|  |
| --- |
| 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

Các hệ thống phần mềm đang trở nên ngày càng phức tạp cả về quy mô lẫn chức năng. Để có thể sản xuất các hệ thống phần mềm như vậy một cách hiệu quả cả về chi phí, chất lượng và thời gian các nhà cung cấp hiện nay sử dụng công nghệ dựa trên thành phần thay vì phát triển tất cả các thành phần của hệ thống ngay từ đầu. Lí do ban đầu khi lựa chọn việc sử dụng các thành phần ngoài mục đích giảm chi phí phát triển thì lí do quan trọng nữa đó là để giảm bớt thời gian đưa sản phẩn ra thị trường. Các thành phần có thể được tái sử dụng các thành phần đã có sẵn hoặc được mua từ bên thứ ba hoặc cũng có thể là sử dụng mã nguồn mở. Một câu hỏi đặt ra là việc sử dụng các thành phần như vậy liệu có thể chắc chắn rằng các thành phần đó thỏa mãn yêu cầu của hệ thống. Hiện nay, có một số phương pháp để trả lời câu hỏi này đó là sử dụng kiểm chứng mô hình một phương pháp khai thác tất cả các trạng thái có thể có của thành phần [18] hoặc phương pháp khác kiểm chứng dựa trên mô hình, một phương pháp xây dựng hệ thống dưới dạng mô hình và sinh các ca kiểm thử dựa trên mô hình [19]. Chúng ta có thể thấy vấn đề đối với hai phương pháp này là làm sao sinh được mô hình cho thành phần cẩn kiểm thử.

Mặt khác với các hệ thống được xây dựng dựa trên mô hình hướng thành phần như vậy để chứng minh được hệ thống thỏa mãn một thuộc tính nào đó dựa trên mô hình của từng thành phần sẽ rất mất thời gian và công sức. Phương pháp sinh giả định–đảm bảo được đề cập trong [7] đưa ra phương pháp sinh giả định như là một mô hình của hệ thống và không cần ghép nối mô hình các thành phần của hệ thống. Thay vì việc phải áp dụng các quy tắc ghép nối để sinh ra mô hình cho hệ thống chúng ta sẽ sử dụng các thuật toán học để sinh mô hình cho thành phần hệ thống. Với các nghiên cứu hiện nay, việc sinh mô hình có thể thực hiện theo một số phương pháp bằng cách sử dụng các thuật toán học [3] [4]. Điểm chung của hai phương pháp này là đều tương tác với Teacher, người biết được mô hình cần học và sẽ trả lời các câu hỏi để các thuật toán học này sinh ra được mô hình. Trong [3] thuật toán học sinh ra mô hình một cách tường minh bằng cách biểu diễn mô hình bằng hệ chuyển trạng thái gắn nhãn – LTS, ưu điểm của cách biểu diễn này là mô hình được biểu diễn một cách rất trực quan tuy nhiên đối với các hệ thống lớn thì việc dùng mô hình không mang lại hiệu quả cao vì việc dựng mô hình đòi hỏi chi phí và thời gian không hề nhỏ, hơn nữa việc số trạng thái của hệ thống lớn thì sẽ dẫn đến bài toán bùng nổ không gian trạng thái [6]. Vì vậy việc sinh mô hình trong [3] sẽ rất hiệu quả và phù hợp với các hệ thống vừa và nhỏ. Trái ngược lại với [3], mô hình được sinh ra trong nghiên cứu [4] được biểu diễn một cách không tường minh bằng cách sử dụng hàm logic. Các trạng thái, hành vi của hệ thống sẽ được biểu diễn dưới dạng các biến và các hàm logic. Chúng ta có thể thấy nhược điểm của phương pháp chính là không trực quan như [3] nhưng bù lại khi biểu diễn các hệ thống lớn sẽ mang lại hiệu quả cao hơn do số lượng biến logic cần dùng để biểu diễn n trạng thái của hệ thống chỉ là [*log2n*]. Nhận thấy ưu và nhược điểm của các phương pháp sinh giả định như vậy, luận văn của em đưa ra phương pháp chuyển đổi qua lại giữa các dạng tả hình thức trong phát triển phần mềm. Đối với hệ thống lớn (số trạng thái và hành vi) thì việc áp dụng thuật toán học L\* để sinh mô hình là không hiệu quả, do đó nếu chuyển từ dạng đặc tả sử dụng LTS sang dạng đặc tả hệ thống sử dụng hàm logic rồi mới áp dụng thuật toán sinh mô hình sẽ mang lại hiệu quả tốt hơn. Mặt khác với các hệ thống vừa và nhỏ, việc đặc tả bằng LTS sẽ mang lại tính trực quan hơn, do đó nếu các hệ thống này đang được biểu diễn bằng các hàm logic thì tốt hơn hết là chuyển chúng sang dạng đặc tả bằng LTS rồi sinh giả định sẽ tối ưu hơn.

