|  |
| --- |
| 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: 60 48 10  LUẬN VĂN THẠC SĨ  **NGƯỜI HƯỚNG DẪN KHOA HỌC: PGS.TS. PHẠM NGỌC HÙNG**  Hà nội – 2016 |

# MỤC LỤC

[MỤC LỤC i](#_Toc464489330)

[LỜI CẢM ƠN ii](#_Toc464489331)

[LỜI CAM ĐOAN iii](#_Toc464489332)

[DANH MỤC THUẬT NGỮ VIẾT TẮT iv](#_Toc464489333)

[DANH MỤC HÌNH VẼ v](#_Toc464489334)

[DANH MỤC BẢNG vi](#_Toc464489335)

[Chương 1: Giới thiệu 1](#_Toc464489336)

[Chương 2: Kiến thức cơ sở 3](#_Toc464489337)

[**2.1. Dạng đặc tả sử dụng hệ chuyển trạng thái được gắn nhãn** 3](#_Toc464489338)

[**2.2. Dạng đặc tả sử dụng hàm logic (Boolean)** 11](#_Toc464489339)

[Chương 3: Các phương pháp sinh mô hình 18](#_Toc464489340)

[**3.1. Phương pháp sinh giả định sử dụng thuật toán học L\* [3]** 19](#_Toc464489341)

[**3.1.1. Thuật toán học L\*** 19](#_Toc464489342)

[**3.1.2. Sinh giả định dựa trên thuật toán học học L\*** 22](#_Toc464489343)

[**3.1.3. Ví dụ minh họa việc sinh ngữ cảnh sử dụng thuật toán học L\*** 25](#_Toc464489344)

[**3.2. Phương pháp sinh đặc tả sử dụng thuật toán CNDF [3]** 31](#_Toc464489345)

[Chương 4: Phương pháp chuyển đổi 32](#_Toc464489346)

[**4.2. Ví dụ về việc chuyển đổi qua lại giữa các dạng đặc tả** 32](#_Toc464489347)

[**4.3. Giới thiệu về hệ thống** 32](#_Toc464489348)

[Chương 5: Xây dựng công cụ chuyển đổi 47](#_Toc464489349)

[Chương 6: KẾT LUẬN 49](#_Toc464489350)

[TÀI LIỆU THAM KHẢO 50](#_Toc464489351)

# 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

- Giới thiệu về phát triển phần mềm hướng thành phần

- Kiểm chứng cho các thành phần trong phần mềm hướng thành phần sử dụng: Kiểm chứng mô hình và kiểm chứng dựa trên mô hình

- Giới thiệu về bài toán: Kiểm chứng giả định đảm bảo

- Ưu và nhược điểm của sinh giả định bằng thuật toán học L\*

- Ưu và nhược điểm của sinh giả định bằng thuật toán CDNF

- Vấn đề cần chuyển đổi qua lại giữa 2 phương pháp để tận dụng ưu nhược điểm của mỗi phương pháp

- Trình bày cấu trúc luận văn

# Chương 2: Kiến thức cơ sở (a đã review cho em xong)

## **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 [2])

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 khởi 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 khởi đầu.



Hình 2.1: Một hệ chuyển trạng thái được gán nhãn

***Định nghĩa 2.2*:** Kích thước của một tập hợp [2].

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 [3].

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.

Vết của một LTS M = (Q, , δ, q0) là một chuỗi hữu hạn các sự kiện có dạng σ = ω0ω1…ωn với 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 khởi 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 khởi đầu.

Ta thấy, chuỗi các hành động *openWindow start stop* là một vết của M, bởi vì tại trạng thái khởi 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 khởi 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.

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 [2].

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 khởi 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 khởi 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 [2].

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= αM1  αM2, 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 khởi 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 khởi 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 khởi 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 khởi 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].

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

***Định nghĩa 2.7***: Thuộc tính an toàn [2].

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 tập tất cả các hành vi được đoán nhận trên αp.

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

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**: Ví dụ về một thuộc tính an toàn và một thuộc tính lỗi

Hình 2.8 biểu diễn một thuộc tính an toàn và một thuộc tính lỗi. Thuộc tính lỗi này là kết quả của việc chuyển từ một thuộc tính an toàn.



Hình 2.7: Minh họa một thuộc tính an toàn p và một thuộc tính lỗi tương ứng perr

***Định nghĩa 2.9***: Tính thỏa mãn một thuộc tính của LTS [2]

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 logic (Boolean)**

***Định nghĩa 2.10***: Hàm logic [3].

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

**Ví dụ 2.10:** Ví dụ về hàm logic.

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

***Định nghĩa 2.11*:** Phép gán [3].

Với X là tập hợp các biến logic, 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 [2].

Với Φ(X) là hàm một logic 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 logic, trong đó X’ = {x’ | x ∈ X}, ψ(X, X’) là hàm logic 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}, X’ = {x’} là các tập hợp biến logic, Φ(x) = là một hàm logic trên tập X. Nếu υ(x) = T thì Φ[υ] = F và nếu υ(x) = F thì Φ[υ] = T. Với ψ(x, x’) = x ∨ x’ là một hàm logic trên tập X và X’, nếu υ(x) = T, υ’(x’) = F thì ψ[υ, υ’] = T ∨ F = T.

