ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

PHẠM THỊ TỐ NGA

ĐẶC TẢ VÀ KIỂM CHỨNG CÁC HỆ THỐNG THỜI GIAN THỰC SỬ DỤNG UPPAAL

Ngành: Công nghệ thông tin
Chuyên ngành:Kỹ Thuật Phần Mềm
Mã số: 60480103

TÓM TẮT LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

NGƯỜI HƯỚNG DẪN KHOA HỌC: PGS.TS PHẠM NGỌC HÙNG

Hà Nội – 2017

1. Giới thiệu

1.1 Đặt vấn đề

Trong thời đại ngày nay, các hệ thống có yếu tố thời gian và đặc biệt các hệ thống thời gian thực là một trong những lĩnh vực nhận được rất nhiều sự quan tâm của giới khoa học nói chung và giới khoa học nghiên cứu về công nghệ nói riêng. Thật vây, hệ thống thời gian thực được ứng dụng rất nhiều trong đời sống xã hội, trong sản xuất, trong y tế, trong hàng không vũ trụ và trong quân sự, gần như trong mọi lĩnh vực ta đều thấy có sự góp mặt của những ứng dụng trong hệ thống thời gian thực. Không chỉ góp mặt trong nhiều lĩnh vực mà sự góp mặt của nó còn có tầm quan trọng rất lớn đối với hệ thống. Trong hệ thống thời gian thực, các công vệc và các tác vụ cần phải hoàn thành trong một khoảng thời gian cho phép (deadline), nếu không đáp ứng được yêu cầu thời gian thì hệ thống sẽ sụp đổ hoặc sẽ gây ra hậu quả nghiêm trọng (hệ Hard Real- Time) hoặc sẽ bị suy giảm về chất lượng dịch vụ (hệ Soft Real-Time).

Chính vì tầm quan trọng của yếu tố thời gian trong hệ thống thời gian thực như vậy nên việc kiểm tra tính đúng đắn đối với hệ thống này là rất cần thiết. Tuy nhiên, việc đặc tả và kiểm chứng một hệ thống thời gian thực là một bài toán khó và một trong những mối quan tâm lớn hiện nay là làm thế nào để đặc tả và kiểm chứng tự động cho các hệ thống thời gian thực.

Bộ công cụ Uppaal ra đời phần nào đáp ứng được yêu cầu đó, công cụ này giúp ta có thể kiểm định những hệ thống được mô hình hóa thành những hệ thống automata thời gian với nhứng biến số nguyên, cấu trúc dữ liệu, hàm người dùng và đồng bộ các kênh. Với việc đặc tả và kiểm chứng một hệ thống thời gian thực thì bộ công cụ Uppaal được đánh giá là tốt nhất hiện nay và được sử dụng rộng rãi trong công nghiệp.

1.2 Nội dung nghiên cứu của luận văn

Trong luận văn này tác giả đã tập trung tìm hiểu về bộ công cụ kiểm chứng Uppaal, đi sâu vào tìm hiểu ngôn ngữ đặc tả của Uppaal cũng như tìm hiểu cơ chế kiểm chứng của bộ công cụ này cho các hệ thống thời gian thực. Từ đó tác giả xây dựng một số ví dụ (cụ thể tác giả đã xây dựng được 4 ví dụ) về một số hệ thống thời gian thực áp dụng vào đặc tả và kiểm chứng hệ thống đó bởi bộ công cụ Uppaal.

Đối với mỗi ví dụ tác giả giả định là một hệ thống thời gian thực, tiến hành đặc tả, mô hình hóa dưới hệ ô-tô-mát thời gian trên trình soạn thảo của Uppaal sau đó chạy mô phỏng và kiểm chứng sự hoạt động của hệ thống đó.

1.3 Cấu trúc của luận văn

Luận văn được trình bày thành 4 chương:

Chương 1: Giới thiệu

Chương 2: Đặc tả và kiểm chứng trong Uppaal

Trình bày những hiểu biết của tác giả về bộ công cụ Uppaal cũng như cách đặc đả và kiểm chứng trong Uppaal.

Chương 3: Một số ví dụ áp dụng

Trình bày 4 ví dụ mà tác giả đã xây dựng được về 4 hệ thống thời gian và tiến hành đặc tả và kiểm chứng các hệ thống đó qua công cụ Uppaal.

Chương 4: Kết luận

2. Tìm hiểu về Upppaal

2.1 Bộ công cụ Uppaal

2.1.1 Giới thiệu về bộ công cụ Uppaal

Uppaal là một phần mềm để kiểm tra những hệ thống thời gian thực, được phát triển bởi Đại học Uppsala và Đại học Aalborg. Phần mềm được ứng dụng thành công trong những nghiên cứu ở nhiều lĩnh vực như bộ điều khiển, giao thức truyền thông hay ứng dụng multimedia – những lĩnh vực mà yếu tố thời gian là rất quan trọng. Công cụ này dùng để kiểm định những hệ thống được mô hình hóa thành hệ thống những automat thời gian với những biến số nguyên, cấu trúc dữ liệu, hàm người dùng, và đồng bộ các kênh.

2.1.2 Tổng quan về bộ công cụ Uppaal

UPPAAL sử dụng kiến trúc máy trạm-máy chủ mà trong đó phân chia chương trình ra gồm 2 phần: giao diện đồ họa và hệ thống kiểm tra mô hình. Giao diện đồ họa, hay client, viết bằng Java, và server được đóng gói tùy vào HĐH (Linux, Windows, Solaris). Do có kiến trúc như vậy nên hai phần của CT sẽ kết nối với nhau qua giao thức TCP/IP. Cũng có phiên bản sử dụng dòng lệnh.

2.1.2.1 Java Client

Ý tưởng của chương trình là mô hình hóa hệ thống ô-tô-mát thời gian sử dụng công cụ đồ họa, mô phỏng và kiểm chứng sự hoạt động của nó, và cuối cùng là kiểm tra xem nó có thỏa mãn những tính chất cho trước hay không. GUI của Java client thực hiện ý tưởng này bằng cách chia nó lám 3 phần: Khung soạn thảo (Editor), Mô phỏng (Simulator) và Kiểm chứng (Verifier) được xếp vào các tab.