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 đưa ra ở phía sau. Nội dung chương 3 sẽ trình bày về các thuật toán sinh giả định cho các thành phần phần mềm, bao gồm nội dung của các thuật toán, chi tiết về việc sinh giả định dựa trên các thuật toán đó, đồng thời đưa ra các ví dụ sinh giả định dựa trên các thuật toán này. 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 logic, đồng thời đưa ra ví dụ minh họa cho phương pháp chuyển đổi này. Trong quá trình đưa ra phương pháp chuyển đổi này, em cũng đã cài đặt một công cụ giúp cho việc chuyển đổi qua lại giữa các dạng đặc tả được tự động hóa, đây cũng chính là nội dung chương 5. 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 thảm khảo bao gồm các bài báo, tạp chí được em tìm hiểu trong quá trình nghiên cứu và làm luận văn.

# 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 [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

# Chương 3: Các phương pháp sinh mô hình

Kiểm chứng đảm bảo giả định là một cách tiếp cận chia để trị nhằm giải quyết bài toán bùng nổ không gian trạng thái trong kiểm chứng mô hình. Để chứng minh một hệ chuyển trạng thái thỏa mãn một tính chất nào đó, cách tiếp cận này sinh ra một giả định đại diện cho môi trường hoạt động của hệ thống. Xét trường hợp đơn giản nhất hệ thống lớn ban đầu là một hệ thống M gồm 2 thành phần M1 và M2 (M1 || M2 = M).



Hình 3.1: Bài toán kiểm chứng giả định–đảm bảo

Bài toán đưa ra là kiểm chứng *hệ thống M có thỏa mãn thuộc tính p nào đó hay không mà không cần ghép nối M1 với M2*. Để làm được điều này, một ngữ cảnh giả định A(p) được sinh ra bằng cách áp dụng một số thuật toán học sao cho:

<A(p)> M1 <p> (1)

<true> M2 <A(p)> (2)

Hiện nay, việc sinh giả định có thể sử dụng các phương pháp sau:

* Phương pháp sinh giả định tường minh: Sử dụng thuật toán học L\*
* Phương pháp sinh giả định không tường minh: Sử dụng thuật toán học CDNF

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

Phương pháp sinh giả định sử dụng thuật toán học L\* được đề xuất bởi Angluin [9] và sau đó được cải tiến bởi Rivest and Schapire [17]. 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 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 3.2 dưới đây mô tả sự tương tác giữa L\* và Teacher.



Hình 3.2: Mô hình sự tương tác giữa L\* và Teacher

### **3.1.1. Thuật toán học L\***

Với T là một bảng ghi chép, 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* và *nhất quán* thành một ứng viên.

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 3.3: Xây dựng một ứng viên từ bảng quan sát *đóng* và *nhất quán*.

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].

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

Để sinh ra giả định hợp lí, phương pháp áp dụng được minh họa như trên hình 3.5. Ở mỗi bước lặp i, giả định ứng viên Ai được sinh ra ra dựa trên một số hiểu biết về hệ thống và kết quả của bước lặp trước đó. Việc sinh Ai được thực hiện theo hai bước như sau. Bước 1 kiểm tra xem M1 có thỏa mãn p trong biểu thức <Ai> M1 <p>. Nếu kết quả là false, điều này có nghĩa ứng viên giả định quá yếu. Do đó, giả định ứng viên Ai phải đươc tăng cường, điều này có nghĩa phải loại bỏ các hành vi trong Ai, phản ví dụ cex sẽ được sinh ra tại bước này. Trong ngữ cảnh của giả định ứng viên tiếp theo là Ai+1, M1 nên ít nhất là không vi phạm các hành vi được phản ánh bởi cex. Ngược lại nếu kết quả là true, điều này có nghĩa là Ai là đủ mạnh để M1 thỏa mãn thuộc tính p.

