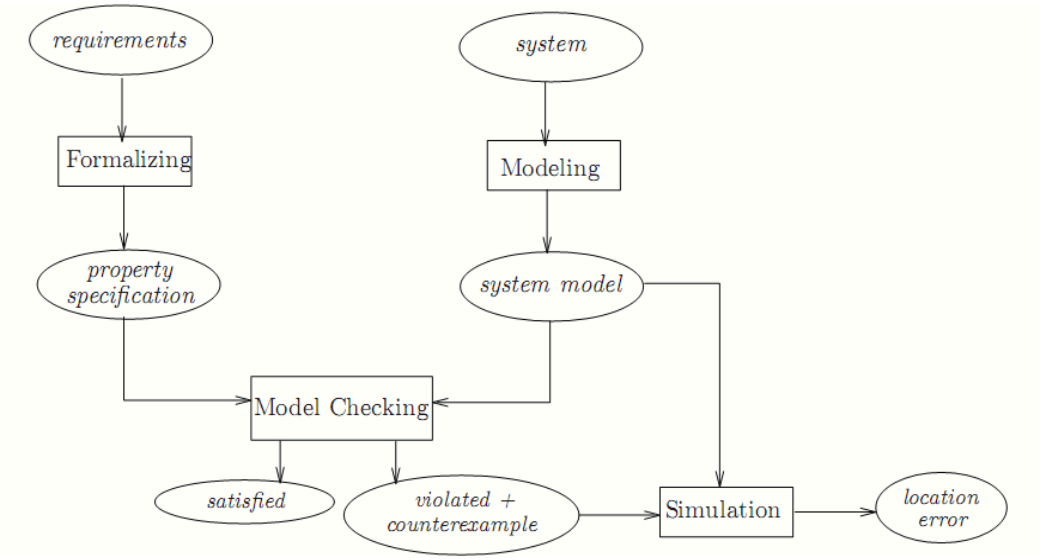
Việc kiểm chứng mô hình dựa trên cơ sở của viêc mô tả chính xác những hành vi của hệ thống một cách không nhập nhằng,điều này giúp phát hiện ra những điều nhập nhằng, không đúng và chưa hoàn thiện của hệ thống.

Kĩ thuật này giúp kiểm chứng trong quà trình thiết kế sản phẩm, nó cũng là một công cụ trong việc kiểm tra những sản phẩm phần mềm bắt buộc khi chạy không có sai sót.

1. 2 CÁCH TIẾN HÀNH

Việc kiểm chứng mô hình được thực hiện bằng việc xác định yêu cầu của hệ thống sau đó ta xây dựng mô hình của hệ thống, từ đó giúp hiểu được các chức năng cũng như hành vi của hệ thống.Ta có thể xây dựng mô hình hệ thống bằng những ngôn ngữ lập trình như *C*, *Promela* hay *java*…

Sau đó công cụ kiểm chứng sẽ sinh ra tất cả những trạng thái có thể có của hệ thống. Kết quả đầu ra sẽ được so sánh với kết quả đầu vào của hệ thống và kiểm tra chúng có thỏa mãn hay không,nếu không thỏa mãn thì bộ kiểm chứng sẽ tìm ra trạng thái không thỏa mãn đó.



Hình1.2 Sơ đồ hoạt động của phương pháp kiểm chứng mô hình

- **Các bước thực hiện của kiểm chứng mô hình.**
  - Từ đặc tả các chức năng,yêu cầu của hệ thống ta xây dựng mô hình.
  - Tạo đầu ra từ các dữ kiện của bài toán
  - So sánh kết quả đầu ra và kết quả thực tế mong muốn
  - Sửa đổi mô hình,tạo thêm ca kiểm thử,dừng kiểm thử,đánh giá chất lượng của phần mềm (Nếu cần).

Kiểm chứng mô hình cũng là một khả năng đánh gia phần mềm hiểu quả.

- **Ưu nhược điểm của kiểm chứng mô hình.**

Trong sản xuất và phát triển phân phần mềm công việc kiểm thử là không thể thiếu được,nhưng nếu kiểm thử bằng phương pháp truyền thống thì sẽ mất rất nhiều thời gian,tiền của… làm cho phần mềm không đáp ứng đủ yêu cầu mà người dùng đưa ra.Chính vì thế kiểm chứng mô hình sẽ khác phục được một số nhược điểm đó:

- *Do quá trình sinh ca kiểm thử là tự động nên quá trình kiểm thử được rút ngắn,đồng thời chất lượng phần mềm được tốt hơn.*
- *Tuy chi phí do việc xây dựng mô hình là lớn nhưng chi phí để bảo trì phần mềm là tốn ít hơn khi hệ thống được đưa vào hoạt động.*
- *Trong kiểm chứng mô hình các ca kiểm thử được tự động sinh ra nên lỗi được phát hiện nhiều hơn.*
- *Lỗi được phát hiện sớm sẽ tăng thời gian giải quyết vàkhác phục sự Do kiểm chứng mô hình là một phương pháp kiểm chứng tổng quát áp*

dụng được cho một phạm vi lớn các ứng dụng:kĩ nghệ phần mềm,thiết kế phần cứng…

- **Bên cạnh đó kiểm chứng mô hình cũng có những nhược điểm:**
  - *Do phải xây dựng mô hình hệ thống một cách chi tiết lên người kiểm thử phải là những người biết phân tích thiết kế hệ thống.*
  - *Do kiểm chứng mô hình dựa trên việc xây dựng mô hình hệ thống,chình vì vậy người kiểm thử phải bỏ thời gian, trí tuệ và tiền bạc vào việc xây dựng mô hình hệ thống.*
  - *Phương pháp chủ yếu được áp dụng những ứng dụng điều khiển là chính,nó không phù hợp với những ứng dụng có khối dữ liệu tăng vô tận.*

CHƯƠNG 2:NGÔN NGỮ PROMELA

2. 1 NGÔN NGỮ PROMELA

Để*SPIN* có thể hiểu được mô hình hệ thống khi chúng ta sử dụng ngôn ngữ *Promela* để xây dựng mô hình.

Chương này chúng ta sẽ biết cách kiểm chứng tự động bằng công cụ *SPIN*.Để có thể làm việc được với *SPIN* chúng ta phải xây dựng mô hình của hệ thống bằng ngôn ngữ *Promela*.Chương này sẽ lần lượt trình bày những khái niệm cơ bản về ngôn ngữ mô hình *Promela*,công cụ *SPIN*,và giao diện người dùng **JSPIN** và ***ISPIN*.**

2. 1. 1 CẤU TRÚC CHƯƠNG TRÌNH PROMELA

Một chương trình *Promela* cơ bản gồm:

- Khai báo kiểu.
- Khai báo biến.
- Khai báo tiến trình.
- [init process].

```
// Các khai báo kiểu và biến
mtype = {MSG,ACK};
chan toS =...
chan toR =...
bool flag;
// Một tiến trình
proctype Sender() {
// Thân một tiến trình
...
}
proctype Receiver() {
...
}
```

```
// Tiến trình init
init {
// Tạo một tiến trình
...
}
```

Biến cũng có thể được khai báo như mảng,ví dụ:

```
int x [10];
```

Khai báo một mảng 10 số nguyên có thể được truy cập trong mảng *Subscript* biểu hiện như:

```
x[0] = x[1] + x[2];
```

Các mảng không thể được liệt kê,nên nó phải được khởi tạo như sau:

```
int x[3];
x[0] = 1;
x[1] = 2;
x[2] = 3;
```

Mảng đa chiều có thể được định nghĩa gián tiếp với sự giúp đỡ của các cấu trúc *typedef.*

2. 1. 2 KIỂU DỮ LIỆU CƠ BẢN

- Bảng 2.1 dữ liệu cơ bản trong *Promela:*

Kiểu dữ liệu	Kích thước(bits)	Dữ liệu
Bit,bool	*1*	*0, 1, false,true*
Byte	*8*	*0…225*
Short	*16*	*-32768…32767*
Int	*32*	$-2^{31} \ldots 2^{31}-1$
Unsigned	*<= 32*	$0 \ldots 2^{n}-1$

2. 1. 3 TOÀN TỬ CƠ BẢN

Do ngôn ngữ*Promela* gần giống như ngôn ngữ C chính vì thế *Promela* có toán tử tương tự với ngôn ngữ C.