Khung soạn thảo (KST) hệ thống được định nghĩa là 1 tập hợp các ô-tô-mát thời gian được tiến hành song song gọi là quá trình. Quá trình được tạo ra từ một template định trước. KST chia làm 2 phần: cửa sổ dạng cây để chọn các template, khai báo khác nhau và 1 bản vẽ. Địa điểm được dán tên. Invariant và edge dán điều kiện guard, đồng bộ hay phép gán. Sơ đồ cây bên trái cho phép ta lựa chọn nhiều phần thông tin của hệ thống:

Mô phỏng (Simulator): Mô phỏng việc thực thi hệ thống một cách ngẫu nhiên, (thông qua mô phỏng này, bước đầu ta kiểm tra được hệ thống có vận hành được không, có xung đột gì không)

Kiểm chứng (Verifier): Cho phép ta kiểm chứng tính thực hiện được của hệ thống, tính đến được của các trạng thái trong mô hình, tính đúng đắn của hệ thống, kiểm tra một trạng thái nào đó có bị deadlock không bằng những câu lệnh cụ thể theo ngôn ngữ của Uppaal.

2.1.2.2 Stand-alone Verifier

Khi thực hiện những bài toán lớn, khó có thể làm hết được trong GUI. Khi đó ta có thể dùng kiểm chứng riêng bằng dòng lệnh bằng chương trình verifier. Nó cho phép ta thực hiện chức năng tương tự như GUI bằng các dòng lệnh. Cách này thích hợp cho những máy có cấu hình yếu và phải chia sẻ bộ nhớ cho những ứng dụng khác.

2.2 Ô-tô-mát thời gian trong Uppaal

2.2.1 Định nghĩa ô-tô-mát thời gian

Một ô-tô-mát thời gian (Time automata-TA) là một tập gồm 6 thành phần (L, l_0, C, A, E, I).

Trong đó:

- L: tập các trạng thái
- l_0: trạng thái ban đầu
- C: tập các đồng hồ
- A: tập các hành động
- E $\in L \times A \times B(C) \times 2^C \times L$: tập các cạnh giữa các trạng thái
- I: $L \rightarrow B(C)$: chỉ định bất biến cho các trạng thái.

2.2.2 Mô tả Ô-tô-mát thời gian trong Uppaal

Ô-tô-mát thời gian thực là một máy hữu hạn trạng thái với một tập các đồng hồ. Mỗi đồng hồ là một hàm số ánh xạ vào một tập số thực không âm, nó ghi lại thời gian trôi qua giữa các sự kiện. Các đồng hồ được đồng bộ hóa về mặt thời gian. Trong UPPAAL, một hệ thống đượ c mô hình là mạng củ a những automat thời gian sắp xếp son g song. Mô hình được mở rộng với các biến rời rạc bị chặn ở trạng thái. Những biến này được sử dụng trong ngôn ngữ lập trình: có thể đọc, ghi và thực hiện các phép tính số học. Một trạng thái của hệ thống định nghĩa bởi vị trí của các automat, giá trị đồng hồ và các biến rời rạc. Mỗi automat có thể thay đổi (không nên hiểu là sự chuyển tiếp) độc lập hay đồng bộ với 1 automat khác, dẫn đến 1 trạng thái khác.

2.3 Đặc tả trong Uppaal

Một hệ thống Ô-tô-mát thời gian được đặc tả trong Uppaal thông qua khung soạn thảo và trình bày dưới dạng các ô-tô-mát thời gian xếp song song, có thể hoạt động độc lập hoặc đồng bộ với nhau thông qua các kênh đồng bộ.

Để đặc tả hệ thống ô-tô-mát thời gian trong Uppaal trên khung soạn thảo ta cần tiến hành các bước:

Bước 1:Phân tích và nhận diện các khuân mẫu có trong hệ thống:

Cần xác định trong hệ thống có những khuân mẫu nào, mỗi khuân mẫu sẽ ứng với một quá trình và được biểu diễn là một ô-tô-mát thời gian trong khung soạn thảo.

Bước 2: Mô hình hóa các khuân mẫu

Mỗi khuân mẫu cần xác định rõ:

- Có những trạng thái nào?
- Bước chuyển trạng thái ra sao?
- Có cần truyền tham số gì không? Từ đó xác định các biến toàn cục và biến địa phương trong hệ thống.

Bước 3: Vẽ mô hình:

2.4 Kiểm chứng trong Uppaal

2.4.1 Kiểm chứng qua mô phỏng sự hoạt động của hệ thống

Sau khi biên tập hệ thống trong phần soạn thảo, Uppaal cho phép kiểm chứng hoạt động của hệ thống qua chức năng Simulator.

Ở chức năng này, bước đầu người dùng phát hiện những lỗi trong thiết kế Ô-tô-mát, lỗi mã nguồn. Nếu ở bước Editor gặp các lỗi này khi chạy Simulator máy sẽ báo lỗi và chỉ rõ lỗi mắc phải qua hộp hội thoại hiện trên màn hình cũng như đánh dấu lỗi trên bản Editor bằng chuyển phông chữ sang màu đỏ và gạch chân. Khi đó, người dùng sẽ biết lỗi ở đâu và sửa lỗi. Chỉ đến khi nào phần biên tập không còn các lỗi soạn thảo thì mới chạy được Simulator.

Trong quá trình chạy Simulator người dùng sẽ bước đầu kiểm tra được sự vận hành của hệ thống qua sự dịch chuyển trạng thái được mô phỏng ngẫu nhiên và các trạng thái của các Ô-tô-mát song song theo thời gian.

Ở bước này nếu hệ thống bị deadlock cũng sẽ được phát hiện ra.

2.4.2 Kiểm chứng bằng dòng lệnh

Uppaal cho phép ta kiểm chứng khả năng hoạt động của ô-tô-mát qua mọi trạng thái, tính an toàn, khả năng rơi vào trạng thái deadlock (không chịu chuyển trạng thái) bởi chức năng Verifier.

Chức năng này cho phép kiểm chứng qua các dòng lệnh (ngôn ngữ C++)

- Kiểm chứng tính có thể đạt được (khả năng tới được 1 trạng thái nhất định)

Cú pháp: E<>φ (trong đó φ là công thức trạng thái)ψ

- Kiểm tra tính an toàn (một điều gì đó luôn luôn đúng)

Cú pháp: A[] và E[]

- Kiểm tra tính liveness của hệ thống (tính chất này đảm bảo một điều gì đó trước sau gì cũng xảy ra)

Cú pháp: A<> φ: Chỉ ra rằng φ luôn được thỏa mãn

φ--> ψ : Khi φ thỏa mãn thì ψ cũng thỏa mãn.

