|  |
| --- |
| ĐẠI HỌC QUỐC GIA HÀ NỘI  **TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**  **LÊ VĂN HÙNG**  **PHƯƠNG PHÁP CHUYỂN ĐỔI QUA LẠI GIỮA CÁC ĐẶC TẢ HÌNH THỨC CHO CÁC HỆ CHUYỂN TRẠNG THÁI**  Ngành: Công nghệ thông tin  Chuyên ngành: Công nghệ phần mềm  Mã Số: 604810  LUẬN VĂN THẠC SĨ  **NGƯỜI HƯỚNG DẪN KHOA HỌC: PGS.TS. PHẠM NGỌC HÙNG**  Hà nội – 2016 |

# MỤC LỤC

# LỜI CẢM ƠN

Trước tiên tôi xin gửi lời cảm ơn chân thành và sâu sắc đến PGS.TS. Phạm Ngọc Hùng – thầy giáo và anh Trần Hoàng Việt – NCS K22, người đã tận tình hướng dẫn, khuyến khích, chỉ bảo và tạo cho tôi những điều kiện tốt nhất từ khi bắt đầu nghiên cứu đề tài đến khi hoàn thành luận văn này.

Tôi xin chân thành cảm ơn các thầy cô giáo khoa Công nghệ thông tin, trường Đại học Công nghệ, Đại học Quốc Gia Hà Nội đã tận tình đào tạo, cung cấp cho tôi những kiến thức vô cùng quý giá, đã tạo điều kiện tốt nhất cho tôi trong suốt quá trình học tập, nghiên cứu tại trường.

Đồng thời tôi xin chân thành cảm ơn những người thân trong gia đình cùng toàn thể bạn bè đã luôn giúp đỡ, động viên tôi trong những lúc gặp phải khó khăn trong việc học tập và nghiên cứu.

# LỜI CAM ĐOAN

Tôi xin cam đoan rằng luận văn thạc sĩ công nghệ thông tin “Phương pháp chuyển đổi qua lại giữa các đặc tả hình thức cho các hệ chuyển trạng thái” là công trình nghiên cứu của riêng tôi, không sao chép lại của người khác. Trong toàn bộ nội dung của luận văn, những điều đã được trình bày hoặc là của chính cá nhân tôi hoặc là được tổng hợp từ nhiều nguồn tài liệu. Tất cả các nguồn tài liệu tham khảo đều có xuất xứ rõ ràng và hợp pháp.

Tôi xin hoàn toàn chịu trách nhiệm và chịu mọi hình thức kỷ luật theo quy định cho lời cam đoan này.

Hà Nội, ngày 26 tháng 09 năm 2016

Lê Văn Hùng

# DANH MỤC THUẬT NGỮ VIẾT TẮT

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Từ viết tắt** | **Từ đầy đủ** | **Ý nghĩa** |
| 1 | LTS | Labeled Transition System. | Hệ thống chuyển trạng thái được gán nhãn. |
| 2 | DFA | Deterministic Finite Automata | Ôtômát hữu hạn đơn định. |

# DANH MỤC HÌNH VẼ

[Hình 2.1: Một hệ chuyển trạng thái được gán nhãn 3](#_Toc464489352)

[Hình 2.2: Minh họa vết của LTS. 4](#_Toc464489353)

[Hình 2.3: Ví dụ về một LTS không đơn định 5](#_Toc464489354)

[Hình 2.4: LTS M1 7](#_Toc464489355)

[Hình 2.5: LTS M2 7](#_Toc464489356)

[Hình 2.6: LTS M1||M2 sau khi ghép nôi 8](#_Toc464489357)

[Hình 2.7: Minh họa một thuộc tính an toàn p và một thuộc tính lỗi tương ứng perr 9](#_Toc464489358)

[Hình 2.8: Mô hình ghép nối M1||Mq||perr 10](#_Toc464489359)

[Hình 2.9: Ví dụ về một LTS. 16](#_Toc464489360)

[Hình 3.2:Mô hình sự tương tác giữa L\* và Teacher 19](#_Toc464489361)

[Hình 3.3: Xây dựng một ứng viên từ bảng quan sát *đóng* và *nhất quán*. 20](#_Toc464489362)

[Hình 3.5: Mô hình sinh giả định dựa trên thuật toán học L\* 23](#_Toc464489363)

[Hình 3.1: LTS Input (M1) 25](#_Toc464489364)

[Hình 3.2: LTS Order 26](#_Toc464489365)

[Hình 3.3: LTS Output (M2) 26](#_Toc464489366)

[Hình 5.1: Ví dụ về file text đầu vào của công cụ chuyển đổi 47](#_Toc464489367)

[Hình 5.2: Ví dụ file text đầu ra của công cụ chuyển đổi 48](#_Toc464489368)

# DANH MỤC BẢNG

[Bảng 2.1: Thành phần Q1 → X1 trong bảng ánh xạ 20](#_Toc463906431)

[Bảng 2.2: Thành phần Q2 → X2 trong bảng ánh xạ 20](#_Toc463906432)

[Bảng 2.3: Bảng 2.3: Thành phần Σ → E trong bảng ánh xạ 20](#_Toc463906433)

[Bảng 2.4: Thành phần δ(q, e, q’) → τ (υ, γ,υ’) trong bảng ánh xạ 21](#_Toc463906434)

# 

# Chương 1: Giới thiệu

Các hệ thống phần mềm đang trở nên phức tạp và cung cấp nhiều chứng năng hơn [1]. Để có thể phát triển các hệ thống như vậy với chi phí hiệu quả, các nhà cung cấp thường hay sử dụng các công nghệ dựa trên thành phần thay vì phát triển tất cả các phần của hệ thống ngay từ đầu. Mục đích của việc sử dụng các thành phần ngay từ đầu là để giảm chi phí phát triển, nhưng sau đó điều quan trọng hơn là giảm thời gian đưa sản phẩm ra thị trường. Các thành phần phần mềm có thể tái sử dụng được từ các thành phần có sẵn hoặc được mua từ một bên thứ ba hoặc cũng có thể sử dụng mã nguồn mở. Điều quan trọng chính là làm thể nào để kiểm chứng được thành phần mà chúng ta sử dụng thỏa mãn được các tính chất của hệ thống.

Phương pháp kiểm chứng giả định–đảm bảo có thể giải quyết được vấn đề này. Phương pháp này không chỉ phù hợp với các phần mềm dựa trên thành phần mà còn giải quyết được bài toán bùng nổ không gian trạng thái [4]. Ý tưởng của phương pháp này là sinh các giả định được xem như môi trường cần thiết để các thành phần thỏa mãn một thuộc tính nào đó. Nếu giả định tồn tại thì hệ thống thỏa mãn tính cần kiểm chứng, ngược lại hệ thống không thỏa mãn. Dựa trên tư tưởng của phương pháp kiểm chứng giả định-đảm bảo chúng ta có một số phương pháp kiểm chứng đó là: Phương pháp kiểm chứng sử dụng đặc tả bằng hệ chuyển trạng thái gắn nhãn (Lable Transition System - LTS) và phương pháp kiểm chứng sử dụng đặc tả bằng hàm lôgic. Với phương pháp kiểm chứng sử dụng đặc tả bằng LTS, chúng ta sẽ sử dụng thuật toán học L\* để sinh giả định, nếu tồn tại giả định thì hệ thống thỏa mãn thuộc tính cần kiểm chứng p, ngược lại hệ thống không thỏa mãn thuộc tính p. Ưu điểm của phương pháp này là giả định được sinh ra trực quan với các điều kiện hệ thống cần kiểm chứng và các thuộc tính của nó được đặc tả bởi các LTS. Tuy nhiên độ phức tạp của phương pháp này vẫn còn rất lớn. Với phương pháp kiểm chứng sử dụng đặc tả bằng hàm lôgic, chúng ta sẽ sử dụng thuật toán CDNF để sinh giả định, thời gian sinh giả định nhanh hơn so với sử dụng thuật toán học L\*. Tuy nhiên, yêu cầu của thuật toán này là hệ thống phải được đặc tả bởi các hàm lôgic. Chính vì thế xuất hiện một nhu cầu là cần kết hợp các ưu điểm và khắc phục nhược điểm của hai phương pháp này. Để có thể tận dụng được các ưu điểm của phương pháp kiểm chứng sử dụng đặc tả bằng hàm lôgic, ta cần chuyển đổi từ dạng đặc tả sử dụng LTS sang dạng đặc tả sử dụng hàm lôgic và để có thể tận dụng được ưu điểm của phương pháp kiểm chứng sử dụng đặc tả bằng LTS, ta cần chuyển đổi từ dạng đặc tả sử dụng hàm lôgic sang dạng đặc tả sử dụng LTS. Đó là lí do em lựa chọn đề tài “Phương pháp chuyển đổi qua lại giữa các đặc tả hình thức cho các hệ chuyển trạng thái”.

Nội dung của luận văn này được trình trong sáu chương. Chương 1 sẽ trình bày về bài toán tổng quan, bao gồm ngữ cảnh của bài toán, lý do chọn đề tài này. Chương 2 là nội dung kiến thức cơ bản bao gồm các khái niệm và định nghĩa cho các khái niệm được sử dụng trong luận văn. Nội dung chương 3 sẽ trình bày về các phương pháp kiểm chứng cho các thành phần phần mềm, bao gồm phương pháp kiểm chứng sử dụng đặc tả bằng LTS và phương pháp kiểm chứng sử dụng đặc tả bằng hàm lôgic, nội dung chính trong chương này sẽ trình bày các phương pháp sinh giả định. Chương 4 tập trung vào việc trình bày phương pháp chuyển đổi qua lại giữa hai dạng đặc tả hình thức trong phát triển phần mềm: Dạng đặc tả sử dụng hệ chuyển trạng thái gắn nhãn – LTS và dạng đặc tả sử dụng hàm lôgic và chứng minh tính đúng đắn của phương pháp chuyển đổi. Thêm vào đó, chương 4 còn đưa ví dụ minh họa cho phương pháp chuyển đổi này. Chương 5 sẽ trình bày về công cụ chuyển đổi qua lại giữa cá dạng đặc. Chương 6 của luận văn sẽ đưa ra kết luận và hướng phát triển tiếp theo của luận văn. Và cuối cùng phần tài liệu tham khảo.

# Chương 2: Kiến thức cơ sở

## **2.1. Dạng đặc tả sử dụng hệ chuyển trạng thái được gắn nhãn**

***Định nghĩa 2.1***: Hệ chuyển trạng thái được gắn nhãn (Labelled Transition System - LTS [5])

Một LTS là một bộ có thứ tự gồm 4 thành phần: M = (Q, , δ, q0), trong đó:

* Q = {q0, q1, ..., qn} là tập các trạng thái,
* = {ω0, ω1, ..., ωn} là tập các sự kiện,
* δ ⊆ Q x x Q là hàm chuyển trạng thái, và
* q0⊆ Q là trạng thái bắt đầu.

