|  |
| --- |
| ĐẠI HỌC QUỐC GIA HÀ NỘI  **TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**  **TRẦN HOÀNG VIỆT**  **PHƯƠNG PHÁP SINH MÔ HÌNH TỰ ĐỘNG CHO PHẦN MỀM DỰA TRÊN THÀNH PHẦN**  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**: **TS. NGUYỄN THẾ LỘC**    Hà nội – 2013 |

# MỤC LỤC

[MỤC LỤC i](#_Toc372056297)

[LỜI CẢM ƠN iii](#_Toc372056298)

[LỜI CAM ĐOAN iv](#_Toc372056299)

[DANH MỤC THUẬT NGỮ VIẾT TẮT v](#_Toc372056300)

[DANH MỤC HÌNH VẼ vi](#_Toc372056301)

[DANH MỤC BẢNG viii](#_Toc372056302)

[Chương 1: Giới thiệu 1](#_Toc372056303)

[Chương 2: Các phương pháp hình thức cho đặc tả phần mềm 5](#_Toc372056304)

[2.1 Hệ thống chuyển trạng thái được gán nhãn 5](#_Toc372056305)

[2.2 Phép ghép nối song song 8](#_Toc372056306)

[2.3 Hệ thống chuyển trạng thái gán nhãn an toàn, thuộc tính an toàn, tính thỏa mãn 10](#_Toc372056307)

[2.4 Thành phần phần mềm và ôtômát hữu hạn trạng thái 13](#_Toc372056308)

[Chương 3: Các phương pháp sinh mô hình tự động cho thành phần phần mềm 17](#_Toc372056309)

[3.1 Phương pháp sinh mô hình dựa trên thuật toán học L\* 18](#_Toc372056310)

[3.1.1 Phương pháp sinh mô hình sử dụng thuật toán L\* 19](#_Toc372056311)

[3.1.2 Thuật toán Vasilevskii-Chow 22](#_Toc372056312)

[3.2 Phương pháp sinh mô hình sử dụng thuật toán Thompson 23](#_Toc372056313)

[3.3 Hạn chế của các phương pháp sinh mô hình tự động theo thuật toán L\* và Thompson 29](#_Toc372056314)

[Chương 4: Nghiên cứu phương pháp sinh mô hình tự động cho thành phần phần mềm sử dụng thuật toán CNNFA 31](#_Toc372056315)

[4.1 Một số khái niệm liên quan 32](#_Toc372056316)

[4.2 Sinh biểu diễn CNNFA cho các biểu thức chính quy thành phần 35](#_Toc372056317)

[4.3 Phương pháp duyệt biểu thức chính quy 37](#_Toc372056318)

[4.4 Sinh mô hình cho thành phần phần mềm 39](#_Toc372056319)

[4.5 Tối ưu hóa mô hình 40](#_Toc372056320)

[4.6 Ví dụ sinh mô hình cho thành phần phần mềm bằng thuật toán CNNFA 44](#_Toc372056321)

[4.6.1 Xây dựng NFA bằng thuật toán CNNFA 44](#_Toc372056322)

[4.6.2 Đơn định hóa NFA 52](#_Toc372056323)

[4.6.3 Tối thiểu hóa DFA 52](#_Toc372056324)

[Chương 5: Thực nghiệm 54](#_Toc372056325)

[5.1 Công cụ sinh mô hình tự động cho phần mềm dựa trên thành phần 54](#_Toc372056326)

[5.2 Thực nghiệm 55](#_Toc372056327)

[5.3 Ý nghĩa của công cụ thực nghiệm 63](#_Toc372056328)

[Chương 6: KẾT LUẬN 64](#_Toc372056329)

[TÀI LIỆU THAM KHẢO 67](#_Toc372056330)

# 

# LỜI CẢM ƠN

Trước tiên tôi xin gửi lời cảm ơn chân thành và sâu sắc đến thầy giáo, TS. Nguyễn Thế Lộc và thầy giáo, TS. Phạm Ngọc Hùng – người đã 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.

Tôi xin trân trọng cảm ơn đề tài mã số QG.12.50 đã tạo điều kiện cho tôi nghiên cứu trong suốt quá trình thực hiện luận văn này.

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 sinh mô hình tự động cho phần mềm dựa trên thành phần” 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 12 tháng 11 năm 2013

Trần Hoàng Việt

# 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 | NFA | Nondeterministic Finite Automata. | Ôtômát hữu hạn không đơn định. |
| 3 | DFA | Deterministic Finite Automata | Ôtômát hữu hạn đơn định. |
| 4 | NNFA | Normal Nondeterministic Finite Automata. | Tên gọi của một loại ôtômát hữu hạn không đơn định |
| 5 | MYNNFA | McNaughton/Yamada NNFA. | Tên gọi của một loại ôtômát hữu hạn không đơn định theo McNaughton và Yamada. |
| 6 | CNNFA | Compressed NNFA. | Tên gọi của một phương pháp chuyển biểu thức chính quy về ôtômát hữu hạn không đơn định. |

# DANH MỤC HÌNH VẼ

[Hình 2.1: Một hệ thống chuyển trạng thái được gán nhãn. 6](#_Toc372056331)

[Hình 2.2: Một LTS không đơn định. 7](#_Toc372056332)

[Hình 2.3: Một LTS đơn định. 7](#_Toc372056333)

[Hình 2.4: Minh họa vết của một LTS. 8](#_Toc372056334)

[Hình 2.5: Ghép nối song song hai LTS. 10](#_Toc372056335)

[Hình 2.6: LTS của thuộc tính p và LTS lỗi tương ứng của p. 11](#_Toc372056336)

[Hình 2.7: Xây dựng LTS ghép nối song song LTS1||LTS2||perr. 12](#_Toc372056337)

[Hình 2.8: Biểu diễn ôtômát hữu hạn. 14](#_Toc372056338)

[Hình 2.9: Chuyển một ôtômát hữu hạn M thành LTS L. 16](#_Toc372056339)

[Hình 3.1: Mô hình thành phần phần mềm trong phương pháp sinh mô hình theo L\*. 19](#_Toc372056340)

[Hình 3.2: Xây dựng một ứng viên từ bảng quan sát *đóng* và *nhất quán*. 20](#_Toc372056341)