Các toán tử cơ bản trong Promela xếp theo thứ tự độ ưu tiên giảm dần:

Toán tử	Tên
()	Dấu ngoặc đơn
[]	Chỉ số mảng
.	Lựa chọn trường
!	Phủ định logic
~	Phân theo bit
++, --	Tăng,giảm
*, /, %	Nhân,chia, modul
<<,>>	Dịch trái,dịch phải
++,-	Cộng,trừ
<,<=,>,>=	Các phép so sánh logic
==,!=	Tương đương,không tương đương
&	Và
^	Loại trừ bit hoặc
|	Gộp bit hoặc
&&	Và logic
||	Trái hoặc logic
=	Gán

2. 1. 4 TÊN, TÊN HẰNG SỐ VÀ BIỂU THỨC

- Tên

Tên có thể là một chữ cái,một ký tự.

- Hằng số

Hằng số là một chuỗi ký tự đại diện cho một số nguyên thập phân.Hằng số tượng trưng có thể được định nghĩa như sau:

#defineMAX999

2. 1. 5 TIẾN TRÌNH

Một tiến trình được khai báo bắt đầu bằng một từ khóa *proctype* và gồm có:

- *Tên*
- *Danh sách thông số chính*
- *Khai báo biến cục bộ*

Cú pháp của một khai báo tiến trình:

```
proctype name( /* formal parameter list */ )
{
/* các khai báo địa phương và các lệnh */
…
}
/* và */ quy định giới hạn chú thích trong promela
```

- **Tiến trình khởi tạo (init)**

Tất cả các chương trình *Promela* đều cần một tiến trình khởi tạo, nó giống như hàm *main()* trong ngôn ngữ C.Việc thực thi một chương trình *promela* được bắt đầu từ tiến trình *init*.

Một tiến trình *init* có dạng:

```
init { /* Các khai báo địa phương và các biểu thức.*/ }
```

Đơn giản nhất có thể là chương trình *Promela* có dạng:

```
init { skip }
```

Skip có nghĩa là không có biểu thức nào trong tiến trình init.

2. 2 XỬ LÝ KÊNH TRONG PROMELA

2. 2. 1 CÚ PHÁP

Kiểu kênh trong *Promela* đi kèm với hai toán tử gửi: !và nhận:?,được khai báo:

Khai báo:

```
Chanch = [dung_luong] of {kieu_du_lieu_1,...,
kieu_du_lieu_n } ;
```

Có 2 kiểu kênh trong *Promela:*

Kênh gửi và nhận:được khai báo với dung lượng bằng 0.

Kênh đệm:được khai báo với dung lượng lớn hơn 0.

Lệnh truyền dữ liệu trên kênh

Gửi:

```
ten_bien_kenh ! bieu_thuc_1, ...,bieu_thuc_n
```

Nhận:

```
ten_bien_kenh?bien_1, ...,bien_n
```

Giá trị của các biểu thức có kiểu tương ứng với kiểu khi kênh được khai báo.

Các biểu thức tronglệnh gửi sẽ được tính toán vàgiá trị thu được sẽ trở thành thông điệp truyền trên kênh,lệnh nhận được thực thi sau lệnh đó sẽ gán các giá trị này cho các biến của nó.

Một lệnh nhận chỉ được thực khi tồn tại thông điệp được gửi lên kênh.

2. 2. 2 KÊNH GỬI VÀ NHẬN.

Kênh gửi và nhận (được khai báo với dung lượng bằng 0) biểu thị rằngnơi gửi (tiến trình chứa lệnh gửi) và nơi nhận thông điệp (tiến trình chứa câu lệnh nhận) truyền dữ liệu một cách đồng bộ.Khi đó,tiến trình chứa lệnh gửi sẽ bị chặn cho đến khi lệnh nhận trong tiến trình nhận được thực thi.

Nếu trong một tiến trình có một lệnh gửi (hay nhận) được khởi tạo mà không có lệnh nhận (hay gửi) nào tương ứng (về của kiểu thông điệp) thì tiến trình đó sẽ bị khóa .

2. 3. CÁC CÚ PHÁP

2. 3. 1 LỆNH printf( )

Câu lệnh prinf

```
printf ("xau_ki_tu",cac_bien) ;
```

trong xâu kí tự có thể chứa các định dạng in tương ứng với các biến như%d

2. 3. 2 LỆNH LỰA CHỌN if

```
if
::bieu_thuc_logic_1 → lenh_11; lenh_12;…lenh_1n
::bieu_thuc_logic_2 → lenh_21; lenh_22;…lenh_2n
...
::bieu_thuc_logic_n → lenh_n1; lenh_n2;…lenh_nn
fi;
```

Ngoài ra,biểu thức logic có thể là*false*hoặc*true*. Chuỗi lệnh theo sauelsesẽ được thực thi nếu các biểu thức logic còn lại đều nhận giá trị false.Chuỗi lệnh theo sau *true* luôn luôn được chọn để thực thi.

Chuỗi lệnh theo sau biểu thức logic có thể trống,khi gặp chuỗi lệnh trống,chương trình bỏqua chuyển sang câu lệnh sau câu lệnh *if*,*skip*có ý nghĩa tương đương với một chuỗi lệnh trống.

2. 3. 3 LỆNH LẶP DO

```
do
::bieu_thuc_logic_1 → lenh_11; lenh_12;…lenh_1n
::bieu_thuc_logic_2 → lenh_21; lenh_22;…lenh_2n
....
::bieu_thuc_logic_n → lenh_n1; lenh_n2;…lenh_nn
::bieu_thuc_logic → break
od;
```

Câu lệnh *if*vàdothể hiện tính không tất định của *Promela*:nếu hai hay nhiều biểu thức logic có giá trịtrue,chuỗi lệnh theo sau một biểu thức bất kỳ trong số đó sẽ được thực thi.

2. 3. 4 LỆNH NHẢY GOTO

Giúp nhảy đến 1 đoạn chương trình sau một nhãn với tên nhãn đặt trước dấu hai chấm.

```
goto:ten_nhan;
```

2. 3. 5 LỆNH define

*define*là lệnh macro dùng để định nghĩa ký hiệu cho một biểu thức và được đặt ởđầu chương trình:

```
#define length 10
```

2. 4. RUN VÀ ATOMIC

2. 4. 1 RUN VÀ TIẾN TRÌNH INIT()

Run là một cách khác để khởi tạo một tiến trình.Trước đó một tiến trình được khai báo mà không có từ khóa *active*

```
proctype P(){
…
}
run P ();
```

Với việc sử dụng *run* ta cóthểkhai báotất cả các tiến trình có trong chương trình sau đó "*run*" (khởi tạo) chúng trong một tiến trình có tên *init().*

Tiến trình *init()*luôn là tiến trình đầu tiên được khởi tạo và do vậy luôn cópidbằng 0.

```
proctype P(){
…
}
proctype Q(){
…
}
```

```
init {
run P();
run Q();
}
```

2. 4. 2 ATOMIC

Chuỗi lệnh đặt trong*atomic{}*sẽ được thực hiện như một câu lệnh độc lập và không bị chen vào bởi một lệnh nào khác ngoài nó. Do vậy,sử dụngatomicgiúp giảm sự phức tạp của mô hình cần kiểm chứng.

Ví dụ 3. 1.Chuỗi lệnh dùng atomic

```
atomic {
temp = n;
n = temp+1
}
```

Đoạn mã trong ví dụ 3. 1.tương đương với lệnh n=n+1;

Ta có thể sử dụng *atomic* để kết hợp các câu lệnh giữa các tiến trình

```
chan ch = [0] of {int};
active proctype P(){ atomic {A;ch!1;B}}
active proctype Q(){ atomic {ch?1 -> C}
```

Trong ví dụ trên,sau khi khối lệnhAtrong tiến trìnhPđược thực thi,dữ liệu kiểu intvới giá trị1được gửi lên kênhchkiểu gặp (câu lệnhch!1;được thực hiện),tiến trình Q nhận dữ liệu trên kênh và khối lệnh C được thực thi,khi khối lệnh Ckết thúc,việc thực thi sẽ tự động chuyển về tiến trìnhPvà thực thi khối lệnh B.