- Kiểm tra ô-tô-mát có rơi vào deadlock hay không

Cú pháp A[] not deadlock

3. Một số ví dụ áp dụng

3.1 Hệ thống phân loại

3.1.1 Ví dụ1. Hệ thống phân loại bóng theo màu sắc.

Một hệ thống có đầu vào là các loại bóng với nhiều màu sắc khác nhau (demo là 7 màu), các quả bóng lần lượt cho chạy qua một sensor nhận biết màu sắc, sensor sẽ thông báo tín hiệu cho các cửa đẩy tương ứng. Bóng sẽ chạy trên một băng chuyền với vận tốc đảm bảo để di chuyển từ cửa này đến cửa kế tiếp là một khoảng thời gian cố định và gặp đúng cửa nó sẽ được đẩy ra khỏi băng chuyền.

Đặc tả: *Một quả bóng có màu sắc i khi đi qua sensor sẽ được sensor phát hiện màu sắc, lập tức sensor sẽ phát tín hiệu tới cửa thứ i. quả bóng qua sensor sẽ được chạy trên băng chuyền với tốc độ đảm bảo đến cửa thứ i sẽ mất đúng i*5s. Sau đúng i*5s kể từ khi nhận được tín hiệu từ sensor cửa thứ i sẽ đẩy ra, quả bóng sẽ bị đẩy xuống rãnh tương ứng.*

Phân tích và nhận diện đối tượng trong hệ thống

Hệ thống có 1 đèn cảm ứng nhận diện màu sắc (Sensor) và 7 cửa đẩy (PushDoor(i), i=1 ÷ 7).

Mô hình hóa các đối tượng

Đối tượng Sensor

- Gồm 2 trạng thái: Safe và SeeColor

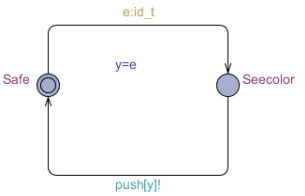
  Safe: trạng thái an toàn, khi không có bóng nào đi qua.

  SeeColor: trạng thái nhìn thấy bóng đi qua.

- Chuyển trạng thái

Khi có một bóng (gán bởi một số thứ tự tương úng với màu sắc) đi qua, sensor phát hiện ra màu và nhớ màu đó (gán y=e) rồi chuyển sang trạng thái Seecolor. Sau đó ngay lập tức gửi tín hiệu đến cửa thứ i (i=y) tương ứng (lệnh push[y]!) và chuyển sang trạng thái Safe.

- Ô-tô-mát Sensor



Đối tượng PushDoor

- Gồm 3 trạng thái: Safedoor; Waiter; PushD.

Safedoor: Trạng thái chưa có yêu cầu báo mở.

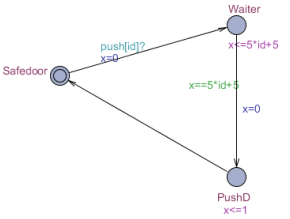
Waiter: Trạng thái có nhận được yêu cầu báo mở nhưng chờ đến thời gian mở.

PushD: đẩy cửa.

- Chuyển trạng thái:

Khi cửa thứ i đang ở trạng thái Safedoor nếu nhận được lệnh có bóng màu thứ i đang đi tới, lập tức chuyển sang trạng thái Waiter và ở trạng thái Waiter đó đúng 5*id+5(s) và lập tức chuyển sang trạng thái PushD. Sau đúng 1(s) thì trở về trạng thái an toàn.

- Ô-tô-mát PushDoor.



- Khai báo các template có trong hệ thống

Mô phỏng sự vận hành của hệ thống

- Mô phỏng sự thay đổi trạng thái của các đối tượng.
- Mô phỏng sự đồng bộ theo thời gian của cá dối tượng

Kiểm chứng hoạt động của hệ thống

- Kiểm chứng tính có thể đạt được (khả năng tới được 1 trạng thái nhất định)

Cú pháp: E<>φ (trong đó φ là công thức trạng thái)ψ

E<> Pushdoor(0).PushD: kiểm tra xem các cửa có đạt được trạng thái PushD hay không

E<>Senor.Seecolor Kiển tra xem Sensor có chuyển sang trạng thái nhìn thấy màu không và có lưu lại màu sắc vừa nhìn không?

E<> Pushdoor(0).PushD and (forall (i:id_t) i!=0 imply Pushdoor(i).Safedoor): Kiểm tra xem khi một cửa đẩy thì các cửa khác có ở trạng thái an toàn không (đảm bảo chỉ có một cửa đẩyr a trong một lúc)

- Kiểm tra tính an toàn (một điều gì đó luôn luôn đúng)

Cú pháp: A[] và E[]

A[] forall(i:id_t) forall(j:id_t) Pushdoor(i).PushD and Pushdoor(j).PushD imply i==j. Tính chất 2 cửa khác nhau không bao giờ cùng đẩy.

- Kiểm tra tính liveness của hệ thống (tính chất này đảm bảo một điều gì đó trước sau gì cũng xảy ra)

Cú pháp: A<> φ: Chỉ ra rằng φ luôn được thỏa mãn

φ-->ψ : Khi φ thỏa mãn thì ψ cũng thỏa mãn.

Pushdoor(0).PushD --> Pushdoor(0).Safedoor. Kiểm tra nếu cửa thứ i đẩy thì cửa thứ i sẽ trở về trạng thái an toàn

- Kiểm tra ô-tô-mát có rơi vào deadlock hay không
  Cú pháp A[] not deadlock

3.1.2 Ví dụ 2. Hệ thống phân loại sản phẩm (sản phẩm đạt chất lượng hay chưa).
Một hệ thống phân loại khoai tây, có đầu vào là các cử khoai tây sau khi thu hoạch, các củ khoai tây đươci cho qua một phễu để chay qua một sensor kiểm tra chất lượng (kích thước, cân nặng, màu sắc), sensor sẽ thông báo tín hiệu cho các cửa đẩy tương ứng. Củ khoai tây sẽ chạy trên một băng chuyền với vận tốc đảm bảo để di chuyển từ cửa này đến cửa kế tiếp là một khoảng thời gian cố định và gặp đúng cửa nó sẽ được đẩy vào đúng rãnh phân loại.