[Hình 3.3: Mô hình giả thiết về thành phần phần mềm C. 23](#_Toc372056342)

[Hình 3.4: Ôtômát tương ứng cho thành phần ε. 26](#_Toc372056343)

[Hình 3.5: Ôtômát tương ứng cho thành phần a. 26](#_Toc372056344)

[Hình 3.6: Ôtômát tương ứng cho thành phần (s).(r). 27](#_Toc372056345)

[Hình 3.7: Ôtômát tương ứng cho thành phần (s) ∪ (r). 27](#_Toc372056346)

[Hình 3.8: Ôtômát cho các biểu thức chính quy cơ sở. 28](#_Toc372056347)

[Hình 3.9: Ôtômát cho thành phần engineOn.engineOff. 28](#_Toc372056348)

[Hình 3.10: Ôtômát cho thành phần engineOn.engineOff.on. 29](#_Toc372056349)

[Hình 3.11: Ôtômát cho thành phần engineOn.engineOff.off. 29](#_Toc372056350)

[Hình 3.12: Ôtômát tương đương với L. 29](#_Toc372056351)

[Hình 4.1: Kiến trúc kiểm chứng mô hình đảm bảo giả định [5]. 31](#_Toc372056352)

[Hình 4.2: Mô hình thành phần phần mềm cho bởi biểu thức chính quy. 32](#_Toc372056353)

[Hình 4.3: Khối cơ bản và khối không cơ bản của biểu thức chính quy. 33](#_Toc372056354)

[Hình 4.4: MYNNFA tương đương của biểu thức chính quy abb(a|b)\*. 34](#_Toc372056355)

[Hình 4.5: Ôtômát đuôi của MYNNFA tương đương với biểu thức chính quy abb(a|b)\*. 34](#_Toc372056356)

[Hình 4.6: Tính toán δ từ lazyδ. 40](#_Toc372056357)

[Hình 4.7: Ôtômát M tương đương của biểu thức chính quy (a|b)\*b. 43](#_Toc372056358)

[Hình 4.8: Ôtômát ở hình 4.7 sau khi đơn định hóa. 43](#_Toc372056359)

[Hình 4.9: Ôtômát đơn định tối thiểu tương đương của biểu thức chính quy (a|b)\*b. 44](#_Toc372056360)

[Hình 4.10: Ôtômát đuôi dạng nén theo thuật toán CNNFA. 51](#_Toc372056361)

[Hình 4.11: Ôtômát không đơn định theo thuật toán CNNFA. 52](#_Toc372056362)

[Hình 4.12: Ôtômát sau khi đã đơn định hóa. 52](#_Toc372056363)

[Hình 4.13: Ôtômát đơn định, tối thiểu cuối cùng. 53](#_Toc372056364)

[Hình 5.1: Kiến trúc công cụ sinh mô hình thành phần phần mềm tự động. 54](#_Toc372056365)

# DANH MỤC BẢNG

[Bảng 4.1: Các luật rút gọn được sử dụng ở bước 2. 38](#_Toc372056366)

[Bảng 5.1: Môi trường thử nghiệm công cụ sinh mô hình thành phần phần mềm. 55](#_Toc372056367)

[Bảng 5.2: Các thông số của kết quả thử nghiệm 57](#_Toc372056368)

[Bảng 5.3: Bảng kết quả thời gian và bộ nhớ thử nghiệm với L\*, Thompson, CNNFA. 57](#_Toc372056369)

# Chương 1: Giới thiệu

Trong ngành công nghiệp phần mềm hiện đại, ngày càng nhiều công ty, tổ chức tham gia phát triển một phần mềm dưới nhiều hình thức như gia công, mua thành phần, thư viện của đối tác phát triển thứ ba. Công ty phát triển sản phẩm chính có thể tập trung vào mảng nghiệp vụ chính của sản phẩm. Các thành phần khác như giao diện, những thuật toán khó, các thư viện hỗ trợ lập trình với các hệ thống lớn có sẵn có thể được mua từ các hãng cung cấp thành phần như DevExpress[[1]](#footnote-1), IndependentSoft[[2]](#footnote-2),v.v. Những phần mềm được phát triển như vậy gọi là phần mềm dựa trên thành phần.

Với những phần mềm dựa trên thành phần, đặc biệt là những phần mềm có cả các thành phần do bản thân công ty tự viết và những thành phần khác được mua từ bên phát triển thứ ba thì vấn đề khó khăn nhất trong quá trình phát triển là làm sao để đảm bảo tính đúng đắn của hệ thống, làm sao để các thành phần có thể cộng tác với nhau được trong môi trường của hệ thống. Đối với các thành phần được phát triển bởi bên thứ ba thì ta không có mã nguồn mà chỉ có được tài liệu của thành phần. Đối với các thành phần công ty tự thiết kế và phát triển thì làm sao để đảm bảo là thành phần thỏa mãn thiết kế. Để làm được việc đó, ta thường dùng phương pháp kiểm chứng mô hình (model checking) và kiểm thử dựa trên mô hình (model-based testing) để đảm bảo tính đúng đắn của phần mềm. Nhưng điểm cốt lõi của hai phương pháp kiểm chứng mô hình và kiểm thử dựa trên mô hình là phải có mô hình của thành phần phần mềm đó.

Ngoài ra, các thành phần phần mềm đều được tiến hóa theo thời gian, việc kiểm thử lại toàn bộ thành phần đó và những phần của hệ thống có sử dụng nó là công việc rất tốn kém về thời gian và công sức. Vì vậy, việc tự động hóa các công việc liên quan đến kiểm thử càng đóng vai trò to lớn mà trung tâm của việc tự động này là việc sinh mô hình một cách tự động cho thành phần phần mềm mỗi khi cần thiết.