Ta kí hiệu qi qj nếu và chỉ nếu có một sự kiện ωi chuyển hệ thống từ trạng thái qi sang trạng thái qj , khi đó (qi , ωi , qj ) ∈δ. Điều này có nghĩa khi một hệ thống đang ở trạng thái qi, nếu có một sự kiện ωi xảy ra thì hệ thống sẽ chuyển sang trạng thái qj. Tương tự, khi hệ thống đang ở trạng thái qj nếu có một hành động ωk xảy ra thì hệ thống sẽ chuyển sang trạng thái qk. Như vậy, chuỗi hai hành động qi qj, qj qk có thể chuyển hệ thống từ trạng thái qi sang trạng thái qk. Khi đó, ta có thể kí hiệu qi qj.

**Ví dụ 2.1:** Ví dụ về một hệ thống chuyển trạng thái được gắn nhãn.

Trên hình 2.1 là một ví dụ về một LTS M = (Q, , δ, q0), trong đó:

* Q = {q0, q1, q2, q3},
* = {openWindow, start, stop, closeWindow},
* δ = {(q0, openWindow, q1), (q1, start, q2), (q2, stop, q3), (q­3, closeWindow, q0)}, và
* q0 là trạng thái bắt đầu.



Hình 2.1: Một hệ chuyển trạng thái được gán nhãn

***Định nghĩa 2.2*:** Kích thước của một tập hợp [5].

Kích thước của một tập hợp Q = {q0, q1, ..., qn} là số phần tử của tập hợp Q, kí hiệu là |Q|.

**Ví dụ 2.2:** Với LTS được cho bởi hình 2.1, tập các trạng thái Q gồm 4 phần tử là Q = {q0, q1, q2, q3} nên |Q| = 4.

***Định nghĩa 2.3***: Kích thước của một LTS [5].

Kích thước của một LTS M = (Q, , δ, q0) là số trạng thái của M, kí hiệu là |M|, trong đó |M| = |Q|.

**Ví dụ 2.3:** Với LTS được cho bởi hình 2.1, kích thước của LTS đó là |M| = |Q| = 4.

***Định nghĩa 2.4*:** Vết của LTS [5].

Vết của một LTS M = (Q, , δ, q0) là một chuỗi hữu hạn các sự kiện có dạng σ = ω0ω1…ωnvới ωk∈Σ và 0 ≤ k ≤ n sao cho ∃qi ∈ Q để q0 qi . Như vậy, vết của LTS M là một chuỗi các sự kiện có thể quan sát được mà M có thể thực hiện được từ trạng thái bắt đầu q0.

**Ví dụ 2.4:** Vết của LTS.

Hình 2.2 minh họa một LTS M = 〈Q, , δ, q0〉, trong đó:

* Q = {q0, q1, q2, q3},
* = {openWindow, start, stop, closeWindow},
* δ = {(q0, openWindow, q1), (q1, start, q2), (q2, stop, q2), (q3, closeWindow, q0)}, và
* q0 là trạng thái bắt đầu.

Ta thấy, chuỗi các hành động *openWindow start stop* là một vết của M, bởi vì tại trạng thái bắt đầu là q0, khi sự kiện *openWindow* xảy ra, hệ thống chuyển sang trạng thái q1, tiếp tục xảy ra sự kiện *start* hệ thống chuyển sang trạng thái q2, khi xảy ra sự kiện *stop* hệ thống chuyển sang trạng thái q3. Chuỗi các hành động *openWindow start stop* chuyển hệ thông từ trạng thái bắt đầu q0 sang trạng thái q3∈ Q nên chuỗi các hành động *openWindow start stop* là một vết của LTS. Tương tự, chuỗi các hành động *openWindow*, *openWindow start, openWindow start stop closeWindow*, *openWindow start stop closeWindow openWindow*, ... đều là vết của M.



Hình 2.2: Minh họa vết của LTS.

***Định nghĩa 2.5*:** Ngôn ngữ của LTS [5]

Ngôn ngữ của LTS M kí hiệu là L(M) được định nghĩa như sau:

L(M) = {α | α là một vết của M}

**Ví dụ 2.5**: Ví dụ về ngôn ngữ của LTS.

Với LTS M như ở hình 2.2, ngôn ngữ của M là:

L(M) = {openWindow, onpenWindow start, openWindow start stop, ...}

***Định nghĩa 2.6***: LTS đơn định và không đơn định [6].

Một LTS M = (Q, αM, δ, q0) là không đơn định nếu nó chứa một chuyển dịch τ hoặc nếu ∃(q, a, q’) và (q, a, q”) ∈δ sao cho q’≠ q”. Trái lại, M là một LTS đơn định.

**Ví dụ 2.6:** Ví dụ về LTS đơn định và LTS không đơn định



Hình 2.3: Ví dụ về một LTS không đơn định

Trên hình 2.3, LTS là một LTS M = 〈Q, , δ, q0〉, trong đó:

* Q = {q0, q1, q2, q3},
* = {openWindow, start, stop, closeWindow},
* δ = {(q0, openWindow, q1), (q1, start, q2), (q1, start, q3), (q2, stop, q3), (q3, closeWindow, q0)}, và
* q0 là trạng thái bắt đầu.

Chúng ta có thể thấy rằng khi hệ thống đang ở trạng thái q1, sự kiện *start* xảy ra hệ thống có thể chuyển sang trạng thái q2 hoặc q3, vì q2 ≠ q3 mà lại tồn tại 2 chuyển trạng thái (q1, start, q2), (q1, start, q3) nên định nghĩa 2.6, LTS M trên hình 2.3 là một LTS không đơn định.

LTS trên hình 2.1 là một LTS M = 〈Q, , δ, q0〉, trong đó:

* Q = {q0, q1, q2, q3},
* = {openWindow, start, stop, closeWindow},
* δ = {(q0, openWindow, q1), (q1, start, q2), (q2, stop, q2), (q3, closeWindow, q0)}, và
* q0 là trạng thái bắt đầu.

Ta thấy không tồn tại bất kì 2 chuyển trạng thái (q, a, q’) và (q, a, q”) ∈δ mà q’≠ q”, vì vậy theo định nghĩa 2.6. LTS M đã cho trên hình 2.1 là một LTS đơn định.

**Chú ý**: Với Σ Act ta ký hiệu ↑Σ là một dẫn xuất thu được bằng cách loại bỏ khỏi tất cả các hành động ω mà ω Σ. Tập tất cả các vết của M được gọi là ngôn ngữ của M, ký hiệu L(M). Một vết = ω1ω2..ωn là một vết hữu hạn trên LTS M. Ta ký hiệu LTS Mσ = (Q, , , q0) với Q = {q0, q1,.., qn} và = {(qi-1, ωi, qi)} với I = 1,..,n. Ta nói rằng một hành động ω được chấp nhận từ một trạng thái q Q nếu tồn tại q’Q sao cho (q, ω, q’). Tương tự vậy, ta nói rằng một vết ω1ω2...ωn được chấp nhận từ trạng thái qi Q nếu tồn tại một dãy các trạng thái qi, qi+1, …, qi+n với qi = q0 sao cho i = thì (qi-1, ai, qi) .

***Định nghĩa 2.7*:** Ghép nối song song LTS [5].

Phép ghép nối song song được kí hiệu là || là một phép toán nối hai thành phần phần mềm bằng cách đồng bộ các hành vi chung trên bảng chữ cái và đan xen các hành động còn lại.

Giả sử có hai LTS là M1 = (Q1, αM1, δ1, q01) và M2= (Q2, αM2, δ2, q02), ghép nối song song giữa M1 và M2, ký hiệu M1||M2 được định nghĩa như sau:

Nếu M1 = Π hoặc M2 = Π thì M1||M2 = Π. Ngược lại, M1||M2 = (Q, αM, δ, q0), trong đó:

Q= Q1×Q2, αM= αM1αM2, q0 = (q01, q02) và hàm δ được xác định như sau:

* Với ∀(q1, a, q2) ∈ δ1 và (q1’,a,q2’) ∈ δ2 thì ((q1, q1’), a, (q2, q2’)) ∈ δ.
* Với (q1, a, q2) ∈δ1, a ∉ αM2 thì ∀q’ ∈ Q2 ta có ((q1,q’), a, (q2,q’)) ∈ δ.
* Với (q1’, a, q2’) ∈ δ2, a ∉αM1 thì ∀q ∈ Q1 ta có ((q, q1’), a, (q, q2’)) ∈ δ.

**Ví dụ 2.7**. Ví dụ về phép ghép nối song song

Cho LTS M1 như trên hình 2.4, M1 = 〈Q1, , δ1, q0〉, trong đó:

* Q1 = {q0, q1, q2, q3},
* = {openWindow, start, stop, closeWindow},
* δ1 = {(q0, openWindow, q1), (q1, start, q2), (q2, stop, q2), (q3, closeWindow, q0)}, và
* q0 là trạng thái bắt đầu.



Hình 2.4: LTS M1



Hình 2.5: LTS M2

Cho LTS M2 như trên hình 2.5, M2 = 〈Q2, , δ2, a〉, trong đó:

* Q2 = {a, b, c},
* = {start, send, stop},
* δ2 = {(a, start, b), (b, send, c), (c, stop, a)}, và
* a là trạng thái bắt đầu.

Bây giờ ta tiến hành ghép nối M1­ và M2 dựa theo định nghĩa 2.7. Sau khi ghép nối, chúng ta tiến hành loại bỏ tất cả các trạng thái không đến được từ trạng thái bắt tạo (q0, a) và tất cả các hành động đưa hệ thống về trạng thái đó ta sẽ thu được một hệ thống chuyển trạng thái ghép nối song song được gán nhãn M1||M2 như trên hình 2.6. Với M1||M2=〈Q, αM, δ, q0〉, trong đó:

* Q = {(q0, a), (q0, b), (q0, c), (q1, a), (q1, b), (q1, c), (q2, a), (q2, b), (q2, c), (q3, a), (q3, b), (q3, c)},
* = {openWindow, closeWindow, start, send, stop},
* δ = {(a, start, b), (b, send, c), (c, stop, a)}, và
* a là trạng thái bắt đầu.



Hình 2.6: LTS M1||M2 sau khi ghép nôi

***Định nghĩa 2.8***: Hệ chuyển trạng thái được gán nhãn an toàn [2].

Chúng ta sử dụng π để kí hiệu một trạng thái lỗi đặc biệt không được phép xuất hiện trong hệ thống [8]

LTS an toàn là một LTS không chứa bất kỳ một trạng thái lỗi π nào.