Bước thứ 2 được áp dụng để kiểm tra nếu mô hình thành phần M2 thỏa mã Ai theo công thức <true> M2 <Ai>. Nếu bước này kết quả trả về là true, thì hệ thống ghép nối M1||M2 thỏa mãn thuộc tính p (M1||M2 |= p) và thuật toán kết thúc. Ngược lại, nếu kết quả là false thì cần có thêm sự phân tích để kiểm tra xem thuộc tính p có thật sự bị vi phạm trong hệ thống M1||M2 hay giả định ứng viên Ai là quá mạnh để M2 thỏa mãn. Phân tích này là hoàn toàn dựa trên phản ví dụ cex được trả lại bởi Teacher. Teacher sẽ kiểm tra xem phản ví dụ cex có thuộc về ngôn ngữ chưa biết U = L(Aw) hay không (cex ϵ L(Aw)?). Để làm điều này, Teacher sẽ chuyển phản ví dụ cex thành một LTS (kí hiệu là [cex]) như sau: Với phản ví dụ cex = a1a2…ak. LTS [cex] được tạo từ phản ví dụ cex được biểu diễn như sau:

[cex] = <Q, α[cex], δ, q0>, trong đó

* Q = {q0, q1, …, qk},
* α[cex] = {a1, a2, …, ak},
* δ = {(qi-1, ai, qi) | 1 ≤ i ≤ k}, và
* q0 = q0

Hình 3.4 minh họa LTS [cex] được tạo từ phản ví dụ cex. Nếu thuộc tính p không thỏa mãn trong hệ thống ghép nối [cex] || M1 ([cex] || M1 ⊭p), điều này có nghĩa là hệ thống ghép nối M1||M2 không thỏa mãn thuộc tính p (M1||M2 ⊭ p). Ngược lại, Ai là quá mạnh để được thỏa mãn bởi M2. Ứng viên giả định Ai cần được làm cho yếu đi trong vòng lặp tiếp theo. Kết quả của việc làm yếu đi này là ít nhất hành vi thể hiện trong phản ví dụ cex sẽ được chấp nhận bởi giả định ứng viên Ai+1.

**…**

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



Hình 3.5: Mô hình sinh giả định dựa trên thuật toán học L\*

Chúng ta đã biêt làm thế nào Teacher có thể trả lời được các câu hỏi từ phía L\*. Vậy thì thuật toán học học L\* làm việc như thế nào hay một câu hỏi tương tự là sinh giả định ứng viên Ai như thế nào tại mỗi bước lặp i trong hình 3.5. Trong phương thức đảm bảo giả định được đề xuất trong [10], L\* học ngôn ngữ của giả định yếu nhất Aw. Ý nghĩa của điều này là L\* học ngôn ngữ chưa biết U = L(Aw) thông qua ngôn ngữ Σ = αAw = (αM1 ∪ α) ∩ αM2. Phương pháp sử dụng các ứng viên được sinh ra bởi thuật toán học L\* làm các giả định ứng viên Ai. Để đưa ra mỗi ứng viên giả định Ai, đầu tiên L\* sinh ra một ứng viên DFA Mi dựa trên bảng quan sát đóng S, E, T, sau đó DFA Mi sẽ được chuyển sang LTS an toàn để làm các giả định ứng viên Ai bằng cách áp dụng định nghĩa được trình bày ở chương 2. Để học Aw, chúng ta cần cung cấp một Teacher, Teacher này có khả năng trả lời 2 loại câu hỏi mà L\* yêu cầu. Loại câu hỏi đầu tiên là câu hỏi truy vấn thành viên, một chuỗi σ có thuộc Σ\* hay không. Câu trả lời là true nếu σ thuộc U, và false nếu σ không thuộc U. Loại câu hỏi thứ hai là một phỏng đoán, một ứng viên DFA Mi sở hữu ngôn ngữ thuật toán được tin tưởng giống với U. Câu trả lời là true nếu L(Mi) = U, ngược lại Teacher trả lại 1 phản ví dụ, phản ví dụ là 1 chuỗi σ đối xứng khác nhau giữa L(Mi) và U. Phương pháp tiếp cận này sử dụng kiểm chứng mô hình để cài đặt một Teacher.

Với kiểu câu hỏi đầu tiên, để trả lời một truy vấn thành viên cho chuỗi σ = a1a2…an xem trong σ 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] = ∑, δ = {(qi-1, ai, qi) | 1 ≤ i ≤ n}, và q0 = q0. Teacher sau đó kiểm tra công thức <[σ]> M1 <p> bằng cách tính toán hệ thống ghép nối [σ]||M1||perr. Nếu không có trạng thái nào trong hệ thống ghép nối đến được trạng thái lỗi 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.

Với kiểu câu hỏi thứ hai, 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)?). Cho mục tiêu 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 1 được minh họa như trong hình 3.5, Teacher sẽ kiểm tra biểu thức <Ai> M <p> bằng cách ghép nối Ai||M||perr. Nếu tồn tại một dẫn xuất nào đó chứa trạng thái lỗi π thì biểu thức <Ai> M <p> trả về false. Nên teacher trả về kết quả là false và 1 phản ví dụ cex. Teacher trả về cho L\* là phỏng đoán Ai không chính xác và cung cấp phản ví dụ cex phù hợp hơn. Ngược lại, Teacher trả lời là True và chuyển sang bước 2.