Đặc tả: *Một củ khoai tây khi đi qua sensor sẽ được sensor phát hiện chất lượng thông qua kích thước, cân nặng và màu sắc, Nếu đảm bảo kích thước, cân nặng và màu sắc tốt thì củ khoai đó được xếp hạng A và lập tức sensor sẽ phát tín hiệu tới cửa hạng A, còn lại sẽ xếp hạng B và được báo tín hiệu đến của hạng B. Củ khoai tây sau khi qua sensor sẽ được chạy trên băng chuyền gặp đúng cửa mở nó sẽ rơi xuống rãnh của cửa đó và được phân loại.*

Phân tích và nhận diện đối tượng trong hệ thống

Hệ thống gồm đèn cảm ứng nhận chất lượng (Sensor) và 2 cửa mở (Adoor và BDoor), các củ khoai được xem là đối tượng tham gia trong hệ thống (Potato).

Mô hình hóa các đối tượng

Đối tượng Potato

- Gồm 4 trạng thái Safe, CrossSensor, CrossA, CrossB

Safe: Trạng thái không ở trong đường truyền.

CrossSensor: Đi qua sensor.

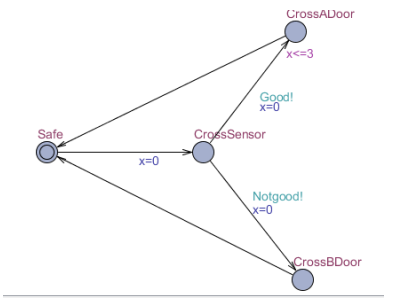
CrossA: Vào rãnh A.

CrossB: Vào rãnh B.

- Chuyển trạng thái

Từ trạng thái safe củ khoai tây được chạy qua phễu và đến vị trí của sensor, bắt đầu kích hoạt đồng hồ, reset về 0. Nếu là củ khoai đạt chất lượng thì báo tín hiệu Good! Và đi đến cửa A, ngược lại, báo tín hiệu Notgood! và đi đến của B. Sau đúng 3 giây thoát khỏi cửa và trở về trạng thái an toàn (đã phân loại).

- Ô-tô-mát Potato



Đối tượng Sensor

- Gồm 2 trạng thái: Free, SeeA và SeeB

  Free: trạng thái an toàn

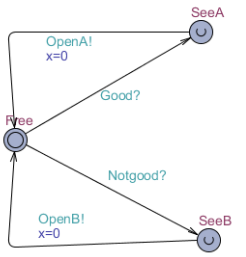
  SeeA: trạng thái nhìn thấy củ khoai đạt chất lượng loại A.

  SeeB: trạng thái nhìn thấy củ khoai đạt chất lượng loại B.

- Chuyển trạng thái

Khi có một củ khoai đi qua, sensor phát hiện ra chất lượng nhờ vào kích thước, cân nặng và màu (được hiểu như củ khoai phát tín hiệu), nếu nhận được tín hiệu Good! Thì chuyển sang SeeA, ngược lại chuyển sang SeeB. Sau đó ngay lập tức gửi tín hiệu đến cửa tương ứng (OpenA! Hoặc OpenB!) và chuyển sang trạng thái Free.

- Ô-tô-mát Sensor



Đối tượng ADoor

- Gồm 2 trạng thái: Close và Open.

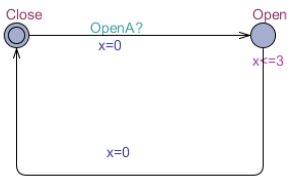
Close: Trạng thái chưa có yêu cầu báo mở.

Open: Mở cửa

- Chuyển trạng thái:

Khi cửa đang ở trạng thái Close nếu nhận được lệnh có củ khoai đạt chất lượng đang đi tới (OpenA?), lập tức reset đồng hồ về 0 và chuyển sang trạng thái Open và ở trạng thái khoảng 3s và sau đó chuyển sang trạng thái Close.

- Ô-tô-mát ADoor.



Đối tượng BDoor

- Gồm 2 trạng thái: Close và Open.

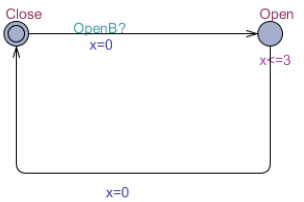
Close: Trạng thái chưa có yêu cầu báo mở.

Open: Mở cửa

- Chuyển trạng thái:

Khi cửa đang ở trạng thái Close nếu nhận được lệnh có củ khoai không đạt chất lượng đang đi tới (OpenB?), lập tức reset đồng hồ về 0 và chuyển sang trạng thái Open và ở trạng thái khoảng 3s và sau đó chuyển sang trạng thái Close.

- Ô-tô-mát BDoor.



Mô phỏng sự vận hành của hệ thống

- Mô phỏng sự thay đổi trạng thái của các đối tượng.
- Mô phỏng sự đồng bộ theo thời gian của cá dối tượng

Kiểm chứng hoạt động của hệ thống

- Kiểm chứng tính có thể đạt được (khả năng tới được 1 trạng thái nhất định)

Cú pháp: E<>φ (trong đó φ là công thức trạng thái)ψ

E<>Potato.CrossSensor kiểm tra xem mọt củ khoai tây có chuyển sang trạng thái CrossSensor được không.

E<>Potato.CrossADoor or Potato.CrossBDoor: kiểm tra xem một củ khoai hoặc là phải ra được cửa A hoặc cửa B.

E<>Senor.Seecolor Kiển tra xem Sensor có chuyển sang trạng thái tây có qua cửa A hay cửa B không.

- Kiểm tra tính an toàn (một điều gì đó luôn luôn đúng)

Cú pháp: A[] và E[]

E[]Potato.CrossADoor and Potato.CrossBDoor đảm bảo một củ khoai tây không bao giờ qua cả hai cửa.

- Kiểm tra tính liveness của hệ thống (tính chất này đảm bảo một điều gì đó trước sau gì cũng xảy ra)

Cú pháp: A<> φ: Chỉ ra rằng φ luôn được thỏa mãn

φ-->ψ : Khi φ thỏa mãn thì ψ cũng thỏa mãn.

Sensor.SeeA-->ADoor.Open: Đảm bảo nếu sensor phát hiện là là củ đạt chất lượng A thì cửa A phải mở.

Sensor.SeeB-->BDoor.Open: Đảm bảo nếu sensor phát hiện là là củ đạt chất lượng B thì cửa B phải mở.