***Định nghĩa 2.7***: Thuộc tính an toàn [5].

Thuộc tính an toàn là thuộc tính đảm bảo không có lỗi xảy ra trong quá trình thực hiện của hệ thống. Một thuộc tính an toàn *p* được biểu diễn dưới dạng một hệ chuyển trạng thái được gán nhãn an toàn *p* = 〈Qp, αp, δp, q0〉. Ngôn ngữ của nó L(*p*) là chuỗi các hành động được đoán nhận trên αp.

***Định nghĩa 2.8***: Hệ chuyển trạng thái được gán nhãn lỗi [5].

Hệ chuyển trạng thái được gán nhãn lỗi của một thuộc tính *p* = 〈Q, αp, δ, q0〉 được kí hiệu là perr = 〈Q {π}, αperr, δ’, q0〉, trong đó:

αperr = αp, δ’ = δ {(q, a, π) **|** a αp và q’ Q sao cho (q, a, q’) δ}.

**Ví dụ 2.8**: Hinh 2.7 biểu diễn việc chuyển đổi từ một LTS an toàn p sang một LTS lỗi perr.



Hình 2.7: Minh họa việc chuyển LTS an toàn p sang mộtLTS lỗi perr

***Định nghĩa 2.9***: Tính thỏa mãn một thuộc tính của LTS [5]

Một LTS M được gọi là thỏa mãn thuộc tính *p*, kí hiệu M╞ *p* khi và chỉ khi ∀σ∈L(M) sao cho: (σ↑αp) ∈L(p).

Để kiểm tra một LTS M có thỏa mãn thuộc tính p hay không, ta thực hiện các bước như sau. Đầu tiên, chuyển thuộc tính p sang thuộc tính lỗi perr. Sau đó, tiến hành ghép nối M và perr. LTS sau khi ghép nối sẽ là M||perr. Nếu LTS này tồn tại một dẫn xuất nào đó có thể tới được trạng thái π thì ta kết luận LTS M không thỏa mãn thuộc tính p. Ngược lại, LTS M thỏa mã thuộc tính p.

**Ví dụ 2.9:** Ví dụ về tính thỏa mãn một thuộc tính của LTS.

Ta thử kiểm tra tính thoản mãn của LTS ghép nối M1||M2 trong ví dụ 2.7 (hình 2.7) đối với thuộc tính p trong ví dụ 2.8 (hình 2.8). Áp dụng đúng theo định nghĩa, ta tiến hành chuyển p sang perr và tiến hành ghép nối M1||M2||perr ta được LTS như trên hình 2.9. Các trạng thái được ghép với trạng thái π đều được gọi chung là trạng thái π. Kết quả ghép nối trên hình 2.9, chúng ta thấy không tồn tại một dẫn xuất đến được trạng thái π, vì thế ta có thể kết luận M1||M2╞ *p*



Hình 2.8: Mô hình ghép nối M1||Mq||perr

## **2.2. Dạng đặc tả sử dụng hàm lôgic (Boolean)**

***Định nghĩa 2.10***: Hàm lôgic [7].

B = {T, F} là miền giá trị lôgic. Với X là tập hợp các biến lôgic, một hàm lôgic θ(X) được định nghĩa θ(X): B|X| → B.

**Ví dụ 2.10:** Ví dụ về hàm lôgic.

Với X là tập hợp gồm 3 phần tử, X = {x, y, z} trong đó x, y, z ∈ B. Hàm lôgic θ(x, y, z) = x ∧y ∨ z chính là một ánh xạ θ(X): B3 → B.

***Định nghĩa 2.11*:** Phép gán [7].

Với X là tập hợp các biến lôgic, phép gán υ được đinh nghĩa υ: X → B.

**Ví dụ 2.11:** Với X là tập hợp gồm 3 phần tử, X = {x, y, z} trong đó x, y, z ∈ B, υ1(x) = T, υ2(x) = F, υ1(y) = T, υ2(y) = F, υ1(z) = T và υ2(z) = F, ... là các phép gán trên tập X.

***Định nghĩa 2.12:*** Phép gán hàm [7].

Với Φ(X) là hàm một lôgic trên tập X, υ là một phép gán trên tập X, phép gán hàm kí hiệu Φ[υ] là kết quả thu được khi thay các phần tử x ∈ X bởi υ(x). Với X và X’ là các tập biến lôgic, trong đó X’ = {x’| x ∈ X}, ψ(X, X’) là hàm lôgic trên hai tập X và X’, với υ(x) và υ’(x’) lần lượt là các phép gán trên tập X và X’, kí hiệu ψ[υ, υ’] là kết quả thu được khi thay một cách tương ứng các phần tử x ∈ X bởi v(x) và x’ ∈ X’ bởi υ’(x’).

**Ví dụ 2.12**: Với X = {}, X’ = {} là các tập hợp biến lôgic, Φ() = là một hàm lôgic trên tập X. Nếu υ() = T thì Φ[υ] = F và nếu υ() = F thì Φ[υ] = T. Với ψ(, ’) = ∨ là một hàm lôgic trên tập X và X’, nếu υ() = T, υ’() = F thì ψ[υ, υ’] = T ∨ F = T.

Một cách tổng quát, với n tập các biến lôgic X, X1, X2, ..., Xn trong đó Xi = {∈ X}, ψ(X, X1, X2, ..., Xn) là hàm lôgic tương ứng trên các tập biến lôgic X, X1, X2, ..., Xn, ta kí hiệu ψ[υ1, υ2, ..., υn] là kết quả thu được khi thay một cách tương ứng các phần tử ∈ X1 bởi υ1(), ∈ X2 bởi υ2(), ... và ∈ Xn bởi υn().

***Định nghĩa 2.13***: Dạng đặc tả sử dụng hàm lôgic [7]

Định nghĩa của dạng đặc tả sử dụng hàm lôgic được đưa ra trong [7], tuy nhiên, để cho phù hợp với bài toán chuyển đổi, em sửa đổi một chút định nghĩa vì thế dạng đặc tả sử dụng hàm lôgic được định nghĩa là một bộ có thứ tự gồm 4 phần tử:

N = 〈X, E, τ(X,E, X’), ι(X)〉, trong đó:

* X là tập các biến lôgic dùng để biểu diễn các trạng thái của hệ thống. X = {},
* E là tập các biến lôgic dùng để biểu diễn các hành vi của hệ thống. E = {},
* τ (X, E, X’) là hàm lôgic biểu diễn việc chuyển trạng thái của hệ thống, và
* ι(X) là hàm lôgic dùng để biểu diễn các trạng thái bắt đầu của hệ thống.

**Ví dụ 2.13:** Ví dụ về dạng đặc tả sử dụng hàm lôgic.

Với dạng đặc tả sử dụng hàm lôgic N = 〈X, E, τ (X, E, X’), ι(X)〉, trong đó:

* X = {}, X’ = {},
* E = {},
* τ (X, E, X’) = (, và
* ι(X) =

***Định nghĩa 2.14:*** Vết của dạng đặc tả sử dụng hàm lôgic [7]

Với dạng đặc tả sử dụng hàm lôgic N = 〈X, E, τ(X,E, X’), ι(X)〉, υ là phép gán cho hàm biểu diễn trạng thái trên tập X, γ là phép gán cho hàm biểu diễn sự kiện trên tập E, một chuỗi hữu hạn ξ = γ0γ1 ...γn được gọi là vết của N khi và chỉ khi tồn tại tập các phép gán υ0, υ1, ..., υn+1, γ0, γ1, ..., γn sao cho ι[υ0] = T và τ[υi , γi, υi+1] = T với 0 ≤ i ≤ n.

**Ví dụ 2.14:** Ví dụ về vết của dạng đặc tả sử dụng hàm lôgic [7].

Cho dạng đặc tả sử dụng hàm lôgic N = 〈X, E, τ (X, E, X’), ι(X)〉, trong đó:

* X = {, }, X’ = {, },
* E = {, },
* *τ* (X,E, X’) = {( và
* *ι*(X) = ∧.

Vì ι(X) = ∧ , với υ0 là phép gán trên tập X sao cho υ0() = F và υ0() = F nên ι[υ0] = T ∧ T = T. Mặt khác, gọi υ1 là phép gán trên tập X sao cho υ1() = T, υ1() = F, γ0 là phép gán trên tập E sao cho γ0() = F và γ() = F. Khi đó, τ [υ0, γ0,υ1] = T nên ξ = FFlà một vết của N. Mặt khác, với phép gán υ1, γ1, υ2 sao cho υ1(x1) = T, υ1() = F, γ1() = T, γ1() = F, υ2() = F, υ2() = T thì τ [υ1, γ1, υ2] = T. Do đó ξ = FFTF cũng là một vết của N. Một cách hoàn toàn tương tự chúng ta có thể tìm được các vết tiếp theo của N.

***Định nghĩa 2.15***: Ngôn ngữ của dạng đặc tả sử dụng hàm lôgic.

Cho dạng đặc tả sử dụng hàm lôgic N = 〈X, E, *τ*(X,E, X’), *ι*(X)〉, tập hợp tất cả các vết của N được gọi là ngôn ngữ của N và được kí hiệu là L(N). Ta có: L(N) = {*ξ* | *ξ* là một vết của N}.

**Ví dụ 2.15**: Với dạng đặc tả sử dụng hàm lôgic N cho bởi ví dụ 2.10 thì ngôn ngữ của N là L(N) = {FF, FFTF, FFTFFT, ...}

***Định nghĩa 2.16***. Ghép nối hệ thống được biểu diễn bởi dạng đặc tả sử dụng hàm lôgic [3]

N1 = 〈X1, E1, τ1(X1, E1, X1’), ι1(X1)〉, trong đó:

* X1 là tập các biến lôgic dùng để biểu diễn các trạng thái của hệ thống. X1 = {},
* E1 là tập các biến lôgic dùng để biểu diễn các hành vi của hệ thống. E1 = {},
* τ1(X1, E1, X1’) là hàm lôgic biểu diễn việc chuyển trạng thái của hệ thống, và
* ι(X1) là hàm lôgic dùng để biểu diễn các trạng thái bắt đầu của hệ thống.

N2 = 〈X2, E2, τ2(X2, E2, X2’), ι2(X2)〉, trong đó:

* X2 là tập các biến lôgic dùng để biểu diễn các trạng thái của hệ thống. X2 = {},
* E2 là tập các biến lôgic dùng để biểu diễn các hành vi của hệ thống. E2 = {},
* τ2(X2, E2, X2’) là hàm lôgic biểu diễn việc chuyển trạng thái của hệ thống, và
* ι2(X2) là hàm lôgic dùng để biểu diễn các trạng thái bắt đầu của hệ thống.