Hiện nay, có nhiều hướng nghiên cứu liên quan đến việc sinh mô hình cho phần mềm hướng thành phần đã được đề xuất bởi nhiều tác giả. Một hướng là tập trung vào nghiên cứu các phương pháp sinh mô hình cho thành phần phần mềm. Với cách tiếp cận này, ta có thể kể đến các phương pháp sinh mô hình được đề cập trong [6], [11], [12], và [13]. Trong [6], các tác giả đã đặt ngữ cảnh là xây dựng mô hình cho thành phần phần mềm được cho dưới dạng một hộp đen và ta có thể thử nghiệm thực thi các hành động trên nó để có thể xây dựng được một tập các chuỗi hành động của thành phần phần mềm. Sau đó, tập các chuỗi hành động của thành phần phần mềm thu được có thể được coi là một biểu thức chính quy đặc tả hành vi của thành phần phần mềm, các tác giả sau đó đã sử dụng thuật toán Thompson để sinh mô hình cho thành phần phần mềm được cho bởi biểu thức chính quy đó. Phương pháp này bị giới hạn bởi độ dài tối đa của chuỗi các hành động có thể thử nghiệm trên thành phần phần mềm. Nghiên cứu trong [11] trình bày một thuật toán gọi là GK-tail mà tự động sinh mô hình cho thành phần phần mềm dưới dạng các EFSM (Extended Finite State Machine) từ các chuỗi tương tác của nó. EFSM mô hình hóa sự tương tác giữa các giá trị dữ liệu và thành phần phần mềm bằng cách ghi chú lên các cạnh của ôtômát hữu hạn đó với các điều kiện trên các giá trị dữ liệu. Trong nghiên cứu này, các tác giả đã đề cập một khía cạnh rất quan trọng của phần mềm. Đó là mô hình hóa các lời gọi hàm trong quan hệ với các tham số của nó. Phương pháp này dựa vào một phần mềm gọi là phần mềm giám sát để có thể sinh ra được các chuỗi tương tác mà được dùng như là đầu vào của nó. Nghiên cứu [12] giới thiệu một tập tích hợp các chương trình phân tích, chuyển đổi thành phần gọi là Bandera mà tự động trích xuất mô hình cho chương trình phần mềm dựa trên mã nguồn. Trong nghiên cứu này, Bandera lấy mã nguồn Java như là đầu vào để sinh mô hình dưới dạng đầu vào cho một số công cụ khác. Ngoài ra, Bandera cũng tham chiếu trở lại mã nguồn ban đầu với mô hình đã được sinh ra. Phương pháp này rõ ràng là phụ thuộc vào mã nguồn của phần mềm cần phân tích. Do đó, đối với các phần mềm hướng thành phần không có mã nguồn của một số thành phần mua từ bên phát triển thứ ba thì phương pháp này rất khó khả thi. Nghiên cứu [13] giới thiệu một công cụ gọi là Bandera Environment Generator (BEG). Công cụ này tự động hóa việc sinh mô hình môi trường để cung cấp một dạng hạn chế của việc kiểm chứng mô hình các mô đun của chương trình Java. Công cụ này sinh mô hình cho đơn vị chương trình Java dựa trên một số giả định về môi trường được cung cấp bởi người dùng. Mô hình đã được sinh ra có thể được dùng trong việc phân tích ảnh hưởng của môi trường lên đơn vị trong phương pháp kiểm chứng mô hình. Đây là một vấn đề rất thách thức trong thực tế phát triển phần mềm vì hệ thống phần mềm luôn phải chạy trên một sự kết hợp của rất nhiều hệ thống khác như hệ điều hành, các hệ thống khác,v.v. Người dùng, thậm chí cả người thiết kế phần mềm cũng khó có thể nhận biết được những thông tin đầy đủ về môi trường trong thời gian làm thiết kế hệ thống.

Một hướng tiếp cận khác là sinh mô hình trong khi thực hiện kiểm chứng mô hình hay trong khi thực hiện kiểm thử dựa trên mô hình [5], [14], và [15]. Trong [5], các tác giả đã sử dụng thuật toán học L\* để học đặc tả của một thành phần phần mềm thông qua một biểu thức chính quy để sinh ra mô hình cho thành phần đó. Biểu thức chính quy đó là kết quả của khâu thiết kế, có thể được sinh ra từ từ biểu đồ tuần tự theo phương pháp được đề cập trong [5]. Tuy phương pháp này sinh được mô hình cho thành phần phần mềm, nhưng sử dụng nhiều thời gian và bộ nhớ. Đặc biệt là phương pháp này bị giới hạn bởi độ dài tối đa của một chuỗi hành vi của phần mềm. Do đó, phương pháp này rất khó áp dụng được trong thực tế. Nghiên cứu [14] đặt vấn đề cho việc kiểm thử hộp đen. Trong nghiên cứu này, nhiều chiến lược được trình bày để kiểm chứng phần mềm từ khi chúng ta chưa có mô hình. Mô hình được sinh ra trong các lần lặp kiểm chứng phần mềm. Nghiên cứu [15] trình bày một phương pháp sinh mô hình thành phần phần mềm trong quá trình thành phần đó tiến hóa. Những mô hình này được sinh ra sử dụng các mô hình chưa đúng hiện có dựa vào các kỹ thuật kiểm thử hộp đen và học máy. Tuy nhiên, phương pháp này sinh mô hình cho toàn bộ phần mềm. Với những phần mềm lớn thì phương pháp này có thể dẫn đến sự bùng nổ trạng thái của mô hình. Với cách tiếp cận này, những mô hình được sinh ra như là một phần của quá trình khác như kiểm thử hộp đen, kiểm chứng mô hình. Luận văn này tập trung vào việc chỉ sinh mô hình cho thành phần phần mềm. Bằng cách này, chúng ta tập trung vào việc có được mô hình bằng một cách thực tế hơn như từ biểu đồ tuần tự [5]. Những mô hình này sau đó có thể được dùng như là đầu vào cho các phương pháp khác như kiểm chứng mô hình, kiểm thử dựa trên mô hình.

Như vậy, về khía cạnh khoa học và thực tiễn, việc sinh mô hình cho thành phần phần mềm mềm một cách tự động đều đóng vai trò then chốt và to lớn cho hàng loạt các vấn đề sau đó như kiểm thử dựa trên mô hình, kiểm chứng mô hình, tiến hóa phần mềm. Đó chính là lý do tại sao tôi lựa chọn đề tài **“Phương pháp sinh mô hình tự động cho phần mềm dựa trên thành phần”** cho nghiên cứu của mình.