- Kiểm tra ô-tô-mát có rơi vào deadlock hay không

Cú pháp A[] not deadlock

3.2 Hệ thống điều khiển sử dụng vùng tài nguyên

3.2.1 Ví dụ 3. Hệ thống điều khiển việc sử dụng chung vùng tài nguyên (có ràng buộc về thời gian sử dụng nguồn tài nguyên).

Có n quá trình (demo là 2) cùng có nhu cầu sử dụng 1 vùng tài nguyên, hệ thống đảm bảo việc điều khiển sao cho tại một thời điểm chỉ có một quá trình được sử dụng vùng tài nguyên và các quá trình gửi yêu cầu trước sẽ được bố trí sử dụng trước (giả thiết là các quá trình cần sử dụng vùng tài nguyên với thời gian khác nhau).

Đặc tả: Có n quá trình –Process (demo là n=2) đều có nhu cầu sử dụng một nguồn tài nguyên – Resource. Khi một quá trình có nhu cầu sử dụng nó sẽ gửi tín hiệu thông báo cho bộ điều khiển, bộ điều khiển tiếp nhận tín hiệu và tiến hành xử lí tín hiệu đó, nếu trong thời điểm đó nguồn tài nguyên đang rảnh nó sẽ báo lại tín hiệu cho quá trình được phép sử dụng nguồn tài nguyên, nếu hiện đang có quá trình đang sử dụng nó, bộ điều khiển sẽ lưu thứ tự các quá trình có nhu cầu vào một hàng đợi và đến khi nguồn tài nguyên rảnh nó sẽ gọi quá trình đầu tiên ra sử dụng trước. Khi quá trình sử dụng xong vùng tài nguyên (mặc định là sau một khoảng thời gian cho trước tương ứng với quá trình đó) nó sẽ báo lại cho bộ điều khiển biết và rời khỏi cùng tài nguyên.

Yêu cầu: Quá trình nào có nhu cầu đều được bố trí sử dụng nguồn tài nguyên, không có sự xung đột, đảm bảo tại một thời điểm chỉ có một quá trình được sử dụng.

Phân tích và nhận diện đối tượng trong hệ thống

Hệ thống có 2 quá trình (Process1 và Process2) và 1 nguồn tài nguyên (Resource). Các Process1 và Process2 hoạt động song song, và được đồng bộ với Resource thông qua các kênh: báo tín hiệu yêu cầu sử dụng (appr[i]), dừng (stop[i]), được phép sử dụng(go[i]) và rời khỏi vùng tài nguyên (leave[i]).

Mô hình hóa các đối tượng

Đối tượng Process1

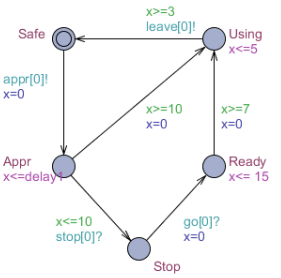
Được truyền tham số là thời gian mà quá trình này sẽ sử dụng vùng tài nguyên (delay1), Biến địa phương là đồng hồ x

- Gồm 5 trạng thái: Safe, Appr, Stop, Ready, Using
  Safe: Trạng thái chưa có nhu cầu sử dụng nguồn tài nguyên.
  Appr: Đăng kí sử dụng nguồn tài nguyên
  Stop: Chờ đến lượt sử dụng
  Using: sử dụng nguồn tài nguyên
- Chuyển trạng thái

Quá trình 1 ở trạng thái Safe, nếu có nhu cầu sử dụng nguồn tài nguyên nó sẽ gửi tín hiệu (appr[0]!) đến bộ điều khiển và chuyển sang trạng thái Appr, tại đây đồng hồ x sẽ được giới hạn trong 1 khoảng thời gian delay1 (là thời gian mà quá trình này sẽ dùng để sử dụng vùng tài nguyên +10s chờ tín hiệu). Nếu trong vòng 10s nó nhận được tín hiệu yêu cầu dừng (stop[0]?) từ bộ điều khiển thì lập tức chuyển sang trạng

thái Stop, và chờ ở đó đến khi nhận được tín hiệu cho phép sử dụng (go[0]?) thì chuyển sang trạng thái Ready và ở đó sau 7s sẽ được sử dụng vùng tài nguyên và chuyển sang trạng thái Using. Nếu sau 10s mà không thấy có tín hiệu (stop[0]) thì nó chuyển sang trạng thái được sử dụng Using. Ở trạng thái Using đúng 5s (thời gian để ra khỏi vùng tài nguyên) thì báo cho bộ điều khiển tín hiệu đã sử dụng xong (leave[0]!) thì chuyển sang trạng thái Safe.

- Ô-tô-mát Process1.



Đối tượng Process2

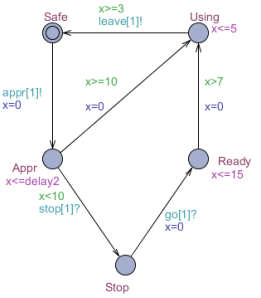
Được truyền tham số là thời gian mà quá trình này sẽ sử dụng vùng tài nguyên (delay2), Biến địa phương là đồng hồ x

- Gồm 5 trạng thái: Safe, Appr, Stop, Ready, Using
  Safe: Trạng thái chưa có nhu cầu sử dụng nguồn tài nguyên.
  Appr: Đăng kí sử dụng nguồn tài nguyên
  Stop: Chờ đến lượt sử dụng
  Using: sử dụng nguồn tài nguyên
- Chuyển trạng thái

Quá trình 1 ở trạng thái Safe, nếu có nhu cầu sử dụng nguồn tài nguyên nó sẽ gửi tín hiệu (appr[1]!) đến bộ điều khiển và chuyển sang trạng thái Appr, tại đây đồng hồ x sẽ được giới hạn trong 1 khoảng thời gian delay1 (là thời gian mà quá trình này sẽ dùng để sử dụng vùng tài nguyên +10s chờ tín hiệu). Nếu trong vòng 10s nó nhận được tín hiệu yêu cầu dừng (stop[1]?) từ bộ điều khiển thì lập tức chuyển sang trạng thái Stop, và chờ ở đó đến khi nhận được tín hiệu cho phép sử dụng (go[1]?) thì chuyển sang trạng thái Ready và ở đó sau 7s sẽ được sử dụng vùng tài nguyên và chuyển sang trạng thái Using. Nếu sau 10s mà không thấy có tín hiệu (stop[1]) thì nó chuyển sang trạng thái được sử dụng Using. Ở trạng thái Using đúng 5s (thời gian để

ra khỏi vùng tài nguyên) thì báo cho bộ điều khiển tín hiệu đã sử dụng xong (leave[1]!) thì chuyển sang trạng thái Safe.