Việc ghép nối N0 và N1 kí hiệu là N0||N­1 là một hệ thống được biểu diễn dưới dạng đặc tả sử dụng hàm lôgic N = 〈X, E, τ(X,E, X’), ι(X)〉, trong đó:

* X = X1 X2là tập các biến lôgic dùng để biểu diễn các trạng thái của hệ thống. X = {},
* E = E1 E2là tập các biến lôgic dùng để biểu diễn các hành vi của hệ thống. E = {},
* τ(X, E, X’) là hàm lôgic biểu diễn việc chuyển trạng thái của hệ thống, τ(X, E, X’) = τ1(X1, E1, X1’) ∧τ2(X2, E2, X2’), và
* ι(X) là hàm lôgic dùng để biểu diễn các trạng thái bắt đầu của hệ thống,ι(X) = ι(X1)∧ι2(X2).

**Ví dụ 2.16:** Ví dụ về việc ghép nối 2 hệ thống được biểu diễn dưới dạng đặc tả sử dụng hàm lôgic**.**

Hệ thống N1 = 〈X1, E1, τ1(X1, E1, X1’), ι1(X1)〉, trong đó:

* X1 = {}, X1’ = {},
* E1 = {
* τ1(X1,E1, X1’) = {( và
* ι1(X1) =

Hệ thống N2 = 〈X2, E2, τ2(X2, E2, X2’), ι2(X2)〉, trong đó:

* X2 = {}, X2’ = {},
* E2= {, },
* τ2(X2, E2, X2’) = {(và
* ι2(X2) = .

Khi đó, hệ thống N =N1||N2 = 〈X, E, τ(X,E, X’), ι(X)〉, trong đó:

* X1 = {}, X1’ = {, },
* E1 = {
* τ1(X1, E1, X1’) = {( và
* ι1(X1) =

***Định nghĩa 2.17*: Tính thỏa mãn một thuộc tính của hệ thống được biểu diễn bởi dạng đặc tả sử dụng hàm lôgic [3]**

Một thuộc tính π(X) là một hàm lôgic trên tập X. Với N là dạng đặc tả sử dụng hàm lôgic của hệ thống, ta nói rằng N thỏa mãn π (kí hiệu là N╞π) nếu với bất kì chuỗi ξ = γ0γ1 ...γt là một vết của N và π[υi, γi, υi+1] = T với 0 ≤ i ≤ t và υi và υi+1 là các phép gán trên tập X.

**Ví dụ 2.18**: Ví dụ về tính thỏa mãn thuộc tính π của hệ thống biểu diễn dưới dạng đặc tả sử dụng lôgic.

Xét một hệ thống N được biểu diễn bởi dạng đặc tả sử dụng hàm lôgic như sau:

N = 〈X, E, τ(X,E, X’), ι(X)〉,

* X = {}, X’ = {}
* E = {},
* τp(X, E, X’) = {(}|,
* ιp(X) = .

Và một thuộc tính π được biểu diễn như sau:

π =

Xét 1 chuỗiξ, ξ∈ L(N) và ξ = γ0γ1 = FFFTFF. Khi đó tồn tại các phép gán υ0, υ1, υ2 sao choτ[υ0,γ0, υ1] = T và τ[υ1,γ1, υ2] = T thì với γ0, γ1 là phép gán trên tập E sao cho γ0[] = F, γ0[] = F, γ0[] = F, γ1[] = T, γ1[] = F, γ1[] = F, khi đó π[υ0,γ0, υ1] = T và π[υ1,γ1, υ2] = T nên ξ ∈ L(N) thỏa mãn thuộc tính π. Nếu tất cả các chuỗi ξ∈ L(N) đều thỏa mãn tính chất π thì chúng ta kết luận N╞π.

***Định nghĩa 2.19:*** Bảng ánh xạ

Bảng ánh xạ (mapping) là một bảng dùng để lưu lại các ánh xạ khi chuyển đổi từ dạng đặc tả sử dụng LTS sang dạng đặc tả sử dụng hàm lôgic và ngược lại. Gọi Map là kí hiệu của bảng ánh xạ. Với một LTS M = 〈Q, Σ, δ, q0〉 trong đó Q = Q1∪ Q2 với Q1 là tập các trạng thái đầu vào, Q2 là tập các trạng thái đầu ra và một dạng đặc tả sử dụng hàm lôgic N =〈X, E, τ(X,E, X’), ι(X)〉, trong đó X = X1∪ X2 với X1 là tập các biến lôgic dùng để biểu diễn các trạng thái đầu vào của hệ thống, X2 là tập các biến lôgic biểu diễn các trạng thái đầu ra của hệ thống. Ta định nghĩa:

Q1­ X1

Q2 X2

Σ E

δ(q, ω, q’) *τ*(X, E, X’)

MAP =

Trong đó:

* Q1 X1 là một song ánh từ tập các trạng thái đầu vào đến tập tập các biến lôgic dùng để biểu diễn các trạng thái đầu vào của hệ thống,
* Q2 X2 là một song ánh từ tập các trạng thái đầu ra đến tập tập các biến lôgic dùng để biểu diễn các trạng thái đầu ra của hệ thống,
* Σ E là một song ánh từ tập các sự kiện đến tập các biến lôgic dùng để biểu diễn các sự kiện, và
* (q, ω, q’) τ (υ, γ, υ’) là một ánh xạ từ hàm chuyển trạng thái đến tập các hàm lôgic dùng để biểu diễn việc chuyển trạng thái của hệ thống.

**Ví dụ 2.19**: Ví dụ về bảng ánh xạ.

Cho LTS M = 〈Q, Σ, δ, q0〉 như trên hình 2.3, trong đó:

* Q = {q0, q1, q2}, Q1 = {q0, q1, q2}, Q2 = {q1, q2, q0},
* Σ = {send, out, ack},
* δ = {(q0, send, q1),(q1, out, q2),(q2, ack, q0)}, và
* q0 là trạng thái bắt đầu.



Hình 2.9: Ví dụ về một LTS.

Hình 10

Hình 11

Hình 12

Và dạng đặc tả sử dụng hàm lôgic N = 〈X, E, τ(X, E, X’), ι(X)〉, trong đó:

* X = {, , , },
* E = {, },
* τ(X, E, X’) =), và
* ι(X) = ∧.

Khi đó, nếu dạng đặc tả sử dụng LTS M và dạng đặc tả dụng hàm lôgic N là tương đương. Ta có bảng ánh xạ:

|  |  |  |  |
| --- | --- | --- | --- |
| Q1 | q0 | q1 | q2 |
| X1 | ∧ |  |  |

Bảng 2.1: Thành phần Q1 X1 trong bảng ánh xạ

|  |  |  |  |
| --- | --- | --- | --- |
| Q2 | q1 | q2 | q0 |
| X2 |  |  | ∧ |

Bảng 2.2: Thành phần Q2 X2trong bảng ánh xạ

|  |  |  |  |
| --- | --- | --- | --- |
| Σ | send | out | ack |
| E | ∧ |  |  |

Bảng 2.3: Thành phần Σ E trong bảng ánh xạ

|  |  |  |  |
| --- | --- | --- | --- |
| δ(q, ω, q’) | δ(q0, send, q1) | δ(q1, out, q2) | δ(q2, ack, q0) |
| τ(X, E, X’) |  |  |  |

Bảng 2.4: Thành phần δ(q, e, q’) τ (υ, γ,υ’) trong bảng ánh xạ

Bảng 5

Bảng 6

Bảng 7

Bảng 8

Bảng 9

Bảng 10

# Chương 3: Các phương pháp kiểm chứng

## **3.1. Phương pháp kiểm chứng sử dụng đặc tả bằng LTS[3]**

### **3.1.1. Thuật toán học L\***

Thuật toán học L\* được đề xuất bởi Angluin [8] và sau đó được cải tiến bởi Rivest and Schapire [9]. Trong nội dung của luận văn này, thuật toán mà chúng ta đề cập chính là phiên bản L\* đã được cải tiến. Thuật toán L\* học một ngôn ngữ chưa biết và đưa ra một DFA chấp nhận ngôn ngữ đó. Với Σ là tập các chữ cái, U là một ngôn ngữ chưa biết trên Σ, L\* sẽ đưa ra một DFA M sao cho M là DFA đơn định nhỏ nhất và L(M) = U. Để học được ngôn ngữ U, L\* cần tương tác với một Teacher (Minimally Adequate Teacher). Người Teacher này phải trả lời chính xác hai loại câu hỏi. Loại câu hỏi thứ nhất là câu hỏi dạng truy vấn thành viên, nội dung câu hỏi là một chuỗi σ ∈ Σ\* có thuộc U hay không. Câu trả lời của Teacher là True nếu σ ∈ U và False nếu σ ∉ U. Loại câu hỏi thứ hai là một truy vấn ứng viên, một DFA M có ngôn ngữ là L(M), được L\* tin tưởng là giống hệt với U, L\* sẽ hỏi Teacher kiểm tra xem là L(M) có bằng U hay không. Câu trả lời của Teacher sẽ là True nếu L(M) = U, khi đó thuật toán sẽ kết thúc. Ngược lại Teacher sẽ trả về một phản ví dụ để cập nhật lại bảng quan sát T và lặp lại quá trình trên. Hình 3.1 dưới đây mô tả sự tương tác giữa L\* và Teacher.



Hình 3.1: Mô hình sự tương tác giữa L\* và Teacher

### **3.1.2. Sinh giả định dựa trên thuật toán học học L\***

Với T là một bảng ghi chép, T sẽ ghi lại các chuỗi Σ\* có thuộc ngôn ngữ U hay không. L\* sẽ làm nhiệm vụ cập nhật lại bảng T, điều này được thực hiện bằng cách đưa ra các truy vấn thành viên đến Teacher để cập nhật T. Ở một số giai đoạn, L\* sẽ quyết định đưa ra một phỏng đoán bằng cách sinh ra một DFA Mi từ bảng T và hỏi Teacher xem Mi có phải là ứng viên cần tìm hay không (L(Mi) = U?). Nếu Teacher trả lời là True, thuật toán kết thúc. Ngược lại, nếu Teacher trả lời là False, Teacher sẽ đồng thời trả về một phản ví dụ *cex* và L\* sẽ sử dụng phản ví dụ đó để cập nhật bảng T. Phản ví dụ *cex* phản ánh sự khác nhau giữa L(Mi) và U hay nó cách khác *cex* L(Mi)\U hoặc *cex* U\L(Mi).

Một cách chi tiết L\* xây dựng một bảng quan sát (S, E, T) trong đó:

* S ⊆ Σ**\*** là tập các tiền tố, biểu diễn các trạng thái,
* E ⊆ Σ**\*** là tập các hậu tố, biểu diễn các giá trị thể hiện sự phân biệt DFA Mi với U được trả về bởi Teacher, và
* T là một ánh xạ từ tập (S S.Σ).E → {*true*, *false*}, với mỗi chuỗi s Σ\* thì T(s) = *true* nếu s U, trái lại T(s) = *false*.