Bước 2 Teacher kiểm tra biểu thức <true> M2 <Ai­> như trong hình 3.5 Nếu biểu thức này trả về kết quả là True. Framework của chúng ta kết thúc việc kiểm chứng bởi vì dựa trên luật ghép nối, M1||M2 ||= p. Ngược lại bước này trả về 1 phản ví dụ cex. Teacher sẽ tiến hành phân tích để xác định xem p có thật sự vi phạm M1 || M2 hay Ai có quả quá mạnh để M2­ thỏa mãn. Việc phân tích phản ví dụ được thực hiện bởi Teacher bằng cách tương tự các sử dụng để trả lời truy vấn thành viên. Với cex là phản ví dụ được trả lại bởi bước 2. Đầu tiên, Teacher sẽ tạo 1 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 1 dẫn xuất nào đó 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. cex ↑ Σ được trả lại như một phản ví dụ cho phỏng đoán Ai

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

Giả sử chúng ta có 3 LTS như trên hình dưới đây

in



Hình 3.6: LTS Input (M1)



Hình 3.7: LTS Order



Hình 3.8: LTS Output (M2)

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

Bước 1. Thực hiện ghép nối LTS Output và Perr ta được



Hình 3.9: Mô hình ghép nối Input và Order

Xây dựng bảng quan sát (S, E, T), kết quả từ cột E có được là nhờ các truy vấn thành viên từ L\*, teacher sẽ kiểm tra trong mô hình ghép nối Input và Order ở hình 3 và đưa ra kết quả.

S

S. Σ

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

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

Ta nhận thấy bảng quan sát này chưa đóng, đưa out lên trên tập S

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 |

Từ bảng quan sát trên LTS tương ứng được sinh ra là A1 có dạng

ack

send

Hình 4. Mô hình LTS A1 ra từ bảng quan sát

Ta tiến hành kiểm tra tính đúng đắn của biểu thức <A1> M1 <P>.

Thực hiện việc ghép nối A1, M1, và P ta được , ta thấy A1||Input ≡ Input

out

in

send

out

in

out

in

out

ack

ack

send

out

in

out

Hình 7. Mô hình ghép nối A1||Input||Order

Chúng ta thấy tồn tại dẫn xuất đến trạng thái lỗi π là <in, send, ack, out>. Chúng ta dùng luôn dẫn xuất này làm phản ví dụ. Ta có, cex = <in, send, ack, out>

Phản ví dụ mà teacher trả lại cho L\* có dạng cex↑Σ = <send, ack>

Đưa send vào tập tiền tố send và ack vào tập hậu tố ta được bảng quan sát như sau:

S

S. Σ

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

Bảng này đã đóng, DFA A2 sinh ra sẽ có dạng

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

3.2.1. Thuật toán học CNDF

Với X là tập các biến logic và một hàm logic λ(X) trên tập X. Thuật toán học tính toán một đại diện của λ(X) trong một số bước hữu hạn. Thuật CDNF là một thuật toán học chính xác cho các hàm logic. Giống như thuật toán L\*, thuật toán CDNF sử dụng một mô hình học chủ động. Trong mô hình, giả sử có một Teacher, là người biết hàm logic mục tiêu λ(X), Teacher sẽ trả lời các câu hỏi cho thuật toán CNDF:

* Câu hỏi truy vấn ứng viên MEM(v) cho hàm mục tiêu λ(X) với υ là phép gán trên tập X. Nếu λ[υ] = T, teacher sẽ trả lời YES, ngược lại Teacher sẽ trả lời NO.
* Câu hỏi truy vấn tương đương EQ(θ) cho hàm mục tiêu λ(X) với θ(X) là hàm logic trên tập X. Nếu phỏng đoán θ(X) là tương đương với hàm logic mục tiêu λ(X), Teacher trả lời YES. Ngược lại Teacher sẽ đưa ra một phép gán υ trên tập X sao cho θ[υ] ≠ λ[υ] . Phép gán υ chính là một phản ví dụ mà Teacher trả lại cho truy vấn tương đương EQ(θ).

Ví dụ:

Giả sử λ(x, y) = là một hàm logic mục tiêu trên x, y. Kí hiệu y) = FF tương đương với ) = ) = T. Với truy vấn thành viên MEM(υ) mà y) = FF thì λ[= F nên Teacher trả lời là NO. Với truy vấn thành viên MEM(υ) mà y) = TF thì λ[= T nên Teacher trả lời là YES. Với phỏng đoán θ(x, y) = , truy vấn tương đương EQ(θ(x, y)), Teacher trả lời NO và đưa ra phép gán y) = TT như là một phản ví dụ vì θ[] = T T = T ≠ λ[] (λ[] = (T F) (F T) = F). Với thì với truy vấn tương đương EQ(θ(x, y)) Teacher sẽ trả lời là YES.