*Atomic*có thể đượcsử dụng để khởi tạo một số các tiến trình và đảm bảo rằng không một tiến trình nào bắt đầu chạy cho tới khi tất cả các tiến trình được khởi tạo hết.

Tiến trình*init*đã được khởi tạo và có thể chạy,một trong hai tiến trình P,Q có thể được chạy trước khi tiến trình còn lại được khởi tạo,điều đó không có lợi cho chúng ta. Bổ sungatomicsẽ loại bỏ được điều này:

```
proctype P(){
…
}
     proctype Q(){
…
}
init {
atomic {
run P();
run Q();
}
```

CHƯƠNG 3 BỘ KIỂM CHỨNG MÔ HÌNH

3. 1 BỘ KIỂM CHỨNG MÔ HÌNH SPIN

Để kiểm chứng mô hình chúng ta có rất nhiều công cụ dể kiểm chứng nhưng trong đồ án này để kiểm chứng trên hệ tương tranh em sử dụng bộ kiểm chứng SPIN trên 2 công cụ là JSPIN và ISPIN.

3. 1. 1 GIỚI THIỆU VỀ SPIN

SPIN là công cụ mã nguồn mở,phổ biến và được sử dụng bởi hàng ngàn người trên toàn cầu.Nó được sử dụng cho việc xác minh thẩm định của các hệ thống phân phối phần mềm.

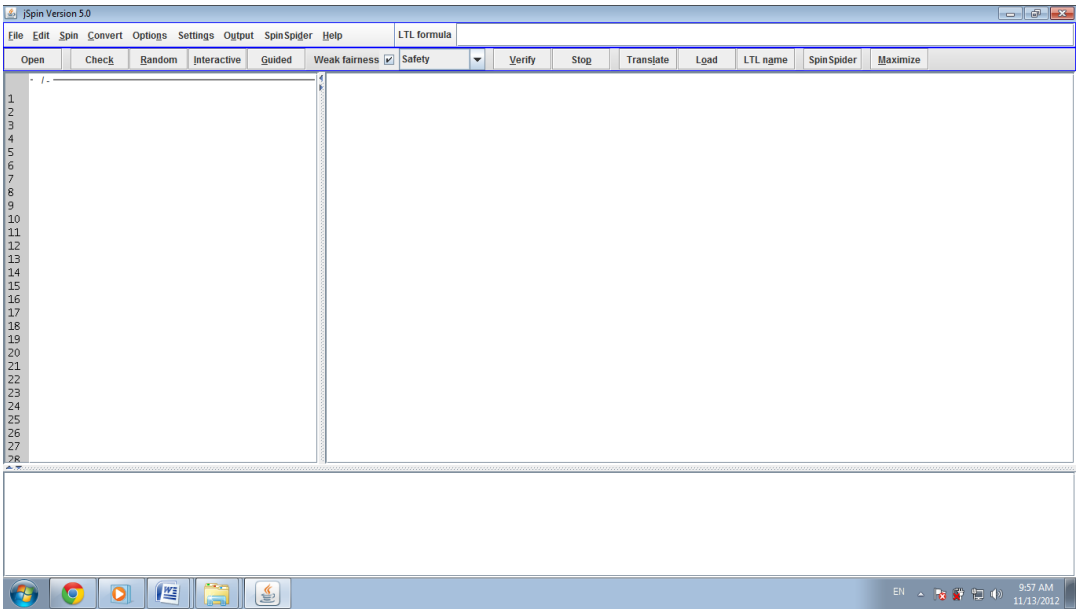
Từ năm 1980 SPIN được phát triển tại Bell Labs của nhóm Unix thuộc trung tâm nghiên cứu Khoa học máy tính.Phần mềm này phát triển rộng rãi từ năm 1991.

Tháng 4 năm 2002 SPIN được trao giải thưởng phần mềm uy tín cho năm 2001 của ACM.

SPIN cung cấp một giả lập cho phép các nhà thiết kế đạt được kết quả phản hồi từ hệ thống mô hình của họ.Những kết quả phản hồi đó đóng vai trò quan trọng trong sự hiểu biết của các nhà thiết kế về hệ thống trước khi họ đầu tư vào phân tích chính thức.

3. 1. 2 CÔNG CỤ JSPIN

Có giao diện thân thiện người dùng cửa sổ soạn thảo chính với khả năngchỉnh sửa và tìm kiếm là đơn giản.ISPIN là công cụ thuận tiện để tiếp cận với SPIN.



Hình3.1 Giao diện JPIN

- **Check**

Kiểm tra cú pháp ngôn ngữ Promela.

Luôn luôn là bước đầu tiên sau khi thay đổi chương trình Promela.

- **Random**

Chạy chế độrandom,để có thể kiểm tra lỗi của bản thân chương trình Promela.

LTL Temporal Logic Formulae

LTL = Mệnh đề Logic + toán tử điều khiển thời gian.

Giúp chỉnh sửa và bảo trì các công thức logic.

Theo thời gian trong SPIN

- Bước 1:Chạy "*LTL Property Manager*"
- Bước 2:Nhập vào đặc tính thời gian mà bạn muốn thẩm định.Chú ý phải là biểu thức bất biến và tên bằng chữ thường.
- Bước 3:Chỉ ra là có hay không đặc tính thời gian cần giữ:

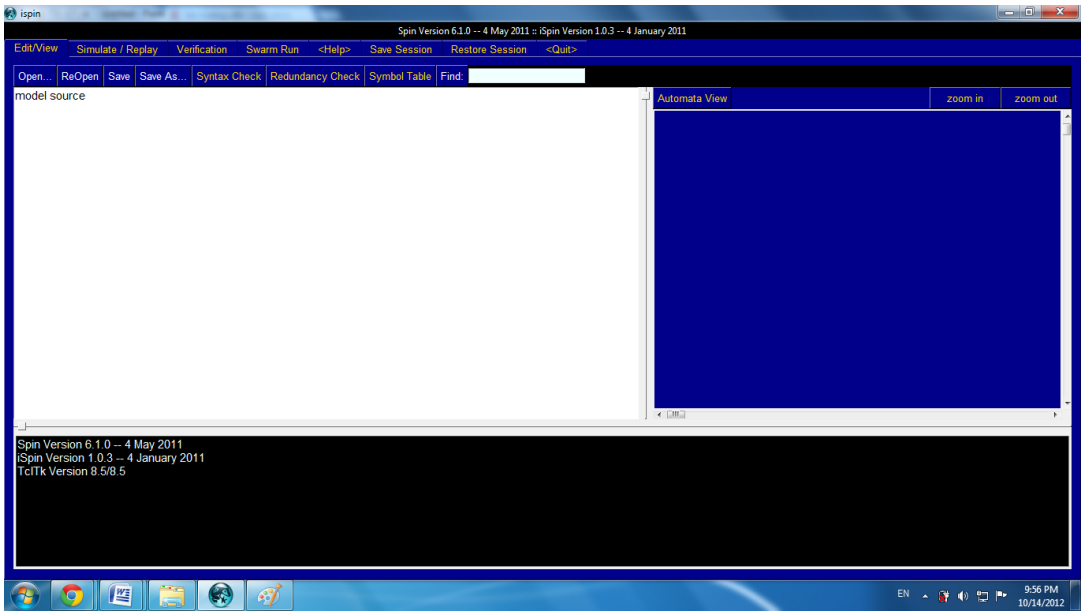
all executions (desired behaviour) hoặc *no executions (error behaviour)*

- Bước 4:Nhập vào một định nghĩa vi mô đối với mỗi hằng số mệnh đề trong của sổ phụ "*Symbol Definitions*"

- Bước 5:Ấn vào nút "*RunVerification*" và tiếp tục ấn vào nút "*Run*" trong của sổ "*LTLVerification*"