Một bảng quan sát (S, E, T) gọi là *đóng* nếu S, Σ thì ’ S, E sao cho T() = T(′*e*).

Nếu bảng quan sát (S, E, T) *đóng* s’ biểu diễn trạng thái kế tiếp của s sau khi thực hiện hành động *a,* tất cả hậu tố của ’ và là hoàn toàn giống nhau. Trực quan chúng ta dễ nhận thấy một bảng quan sát (S, E, T) là *đóng* nếu mọi dòng S.Σ đều tồn tại một dòng ’ S tương ứng với nó.

**Ví dụ 3.1**: Hình 3.2 minh họa cách chuyển một bảng quan sát *đóng* thành một ứng viên DFA.

S

S. Σ



|  |  |
| --- | --- |
|  | E |
| T | λ |
| λ | True |
| out | False |
| ack | True |
| out | False |
| send | True |
| out, ack | False |
| out, out | False |
| out, send | False |

Hình 3.2: Xây dựng một ứng viên từ bảng quan sát đóng

Thuật toán 3.1: Thuật toán học L\*

|  |
| --- |
| **Thuật toán 3.1**: Thuật toán học L\*  **Input**: U, Σ: Với U là một ngôn ngữ chưa biết, Σ là bảng chữ cái  **Output**: M: Với M là một DFA sao cho M là một automata đơn định nhỏ nhất tương ứng với U và L(M) = U  1: Khởi tạo, S = {λ}, E = {λ}  2: loop  3: Cập nhật T sử dụng truy vấn thành viên  4: **while** (S,E,T) chưa đóng  5: Thêm sa vào S to để S đóng, với s ∈ S and a ∈ Σ  6: Cập nhật bảng T sử dụng truy vấn thành viên  7: **end while**  8: Xây dựng ứng viên DFA Mi từ bảng quan sát (S,E,T)  9: Sử dụng câu hỏi truy vấn kiểm tra thành viên, kiểm tra Mi (L(Mi) = U?)  10: if Teacher trả lời YES  11: return Mi  12: else  13: Thêm e ∈ Σ∗ (lấy từ phản ví dụ *cex*) vào E  14: end if  15: end loop |

**Chi tiết của thuật toán 3.1 như sau:**

Đầu tiên, thuật toán khởi tạo các giá trị S và E là {λ} (dòng 1), với λ là một xâu rỗng, sau đó thuật toán dùng câu hỏi truy vấn thành viên cho λ để cập nhật bảng quan sát T đầu tiên. Câu hỏi truy vấn thành viên sẽ tạo ra một ánh xạ (S S.Σ).E → {*true*, *false*} (dòng 3). Sau đó, thuật toán sẽ kiểm tra xem bảng quan sát (S, E, T) đã đóng chưa (dòng 4). Nếu (S, E, T) chưa đóng, hàng sas’ sẽ được thêm vào trong S với S, Σ, ’ S. Vì bảng quan sát (S, E, T) vừa được cập nhật nên tại thuật toán sẽ tiến hành cập nhật lại bảng T bằng cách sử dụng câu hỏi truy vấn thành viên (dòng 5). Khi bảng T đã đóng, một DFA Mi sẽ được sinh ra từ bảng T (dòng 8).

DFA Mi sinh ra sẽ có dạng: Mi = (Q, Mi, , q0, F) như sau:

* Q = S,
* Mi = Σ,
* được định nghĩa như sau (, a) = ′ nếu *e*  E thì T(*e*) = T(′*e*),
* q0 = λ, và
* F = { S sao cho T() = *true*}.

Sau khi DFA Mi sinh ra, L\* sẽ sử dụng câu hỏi truy vấn ứng viên để hỏi Teacher xem Mi có phải là ứng viên cần tìm không. Nếu câu trả lời của Teacher là YES (dòng 10), điều này có nghĩa L(Mi) = U thì thuật toán sẽ dừng lại (dòng 11). Ngược lại, khi câu trả lời của Teacher là No, L\* nhận được một phản ví dụ *cex* Σ từ Teacher. Phản ví dụ *cex* sẽ được L\* phân tích để tìm hậu tố e, hậu tố e này chính là bằng chứng về sự khác nhau U và L(Mi). Sau đó, e sẽ được thêm vào E (dòng 13). Thuật toán tiếp tục vòng lặp tiếp theo (dòng 3).

**Độ phức tạp**: Độ phức tạp của thuật toán này là O(kn2 + nlogm), trong đó k = |Σ|, n là số trạng thái của mô hình sinh ra, m là độ dài lớn nhất của phản ví dụ [2].



Hình 3.3: Mô hình sinh giả định dựa trên thuật toán học L\*

Với loại câu hỏi truy vấn ứng viên, để trả lời rằng một chuỗi σ = a1a2…an có thuộc Σ\* = L(Aw) hay không, Teacher minh họa truy vấn bằng cách ghép nối M1||perr. Với chuỗi σ, đầu tiên Teacher xây dựng một LTS [σ] = 〈Q, α[σ], δ, q0〉, trong đó Q = {q0, q1, …, qn}, α[σ] = Σ, δ = {(qi-1, ai, qi) | 1 ≤ i ≤ n}, và q0 = q0. Teacher sau đó kiểm tra công thức 〈[σ]〉 M1 〈p〉 bằng cách ghép nối [σ]||M1||perr. Nếu hệ thống đã ghép nối [σ]||M1||perr không tồn tại một dẫn xuất đến được trạng thái π (tức là công thức trả về True), điều đó có nghĩa là σ thuộc L(AW). Trong trường hợp này, Teacher trả về true bởi vì M1 không vi phạm thuộc tính p trong ngữ cảnh của σ. Ngược lại câu trả lời để truy vấn thành viên là false.

Loại câu hỏi hai sẽ kiểm tra xem ứng viên Mi có ngôn ngữ là Mi thì L(Mi) có bằng U hay không (L(Mi) = U?). Với mỗi DFA Mi được sinh ra bởi L\* từ bảng quan sát (S, E, T) tại mỗi vòng lặp i, Teacher phải kiểm tra xem DFA Mi có phải là môt DFA ứng viên của vòng lặp i hay không (L(Mi) = L(AW)?). Để làm được điều này, Teacher đầu tiên chuyển DFA Mi sang LTS an toàn Ai. Sau đó sử dụng LTS an toàn Ai làm ứng viên giả định cho quy tắc ghép nối. Teacher áp dụng hai bước của quy tắc ghép nối và phân tích phản ví dụ để trả lời phỏng đoán như sau:

Bước thứ nhất: Với giả định ứng viên Ai sinh ra, Teacher sẽ tiến hành kiểm tra xem M1 có thỏa mãn thuộc tính p với giả định Ai hay không bằng cách tính toán biểu thức thức 〈Ai〉 M 〈p〉 và áp dụng định nghĩa 2.9 để kiểm tra. Nếu M1 thỏa mãn p với giả định Ai, Teacher trả về True thì nghĩa là giả định ứng viên Ai đủ mạnh để M1 thỏa mãn p và chuyển sang bước hai. Ngược lại Teacher sẽ trả về kết quả là False, điều này chứng tỏ giả định ứng viên Ai quá yếu và phải được làm cho mạnh thêm. Việc làm cho giả định ứng viên Ai mạnh thêm được thực hiện bằng cách loại bỏ bớt các hành vi từ trong Ai, đây cũng chính là bước để sinh ra phản ví dụ *cex*. Với việc loại bỏ bớt các hành vi từ trong Ai thì giả định ứng viên Ai+1 sẽ không thể hiện các hành vi làm M1 không thỏa mãn p.

Bước thứ hai: Teacher kiểm tra biểu thức 〈true〉 M2 〈Ai­〉 như trong hình 3.5 bằng cách thực hiện việc ghép nối M2 với giả định ứng viên Aierr (Aierr là LTS lỗi được chuyển từ LTS Ai). Nếu kết quả sau khi ghép nối không tồn tại bất kì một dẫn xuất nào đến được trạng thái π thì kết quả là True. Điều này chứng tỏ 〈true〉 M2 〈Ai­〉 trả về kết quả True, đồng thời M1||M2 ||= p. Ngược lại bước này trả về một phản ví dụ *cex*. Teacher sẽ tiến hành phân tích để xác định xem p có thật sự vi phạm M1||M2 hay giả định Ai quá mạnh để M2­ thỏa mãn. Việc phân tích phản ví dụ được thực hiện bởi Teacher bằng cách tương tự cách sử dụng để trả lời truy vấn thành viên. Với *cex* là phản ví dụ được trả lại bởi bước hai. Đầu tiên, Teacher sẽ tạo một LTS an toàn [*cex*↑Σ] từ phản ví dụ *cex* được minh họa trên hình 3.4. Sau đó, Teacher kiểm tra biểu thức 〈[*cex*↑Σ]〉 M1 〈p〉 bằng cách ghép nối [*cex*↑Σ]||M1 ||perr. Nếu không tồn tại bất kì một dẫn xuất nào của hệ thống sau khi ghép nối chứa trạng thái lỗi π thì chứng tỏ M1||M2 không thỏa mãn p. Ngược lại, Ai quá mạnh để M2 thỏa mãn trong ngữ cảnh của *cex*. Do đó, *cex*↑Σ được trả lại như một phản ví dụ cho phỏng đoán Ai. Quá trình sinh ra giả định lại bắt đầu từ bước thứ nhát.