Đề tài này nhằm mục đích nghiên cứu phương pháp sinh mô hình chính xác đặc tả cho hành vi của phần mềm dựa trên thành phần một cách tự động làm cơ sở cho việc kiểm chứng mô hình, kiểm thử dựa trên mô hình và tiến hóa phần mềm nhằm nâng cao chất lượng phần mềm và giảm thiểu chi phí kiểm thử trong công nghiệp phần mềm hiện đại. Đề tài này đặt giả thiết rằng thành phần phần mềm sẽ được sinh mô hình đã được đặc tả bằng một biểu thức chính quy các hành vi của nó. Đặc tả này có thể có được trong quá trình thiết kế, từ biểu đồ tuần tự hoặc là đặc tả của thành phần phần mềm ta nhận được của bên thứ ba khi mua thư viện, thành phần phần mềm của họ. Từ mô hình của các thành phần phần mềm đã được sinh ra, ta có thể sinh được mô hình cho toàn bộ phần mềm.

Nội dung luận văn này được trình bày trong sáu chương. Chương 1 giới thiệu về đề tài. Chương này trình bày các ngữ cảnh, các nghiên cứu đã có về vấn đề cần giải quyết, lý do lựa chọn đề tài, mục tiêu của đề tài và cấu trúc nội dung của luận văn. Chương 2 trình bày các khái niệm cơ bản phục vụ cho đề tài. Chương này mô tả các phương pháp đặc tả hình thức cho thành phần phần mềm, các khái niệm về thành phần phần mềm, đặc tả hình thức cho thành phần phần mềm, máy hữu hạn trạng thái, hệ chuyển trạng thái được gán nhãn, ôtômát hữu hạn trạng thái và các khái niệm liên quan. Một số phương pháp sinh mô hình tự động hiện nay cho phần mềm dựa trên thành phần được giới thiệu trong chương 3. Chương này trình bày tóm tắt hai phương phương pháp sinh mô hình tự động cho thành phần phần mềm hiện nay. Phương pháp thứ nhất là sinh mô hình tự động bằng việc áp dụng thuật toán học L\* [2], [5]. Phương pháp thứ hai là sinh mô hình tự động cho thành phần phần mềm sử dụng thuật toán Thompson [3], [6]. Chương 3 cũng đưa ra phân tích về một số hạn chế của hai phương pháp này. Chương 4 nghiên cứu một phương pháp khác để sinh mô hình tự động cho thành phần phần mềm sử dụng thuật toán CNNFA [4], [9]. Phương pháp này sinh tự động mô hình chính xác đặc tả cho hành vi của một thành phần phần mềm được cho bởi biểu thức chính quy. Ngoài ra, chương này cũng đưa ra một số thuật toán nhằm tối ưu hóa mô hình đã được sinh ra bằng phương pháp CNNFA ở trên. Chương 5 giới thiệu về công cụ thực nghiệm và kết quả thực nghiệm của các phương pháp sinh mô hình tự động được trình bày trong chương 3 và chương 4. Chương 6 đưa ra kết luận, định hướng phát triển cho đề tài. Cuối cùng là tài liệu tham khảo.

# Chương 2: Các phương pháp hình thức cho đặc tả phần mềm

## 2.1 Hệ thống chuyển trạng thái được gán nhãn

Có nhiều phương pháp đặc tả thành phần phần mềm, chương 2 này chỉ trình bày một phương pháp được sử dụng trong luận văn này. Mỗi thành phần phần mềm được đặc tả bằng một mô hình tương ứng mô tả các hành vi quan sát được của thành phần đó. Mô hình của thành phần phần mềm đó có thể được mô tả bằng một hệ chuyển trạng thái được gán nhãn. Đó là một đồ thị có hướng với các nhãn được gắn trên các cạnh của nó. Mỗi nhãn trên cạnh được biểu diễn cho một hành vi có thể thực hiện được của thành phần đó. Mỗi đỉnh của đồ thị là một trạng thái biểu diễn cho trạng thái của hệ thống.

Tập tất cả các hành động của thành phần phần mềm gọi là bảng chữ cái. Ta kí hiệu *Act* là tập tất cả các hành động quan sát được của thành phần phần mềm, là hành động không quan sát được trong môi trường của thành phần phần mềm, là trạng thái lỗi đặc biệt của hệ thống, ta kí hiệu LTS Π = <{π}, Act, , π>.

***Định nghĩa 2.1***: Hệ thống chuyển trạng thái được gán nhãn (Labeled Transition System – LTS [10]).

Một LTS là một bộ gồm bốn thành phần (Q, , , q0),trong đó:

* *Q* là một tập khác rỗng của các trạng thái,
* là tập các hành động quan sát được, gọi là bảng chữ cái của M,
* *Q x x Q* là hàm chuyển trạng thái, và
* *q0 Q* là trạng thái ban đầu.

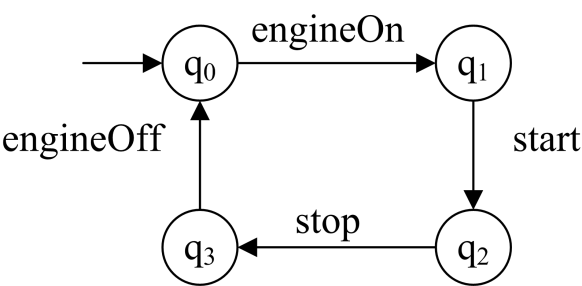
Ta kí hiệu qi qj nếu có một hành động với nhãn là *a* chuyển hệ thống từ trạng thái qi sang trạng thái qj, khi đó (qi, *a*, qj) . Điều này có nghĩa là khi hệ thống đang ở trạng thái qi, nó có thể thực hiện một hành động a và chuyển sang trạng thái qj. Tương tự khi hệ thống đang ở trạng thái qj có thể thực hiện một hành động *a’* và chuyển sang trạng thái qk. Như vậy, ta có một chuỗi các hành động qi qj qk. Khi đó, ta có thể kí hiệu qi qk.

Một cách tổng quát, nếu ta có một chuỗi hành động qi qk biểu diễn hệ thống thì khi nó ở trạng thái qi, nó có thể thực hiện một chuỗi các hành động *a1.a2….an* và kết thúc ở trạng thái qk.

Xét một LTS P = **(**Q, , , q0**)** với q, q′ Q, , i , với 1 ≤ i ≤ n, ta có:

1. q q′ (q, , q′ ) .
2. q q′ q0, q1, …, qn : q = q0 q1 q2  … qn = q′.