3. 2. 3 CÔNG CỤ ISPIN

Công cụ ISPIN có giao diện



Hình 3.2 Giao diện ISPIN

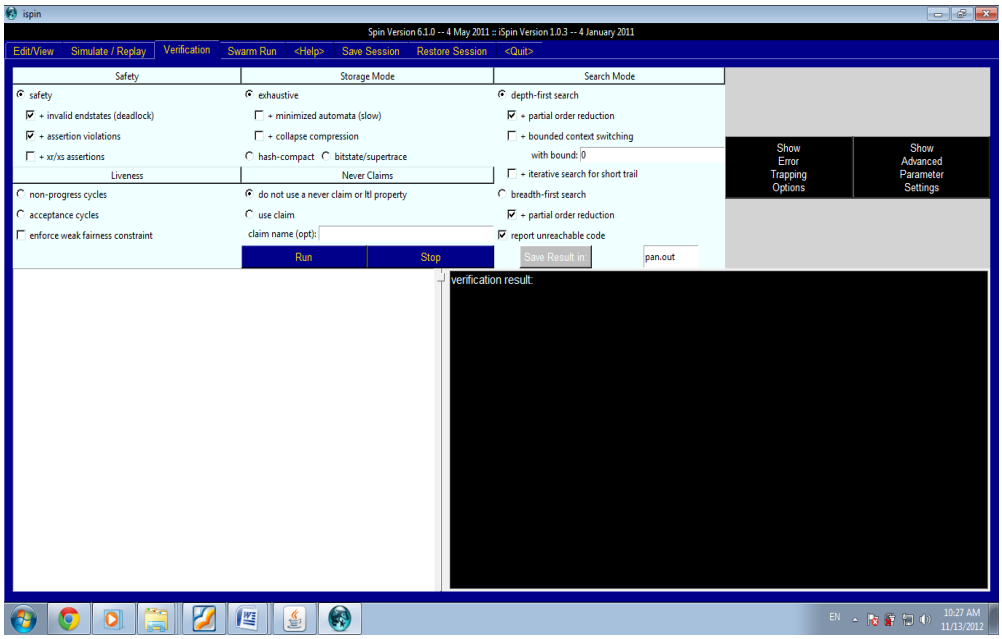
- **Run Syntax Check**

Giống như chức năng của check trong **JSPIN**

- Kiểm tra cú pháp ngôn ngữ *Promela*.
- Luôn luôn là bước đầu tiên sau khi thay đổi chương trình Promela.
- Lập thông số Verification kiểm tra mô hình.
- Đảm bảo thực hiện an toàn và xác minh tính chất.

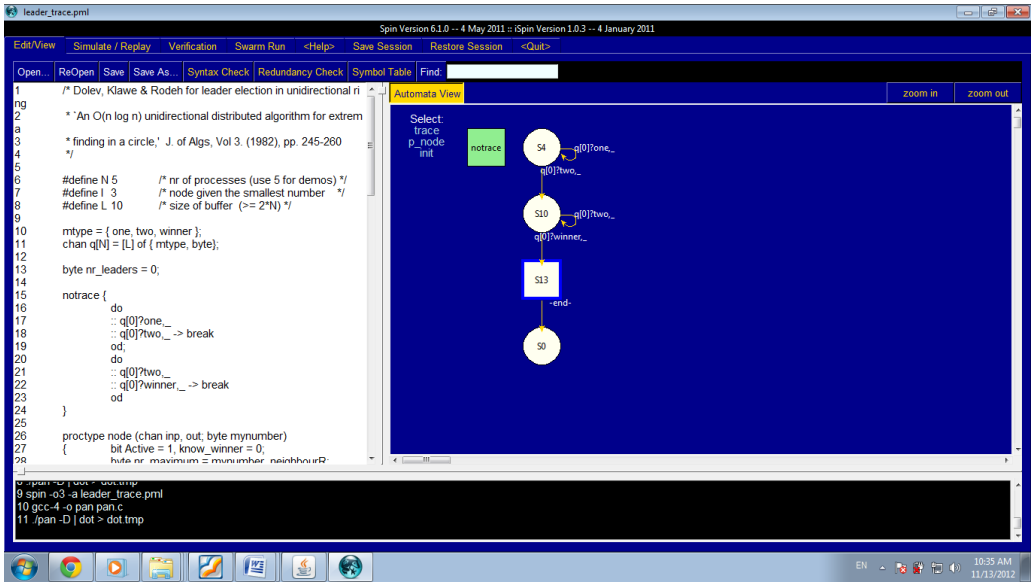
- **Correctness Properties:Safety,Liveness.**
- **Search mode.**
- **A full queue.**
- **Verify an LTL property.**

Set advance options.



Hình3.3 Cửa sổ của Verification

- **View SPIN Automaton (FSM View)**



Hình 3.4 Của sổ chạy chức năng View SPIN Atomaton

3. 2 DÙNG SPIN ĐỂ KIỂM CHỨNG

3. 2. 1 GIẢ LẬP NGẪU NHIÊN

Trong chế độ giả lập,SPIN sẽ biên dịch và chạy một chương trình*Promela*, sau đó in ra các trạng thái của chương trình.Khóa luận này em sử dụng công cụ **JSPIN** và **ISPIN** để xem xét kết quả kiểm chứng một cách trực quan.

Một chương trình bao gồm rất nhiều trạng thái và mỗi trạng thái và mỗi trạng thái bao gồm một bộ gồm các giá trị gồm các biến.Một tính toán là một chuỗi các trạng thái khởi tạo và tiếp tục với những trạng thái xuất hiện khi mỗi câu lệnh được thực hiện.

3. 2. 2 Verify

Assertion là cách đơngiản để kiểm tra chương trình.Một *assertion* là một mệnh đề được đặttrong1 chương trình mà tacho rằng mệnh đề sẽ luôn đúng tại vị trí đó. SPIN sẽ tính toán các *assertion* trong quá trình tìm kiếm phản ví dụ trong không gian trạng thái của chương trình.

Ví dụ 3. 1 Nghịch đảo các số

```
1    active proctype P() {
2    int value = 123;
3    int reversed;
4    reversed =
5         (value % 10)*100 +
6         (value / 10)%10 + 10 +
7         (value / 100);
8    printf("value  =  %d,reversed  =  %d\n",value,
reversed)
9}
```

Trong chương trình này ta khai báo các biến,*value* và *reversed* thuộc kiểu *int*,biến đầu tiên được khởi tạo giá trị ban đầu.Giá trị gán cho biến *reversed* được tính toán từ giá trị của biến value sử dụng phép chia và phép chia có dư,sau đó,những giá trị của biến này được in ra màn hình.Câu lệnh *printf* được lấy ra từ ngôn ngữ C:Một xâu trong dấu ngoặc kép,tiếp theo đó là một danh sách các biến,

danh sách các biến này cần thỏa mãn định dạng được kí hiệu trong xâu.Đặc tả để in ra giá trị nguyên %d,và xâu có thể được kết thúc với kí hiệu bắt đầu trong dòng mới \n.

Nếu chạy toàn bộ mô phỏng ngẫu nhiên của chương trình SPIN sẽ in ra kết quả:

```
Value = 123, reversed = 321
```

Trong ISPIN ta chạy Verify,kết quả không có lỗi vi phạm nào được thông báo từ chương trình.

```
(SPIN Version 6. 1. 0 -- 4 May 2011)
+ Partial Order Reduction
Full statespace search for:
never claim - (not selected)
assertion violations+
cycle checks- (disabled by -DSAFETY)
invalid end states+
State-vector 20 byte,depth reached 2,errors:0
3 states,stored
0 states,matched
3 transitions (= stored+matched)
0 atomic steps
hash conflicts:0 (resolved)
2. 539memory usage (Mbyte)
unreached in proctype P
(0 of 3 states)
pan:elapsed time 0. 001 seconds
No errors found -- did you verify all claims?
```

Trong ví dụ 3. 1 ta cũng tìm nghịch đảo các số nhưng khi biễn value được gán giá trị value =(value % 10)*100 + (value / 10)%10 + 10 +1000(value / 100)do vậy khi in giá tri vulue và reversed không thỏa mãn.