Thuật toán 3.2:

# Chương 4: Phương pháp chuyển đổi

Chương 3 đã trình bày rõ phương pháp sinh giả định dựa vào 2 thuật toán L\* và CDNF

4.1. Phương pháp chuyển đổi

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

## **4.3. Giới thiệu về hệ thống**

4.2. Ví dụ minh họa

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 |

Bảng 4.1: Các chuyển trạng thái của tiến trình

**4.2 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 4.1. Đồng thời sau khi mã hóa

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

Bảng 4.2: Thành phần Q1 X1 trong bảng ánh xạ

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 sau:

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

Bảng 4.3: 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 4.4: 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:

|  |  |  |
| --- | --- | --- |
| 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 4.5: Thành phần δ(q, e, q’) τ (υ, γ, υ’) trong bảng ánh xạ

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.

**4.3 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. (\*)

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

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). (\*\*\*\*)

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.

3.1 Thuật toán

Đầu vào (Input): Một LTS.

Đầu ra (Output): Dạng đặc tả sử dụng hàm logic 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.

3.1.1 Thuật toán mã hóa một tập hợp

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 log2(|A|) là số nguyên then

5 z = log2(|A|)

6 else

7 z = [log2(|A|)] + 1

8 end

9 end

10 for mỗi phần tử ai trong tập A do

11 Lưu ai vào bảng ánh xạ ứng với thành phần A

12 αi = True

13 k = Thứ tự của ai 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 αi = αi ∧ xj

18 else

19 αi = αi ∧ xj

20 end

21 end

22 Lưu αi vào bảng ánh xạ ứng với vị trí của phần tử ai 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 logic cần dùng để mã hóa tất cả các phần tử của tập hợp A, gọi z là số biến logic cần dùng. Gọi X = {x1, x2, ..., xz} là tập các biến logic 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 logic 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.

3.2 Chứng minh

Với thuật toán mã hóa đưa ra ở chương phía trước chúng ta hoàn toàn có thể chuyển đổi một dạng đặc tả sử dụng LTS sang dạng đặc tả sử dụng hàm logic và ngược lại từ dạng đặc tả sử dụng hàm logic kết hợp với bảng ánh xạ để chuyển đổi sang dạng đặc tả sử dụng LTS. Tuy nhiên, hai dạng đặc tả này sau khi chuyển đổi liệu có tương đương với nhau? Để chứng minh ngôn ngữ của dạng đặc tả được biểu diễn bằng LTS tương đương với ngôn ngữ của dạng đặc tả sử dụng hàm logic, chúng ta cần chứng minh 2 mệnh đề:

• Mệnh đề 1: Ngôn ngữ của dạng đặc tả hệ thống biểu diễn bằng LTS sau khi chuyển đổi sang dạng đặc tả biểu diễn bằng hàm logic được đoán nhận bởi dạng đặc tả sử dụng hàm logic. (1)

• Mệnh đề 2: Ngôn ngữ của dạng đặc tả sử dụng hàm logic sau khi chuyển sang dạng đặc tả biểu diễn bằng LTS được đoán nhận bởi LTS.