**…**

Hình 3.4. LTS [*cex*] được tạo từ phản ví dụ *cex*

### **3.1.3. Ví dụ minh họa việc sinh ngữ cảnh sử dụng thuật toán học L\***

Giả sử chúng ta có 3 LTS là Input (hình 3.6), Order (hình 3.7) và Output (hình 3.8) được biểu diễn như ở dưới đây:

in



Hình 3.5: LTS Input (M1)



Hình 3.6: LTS Order



Hình 3.7: LTS Output (M2)

Nhiệm vụ của chúng ta là kiểm tra xem Input || Output ⊨ Order hay không. Gọi A là LTS ngữ cảnh giả định với Σ là tập tập các hành vi. Ta có Σ = (αM1 αP) ∩ αM2 = {start, stop, find}. Ta thực hiện lần lượt theo các bước sau:

Bước 1: Xây dựng bảng quan sát (S, E, T)

|  |  |
| --- | --- |
|  | E |
| T | λ |
| λ | ? |
| **start** | **?** |
| stop | ? |
| find | ? |

Bảng 3.1: Bảng quan sát (S, E, T) lúc khỏi tạo

Xét 1 truy vấn thành viên σ = start, Teacher sẽ minh họa σ thành LTS A có dạng



Hình 3.8: LTS [σ] được xây dựng từ σ = start

Bước 2 : Teacher tiến hành ghép nối [σ]||Input||Order, kết quả thu được như trên hình 3.10



Hình 3.9: LTS [σ]||Input||Order

Kết quả Teacher trả về là False vì tồn tại dẫn xuất chứa start và đến được trạng thái lỗi π. Một cách hoàn toàn tương tự ta được bảng quan sát (S, E, T)

|  |  |
| --- | --- |
|  | E |
| T | λ |
| λ | True |
| **start** | **True** |
| stop | True |
| find | False |

Bảng 3.2: L\* cập nhật bảng quan sát lần 1

Nhận thấy bảng quan sát 3.11 chưa đóng,vì thế chúng ta phải tiến hành làm đóng bảng 3.11, L\* đưa find lên tập S và tiếp tục thực hiện các truy vấn ứng viên ta được bảng 3.12.

|  |  |
| --- | --- |
|  | E |
| T | λ |
| λ | True |
| find | False |
| **start** | **True** |
| stop | True |
| find | False |
| find start | False |
| find stop | False |
| find find | False |

Bảng 3.3: L\* cập nhật bảng quan sát lần 2

Bước 3: Sinh ứng viên từ bảng quan sát  
Bảng 3.3 là bảng quan sát đóng, vì thế ứng viên DFA M1 sinh ra như hình 3.19



Hình 3.10: DFA M1 sinh ra từ bảng 3.3

Bước 4 : L\*thực hiện truy vấn ứng viên hỏi Teacher xem L(M­1) = U ?

Teacher chuyển M1 thành LTS A1 như hình 3.11 rồi tiến hành ghép nối A1 || Input || Order như trong hình 3.12. Ta thấy vẫn còn tồn tại dẫn xuất đến được trạng thái lỗi π nên Teacher trả về một phản ví dụ *cex* = **open start stop open***,* *cex*↑Σ = **start stop**.



Hình 3.11: LTS M1 được Teacher chuyển thành LTS A1



Hình 3.12: LTS A1 || Input || Order

Bước 5. Cập nhật bảng quan sát và sinh ứng viên DFA M2

Kết quả cập nhật bảng quan sát như ở bảng 3.4. Ta thấy, bảng quan sát này đã đóng vì thế DFA M2 sinh ra như trên hình 3.13.

|  |  |  |
| --- | --- | --- |
|  | E | |
| T | λ | stop |
| λ | True | True |
| find | False | False |
| start | True | False |
| start | True | False |
| stop | True | True |
| find | False | False |
| find start | False | False |
| find stop | False | False |
| find find | False | False |
| start start | True | True |
| start stop | False | False |
| start find | True | True |

Bảng 3.4: Cập nhật bảng quan sát từ phản ví dụ mà Teacher trả về



Hình 3.13: DFA M2

Bước 6: L\* sẽ thực hiện truy vấn ứng viên để hỏi Teacher xem với DFA M2 thì L(M2) = U?

Đầu tiên, Teacher sẽ chuyển M2 thành LTS A2 như trên hình 3.14



Hình 3.14: LTS A2 được Teacher chuyển từ DFA M2

Sau đó, Teacher tiến hành ghép nối A2||Input||Order, kết quả sau khi ghép nối được thể hiện như ở hình 3.15



Hình 3.15: LTS A2err chuyển từ LTS A2

Ta thấy LTS A2||Input||Order không tồn tại một dẫn xuất đến được trạng thái lỗi π, vì thế A2 || Input╞ Order

Bước 7: Kiểm tra kết quả của biểu thức 〈true〉 M2 〈A2­〉 bằng cách chuyển LTS A2 sang LTS A2err như trên hình 3.16, sau đó Teacher tiến hành ghép nối M2 || A2err như trên hình 3.17.

## **3.2. Phương pháp kiểm chứng sử dụng đặc tả bằng hàm logic**

### **3.2.1. Thuật toán học CNDF [3]**

Với X là một tập cố định các biến logic và f(X) là một hàm logic trên tập X, thuật toán học cho hàm logic sẽ tính toán một hàm f’(X) làm đại diện cho f(X) trong một số bước hữu hạn. Thuật toán CDNF học chính xác hàm logic. Giống như thuật toán L\*, thuật toán CDNF sử dụng một mô hình học chủ động. Trong mô hình này, có một Teacher được giả định. Teacher này biết hàm logic cần học f(X) và đưa cho thuật toán học câu trả lời dựa vào các kiểu câu hỏi:

Kiểu câu hỏi thành viên MEM(v) cho f(X) với υ là một phép gán trên X. Nếu f[v] = T, Teacher trả lời YES và nếu ngược lại Teacher trả lời là NO. Kiểu truy vấn tương đương EQ(θ) cho hàm f(X), trong đó θ(X) là một hàm logic trên tập X. Nếu phỏng đoán θ(X) tương đương với hàm f(X), Teacher trả lời là YES, ngược lại Teacher trả lời NO và đưa ra một phản ví dụ υ với υ là phép gán trên tâp X mà θ[X] = f[X].

Ví dụ, giả sử f(x, y) = là hàm logic cần học trên x và y. Teacher trả lời NO đến truy vấn MEM(υ) trong đó υ(xy) = FF (υ(x) = F, υ(y) = F), khi đó Teacher kiểm tra f[υ(x), υ(y)] = f(F, F) = F nên Teacher trả lời NO. Với một truy vấn khác υ(xy) = TF, Teacher trả lời YES. Một ví dụ về truy vấn tương đương, EQ(). Teacher trả lời NO và trả về một phản ví dụ là phép gán υ(xy) = TT vì f(T, T) = F EQ(TT) = T. Với một truy vấn dương đương khác EQ(, Teacher trả lời YES. Với f(X) là một hàm logic trên tập X, |f(X)|DNF và |f(X)|CNF được kí hiệu là kích thước của f(X) lần lượt là dạng bình thường rời rạc và dạng bình thường nối tiếp của f(X). Dưới phương thức học tập cố định ở trên, thuật toán CDNF tính toán ra một đại diện cho hàm logic f(X) bất kì thông qua một chuỗi các truy vấn trong |f(X)|DNF và |f(X)|CNF, |X|

### **3.2.2. Sinh giả dịnh dựa trên thuật toán học CDNF**

- Trình bày Leaner

- Trình bày Teacher

### **3.2.3. Ví dụ minh họa việc sinh giả định sử dụng thuật toán học CDNF**

Lấy ví dụ, trình bày từng bước để sinh giả định.

# Chương 4: Chuyển đổi giữa dạng đặc tả sử dụng LTS và dạng đặc tả sử dụng hàm lôgic

## **4.1. Phương pháp chuyển đổi**

### **4.1.1. Thuật toán chuyển đổi**

Đầu vào (Input): Một LTS.

Đầu ra (Output): Dạng đặc tả sử dụng hàm lôgic và bảng ánh xạ.

Thuật toán chia làm 4 bước:

* Bước 1: Mã hóa tập các trạng thái đầu vào - Thuật toán 3.1.
* Bước 2: Mã hóa tập các trạng thái đầu ra - Thuật toán 3.1.
* Bước 3: Mã hóa tập các sự kiện - Thuật toán 3.1.
* Bước 4: Mã hóa tập các chuyển trạng thái - Thuật toán 3.2.

Tập các trạng thái đầu vào, tập các trạng thái đầu ra hay tập các sự kiện gọi chung là một tập hợp. Vì các bước tiến hành mã hóa một tập hợp là giống nhau nên luận văn chỉ trình bày một thuật toán chung. Khi tiến hành mã hóa thì tùy từng mục đích mã hóa tập đầu vào sẽ thay đổi. Cụ thể, nếu chúng ta tiến hành mã hóa tập các trạng thái đầu vào Q1 thì đầu vào cho thuật toán mã hóa sẽ là tập các trạng thái đầu vào Q1, hoặc nếu chúng ta tiến hành mã hõa tập các trạng thái đầu ra Q2 thì đầu vào cho thuật toán mã hóa sẽ là tập các trạng thái đầu ra Q2 và nếu chúng ta tiến hành mã hóa tập các sự kiện Σ thì đầu vào cho thuật toán mã hóa sẽ là tập các sự kiện Σ.