- Ô-tô-mát Process1.

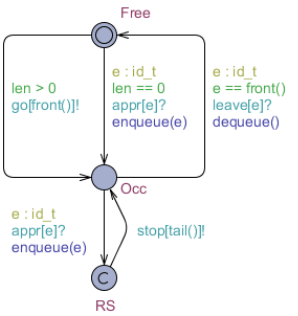


Đối tượng Resource

- Khuân mẫu này gồm có 3 trạng thái Free, Occ, RS
  Free: trạng thái nguồn tài nguyên rảnh.
  Occ: trạng thái tiếp nhận thông tin đăng kí xếp hàng
  RS: Xếp hàng cho các quá trình.
- Chuyển trạng thái

Nếu nguồn tài nguyên đang ở trạng thái rảnh mà số quá trình đang có nhu cầu xử dụng trong hàng đợi >0 thì gọi ra quá trình đầu tiên cho xử dụng trước. Nếu không có quá trình nào trong hàng đợi, đồng thời nhận được tín hiệu báo có nhu cầu sử dụng thì xếp nó vào hàng đợi và chuyển sang trạng thái Occ. Từ trạng thái Occ nếu vẫn tiếp tục nhận được tín hiệu có nhu cầu sử dụng thì xếp vào hàng đợi chuyển qua trạng thái RS- Trạng thái Commited không cho phép trễ, đồng thời gửi tín hiệu stop! cho quá trình đó rồi lập tức trở về trạng thái Occ. Từ trạng thái Occ nếu nhận được tín hiệu leave[i]? lập tức xóa quá trình đó khỏi hàng đợi và chuyển về trạng thái Free.

- Ô-tô-mát Resource.



- Khai báo các quá trình và các hằng được sử dụng trong hệ thống

Mô phỏng sự vận hành của hệ thống

- Mô phỏng sự thay đổi trạng thái của các đối tượng.
- Mô phỏng sự đồng bộ theo thời gian của các đối tượng

Kiểm chứng hoạt động của hệ thống

- Kiểm chứng tính có thể đạt được (khả năng tới được 1 trạng thái nhất định)

Cú pháp: E<>φ (trong đó φ là công thức trạng thái)ψ

E<> P1.Using kiểm tra xem một quá trình có chuyển sang trạng thái Using được không.

E<> Resource.Occ: kiểm tra xem một quá trình có chuyển sang trạng thái Occ được không?

- Kiểm tra tính an toàn (một điều gì đó luôn luôn đúng)

Cú pháp: A[] và E[]

A[]P1.Using && P2.Using: đảm bảo tại một thời điểm chỉ có nhiều nhất một quá trình được sử dụng vùng tài nguyên (tính không xung đột).

A[] Resource.list[N] == 0 đảm bảo không có quá n quá trình trong hàng đợi.

- Kiểm tra tính liveness của hệ thống (tính chất này đảm bảo một điều gì đó trước sau gì cũng xảy ra)

Cú pháp: A<> φ: Chỉ ra rằng φ luôn được thỏa mãn

φ-->ψ : Khi φ thỏa mãn thì ψ cũng thỏa mãn.

P1.Appr --> P1.Using: Đảm bảo một quá trình 1 khi có nhu cầu sử dụng thì sẽ được sử dụng.

P2.Appr --> P2.Using

E<> P1.Using and P2.Using: Đảm bảo 2 quá trình khác nhau sẽ không cùng được sử dụng.

E<> P1.Using and P2.Stop:

E<> P2.Using and P1.Stop: đảm bảo một quá trình đang sử dụng thì tất cả các quá trình khác đều trong trạng thái phải chờ.

- Kiểm tra ô-tô-mát có rơi vào deadlock hay không

Cú pháp A[] not deadlock

3.2.2 Ví dụ 4. Hệ thống điều khiển việc sử dụng chung vùng tài nguyên (có nhiều nhóm quá trình có ràng buộc về thời gian sử dụng nguồn tài nguyên).

Có n quá trình (demo là 4) cùng có nhu cầu sử dụng 1 vùng tài nguyên, trong đó có nhiều nhóm quá trình muốn sử dụng vùng tài nguyên với thời gian khác nhau (demo là 2 nhóm: Nhóm 1 có n1 quá trình muốn dùng vùng tài nguyên trong thời gian delay1, nhóm 2 có n2 quá trình muốn sử dụng vùng tài nguyên với thời gian delay2), hệ thống đảm bảo việc điều khiển sao cho tại một thời điểm chỉ có một quá trình được sử dụng vùng tài nguyên và các quá trình gửi yêu cầu trước sẽ được bố trí sử dụng trước.

Đặc tả: Có 2 nhóm quá trình –Process1 và Process2 (với các mã quá trình được đánh số theo cách: nhóm 1 từ 0 đến n1-1; nhóm 2 từ n1 đến n2-1) đều có nhu cầu sử dụng một nguồn tài nguyên – Resource. Khi một quá trình có nhu cầu sử dụng nó sẽ gửi tín hiệu thông báo cho bộ điều khiển, bộ điều khiển tiếp nhận tín hiệu và tiến hành xử lí tín hiệu đó, nếu trong thời điểm đó nguồn tài nguyên đang rảnh nó sẽ báo lại tín hiệu cho quá trình được phép sử dụng nguồn tài nguyên, nếu hiện đang có quá trình đang sử dụng nó, bộ điều khiển sẽ lưu thứ tự các quá trình có nhu cầu vào một hàng đợi và đến khi nguồn tài nguyên rảnh nó sẽ gọi quá trình đầu tiên ra sử dụng trước. Khi quá trình sử dụng xong vùng tài nguyên (mặc định là sau một khoảng thời gian cho trước tương ứng với quá trình đó) nó sẽ báo lại cho bộ điều khiển biết và rời khỏi cùng tài nguyên.

Yêu cầu: Quá trình nào có nhu cầu đều được bố trí sử dụng nguồn tài nguyên, không có sự xung đột, đảm bảo tại một thời điểm chỉ có một quá trình được sử dụng.

