# 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