* Thuật toán 3.1: Thuật toán mã hóa một tập hợp
* Đầu vào: Một tập hợp A (A là tập các trạng thái đầu vào hoặc A là tập các trạng thái đầu ra hoặc A là tập các sự kiện)
* Đầu ra : Tập hợp các phần tử của tập hợp A đã được mã hóa và bảng ánh xạ
* 1 **if** |A| = 1 **then**
* 2 | z = 1
* 3 **else**
* 4 **if** *log*2(|A|) *là số nguyên* **then**
* 5 z = *log*2(|A|)
* 6  **else**
* 7 z = [log2(|A|)] + 1
* 8 **end**
* 9 **end**
* 10 **for** *mỗi phần tử a*i *trong tập A* **do**
* 11 Lưu *a*i vào bảng ánh xạ ứng với thành phần A
* 12 *α*i = True
* 13 k = Thứ tự của *a*i trong A
* 14 Chuyển k sang số nhị phân với độ dài z bit
* 15 **for** *mỗi bit trong chuỗi nhị phân biểu diễn k* **do**
* 16  **if** *bit = 0* **then**
* 17 *α*i *= α*i ∧ *x*j
* 18 **else**
* 19 *α*i *= α*i ∧ *x*j
* 20 **end**
* 21 **end**
* 22 Lưu *α*ivào bảng ánh xạ ứng với vị trí của phần tử *a*i trong thành phần **A**
* 23 **end**
* Với A là tập các phần tử cần mã hóa A = {a0, a1, ..., an}. Từ bước (1) - (9), ta sẽ xác định được số biến lôgic cần dùng để mã hóa tất cả các phần tử của tập hợp A, gọi z là số biến lôgic cần dùng. Gọi X = {x1, x2, ..., xz} là tập các biến lôgic dùng để mã hóa các phần tử của A. Mỗi phần tử ai trong A sẽ được biểu diễn dưới dạng x1 ∧ x2 ∧ ... ∧ xz. Xét một phần tử bất kì ai trong tập A, theo bước (13) ta xác định được thứ tự của ai trong tập A là k, theo bước (14) thì k sẽ biểu diễn dưới dạng số nhị phân z bit. Theo bước (15) - (20) chúng ta sẽ mã hóa được phần tử ai . Thêm vào đó, theo bước (22) dạng biểu diễn của phần tử ai sẽ được lưu vào trong bảng ánh xạ. Bởi vì, thứ tự của mỗi phần tử trong ai là duy nhất nên số nhị phân biểu diễn thứ tự của phần tử ai cũng sẽ là duy nhất, vì thế dạng mã của phần tử ai là duy nhất. Một cách tương tự cho các phần tử khác trong A, sau khi vòng lặp ở bước (1) kết thúc chúng ta được dạng mã hóa được tất cả các trạng thái của tập A. Thêm vào đó, sau khi mã hóa, các thông tin về các trạng thái và dạng mã hóa tương ứng của từng trạng thái này đều được lưu vào bảng ánh xạ. Chúng ta sẽ thấy αi = αi n ∧ j=1 xj là một hàm lôgic biểu diễn phần tử ai và các phần tử xj biểu diễn hàm αi đều thuộc tập X.
* **Độ phức tạp**: Độ phức tạp của thuật toán là O(n), trong đó n là kích thước của tập A cần mã hóa.
* **3.1.2 Thuật toán mã hóa tập các truyển trạng thái**
* **Thuật toán 3.2**: Mã hóa tập các chuyển trạng thái
* **Đầu vào**: Tập hợp các chuyển trạng thái của LTS.
* **Đầu ra** : Hàm chuyển trạng thái τ (X,E, X’)
* 1 **for** *mỗi chuyển trạng thái được biểu diễn δ*(*q*, *w*, *q’*) **do**
* 2 Lấy dạng biểu diễn *αi* của *q* từ bảng ánh xạ của tập các trạng thái đầu vào
* 3 Lấy dạng biểu diễn *ei* của *wi* từ bảng ánh xạ của tập các sự kiện
* 4 Lấy dạng biểu diễn *αi+1* của *q’* từ bảng ánh xạ của tập các trạng thái đầu ra
* 5 *τ* = n ∨ i=1 {*αi* ∧ *ei* ∧ *αi+1*}
* 6 **end**
* 7 **return** *τ*
* Một chuyển trạng thái được biểu diễn là một bộ ba (q, w, q’), nên để mã hóa cho mỗi chuyển trạng thái, chúng ta cần tìm dạng mã hóa cho từng thành phần *q*, *w* và *q’*. Dựa theo thuật toán 3.1, chúng ta đã có được bảng ánh xạ lưu thông tin về các trạng thái và dạng mã hóa tương ứng, thông tin về các sự kiện và dạng mã hóa tương ứng. Xét một chuyển trạng thái (*q1*, *wi* , *qi+1’*) Theo bước (2) *qi* sẽ tương ứng với *xi* trong bảng ánh xạ của tập các trạng thái đầu vào. Theo bước (3) sự kiện *wi* sẽ tương ứng với *ei* trong bảng ánh xạ của tập các sự kiện. Theo bước (4) trạng thái *qi+1* sẽ tương ứng với *xi+1* trong bảng ánh xạ của tập các trạng thái đầu ra. Sau bước (5), chuyển trạng thái (*qi*, *wi*, *qi+1’*) sẽ được biểu diễn dưới dạng *αi* ∧ *ei* ∧ *αi+1*.
* **Độ phức tạp**: Độ phức tạp của thuật toán là O(n), trong đó n là kích thước của tập hợp các chuyển trạng thái cần mã hóa.

### **4.1.2. Chứng minh tính đúng đắn của phương pháp chuyển đổi**

## **4.2. Ví dụ về việc chuyển đổi qua lại giữa các dạng đặc tả**

### **4.2.1. Giới thiệu về hệ thống**



Hình 4.1: Một hệ thống chuyển trạng thái được gán nhãn.

Hình 4.1 là một LTS biểu diễn việc chuyển trạng thái của một tiến trình. Tiến trình là một chương trình đang được thực hiện. Một tiến trình có 5 trạng thái cơ bản:

* New là trạng thái khi tiến trình vừa được tạo hay nói cách khác là tiến trình vừa được sinh ra.
* Ready là trạng thái tiến trình được đưa vào hệ thống và được cấp phát đầy đủ tài nguyên (trừ CPU) để xử lí.
* Blocked là trạng thái mà tiến trình chờ đợi để được cấp phát thêm tài nguyên hoặc để một số sự kiên nào đó xảy ra, hay một quá trình vào/ra kết thúc. Khi sự kiện đang chờ xảy ra thì tiến trình sẽ ở trạng thái Ready.
* Running là trạng thái mà tiến trình đang được sở hữu CPU để hoạt động, hay nói cách khác là các chỉ thị của tiến trình đang được thực hiện/ xử lý bởi processor.
* Exit là trạng thái mà tiến trình kết thúc việc xử lí.

Dưới đây là bảng mô tả việc chuyển đổi giữa các trạng thái trong tiến trình.

|  |  |  |  |
| --- | --- | --- | --- |
| Trạng thái bắt đầu | Trạng thái kết thúc | Sự kiện | Nguyên nhân |
| - | New | - | Tiến trình đang được tạo |
| New | Ready | admit | Tiến trình được khởi tạo, đưa vào hệ thống và được cấp phát đầy đủ tài nguyên chỉ thiếu CPU |
| Ready | Running | dispatch | Tiến trình được cấp CPU để bắt đầu thực hiện/xử lý. |
| Running | Blocked | waitEvent | Tiến trình đang chờ một sự kiện nào đó xảy ra hay đang chờ một thao vào/ra kết thúc hay tài nguyên mà tiến trình yêu cầu chưa được hệ điều hành đáp ứng. |
| Blocked | Ready | eventOccurs | Sự kiện mà tiến trình chờ đã xảy ra, thao tác vào/ra mà tiến trình đợi đã kết thúc, hay tài nguyên mà tiến trình yêu cầu đã được hệ điều hành đáp ứng. |
| Running | Ready | timeout | Khi tiến trình đang chạy bị chiếm chỗ bởi tiến trình khác có độ ưu tiên cao hơn hoặc tiến trình đang chạy đã sử dụng quá thời gian cho phép |
| Running | Exit | Release | Tiến trình kết thúc |

Hình 4.20: Một hệ chuyển trạng thái được gán nhãn

### **4.2.2. Chuyển đổi dạng đặc tử sử dụng LTS sang dạng đặc tả sử dụng hàm lôgic**

Xét LTS M = {Q, Σ, δ, q­0} như hình 4.1. Trong đó:

* Q = {New, Ready, Running, Exit, Blocked}, Q1 = {New, Ready, Running, Blocked}, Q2 = {Ready, Running, Exit, Blocked},
* Σ = {admit, dispatch, timeout, release, waitEvent, eventOccurs},
* δ = {(New, admit, Ready), (Ready, dispatch, Running), (Running, release, Exit), (Running, timeout, Ready), (Ready, waitEvent, Blocked), (Blocked, eventOccurs, Ready)}, và
* New là trạng thái bắt đầu.

Chúng ta sẽ mã hóa tập các trạng thái và tập các sự kiện dựa theo thuật toán 3.1. Với đầu vào là tập các trạng thái đầu vào Q1 = {New, Ready, Running, Blocked}, ta có |Q1| = 4, vì log2(4) là số nguyên nên theo bước (4) và bước (5) ta có z = log2(4) = 2 biến. Theo bước (10) chúng ta tiến hành mã hóa từng trạng trạng thái trong tập Q1, trạng thái a0 là trạng thái New, theo bước (11), chúng ta sẽ lưu New vào trong bảng ánh xạ ứng với thành phần Q1 X1, theo bước (12), khởi tạo α0 = True, theo bước (13), k là vị trí của trạng thái New trong tập Q1 nên k = 0, theo bước (14), k sẽ được biểu diễn dưới dạng số nhị phân có dộ dài 2 bit (z bit) hay k = 00. Theo bước (15), xét bit thứ 1 của k, giá trị của bit này bằng 0 nên theo bước (17) α0 = α0 ∧ 1, bit tiếp theo là bit thứ 2, giá trị của bit này bằng 0, nên theo bước (17) α0 = α0 ∧ 1 ∧ 2 = 1 ∧ 2. Theo bước (22), chúng ta sẽ lưu α0 vào trong bảng ánh xạ ứng với vị trí của trạng thái New trong thành phần Q1 X1. Một cách hoàn toàn tương tự, chúng ta thu được dạng mã hóa của trạng thái Ready là α1 = 1 ∧ 2, dạng mã hóa của trạng thái Running là α2 = 1 ∧ 2 và dạng mã hóa của trạng thái Blocked là α3 = 1 ∧ 2. Sau khi mã hóa xong tập các trạng thái Q1 ta thu đươc thành phần Q1 X1 của bảng ánh xạ như bảng 4.2. Đồng thời sau khi mã hóa xong tập các trạng thái đầu vào Q1, chúng ta cũng thu được hàm lôgic dùng để biểu diễn trạng thái bắt đầu của hệ thống (cũng chính là dạng mã hóa của trạng thái q0) là ι(X) = 1 ∧ 1. Một cách hoàn toàn tương tự, áp dụng thuật toán 3.1 với đầu vào là tập các trạng thái đầu ra Q2 = {Ready, Running, Exit, Blocked} và tập các sự kiện Σ = {admit, dispatch, timeout, release, waitEvent, eventOccurs} ta thu các thành phần Q2 X2 và thành phần Σ E của bảng ánh xạ như trên hình 4.3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Q1 | New | Ready | Running | Blocked |
| X1 | 1 ∧ 2 | 1 ∧ 2 | 1 ∧ 2 | 1 ∧ 2 |

Bảng 2.11: Thành phần Q1 X1 trong bảng ánh xạ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Q2 | Ready | Running | Exit | Blocked |
| X2 | 6 ∧ 7 | 6 ∧ 7 | 6 ∧ 7 | 6 ∧ 7 |

Bảng 4.12: Thành phần Q2 X2 trong bảng ánh xạ

|  |  |  |  |
| --- | --- | --- | --- |
| Σ | admit | dispatch | release |
| E | 3 ∧ 4 ∧ 5 | 3 ∧ 4 ∧ 5 | 3 ∧ 4 ∧ 5 |
| Σ | timeout | waitEvent | eventOccurs |
| E | 3 ∧ 4 ∧ 5 | 3 ∧ 4 ∧ 5 | 3 ∧ 4 ∧ 5 |

Bảng 4.13: Thành phần Σ ↦ E trong bảng ánh xạ

Công việc còn lại sẽ là mã hóa tập các chuyển trạng thái. Áp dụng thuật toán 3.2 với đầu vào là tập các chuyển trạng thái δ = {(New, admit, Ready), (Ready, dispatch, Running), (Running, release, Exit), (Running, timeout, Ready), (Ready, waitEvent, Blocked), (Blocked, eventOccurs, Ready)}. Theo bước (1) xét lần lượt từng chuyển trạng trong tập δ, với chuyển trạng thái đầu tiên (New, admit, Ready), theo bước (2), ta lấy được dạng biểu diễn của New từ thành phần Q1 X1 của bảng ánh xạ là 1 ∧ 2. Theo bước (3), ta lấy được dạng biểu diễn của sự kiện admit từ thành phần Σ E của bảng ánh xạ là 3 ∧ 4 ∧ 5. Theo bước (4), ta lấy được dạng biểu diễn của Ready từ thành phần Q2 X2 của bảng ánh xạ là 6 ∧ 7. Theo bước (5), ta thu được dạng mã hóa của chuyển trạng thái (New, admit, Ready) là1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7. Áp dụng một cách hoàn toàn tương tự cho các chuyển trạng thái khác trong tập δ, cuối cùng ta thu được τ (X, E, X’) = {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7} | {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7} | {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7} | {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7} | {1 ∧ 2 ∧ 3 ∧ x4 ∧ 5 ∧ 6 ∧ 7} | {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7}. Và thành phần δ (q, e, q’) τ (υ, γ, υ’) của bảng ánh xạ như trong bảng 4.5.