Phân tích và nhận diện đối tượng trong hệ thống

Hệ thống có 2 nhóm quá trình Process1 và Process2 (trong đó nhóm 1 có n1 quá trình, nhóm 2 có n2 quá trrình) và 1 nguồn tài nguyên (Resource). Các Process1 và Process2 hoạt động song song, và được đồng bộ với Resource thông qua các kênh: báo tín hiệu yêu cầu sử dụng (appr1[i] hoặc appr2[i]), dừng (stop1[i] hoặc stop2[i]), được phép sử dụng(go1[i] hoặc go2[i]) và rời khỏi vùng tài nguyên (leave[i]).

Mô hình hóa các đối tượng

Khai báo biến toàn cục

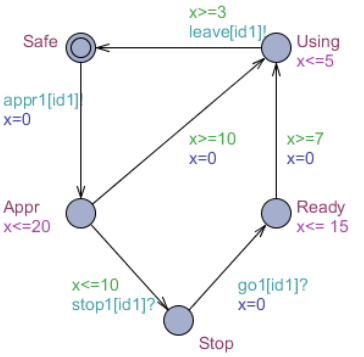
Đối tượng Process1

Được truyền tham số là mã quá trình (đánh số từ 0 đến n1-1), Biến địa phương là đồng hồ x

- Gồm 5 trạng thái: Safe, Appr, Stop, Ready, Using
  Safe: Trạng thái chưa có nhu cầu sử dụng nguồn tài nguyên.
  Appr: Đăng kí sử dụng nguồn tài nguyên
  Stop: Chờ đến lượt sử dụng
  Using: sử dụng nguồn tài nguyên
- Chuyển trạng thái

Quá trình 1 ở trạng thái Safe, nếu có nhu cầu sử dụng nguồn tài nguyên nó sẽ gửi tín hiệu (appr1[i]!) đến bộ điều khiển và chuyển sang trạng thái Appr, tại đây đồng hồ x sẽ được giới hạn trong 1 khoảng thời gian delay1(demo là 20) (là thời gian mà quá trình này sẽ dùng để sử dụng vùng tài nguyên +10s chờ tín hiệu). Nếu trong vòng 10s nó nhận được tín hiệu yêu cầu dừng (stop1[i]?) từ bộ điều khiển thì lập tức chuyển sang trạng thái Stop, và chờ ở đó đến khi nhận được tín hiệu cho phép sử dụng (go1[i]?) thì chuyển sang trạng thái Ready và ở đó sau 7s sẽ được sử dụng vùng tài nguyên và chuyển sang trạng thái Using. Nếu sau 10s mà không thấy có tín hiệu (stop1[i]) thì nó chuyển sang trạng thái được sử dụng Using. Ở trạng thái Using đúng 5s (thời gian để ra khỏi vùng tài nguyên) thì báo cho bộ điều khiển tín hiệu đã sử dụng xong (leave[i]!) thì chuyển sang trạng thái Safe.

- Ô-tô-mát Process1.



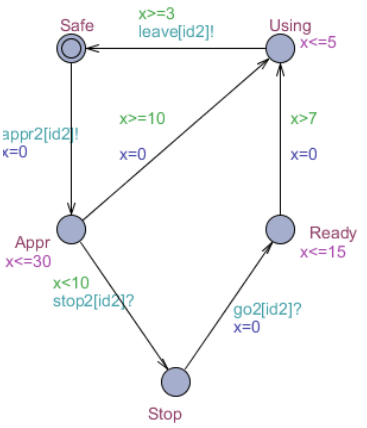
Đối tượng Process2

Được truyền tham số là mã quá trình (được đánh số thứ tự từ n1 đến n2-1), Biến địa phương là đồng hồ x

- Gồm 5 trạng thái: Safe, Appr, Stop, Ready, Using
  Safe: Trạng thái chưa có nhu cầu sử dụng nguồn tài nguyên.
  Appr: Đăng kí sử dụng nguồn tài nguyên
  Stop: Chờ đến lượt sử dụng
  Using: sử dụng nguồn tài nguyên
- Chuyển trạng thái

Quá trình 1 ở trạng thái Safe, nếu có nhu cầu sử dụng nguồn tài nguyên nó sẽ gửi tín hiệu (appr2[i]!) đến bộ điều khiển và chuyển sang trạng thái Appr, tại đây đồng hồ x sẽ được giới hạn trong 1 khoảng thời gian delay1 (demo là 30s, là thời gian mà quá trình này sẽ dùng để sử dụng vùng tài nguyên +10s chờ tín hiệu). Nếu trong vòng 10s nó nhận được tín hiệu yêu cầu dừng (stop2[i]?) từ bộ điều khiển thì lập tức chuyển sang trạng thái Stop, và chờ ở đó đến khi nhận được tín hiệu cho phép sử dụng (go2[i]?) thì chuyển sang trạng thái Ready và ở đó sau 7s sẽ được sử dụng vùng tài nguyên và chuyển sang trạng thái Using. Nếu sau 10s mà không thấy có tín hiệu (stop2[i]) thì nó chuyển sang trạng thái được sử dụng Using. Ở trạng thái Using đúng 5s (thời gian để ra khỏi vùng tài nguyên) thì báo cho bộ điều khiển tín hiệu đã sử dụng xong (leave[i]!) thì chuyển sang trạng thái Safe.

- Ô-tô-mát Process2.



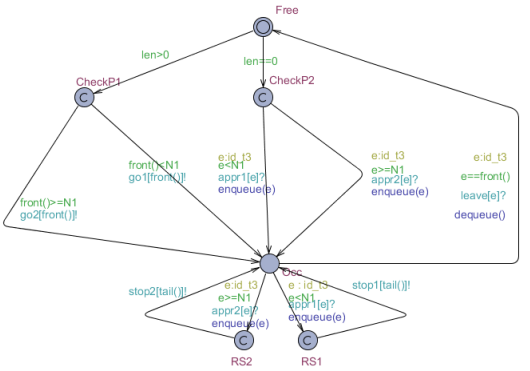
Đối tượng Resource

Khai báo biến và hàm sử dụng trong khuân mẫu

- Khuân mẫu này gồm có 6 trạng thái Free, CheckP1, CheckP2, Occ, RS1, RS2
  Free: trạng thái nguồn tài nguyên rảnh.
  CheckP1, CheckP2: trạng thái kiểm tra xem quá trình xếp hàng đầu tiên là thuộc nhóm nào.
  Occ: trạng thái tiếp nhận thông tin đăng kí xếp hàng
  RS1, RS2: Xếp hàng cho các quá trình thuộc nhóm 1 và nhóm 2.
- Chuyển trạng thái