Một cách tổng quát, với n tập các biến logic X, X1, X2, ..., Xn trong đó Xi = {xi | x ∈ X}, ψ(X, X1, X2, ..., Xn) là hàm logic tương ứng trên các tập biến logic 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 ∈ X1 bởi υ1(x1), x2 ∈ X2 bởi υ2(x2), ... và xn ∈ Xn bởi υn(xn).

***Định nghĩa 2.13***: Dạng đặc tả sử dụng hàm logic.

Dạng đặc tả sử dụng hàm logic 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 logic dùng để biểu diễn các trạng thái của hệ thống. X = {x0, x1, ..., xn},
* E là tập các biến logic dùng để biểu diễn các hành vi của hệ thống. E = {e0, e1, ..., en},
* τ (X, E, X’) là hàm logic biểu diễn việc chuyển trạng thái của hệ thống, và
* ι(X) là hàm logic dùng để biểu diễn các trạng thái khởi đầu của hệ thống.

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

Với dạng đặc tả sử dụng hàm logic 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 logic [3]

Với dạng đặc tả sử dụng hàm logic 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 = γ0 γ1 ...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 logic.

Cho dạng đặc tả sử dụng hàm logic 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(x2) = 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 logic.

Cho dạng đặc tả sử dụng hàm logic 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 logic 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 logic [3]

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

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

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

* X2 là tập các biến logic dùng để biểu diễn các trạng thái của hệ thống. X2 = {},
* E2 là tập các biến logic dùng để biểu diễn các hành vi của hệ thống. E2 = {},
* τ2(X2, E2, X2’) là hàm logic biểu diễn việc chuyển trạng thái của hệ thống, và
* ι2(X2) là hàm logic dùng để biểu diễn các trạng thái khởi 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 logic N = 〈X, E, τ(X,E, X’), ι(X)〉, trong đó:

* X = X1 X2 là tập các biến logic dùng để biểu diễn các trạng thái của hệ thống. X = {},
* E = E1 E2 là tập các biến logic dùng để biểu diễn các hành vi của hệ thống. E = { },
* τ(X, E, X’) là hàm logic 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 logic dùng để biểu diễn các trạng thái khởi 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 logic.**

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 logic [3]**

Một thuộc tính π(X) là một hàm logic trên tập X. Với N là dạng đặc tả sử dụng hàm logic 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 logic.

Xét một hệ thống N được biểu diễn bởi dạng đặc tả sử dụng hàm logic 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 chuỗi ξ, ξ ∈ L(N) và ξ = γ0γ1 = FFFTFF. Khi đó tồn tại các phép gán υ0, υ1, υ2 sao cho τ[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 logic 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 logic N = 〈X, E, τ(X,E, X’), ι(X)〉, trong đó X = X1 ∪ X2 với X1 là tập các biến logic dùng để biểu diễn các trạng thái đầu vào của hệ thống, X2 là tập các biến logic 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 logic 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 logic 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 logic 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 logic 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 khởi đầu.



Hình 2.9: Ví dụ về một LTS.

Và dạng đặc tả sử dụng hàm logic 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 logic 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 X2 trong 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 5fdasf

Bảng 6ffsdfa

Bảng 7

Bảng 8

Bảng 9

Bảng 10

# Chương 3: Các phương pháp sinh giả định

## **3.1. Phương pháp sinh giả định sử dụng thuật toán học L\* [3]**

### **3.1.1. Thuật toán học L\***

Trình bày và giải thích thuật toán

+ Trình bày thuật toán: Đưa ra khái niệm bảng quan sát, bảng quan sát đóng, giải thích tư tưởng của thuật toán

+ Độ phức tạp của thuật toán

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

- Trình bày hai bước sinh giả định Ai dựa theo mô hình:



B1: <Ai> M1 <p>

B2: <true> M2 <Ai>

- Làm thế nào Teacher đưa ra câu trả lời cho L\*

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

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

## **3.2. Phương pháp sinh đặc tả sử dụng thuật toán CNDF [3]**

### **3.2.1. Thuật toán học CNDF**

Trình bày và giải thích thuật toán

+ Trình bày thuật toán: Đưa ra khái niệm bảng quan sát, bảng quan sát đóng, giải thích tư tưởng của thuật toán

+ Độ phức tạp của thuật toán

### **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: Phương pháp chuyển đổi

## **4.1. Phương pháp chuyển đổi**

### **4.1.1. Thuật toán chuyển đổi**

### **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 4.1: 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 logic**

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 khởi tạo.

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 logic dùng để biểu diễn trạng thái khởi tạo 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 logic 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 logic sang dạng đặc tả sử dụng LTS**

Giả sử chúng ta đã có dạng đặc tả sử dụng hàm logic 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 4.6: Thành phần Q1 X1 trong bảng ánh xạ

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

Bảng 4.7: 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 logic 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 khởi tạo 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 4.8: 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 4.9: 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 khởi tạo.

# Chương 5: Xây dựng công cụ chuyển đổi

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, Magnus Larsson, “Component-Based Software Engineering – New Paradigm of Software Development”, Department of Computer Engineering, Mälardalen University.

[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.

[3] 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.

[4] 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

[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", Information and Computation, vol. 75, no. 2, pp. 87-106 (Nov. 1987).

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

[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).

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

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