Chứng minh mệnh đề 1: Với một LTS M = hQ, Σ, δ, q0i, L(M) là ngôn ngữ của của M và một dạng đặc tả sử dụng hàm logic N = hX, E, τ (X,E, X’), ι(X)i, L(N) là ngôn ngữ của N, N được chuyển đổi từ M. Với α = w0w1...wn là vết của LTS M, áp dụng thuật toán 3.1 ta thu được ξ = γ 0γ 1 ...γn, ta cần chứng minh ξ được đoán nhận bởi L(N). Thật vậy, xét q0 là trạng thái khởi đầu của M, áp dụng thuật toán 3.1, chúng ta sẽ mã hóa q0 thành α0. Vì α0 là một hàm logic nên tồn tại một phép gán υ 0 cho hàm α0 trên tập X sao cho ι[υ 0 ] = T. (\*) Mặt khác, gọi δ(qi , wi , qi+1) là một chuyển trạng thái bất kì trong tập các chuyển trạng thái của LTS, áp dụng thuật toán 3.2, δ(qi , wi , qi+1) sẽ được mã hóa thành τ (αi , ei , αi+1). Vì αi , ei , αi+1 là hội của các biến logic nên tồn tại các phép gán υ i cho hàm αi trên tập X, υ i+1 cho hàm αi+1 trên tập X’, γ i là phép gán cho hàm ei trên tập E sao cho υ i = T, υ i+1 = T, γ i = T để τ [υ i , γi , υi+1] = T (\*\*). Từ (\*), (\*\*) và định nghĩa 2.10, ξ = γ 0γ 1 ...γn là một vết của N hay ξ ∈ L(N). Chứng minh mệnh đề 2: Với một LTS M = hQ, Σ, δ, q0i, L(M) là ngôn ngữ của của M và một dạng đặc tả sử dụng hàm logic N = hX, E, τ (X,E, X’), ι(X)i, L(N) là ngôn ngữ của N và N được chuyển đổi từ M. Gọi Map là bảng ánh xạ lưu các ánh xạ khi chuyển đổi từ M sang N. Gọi αi là dạng mã hóa của trạng thái qi . Gọi ξ = γ 0γ 1 ...γn là một vết của N. Trong đó, γ i là phép gán cho hàm e i biểu diễn hành vi thứ i của hệ thống. Theo định nghĩa vết của dạng đặc tả đặc tả sử dụng hàm logic 2.10 vì ξ = γ 0γ 1 ...γn là một vết của N nên tồn tại các phép gán υ 0 , υ 1 , ..., υ n+1 , γ 0 , γ 1 , ..., γ n sao cho ι[υ 0 ] = T và τ [υ i , γi , υi+1] = T với ∀i: 0 ≤ i ≤ n. Vì ι[υ 0 ] = T nên α0[υ 0 ] = T mà α0 lại là dạng mã hóa của trạng thái q0 nên q0 là trạng thái khởi đầu của M. Thêm vào đó, vì τ [υ i , γi , υi+1] = T nên dựa vào bảng ánh xạ Map ta thu được các hàm chuyển trạng thái δ(qi , wi , qi+1) tương ứng. Vì 0 ≤ i ≤ n nên ta có chuỗi các sự kiện σ = w0w1...wi là một vết của M. Với q0 là trạng thái khởi đầu, sau khi sự kiện w0 xảy ra hệ thống chuyển sang trạng thái q1 (do sự tồn tại của chuyển trạng thái δ(q0, w0, q1)), từ trạng thái q1 khi xảy ra sự kiện w1 hệ thống chuyển sang trạng thái q2 (do sự tồn tại của chuyển trạng thái σ(q1, w1, q2)), một cách tương tự với các sự kiện từ w2 đến wi−1, sau khi sự kiện wi−1 xảy ra hệ thống chuyển sang trạng thái qi , khi sự kiện wi xảy ra, do sự tồn tại của chuyển trạng thái σ(qi , wi , qi+1) nên hệ thống chuyển sang trạng thái qi+1. Vì thế, với ∀i: 0 ≤ i ≤ n, tồn tại trạng thái qi ∈ Q sao cho q0 σ→ qi nên theo định nghĩa 2.4, σ = w0w1...wn là một vết của M.