Ví dụ 4. 1 Chương trình nghịch đảo các số có chứa lỗi:

```
    active proctype P() {
    intvalue = 123;
    int reversed;
    value =
    (value % 10)*100 +
    (value / 10)%10 + 10 +1000
    (value / 100);
    printf("value   =   %d,reversed   =   %d\n",value,
reversed)
    }
    Khi chạy Verify, ISPIN sẽ đưa ra thông báo lỗi vi
phạm
    verification result:
    SPIN -athu
    SPIN:thu:7,Error:syntax errorsaw ''(' = 40'
    SPIN:thu:10,Error:no runable process
    gcc-4 -DMEMLIM=1024 -O2 -DXUSAFE -DSAFETY -DNOCLAIM
-w -o pan pan. c
    pan. c:435:17:error:expected expression before â€˜,
â€™ token
```

3. 3 GIỚI THIỆU VỀ LTL(Linear Temporal Logic)

Linear Temporal Logic (viết tắt *LTL*) được đề xuất bởi Amir Pnueli (1941-2009),một nhà khoa học người Israel,là một loại logic áp dụng cho thời gian,người ta có thể xây dựng các công thức về tương lai.Ví dụ:một điều kiện cuối cùng sẽ

đúng hoặc một điều kiện sẽ đúng cho đến khi một điều kiện khác đúng, ... *LTL* là một phần của *CTL** (một loại logic có thêm các lượng từ và nhánh thời gian).*LTL* đầu tiên được đề xuất dùng trong kiểm chứng hình thức bởi Amir Pnueli năm 1977.

3. 3. 1 CÚ PHÁP

LTL được xây dựng từ các biến mệnh đề AP (Atomic Proposition),các toán tử logic và các toán tử thời gian X,U.

Một cách hình thức,các công thức LTL được định nghĩa như sau:

Nếu p là một mệnh đề nguyên tử thì p (p $\in$ AP) là một công thức LTL.

Nếu ψ và φ là các công thức LTL thì $\neg\psi$,$\varphi \vee \psi$,**X** ψ và φ **U** ψcũng là các công thức LTL.

$\varphi ::= \text{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg \varphi \mid X\varphi \mid \varphi_1 \text{ U } \varphi_2 \mid$

X đọc là NE**X**T

U đọc là **U**NTIL

Trình tự các toán tử như sau:Toán tử một ngôi có độ ưu tiên cao nhất,toán tử ¬ và toán tử **X**có độ ưu tiên tương đương nhau.Toán tử U có độ ưu tiên hơn các toán tử ∧,∨,và →.Dấu ngoặc đơn "( )" có thể loại bỏ nếu không gây nhầm lẫn.Toán tử U có độ ưu tiên bên phải,tức là công thứcφ_1 U (φ_2 U φ_3) có thểviết thànhφ_1 U φ_2 U φ_3.

Dùng các toán tử ¬,∧ đủ để biểu diễn các công thức trong logic mệnh đề.

$$
\begin{aligned}
\varphi_1 \vee \varphi_2 &\stackrel{\text{def}}{=} \neg(\neg\varphi_1 \wedge \neg\varphi_2) \\
\varphi_1 \rightarrow \varphi_2 &\stackrel{\text{def}}{=} \neg\varphi_1 \ \vee \ \varphi_2 \\
\varphi_1 \leftrightarrow \varphi_2 &\stackrel{\text{def}}{=} (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1) \\
\varphi_1 \oplus \varphi_2 &\stackrel{\text{def}}{=} (\varphi_1 \wedge \neg\varphi_2) \ \vee \ (\varphi_2 \wedge \neg\varphi_1) \\
&\ \ \vdots
\end{aligned}
$$

Ngoài ra,có một số toán tử được bổ sung như **F, G**.Toán tử **F** còn được ký hiệu là ◊,toán tử **G** ký hiệu là □,toán tử **X** ký hiệu là O.

Toàn tử **U** có thể biểu diễn được các toán tử **F, G** như sau:

$$\Diamond\varphi \stackrel{\text{def}}{=} \text{true U } \varphi \quad \text{và} \quad \Box\varphi \stackrel{\text{def}}{=} \neg\Diamond\neg\varphi$$

G đọc là**G**lobally (luôn xảy ra)

F đọc là **F**uture (cuối cùng sẽ xảy ra)

3. 3. 2 NGỮ NGHĨA

Công thức LTL biểu diễn các tính chất của một chuỗi hành động (path,có thể gọi là vết - trace). Điều này có nghĩa là một chuỗi các hành động có thể thoả một công thức LTL hoặc không.Để lập một công thức chính xác khi chuỗi hành động thoả mãn công thức LTL chúng ta tiến hành như sau:

Đầu tiên,ngữ nghĩacủa công thức LTL φ được định nghĩa như một ngôn ngữWords(φ) chứa tất cả các từ vô hạn trên bảng chữ cái 2^{AP}thoả mãn φ,sau đó ngữ nghĩa được mở rộng để diễn giải toàn bộ các trạng thái và các chuỗi hành động của một hệ thống dịch chuyển.

Định nghĩa(Ngữ nghĩa của *LTL*):Cho φlà một công thức LTL trên AP.Tính chất logic thời gian được sinh ra bởi φ là

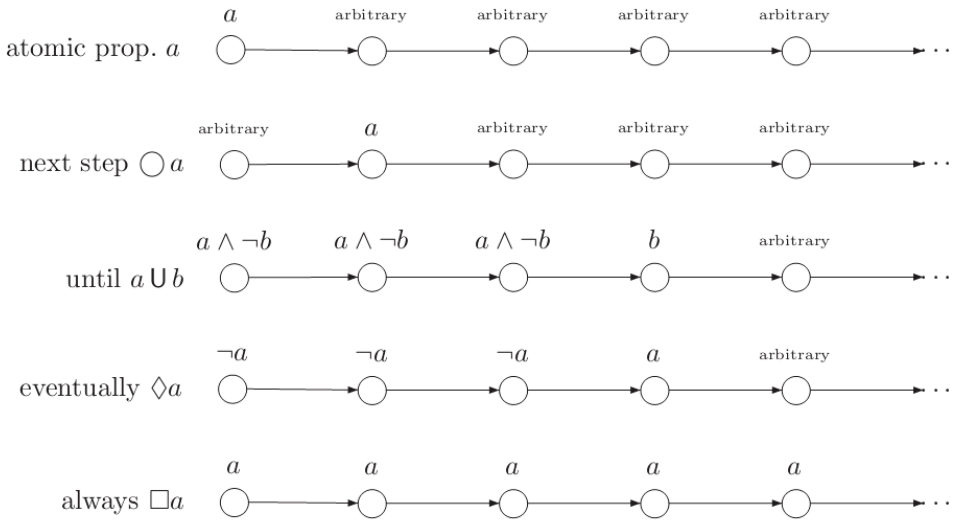
$$\mathbf{Words}(\varphi) = \{\sigma \in (\mathbf{2}^{\boldsymbol{AP}})^{\omega} \mid \sigma \vDash \varphi\}$$

Ở đây,quan hệ thoả được ⊨là tập con của$(2^{AP})^{\omega} \times LTL$ là quan hệ nhỏ nhất với thuộc tính trên bảng 1.Ký hiệu $\vDash \subseteq (2^{AP})^{\omega}\ LTL$.

$$
\begin{array}{lcll}
\sigma & \models & \text{true} & \\
\sigma & \models & a & \text{iff} \quad a \in A_0 \quad (\text{i.e., } A_0 \models a) \\
\sigma & \models & \varphi_1 \wedge \varphi_2 & \text{iff} \quad \sigma \models \varphi_1 \text{ and } \sigma \models \varphi_2 \\
\sigma & \models & \neg \varphi & \text{iff} \quad \sigma \not\models \varphi \\
\sigma & \models & \bigcirc \varphi & \text{iff} \quad \sigma[1\ldots] = A_1 A_2 A_3 \ldots \models \varphi \\
\sigma & \models & \varphi_1 \,\mathsf{U}\, \varphi_2 & \text{iff} \quad \exists j \geqslant 0.\ \sigma[j\ldots] \models \varphi_2 \ \text{ and } \ \sigma[i\ldots] \models \varphi_1, \text{ for all } 0 \leqslant i < j
\end{array}
$$

Ngữ nghĩa LTL (quan hệ thỏa được ⊨) cho các từ vô hạn trên2^{AP}

Giả sử a,b là hai mệnh đề nguyên tử,bảng 2 minh họa ngữ nghĩa của các phép toán đối với các mệnh đề này.