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



Hình 2.1: Một hệ thống chuyển trạng thái được gán nhãn.

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

* Q = {q0, q1, q2, q3},
* = {engineOn, start, stop, engineOff},
* = {(q0, engineOn, q1), (q1, start, q2), (q2, stop, q3), (q3, engineOff, q0)}, và
* q0 là trạng thái khởi đầu.

***Định nghĩa 2.2***: Kích thước của một LTS [10].

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

**Ví dụ 2.2**: Xét một LTS cho bởi hình 2.1, kích thước của LTS đó là |M| = |Q| = 4.

***Định nghĩa 2.3***: LTS đơn định và không đơn định [10].

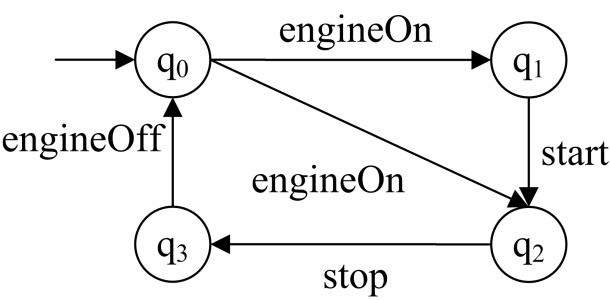
Một LTS M = (Q, , , q0) là không đơn định nếu nó chứa một chuyển dịch τ hoặc nếu (q, a, q’) và (q, a, q”) ∈ δ sao cho q’ q”. Trái lại, M là hệ chuyển trạng thái được gán nhãn đơn định.

**Chú ý 2.1**: Cho hai LTS M = (Q, , δ, q0) và M’ = (Q’, ’, δ’, q0’). Ta nói M chuyển dịch thành M’ với chuyển dịch a nếu và chỉ nếu (q0, a, q0’) ∈ δ, ’, Q = Q’ và δ = δ’. Ta ký hiệu: M M’.

**Ví dụ 2.3**: Một LTS đơn định và không đơn định.

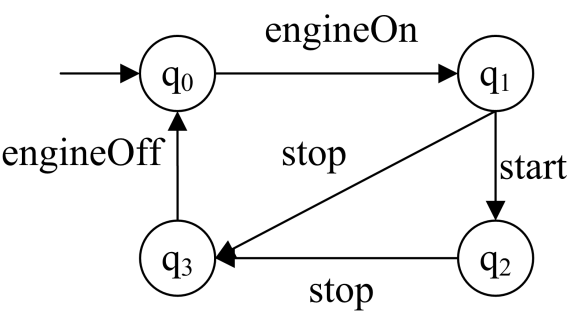
Trên hình 2.2 là một LTS M = (Q, , , q0), trong đó:

* Q = {q0, q1, q2, q3},
* = {engineOn, start, stop, engineOff},
* = {(q0, engineOn, q1), (q0, engineOn, q2), (q1, start, q2), (q2, stop, q3), (q3, engineOff, q0)}, và
* q0 là trạng thái khởi đầu.



Hình 2.2: Một LTS không đơn định.

Khi hệ thống đang ở trạng thái q0, thực hiện một hành động engineOn thì hệ thống có thể chuyển trạng thái đến trạng thái q1 hoặc trạng thái q2. Như vậy, trạng thái tiếp theo của q0 khi ta thực hiện hành động engineOn là không xác định duy nhất hay không đơn định. Ta gọi đó là LTS không đơn định.



Hình 2.3: Một LTS đơn định.

Trên hình 2.3 là một LTS M = (Q, , , q0), trong đó:

* Q = {q0, q1, q2, q3},
* = {engineOn, start, stop, engineOff},
* = {(q0, engineOn, q1), (q1, start, q2), (q1, stop, q3), (q2, stop, q3), (q3, engineOff, q0)}, và
* q0 là trạng thái khởi đầu.

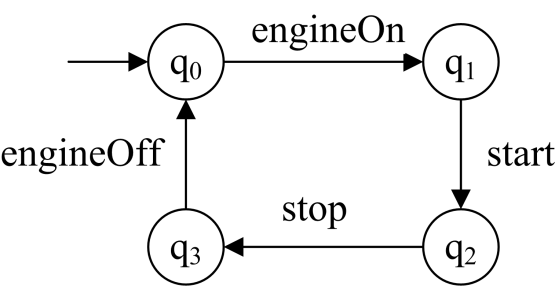
Với LTS cho trong hình 2.3, khi hệ thống đang ở một trạng thái qi bất kỳ, nếu thực hiện một hành động bất kỳ trong thì hệ thống sẽ chuyển sang một trạng thái xác định duy nhất qk. Ta gọi đó là LTS đơn định.

***Định nghĩa 2.4***: Vết của LTS M [10].

Một vết của một hệ chuyển trạng thái được gán nhãn M = **(**Q, , , q0**)** là một chuỗi hữu hạn các hành động a1a2 …an với ai (i = 1, …, n) mà tồn tại một chuỗi các trạng thái bắt đầu bằng trạng thái khởi đầu (ví dụ: q0q1...qn) mà (qi-1, ai, qi) với i = 1, ..., n.

Như vậy, vết của một LTS M là một chuỗi các hành động có thể quan sát được mà M có thể thực hiện được từ trạng thái khởi đầu q0.

**Ví dụ 2.4**: Vết của một LTS.



Hình 2.4: Minh họa vết của một LTS.

Trên hình 2.4 là một LTS M = (Q, , , q0), trong đó:

* Q = {q0, q1, q2, q3},
* = {engineOn, start, stop, engineOff},
* = {(q0, engineOn, q1), (q1, start, q2), (q2, stop, q3), (q3, engineOff, q0)}, và
* q0 là trạng thái khởi đầu.

Ta thấy rằng chuỗi các hành động *engineOn start stop engineOff* là một dẫn xuất trên M. Từ trạng thái q0, thực hiện hành động *engineOn*, hệ thống chuyển sang trạng thái q1, tiếp tục thực hiện hành động *start*, hệ thống chuyển sang trạng thái q2, tiếp tục thực hiện hành động *stop*, hệ thống chuyển sang trạng thái q3, cuối cùng thực hiện hành động *engineOff* thì hệ thống chuyển về trạng thái q0. Tương tự như vậy, chuỗi *engineOn start stop engineOff engineOn start* cũng là một dẫn xuất của M.