Nếu nguồn tài nguyên đang ở trạng thái rảnh mà số quá trình đang có nhu cầu xử dụng trong hàng đợi >0 thì chuyển sang trạng thái kiểm tra xem quá trình đầu tiên thuộc nhóm 1 hay nhóm 2, rồi gọi ra quá trình đầu tiên đó cho sử dụng trước và chuyển sang trạng thái Occ. Nếu không có quá trình nào trong hàng đợi thì chuyển sang trạng thái kiểm tra xem tín hiệu báo nhu cầu sử dụng là thuộc nhóm 1 hay nhóm 2, đồng thời nhận được tín hiệu báo có nhu cầu sử dụng thì xếp nó vào hàng đợi và chuyển sang trạng thái Occ. Từ trạng thái Occ nếu vẫn tiếp tục nhận được tín hiệu có nhu cầu sử dụng thì xếp vào hàng đợi chuyển qua trạng thái RS1- nếu là thuộc nhóm 1 hoặc RS2-nếu thuộc nhóm 2, đồng thời đây là trạng thái Commited không cho phép trễ, đồng thời gửi tín hiệu stop1[i]! - nếu thuộc nhóm 1 hoặc stop2[i]! - nếu thuộc nhóm 2 rồi lập tức trở về trạng thái Occ. Từ trạng thái Occ nếu nhận được tín hiệu leave[i]? lập tức xóa quá trình đó khỏi hàng đợi và chuyển về trạng thái Free.

- Ô-tô-mát Resource.



- Khai báo các quá trình và các hằng được sử dụng trong hệ thống

Mô phỏng sự vận hành của hệ thống

- Mô phỏng sự thay đổi trạng thái của các đối tượng.
- Mô phỏng sự đồng bộ theo thời gian của các đối tượng

Kiểm chứng hoạt động của hệ thống

- Kiểm chứng tính có thể đạt được (khả năng tới được 1 trạng thái nhất định)

Cú pháp: E<>φ (trong đó φ là công thức trạng thái)ψ

E<>Process1(0).Using kiểm tra xem một quá trình có chuyển sang trạng thái Using được không.

E<>Process1(2).Using.

E<> Resource.Occ: kiểm tra xem một quá trình có chuyển sang trạng thái Occ được không?

- Kiểm tra tính an toàn (một điều gì đó luôn luôn đúng)

Cú pháp: A[] và E[]

A[]Process1(0).Using && Process2(2).Using: đảm bảo tại một thời điểm chỉ có nhiều nhất một quá trình được sử dụng vùng tài nguyên (tính không xung đột).

A[] Resource.list[N1+N2] == 0 đảm bảo không có quá n quá trình trong hàng đợi.

- Kiểm tra tính liveness của hệ thống (tính chất này đảm bảo một điều gì đó trước sau gì cũng xảy ra)

Cú pháp: A<> φ: Chỉ ra rằng φ luôn được thỏa mãn

φ--> ψ : Khi φ thỏa mãn thì ψ cũng thỏa mãn.

Process1(0).Appr --> Process1(0).Using: Đảm bảo một quá trình 1 khi có nhu cầu sử dụng thì sẽ được sử dụng.

Process2(2).Appr --> Process2(2).Using

E<>Process1(0).Using imply Process2(3).Stop: đảm bảo một quá trình đang sử dụng thì tất cả các quá trình khác đều trong trạng thái phải chờ.

E<>Process1(0).Using and Process2(2).Using: Đảm bảo 2 quá trình khác nhau sẽ không cùng được sử dụng.

- Kiểm tra ô-tô-mát có rơi vào deadlock hay không

Cú pháp A[] not deadlock

4. KẾT LUẬN

Ngày nay, với sự phát triển ngày càng mạnh trong khoa học, kỹ thuật, quân sự và y tế. Các hệ thống có yếu tố thời gian ngày càng trở nên phổ biến và khẳng định vị trí quan trọng của nó trong mọi lĩnh vực của đời sống xã hội. Việc đặc tả và kiểm chứng một hệ thống thời gian thực trở nên cấp thiết hơn bao giờ hết và việc đưa công cụ để đặc tả và kiểm chứng tự động cho hệ thống thời gian thực là một xu thế tất yếu phù hợp với sự phát triển vũ bão của khoa học công nghệ.

Công cụ kiểm chứng Uppaal với giao diện thân thiện, khả năng kiểm chứng tối ưu dựa trên cơ sở mô phỏng sự vận hành của hệ thống theo thời gian cũng như kiểm chứng được các đặc tính quan trọng của hệ thống như tính an toàn, khả năng đến được, tính not deadlock thông qua các dòng lệnh đơn giản đã khiến Uppaal trở thành một công cụ kiểm chứng tốt nhất hiện nay đối với các hệ thống có yếu tố thời gian.

Việc nắm bắt và sử dụng được một công cụ tốt như Uppaal có ý nghĩa rất quan trọng, hơn nữa việc xây dựng nên các hệ thống đặc trưng có giá trị ứng dụng thực tế, đặc tả và kiểm chứng nó trên công cụ Uppaal là một nhiệm vụ rất cần thiết.

Tác giả đã nghiên cứu và xây dựng được 4 ví dụ về hệ thống thời gian giả định có tính ứng dụng trong thực tế (hệ thống tự động phân loại sản phẩm và hệ thống điều khiển việc sử dụng chung nguồn tài nguyên),vận dụng đặc tả và kiểm chứng các hệ thống đó bằng công cụ Uppaal. Đây là những bước đầu, các ví dụ về các hệ thống còn khá đơn giản. Hơn nữa điểm hạn chế của công cụ này là phải mô phỏng được cả hệ thống thời gian thành hệ ô-tô-mát thời gian (một nhiệm vụ không phải là dễ dàng đối với người sử dụng).

Trong tương lai, tác giả mong muốn sẽ mở rộng ra các hệ thống phức tạp hơn với các ràng buộc thời gian chặt chẽ hơn, đồng thời tìm hiểu để có thể tự động hóa nhiều hơn ngay từ khâu đặc tả.