Chương 4 Ví dụ về việc chuyển đổi giữa các dạng đặc tả 4.1 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 15 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 đầu, 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 Bảng 4.1: Các chuyển trạng thái của tiến trình 4.2 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 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) tta 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 7→ X1, theo bước (12), khởi đầu 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 ∧ x1, bit tiếp theo là bit thứ 2, giá trị của bit này bằng 0, nên theo bước (17) α0 = α0 ∧ x1 ∧ x2 = x1 ∧ x2. 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 7→ X1. Một cách hoàn toàn tương tự, chúng ta thu được dạng mã hóa của trạng thái Ready là α1 = x1 ∧ x2, dạng mã hóa của trạng thái Running là α2 = x1 ∧ x2 và dạng mã hóa của trạng thái Blocked là α3 = x1∧x2. Sau khi mã hóa xong tập các trạng thái Q1 ta thu đươc thành phần Q1 7→ X1 của bảng ánh xạ như bảng 4.1. Đồng thời sau khi mã hóa Q1 New Ready Running Blocked X1 x1 ∧ x2 x1 ∧ x2 x1 ∧ x2 x1 ∧ x2 Bảng 4.2: Thành phần Q1 7→ X1 trong bảng ánh xạ xong tập các trạng thái đầu vào Q1 , chúng ta cũng thu được hàm logic dùng để biểu diễn trạng thái khởi đầu của hệ thống (cũng chính là dạng mã hóa của trạng thái q0) là ι(X) = x1 ∧ x1. 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 7→ X2 và thành phần Σ 7→ E của bảng ánh xạ như sau: Q2 Ready Running Exit Blocked X2 x6 ∧ x7 x6 ∧ x7 x6 ∧ x7 x6 ∧ x7 Bảng 4.3: Thành phần Q2 7→ X2 trong bảng ánh xạ Σ admit dispatch release E x3 ∧ x4 ∧ x5 x3 ∧ x4 ∧ x5 x3 ∧ x4 ∧ x5 Σ timeout waitEvent eventOccurs E x3 ∧ x4 ∧ x5 x3 ∧ x4 ∧ x5 x3 ∧ x4 ∧ x5 Bảng 4.4: Thành phần Σ 7→ 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 7→ X1 của bảng ánh xạ là x1 ∧x2. Theo bước (3), ta lấy được dạng biểu diễn của sự kiện admit từ thành phần Σ 7→ E của bảng ánh xạ là x3 ∧ x4 ∧ x5. Theo bước (4), ta lấy được dạng biểu diễn của Ready từ thành phần Q2 7→ X2 của bảng ánh xạ là x6∧x7. Theo bước (5), ta thu được dạng mã hóa của chuyển trạng thái (New, admit, Ready) là x1∧x2∧x3∧x4∧x5∧x6∧x7. 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’) = {x1 ∧x2 ∧x3 ∧x4 ∧x5 ∧x6 ∧x7} | {x1 ∧x2 ∧x3 ∧x4 ∧x5 ∧x6 ∧x7} | {x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7} | {x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7} | {x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7} | {x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7}. Và thành phần δ(q, e, q’) 7→ τ (υ, γ, υ’) của bảng ánh xạ: δ(q, e, q’) (New, admit, Ready) (Ready, dispatch, Running) τ (υ, γ, υ’) x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7 x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7 δ(q, e, q’) (Running, release, Exit) (Running, timeout, Ready) τ (υ, γ, υ’) x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7 x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7 δ(q, e, q’) (Ready, waitEvent, Blocked) (Blocked, eventOccurs, Ready) τ (υ, γ, υ’) x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7 x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7 Bảng 4.5: Thành phần δ(q, e, q’) 7→ τ (υ, γ, υ’) trong bảng ánh xạ Cuối cùng, kết quả sau khi chuyển đổi chúng ta thu được dạng đặc tả sử dụng hàm logic N = hX, E, τ (X,E, X’), ι(X)i và bảng ánh xạ (Bảng 4.2, 4.3, 4.4 và 4.5). Với N = hX, E, τ (X,E, X’), ι(X)i. Trong đó: • X = {x1, x2}, X’ = {x6, x7}, • E = {x3, x4, x5}, • τ (X,E, X’) = {x1∧x2∧x3∧x4∧x5∧x6∧x7} | {x1∧x2∧x3∧x4∧x5∧x6∧x7} | {x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7} | {x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7} | {x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7} | {x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7}, và • ι(X) = x1 ∧ x2. 4.3 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 = hX, E, τ (X,E, X’), ι(X)i. Trong đó: • X = {x1, x2}, X’ = {x6, x7}, • E = {x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7} | {x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7} | {x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7} | {x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7} | {x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7} | {x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7}}, và • ι(X) = x1 ∧ x2. Và bảng ánh xạ Map: Q1 New Ready Running Blocked X1 x1 ∧ x2 x1 ∧ x2 x1 ∧ x2 x1 ∧ x2 Bảng 4.6: Thành phần Q1 7→ X1 trong bảng ánh xạ Q2 Ready Running Exit Blocked X2 x6 ∧ x7 x6 ∧ x7 x6 ∧ x7 x6 ∧ x7 Bảng 4.7: Thành phần Q2 7→ X2 trong bảng ánh xạ Σ admit dispatch release E x3 ∧ x4 ∧ x5 x3 ∧ x4 ∧ x5 x3 ∧ x4 ∧ x5 Σ timeout waitEvent eventOccurs E x3 ∧ x4 ∧ x5 x3 ∧ x4 ∧ x5 x3 ∧ x4 ∧ x5 Bảng 4.8: Thành phần Σ 7→ E trong bảng ánh xạ δ(q, e, q’) (New, admit, Ready) (Ready, dispatch, Running) τ (υ, γ, υ’) x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7 x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7 δ(q, e, q’) (Running, release, Exit) (Running, timeout, Ready) τ (υ, γ, υ’) x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7 x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7 δ(q, e, q’) (Ready, waitEvent, Blocked) (Blocked, eventOccurs, Ready) τ (υ, γ, υ’) x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7 x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7 Bảng 4.9: Thành phần δ(q, e, q’) 7→ τ (υ, γ, υ’) 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 7→ X1 ta thấy x1 ∧ x2 là dạng mã hóa của trạng thái q0 mà ι(X) = x1 ∧ x2 nên q0 là trạng thái khởi đầu của LTS. (\*) Lại có, dựa vào thành phần Q1 7→ X1, Q2 7→ X2, ta thấy Q1 có 4 trạng thái 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 7→ E, ta thấy LTS có 6 sự kiện Σ = {admit, dispatch, release, timeout, waitEvent, eventOccurs} (\*\*\*). Mặt khác τ (X,E, X’) = {x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7} | {x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7} | {x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7} | {x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7} | {x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7} | {x1 ∧ x2 ∧ x3 ∧ x4 ∧ x5 ∧ x6 ∧ x7}}. Xét lần lượt từng phần tử trong τ (X,E, X’), với τ (υ, γ, υ’) = {x1 ∧x2 ∧x3 ∧x4 ∧x5 ∧x6 ∧x7} (kể cả khi thứ tự các biến mã hóa bị thay đổi). Đối chiếu với thành phần Σ 7→ 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). (\*\*\*\*) Từ (\*) (\*\*) (\*\*\*) và (\*\*\*\*) ta được LTS M như hình 4.1. M = hQ, Σ, δ, q0i 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 đầu.