Cuối cùng, kết quả sau khi chuyển đổi chúng ta thu được dạng đặc tả sử dụng hàm lôgic N = X, E, τ (X, E, X’), ι(X) và bảng ánh xạ (Bảng 4.2, 4.3, 4.4 và 4.5). Với N = X, E, τ (X, E, X’), ι(X). Trong đó:

* X = {1, 2}, X’ = {6, 7},
* E = {3, 4, 5},
* τ (X, E, X’) = {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7} | {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7} | {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7} | {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7} | {1 ∧ 2 ∧ 3 ∧ x4 ∧ 5 ∧ 6 ∧ 7} | {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7}, và
* ι(X) = 1 ∧ 2.

|  |  |  |
| --- | --- | --- |
| δ(q, e, q’) | (New, admit, Ready) | (Ready, dispatch, Running) |
| τ (υ, γ, υ’) | 1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7 | 1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7 |
| δ(q, e, q’) | (Running, release, Exit) | (Running, timeout, Ready) |
| τ (υ, γ, υ’) | 1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7 | 1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7 |
| δ(q, e, q’) | (Ready, waitEvent, Blocked) | (Blocked, eventOccurs, Ready) |
| τ (υ, γ, υ’) | 1 ∧ 2 ∧ 3 ∧ x4 ∧ 5 ∧ 6 ∧ 7 | 1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7 |

Bảng 4.14: Thành phần δ(q, e, q’) τ (υ, γ, υ’) trong bảng ánh xạ

### **4.2.3. Chuyển đổi dạng đặc tử sử dụng hàm lôgic sang dạng đặc tả sử dụng LTS**

Giả sử chúng ta đã có dạng đặc tả sử dụng hàm lôgic N và bảng ánh xạ. Với N = X, E, τ (X,E, X’), ι(X). Trong đó:

* X = {1, 2}, X’ = {6, 7},
* E = {3, 4, 5},
* τ(X, E, X’) = {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7} | {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7} | {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7} | {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7} | {1 ∧ 2 ∧ 3 ∧ x4 ∧ 5 ∧ 6 ∧ 7} | {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7}, và
* ι(X) = 1 ∧ 2. Và bảng ánh xạ Map:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Q1 | New | Ready | Running | Blocked |
| X1 | 1 ∧ 2 | 1 ∧ 2 | 1 ∧ 2 | 1 ∧ 2 |

Bảng 4.15: Thành phần Q1 X1 trong bảng ánh xạ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Q2 | Ready | Running | Exit | Blocked |
| X2 | 6 ∧ 7 | 6 ∧ 7 | 6 ∧ 7 | 6 ∧ 7 |

Bảng 4.16: Thành phần Q2 X2 trong bảng ánh xạ

Nhiệm vụ của chúng ta là tìm ra LTS ban đầu, tức là LTS đã được mã hóa thành dạng đặc tả sử dụng hàm lôgic N đã cho. Dựa vào thành phần Q1 X1 ta thấy x̄1 ∧ x̄2 là dạng mã hóa của trạng thái q0 mà ι(X) = x̄1 ∧ x̄2 nên q0 là trạng thái bắt đầu của LTS. (\*) là Q1 = {New, Ready, Running, Blocked}, Q2 có 4 trạng thái là Q2 = {Ready, Running, Exit, Blocked} mà Q = Q1 ∪ Q2 nên Q = {New, Ready, Running, Exit, Blocked} (\*\*). Thêm vào đó, dựa vào thành phần Σ E, ta thấy LTS có 6 sự kiện Σ = {admit, dispatch, release, timeout, waitEvent, eventOccurs} (\*\*\*). Mặt khác τ (X, E, X’) = {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7} | {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7} | {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7} | {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7} | {1 ∧ 2 ∧ 3 ∧ x4 ∧ 5 ∧ 6 ∧ 7} | {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7}. Xét lần lượt từng phần tử trong τ (X, E, X’), với τ (υ, γ, υ’) = {1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7} (kể cả khi thứ tự các biến mã hóa bị thay đổi). Đối chiếu với thành phần Σ E của bảng ánh xạ ta thấy đây là dạng biểu diễn của sự kiện admit. Một các hoàn tòan toàn tương tự ta sẽ có các chuyển trạng thái còn lại trong LTS là (Ready, dispatch, Running), (Running, release, Exit), (Running, timeout, Ready), (Running, waitEvent, Blocked) và (Blocked, eventOccurs, Ready). (\*\*\*\*)

Lại có, dựa vào thành phần Q1 X1, Q2 X2, ta thấy Q1 có 4 trạng thái

|  |  |  |  |
| --- | --- | --- | --- |
| Σ | admit | dispatch | release |
| E | 3 ∧ 4 ∧ 5 | 3 ∧ 4 ∧ 5 | 3 ∧ 4 ∧ 5 |
| Σ | timeout | waitEvent | eventOccurs |
| E | 3 ∧ 4 ∧ 5 | 3 ∧ 4 ∧ 5 | 3 ∧ 4 ∧ 5 |

Bảng 4.17: Thành phần Σ E trong bảng ánh xạ

|  |  |  |
| --- | --- | --- |
| δ(q, e, q’) | (New, admit, Ready) | (Ready, dispatch, Running) |
| τ (υ, γ, υ’) | 1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7 | 1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7 |
| δ(q, e, q’) | (Running, release, Exit) | (Running, timeout, Ready) |
| τ (υ, γ, υ’) | 1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7 | 1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7 |
| δ(q, e, q’) | (Ready, waitEvent, Blocked) | (Blocked, eventOccurs, Ready) |
| τ (υ, γ, υ’) | 1 ∧ 2 ∧ 3 ∧ x4 ∧ 5 ∧ 6 ∧ 7 | 1 ∧ 2 ∧ 3 ∧ 4 ∧ 5 ∧ 6 ∧ 7 |

Bảng 4.18: Thành phần δ(q, e, q’) τ (υ, γ, υ’) trong bảng ánh xạ

Từ (\*) (\*\*) (\*\*\*) và (\*\*\*\*) ta được LTS M như hình 4.1.

M = Q, Σ, δ, q0, trong đó:

* Q = {New, Ready, Running, Exit, Blocked}, Q1 = {New, Ready, Running, Blocked}, Q2 = {Ready, Running, Exit, Blocked},
* Σ = {admit, dispatch, timeout, release, waitEvent, eventOccurs},
* δ = {(New, admit, Ready), (Ready, dispatch, Running), (Running, release, Exit), (Running, timeout, Ready), (Ready, waitEvent, Blocked), (Blocked, eventOccurs, Ready)}, và
* New là trạng thái bắt đầu.

# Chương 5: Công cụ và thực nghiệm

Trình bày công cụ chuyển đổi

- Giới thiệu

- Cách sử dụng (định dạng file input, output thế nào, …)

- Ví dụ minh họa

# Chương 6: KẾT LUẬN

- Các kết quả đạt được của luận văn

- Hướng phát triển trong tương lai

# TÀI LIỆU THAM KHẢO

[1] Ivica Crnkovic and Magnus Larsson, “Component-Based Software Engineering – New Paradigm of Software Development”, Department of Computer Engineering, Mälardalen University.

[2] Christel Baier and Joost-Pieter Katoen, “Principles of Model Checking”, Cambridge, Massachusetts (pp. 8)

[3] P.N.Hung, “Assume–Guarantee Veriﬁcation of Evolving Component-Based Software”, Japan Advanced Institute of Science and Technology in partial fulﬁllment of the requirements for the degree of Doctor of Philosophy.

[4] Edmund M. Clarke, William Klieber, Milos Novácek, and Paolo Zuliani, “Model checking and the State Explosion Problem”

[5] P. N. Hung, N. V. Ha, T. Aoki and T. Katayama, “On Optimization of Minimized Assumption Generation Method for Component-based Software Verification”, IEICE Trans. on Fundamentals, Special Issue on Software Reliability Engineering, Vol. E95-A, No.9, pp. 1451-1460, Sep. 2012.

[6] D. Angluin, “Learning regular sets from queries and counterexamples”, Information and Computation, 75(2), pp. 87-106, Nov. 1987.

[7] Yu-Fang Chen, Edmund M. Clarke, Azadeh Farzan, Ming-Hsien Tsai, YihKuen Tsay, and Bow-Yaw Wang, Automated Assume-Guarantee Reasoning through Implicit Learning. Addison-Wesley, Reading, Massachusetts, 1993.

[8] Jorge Cuellar, Tom Maibaum, Kaisa Sere, “FM 2008: Formal Methods”, 15th International Symposium on Formal Methods Turku, Finland, pp. 118, May 2008.

[2] P. N. Hung, N. V. Ha, T. Aoki and T. Katayama, “On Optimization of Minimized Assumption Generation Method for Component-based Software Verification”, IEICE Trans. on Fundamentals, Special Issue on Software Reliability Engineering, Vol. E95-A, No.9, pp. 1451-1460, Sep. 2012.

[6] Edmund M. Clarke, William Klieber, Milos Novácek, and Paolo Zuliani, “Model checking and the State Explosion Problem”

[7] P. N. Hung, “Assume-Guarantee Verification of Evolving Component-Based Software”, Japan Advanced Institute of Science and Technology in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

[9] D. Angluin: "Learning Regular Sets from Queries and Counterexamples", Informationand Computation, vol. 75, no. 2, pp. 87-106 (Nov. 1987).

[10]Jamieson M. Cobleigh, Dimitra Giannakopoulou, and Corina S. Păsăreanu, “Learning Assumptions for CompositionalVerification”,

[17] R. L. Rivest and R. E. Schapire: \Inference of Finite Automata using Homing Sequences",Information and Computation, vol. 103, no. 2, pp. 299-347 (Apr. 1993).

[19] “Bhaswati Sadhukhan”, Model Based Testing Practices, Global Business Services (GBS) IBM(I) Pvt. Ltd. Kolkata, India.