Minh hoạ ngữ nghĩa của *LTL*

CHƯƠNG 4 THỰC NGHIỆM

4. 1 MÔ HÌNH MÁY TRẠNG THÁI HƯU HẠN

Máytrạngtháihữuhạn(*Finite State Machine-FSM*) hay còn gọi làmáy tựđộng trạng thái hữu hạn(*FiniteStateAutomaton-FSA*) được sử dụng để mô tả hoạt động của nhiều hệ thống trong thực tế,là một dạng mô hình đa trạng thái,gồm 3 thành phần chính:

- Các trạng thái
- Các cung chuyển trạng thái
- Các hành động (*Actions*)
- Cách thức để chuyển FSM sang mô hình promela như sau:
- Các bước chuyển trạng thái được khai báo trong một kiểu mtype

Tất cả các thông tin về chuyển trạng thái được chuyển dến một biến *message* kiểu kênh nhận thông điệp thuộc loại mtype trên, sau đó *message* sẽ gửi thông điệp trong tiến trình *coltrol()* là tiến trình điều khiển của máy hữu hạn trạng thái.

Trạng thái của máy được thể hiện bởibiếnstatecó giá trị được thay đổi trong tiến trình*action()*,kênh*message*sẽ nhận thông điệp trong tiến trình action().

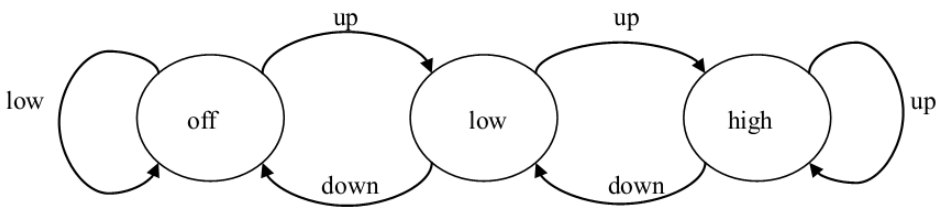
Mỗi khi tiến trình điều khiển *coltrol()*gửi một thông điệp về sự chuyển trạng thái,dựavào thông điệp nhận được và giá trị hiện tại củastate,state sẽ được gán một giá trị mới tương ứng với trạng thái mới.

Vềmặtứng dụng,máy trạng thái hữu hạnđóng vai trò quan trọng trong nhiều lĩnh vực như công nghệ điện tử,ngôn ngữ học,khoa học máy tính,triết học,sinh học,toán học,logic học… Trong khoa học máy tính,máy trạng thái hữu hạnđược sửdụng rộng rãi trong việc mô hình hóa hoạt động của phần mềm,thiết kếhệthống phần cứng,trình biên dịch,công nghệ mạng,xử lý ngôn ngữ tự nhiên …

4. 2 THỰC NGHIỆM VỚI HỆ THỐNG ĐÈN

4. 2. 1 MÔ TẢ BÀI TOÁN

Áp dụng các phương pháp đã đề xuất ở được tìm hiểu trong khóa luận cho bài toán là một hệ thống công tác đèn là một dùng trong gia đình.Đầu vào của thiết bị thông qua 2 nút bấm*up* và *down*.Đầu ra là thông quađèn báo hiệu với 3 mức sáng:tắt *(off)*,sáng yếu *(low)*,sáng mạnh *(high)*.Hoạt động của đèn được thể hiệnbởi mô hình máy hữu hạn trạng thái như sau:



Hình4.1 Mô hình công tắc đèn

Mô hình promela mô tả hoạt động của đèn được xây dựng như sau:mô hình promela gồm 2 tiến trình:*lamp()*và*switch()*(lần lượt tương ứng với*active()*và *control()*.Để thểhiệntương tácgiữacông tắcvà bóng đènhaitiến trình trao đổi thông tin qua kênh updown nhận và gửi thông điệp thuộc kiểu

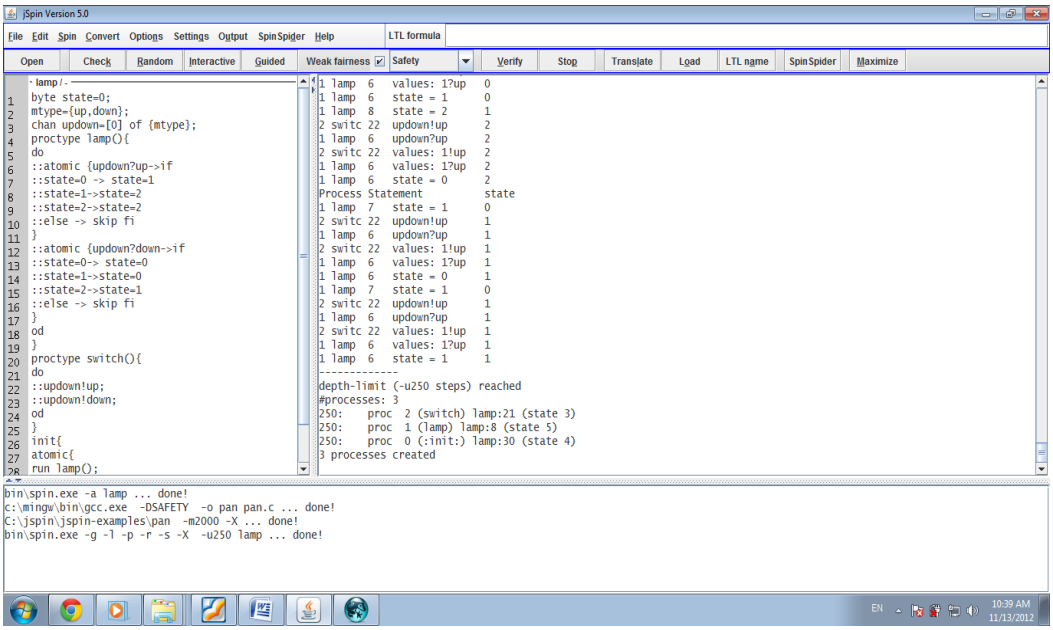
```
mtype = { up,down }
```

Mô hình promela của cái đèn (file lamp. pml):

```
1    byte state=0;
2    mtype={up,down};
3    chan updown=[0] of {mtype};
4    proctype lamp(){
5    do
6         ::atomic {updown?up->if
7         ::state=0 -> state=1
8         ::state=1->state=2
9         ::state=2->state=2
10        ::else -> skip fi
11   }
12        ::atomic {updown?down->if
13        ::state=0-> state=0
14        ::state=1->state=0
15        ::state=2->state=1
16        ::else -> skip fi
```

```
17  }
18  od
19  }
20  proctype switch(){
21  do
22       ::updown!up;
23       ::updown!down;
24  od
25  }
26       init{
27       atomic{
28       run lamp();
29       run switch();}
30  }
```

Trong mô hình trên giá trị của biếnstatelà 0,1,2 lần lượt mô tả ba trạng thái của đèn là *off,low,high*.Với mô hình này,trong *SPIN*ta chạy chế độ*random*,để có thể kiểm tra lỗi của bản thân chương trình Promela. Trong chế độ này,khi tiến trình init được thực thi,*SPIN* giả lập hoạt động của đèn,chuỗi các tín hiệuupvàdownđược sinh ra một cách tùy ý(nhờ sự thực thi tiến trình *switch()*)và từ đó trạng thái của đènthay đổi (nhờ sự thực thi tiến trình*lamp()*).Chương trình không có lỗi cú pháp và *SPIN* dừng sau khi đạt tới giới hạn 250 bước.