Xét chuỗi hành động *engineOn start engineOn start stop engineOff* trên M. Ta thấy rằng, từ trạng thái q0, thực hiện hành động *engineOn*, hệ thống chuyển sang trạng thái q1, tiếp tục thực hiện hành động *start*, hệ thống chuyển sang trạng thái q2, từ trạng thái q2, ta không thể thực hiện hành động *engineOn* trên M. Do đó, chuỗi hành động trên không phải là một dẫn xuất của M.

**Chú ý 2.2**: Với Σ Act ta ký hiệu ↑Σ là một dẫn xuất thu được bằng cách loại bỏ khỏi tất cả các hành động a mà a Σ. Tập tất cả các vết của M được gọi là ngôn ngữ của M, ký hiệu L(M). Một vết = a1a2..an là một vết hữu hạn trên LTS M. Ta ký hiệu LTS Mσ = (Q, , , q0) với Q = {q0, q1,.., qn} và = {(qi-1, ai, qi)} với i=1,..,n. Ta nói rằng một hành động a được chấp nhận từ một trạng thái q Q nếu tồn tại q’Q sao cho (q, a, q’) . Tương tự vậy, ta nói rằng một vết a1a2...an 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) .

## 2.2 Phép ghép nối song song

***Định nghĩa 2.5***: Phép ghép nối song song [10].

Một toán tử ghép nối song song, kí hiệu là || là một phép toán ghép 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 mọi (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.

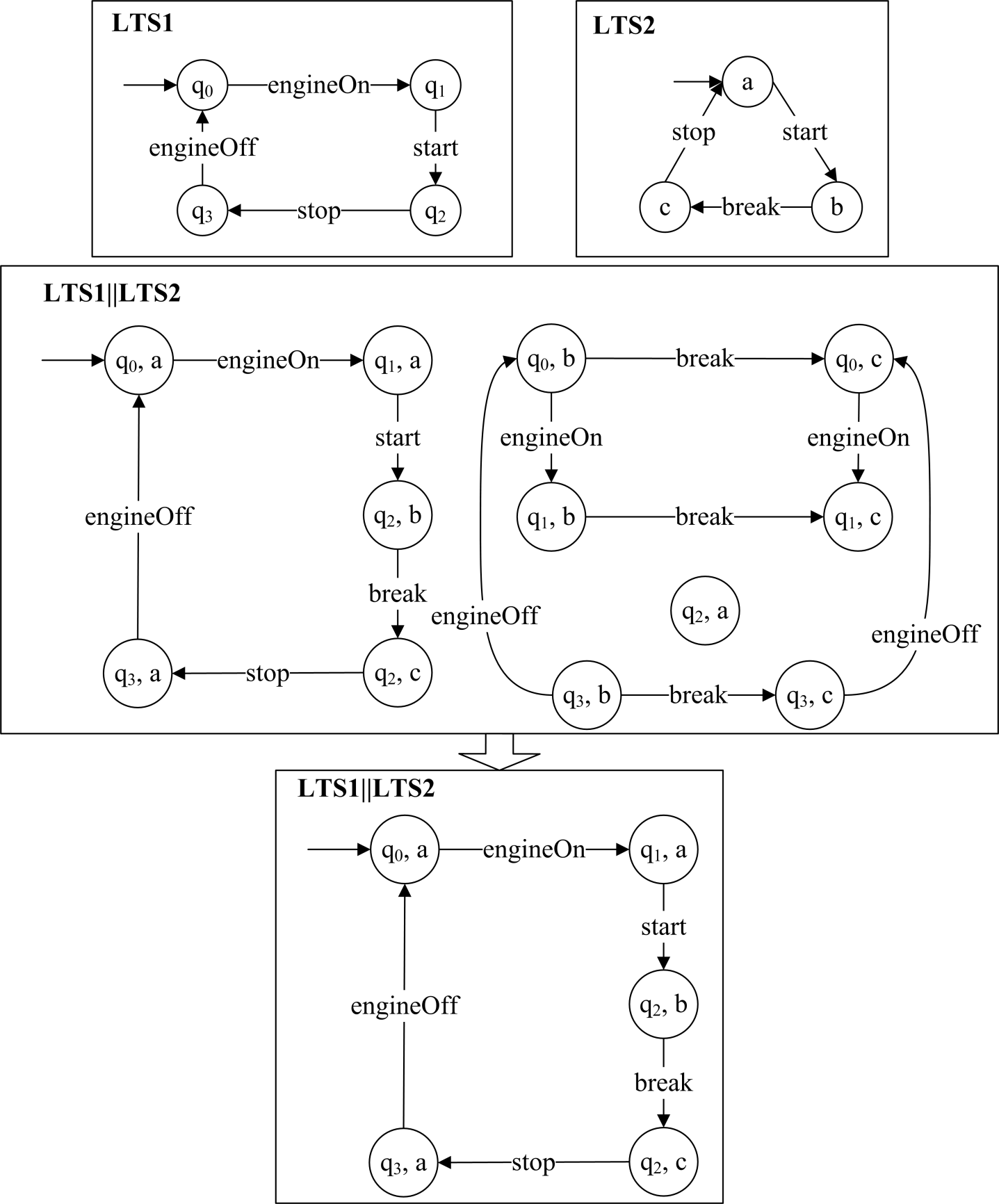
**Ví dụ 2.5**: Mô hình ghép nối song song hai LTS1 và LTS2 được trình bày trên hình 2.5.

Khi ghép nối LTS1 và LTS2 trên hình 2.5, hai hành động start và stop là đồng bộ, các hành động còn lại đan xen nhau. Theo quy tắc trên, ta xác định được hệ chuyển trạng thái song song được gán nhãn M’ = (Q’, αM’, δ’, q0’), trong đó:

* Q’ = Q1xQ2 = {(q0,a), (q0,b), (q0,c), (q1,a), (q1,b), (q1,c), (q2,a), (q2,b), (q2,c), (q3,a), (q3,b), (q3,c)}, αM’ = αM1∪αM2 = {engineOn, start, break, stop, engineOff},
* q0’ = (q0, a), và
* δ’ = {((q0,a), engineOn, (q1,a)), ((q1,a), start, (q2,b)), ((q2,b), break, (q2,c)), ((q2,c), stop, (q3,a)), ((q3,a), engineOff, (q0,a)), ((q0,b), break, (q0,c)), ((q3,b), break, (q3,c)), ((q1,b), break, (q1,c)), ((q0,b), engineOn, (q1,b)), ((q0,c), engineOn, (q1,c)), ((q3,b), engineOff, (q0,b)), ((q3,c), engineOff, (q0,c))}.