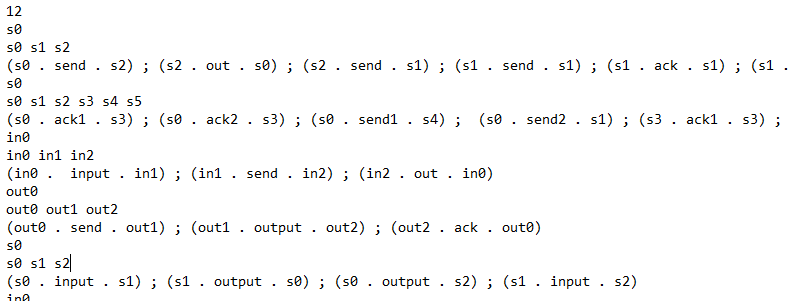
# Chương 5: Xây dựng công cụ chuyển đổi

Công cụ chuyển đổi từ dạng đặc tả sử dụng LTS sang dạng đặc tả sử dụng hàm logic, được cài đặt bằng ngôn ngữ lập trình C# trên công cụ Visual Studio 2015.

Đầu vào là một file text với nội dung như sau:

* Dòng 1: Số lượng LTS cần chuyển sang hệ chuyển trạng thái biểu diễn bằng hàm logic
* Dòng i\*3 + 2: Tên trạng thái khởi tạo
* Dòng i\*3 + 2: Tên các trạng thái của LTS
* Dòng i\*3 + 3: Biểu diễn các chuyển trạng thái, các chuyển trạng thái được biểu diễn như sau: (tên trạng thái bắt đầu . tên sự kiện . tên trạng thái kết thúc). Tập các chuyển trạng thái của LTS được biểu diễn như sau (chuyển trạng thái) ; (chuyển trạng thái) …

Ví dụ về nôi dung của file text đầu vào cho công cụ được mô tả như ở hình 5.1.

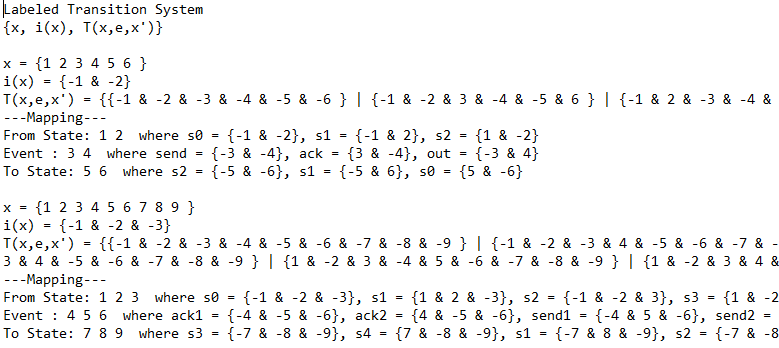


Hình 5.1: Ví dụ về file text đầu vào của công cụ chuyển đổi

Đầu ra của công cụ được biểu diễn như sau:

* 2 dòng đầu mô tả thông tin của hệ chuyển trạng thái sử dụng hàm logic
* 1 dòng trống
* Dòng i\*8 + 4: Tập các biến logic dùng để biểu diễn tập các trạng thái
* Dòng i\*3 + 5: Tập các biến logic dùng để biểu diễn tập các sự kiện
* Dòng i\*3 + 6: Dạng biểu diễn bằng hàm logic của tập các chuyển trạng thái
* Từ dòng i\*8 + 7 đến dòng i\*8 + 10: Thông tin về bảng ánh xạ

Hình 5.2 dưới đây mô tả thông tin về file text đầu ra của công cụ chuyển đổi



Hình 5.2: Ví dụ file text đầu ra của công cụ chuyển đổi

# Chương 6: KẾT LUẬN

Fdsaf

# 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.