Hình 4.2Kết quả chạy giả lập mô hình hệ thống đèn

4. 2. 2 KIỂM CHỨNG MÔ HÌNH HỆ THỐNG ĐÈN BẰNG SPIN

4. 2. 2. A KIỂM CHỨNG MÔ HÌNH HỆ THỐNG CÔNG TẮC ĐÈN ĐÁP ỨNG THUỘC TÍNH

Chạy giả lập ở chế độ *dandom* để kiểm tra chương trinh:

Mô tả các trạng thái của đèn:

off:up:low

low:up:high

low:down:off

high:up:high

high:down:low

Sau khi chạy giả lập ở chế độ random như ở phần trên ta cần kiểm tra tính chất mà hệ thống cần thỏa mãn bằng việc chạy*verify*.Tính chất cần kiểm chứng trong trường hợp này là:khi đèn ở trạng thái*high*,cần phải qua trạng thái*low*rồi mới có thể về trạng thái *off*.

Biểu thức LTL mô tả tính chất trên là:

```
[] !((state == 2) && (state != 1) U (state ==0))
```

(Không bao giờ xảy ra trường hợp biến state bằng 2 và,nó luôn khác 1 cho tới khi nó bằng 0)

Ta thêm vào mô hình promela định nghĩa các kí hiệu

#define a0 (state = = 0)

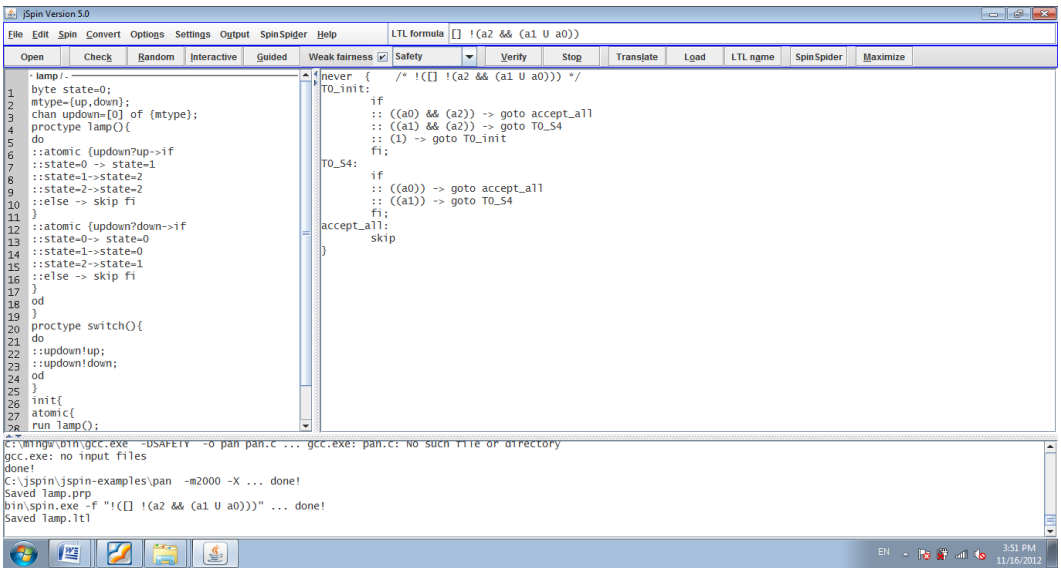
#define a1 (state != 1)

#define a2 (state = = 2)

Biểu thức LTL đưa vào SPIN là:

[] !(a2 && (a1 U a0))

Khi đó trong*JSPIN* ta sử dụng chức năng *Translate* để dịch tử ngôn ngữ*LTL* sang Promela.



Hình4.3 JSPIN dịch từ LTL sang Promela

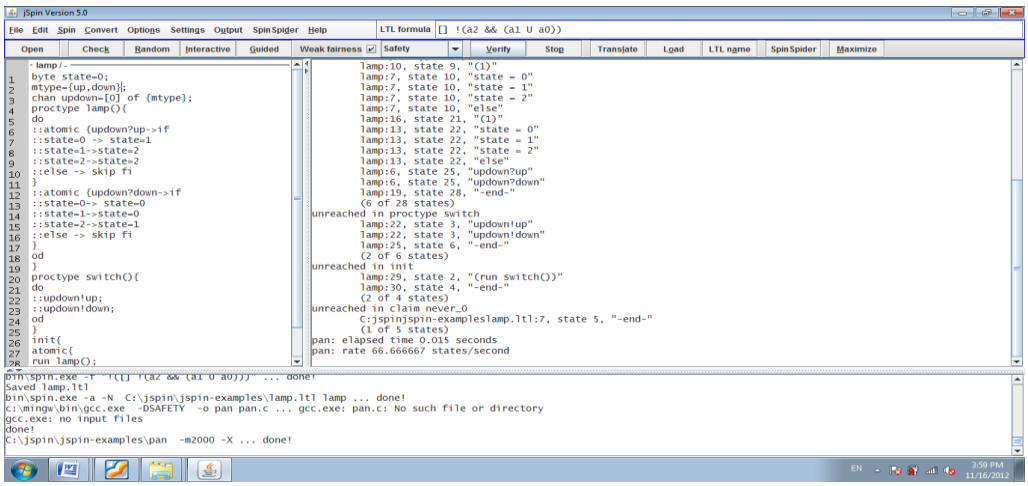
Code sau khi *JSPIN* dịch từ *LTL* sang ngôn ngữ Promela:

never{/* !([] !(a2 && (a1 U a0))) */

```
T0_init:
If
:: ((a0) && (a2)) -> goto accept_all
     :: ((a1) && (a2)) -> goto T0_S4
     :: (1) -> goto T0_init
fi;
```

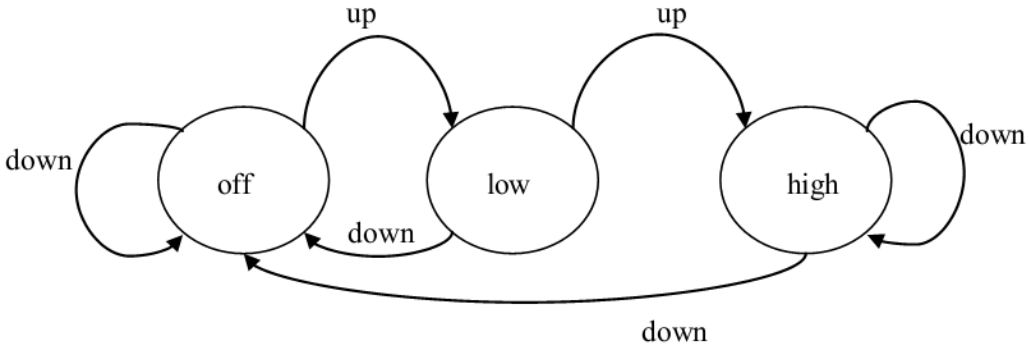
```
T0_S4:
if
        :: ((a0)) -> goto accept_all
        :: ((a1)) -> goto T0_S4
fi;
        accept_all:
        skip
}
```

Sau khi ta *Translate* để dịch tử ngôn ngữ *LTL* sang Promela ta chạy *verify*, kết quả cho thấy không có lỗi vi phạm do vậy hệ thống thỏa mãn tính chất cần kiểm tra.



Hình4.4 Kết quả kiểm tra mô hình hệ thống đèn

4. 2. 2. B KIỂM CHỨNG MÔ HÌNH KHÔNG ĐÁP ỨNG THUỘC TÍNH.



Hình4.5 Mô hình công tắc đèn không đúng

Mô tả các trạng thái của đèn không thỏa mãn đã nêu trong phần 5.2.2.a. Trong mô hình từ trạng thái high,cái đèn có thể chuyển sang trạng thái *off* mà không qua trạng thái *low*.Tệp mô tả trạng thái của đèn là lamp. txt

off:up:low

low:up:high

low:down:off

high:up:high

high:down:off

Sau khi chạy giả lập ở chế độ *random* như ở phần trên ta cần kiểm tra tính chất mà hệ thống cần thỏa mãn bằng việc chạy*verify*.Tính chất cần kiểm chứng trong trường hợp này là:khi đèn ở trạng tháihigh,cần phải qua trạng thái*low*rồi mới có thể về trạng thái *off*.