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 M = (Q, αM, δ, q0), trong đó:

* Q = {(q0,a), (q1,a), (q2,b), (q2,c), (q3,a)},
* αM = {engineOn, start, break, stop, engineOff},
* q0 = (q0, a), và
* δ = {((q0,a), engineOn, (q1,a)), ((q1,a), start, (q2,b)), ((q2,b), break, (q2,c)), ((q2,c), stop, (q3,a)), ((q3,a), engineOff, (q0,a))}.



Hình 2.5: Ghép nối song song hai LTS.

## 2.3 Hệ thống chuyển trạng thái gán nhãn an toàn, thuộc tính an toàn, tính thỏa mãn

***Định nghĩa 2.6***: Hệ chuyển trạng thái được gán nhãn an toàn [10].

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

***Định nghĩa 2.7***: Thuộc tính an toàn [10].

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 [10].

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

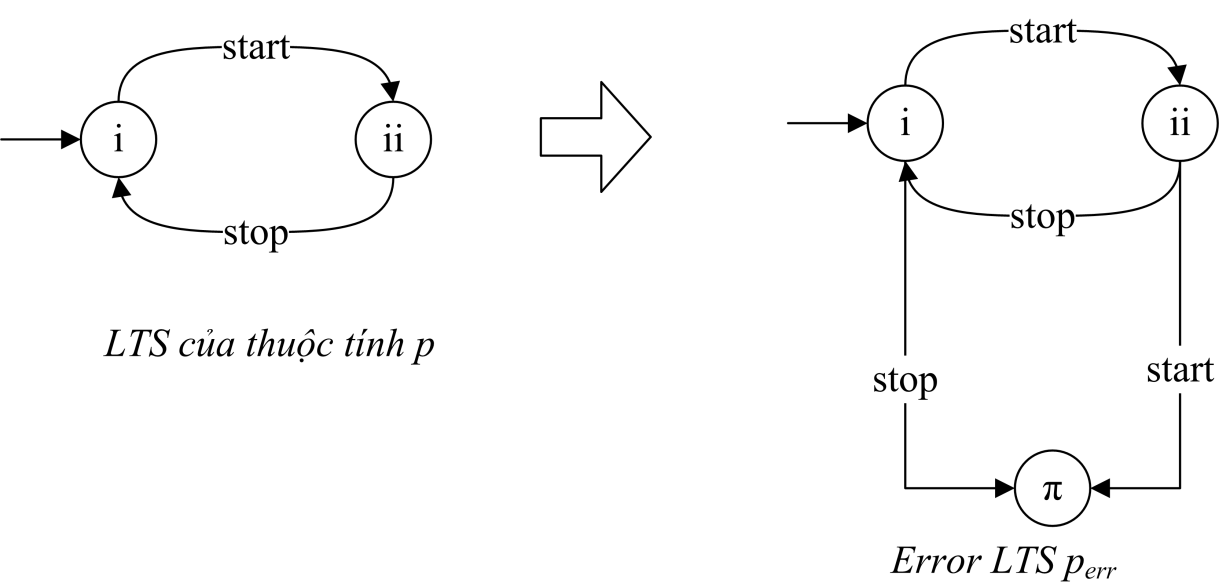
αperr = αp, δ’ = δ {(q, a, π) **|** a αp và q’ Q sao cho (q, a, q’) δ}.

**Ví dụ 2.6**: Hình 2.6 mô tả việc xây dựng hệ chuyển trạng thái được gán nhãn lỗi từ một hệ chuyển trạng thái ứng với thuộc tính p.

***Định nghĩa 2.9***: Tính thỏa mãn [10].

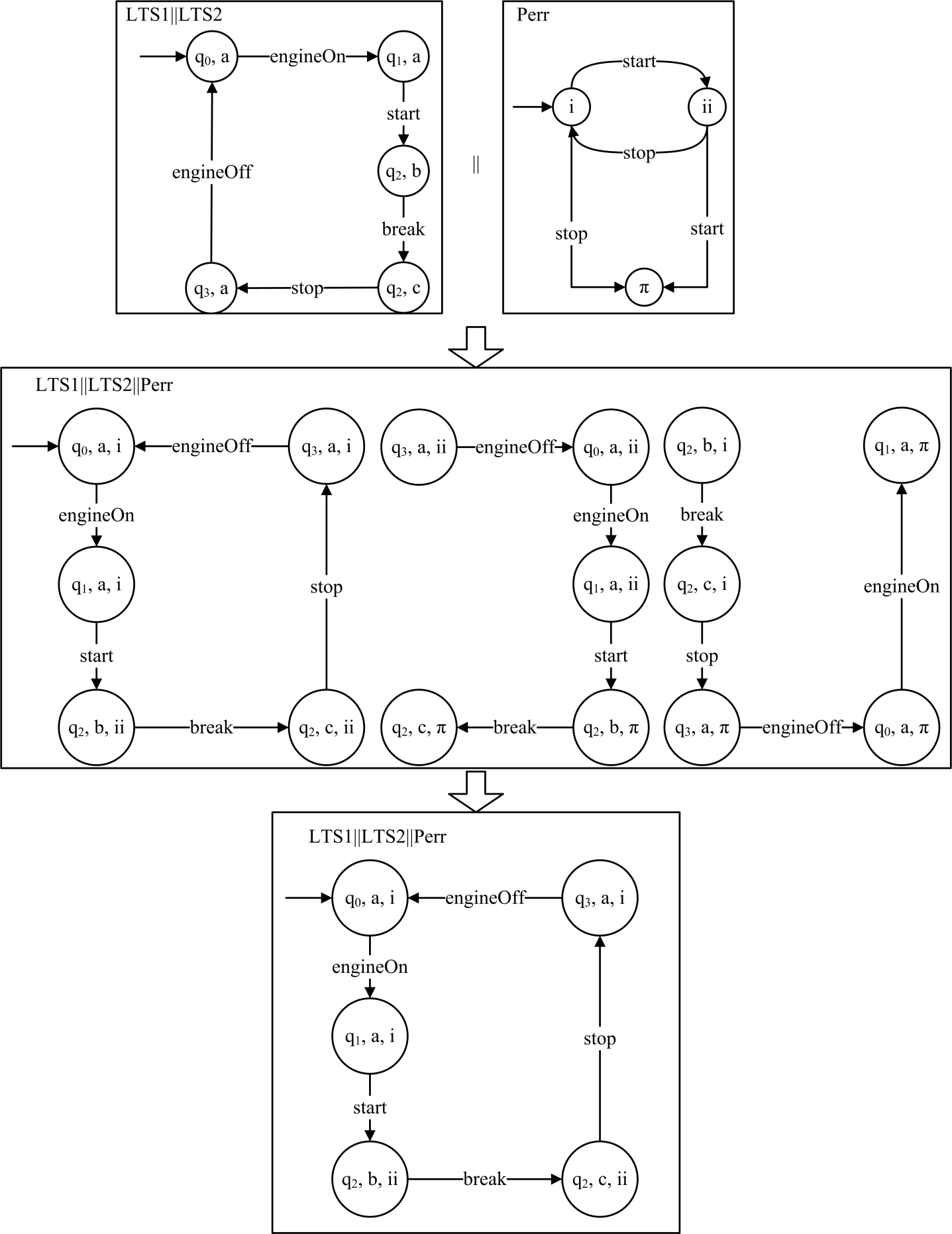
Một LTS M được gọi là thỏa mãn thuộc tính *p*, ký hiệu M╞ *p* khi và chỉ khi ∀∂ ∈ L(M) sao cho: (∂↑αp) ∈ L(p).