Tệp lamp1. txt của hệ thống đèn không đáp ứng được thuộc tính:

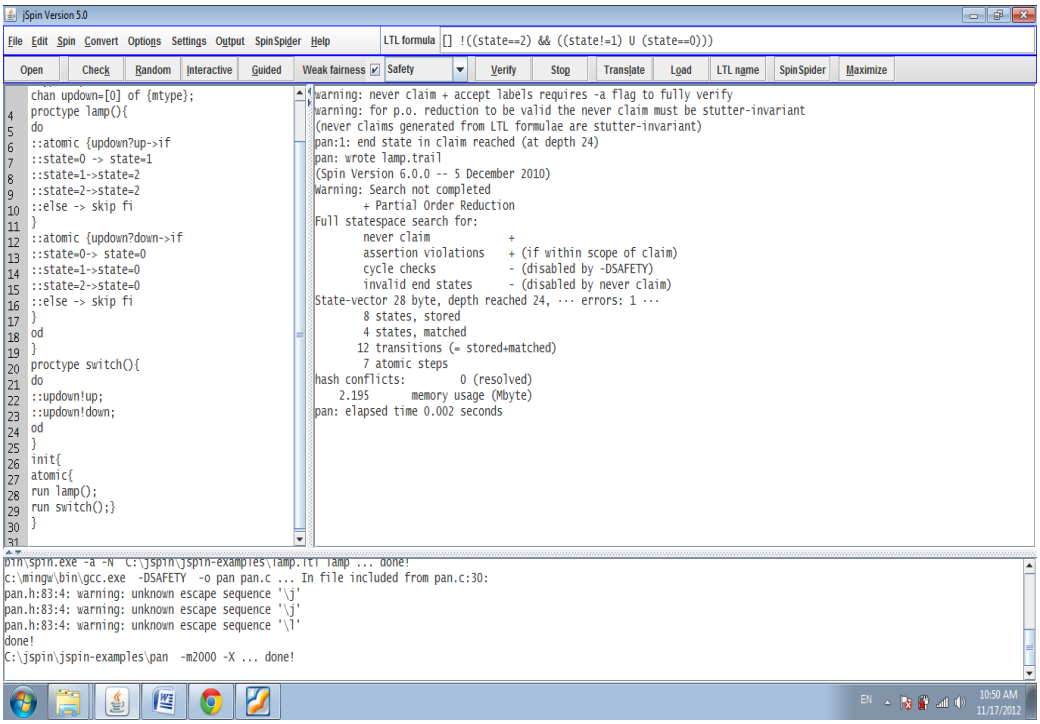
```
proctype lamp(){
do
     ::atomic {updown?up->if
     ::state=0 -> state=1
     ::state=1->state=2
     ::state=2->state=2
     ::else -> skip fi
}
atomic {updown?down->if
     ::state=0-> state=0
     ::state=1->state=0
     ::state=2->state=0
else -> skip fi
```

```
}
od
}
proctype switch(){
do
     ::updown!up;
     ::updown!down;
od
}
init{
atomic{
run lamp();
run switch();}
}
```

Biểu thức *LTL*đưa vào *SPIN:*

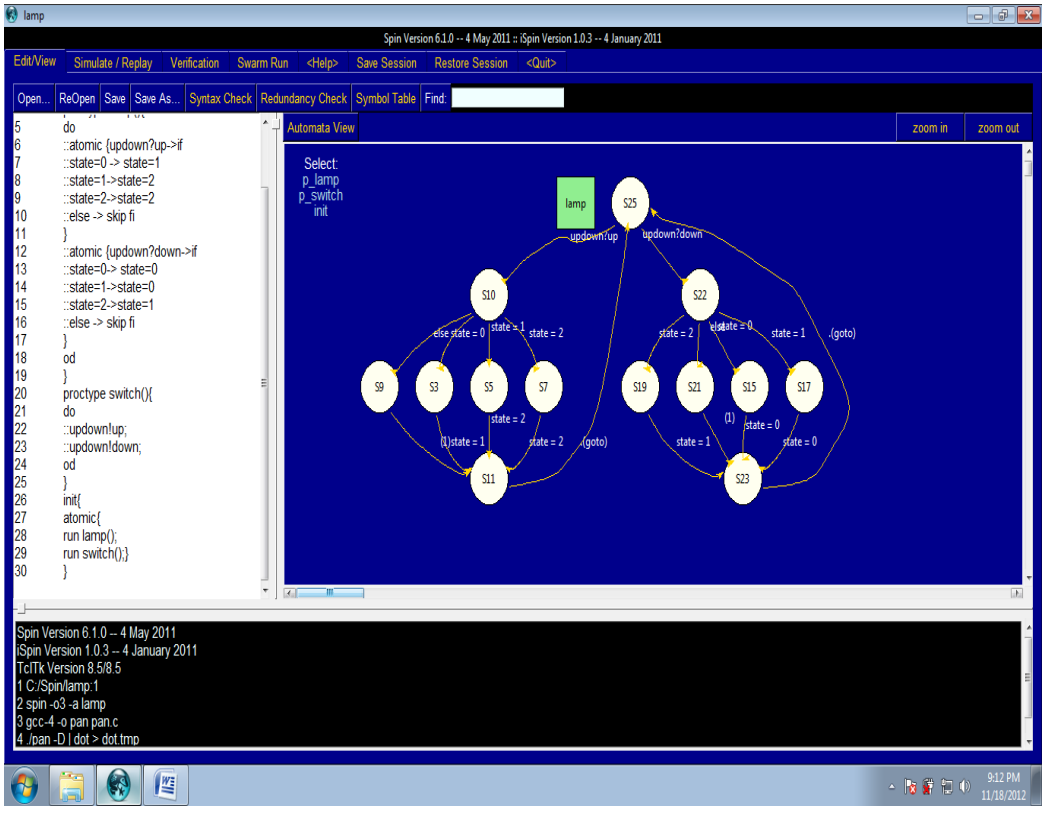
[] !(a2 && (a1 U a0))

Khi đó trong *JSPIN* ta sử dụng chức năng Translate để dịch tử ngôn ngữ *LTL* sang *Promela*.Sau đó ta chạy *verity* thì chương trình sẽ chạy chương trình *promle* của hệ thống đèn và đồng thời so sánhbiểu thức *LTL*,kết quả cho thấy *SPIN* báo lỗi vi phạm,suy ra mô hình đưa vào không thảo mãn tính chất cần kiểm tra.



Hình4.6 Kết quả kiểm chứng mô hình hệ thống đèn không thỏa mãn

4. 2. 3 BẢNG CHUYỂN ATOMATA



Hình4.7 Kết quả bảng chuyển Atomata

KẾT LUẬN

- **Kết quả của khóa luận**

Đồ án tìm hiểu bộ kiểm chứng mô hình *SPIN*, trong đó các khái niệm về kiểm chứng mô hình, các kĩ thuật kiểm chứng. Đồ án đi sâu vào tìm hiểu hệ tương tranh và mô hình hóa hệ thốngbằng Promela đồng thời em tìm hiểu cách kiểm chứng bằng công cụ **JSPIN** và **ISPIN**. Để thực nghiệm các kết quả nghiên cứu trong đồ án em đã xây dựng thực nghiệm về công tắc đèn.

- **Hướng nghiên cứu tiếp theo**

Hướng nghiên cứu tiếp theo của đồ án là tiếp tục tìm hiểu và xây dựng SPIN để đặc tả các hệ thống thời gian thực.

TÀI LIỆU THAM KHẢO

[1] Spin Model Checker The Primer and Reference Manual.

[2] Software reliability methods.

[3]Bài giảng môn Công Nghệ Phần Mềm của thầy Nguyễn Văn Vỵ

[4] http://spinroot.com

[5] Principles_of_model_checking.

[6] Model Cheking của Edmund M Jr Clarke,Orna Grumberg,Doron A. Peled.