Để kiểm tra một LTS M thỏa mãn thuộc tính p hay không, ta biểu diễn M và perr dưới dạng LTS an toàn (safe LTS), sau đó thực hiện phép ghép song song M||perr. Nếu LTS thu được sau phép ghép nối tồn tại một dẫn xuất có thể tới được trạng thái π, khi đó ta nói rằng LTS M vi phạm thuộc tính p. Ngược lại, M thỏa mãn thuộc tính p.



Hình 2.6: LTS của thuộc tính p và LTS lỗi tương ứng của p.

**Ví dụ 2.7**: Kiểm chứng hệ chuyển trạng thái ghép nối song song được gán nhãn thu được như trên ví dụ 2.5 có thỏa mãn thuộc tính p được đề cập trong ví dụ 2.6 hay không, ta xây dựng một LTS ghép nối song song của LTS1||LTS2||perr như trên hình 2.7. Kết quả cuối cùng là sau khi đã bỏ đi những trạng thái không đến được từ trạng thái ban đầu.



Hình 2.7: Xây dựng LTS ghép nối song song LTS1||LTS2||perr.

Khi ghép nối hai (LTS1||LTS2) và perr ở trên, hai hành động start và stop là đồng bộ, các hành động còn lại đan xen nhau. Theo quy tắc trên, ta xác định được hệ chuyển trạng thái song song được gán nhãn M’’ = (Q’’, αM’’, δ’’, q0’’), trong đó:

* Q’’ = {(q0,a,i), (q1,a,i), (q2,b,i), (q2,c,i), (q3,a,i), (q0,a,ii), (q1,a,ii), (q2,b,ii), (q2,c,ii), (q3,a,ii), (q0,a,π), (q1,a,π), (q2,b,π), (q2,c,π), (q3,a,π)},
* αM’’ = {engineOn, start, break, stop, engineOff},
* q0’’ = (q0,a,i), và
* δ’’ = {((q0,a,i), engineOn, (q1,a,i)), ((q1,a,i), start, (q2,b,ii)), ((q2,b,ii), break, (q2,c,ii)), ((q2,c,ii), stop, (q3,a,i)), ((q3,a,i), engineOff, (q0,a,i)), ((q3,a,ii), engineOff, (q0,a,ii)), ((q0,a,ii), engineOn, (q1,a,ii)), ((q1,a,ii), start, (q2,b, π)), ((q2,b, π), break, (q2,c, π)), ((q2,b, i), break, (q2,c,i)), ((q2,c, i), stop, (q3,a,π)), ((q3,a,π), engineOff, (q0,a,π)), ((q0,a,π), engineOn, (q1,a,π))}.

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 M = (Q, αM, δ, q0), trong đó:

* Q = {(q0,a,i), (q1,a,i), (q2,b,ii), (q2,c,ii), (q3,a,i)},
* αM = {engineOn, start, break, stop, engineOff},
* q0 = (q0,a,i), và
* δ = {((q0,a,i), engineOn, (q1,a,i)), ((q1,a,i), start, (q2,b,ii)), ((q2,b,ii), break, (q2,c,ii)), ((q2,c,ii), stop, (q3,a,i)), ((q3,a,i), engineOff, (q0,a,i))}.

Ta thấy rằng LTS thu được cuối cùng không chứa trạng thái π nên LTS1||LTS2 là thỏa mãn p.

## 2.4 Thành phần phần mềm và ôtômát hữu hạn trạng thái

Trong hoàn cảnh của ngành công nghệ phần mềm hiện nay, một công ty, tổ chức khó có thể một mình làm được sản phẩm hoàn thiện toàn bộ mà họ chỉ tập trung vào làm phần nghiệp vụ chính, phần còn lại như thư viện giao diện người sử dụng, phần giao tiếp với các hệ thống lớn khác thì có thể mua của bên phát triển thứ ba. Một thành phần phần mềm như vậy có thể được biểu diễn bằng một ôtômát hữu hạn trạng thái được định nghĩa như sau:

***Định nghĩa 2.10***: Ôtômát hữu hạn trạng thái (Finite State Automata) [10].

Ôtômát hữu hạn trạng thái là một bộ 5 có thứ tự M = (Q, , , q0, F), trong đó:

* Q là tập hữu hạn khác rỗng các trạng thái,
* Act là một tập hữu hạn khác rỗng các hành động, còn gọi là bảng chữ cái của M,
* q0  Q là trạng thái ban đầu,
* là hàm chuyển trạng thái có dạng:
  + Nếu hàm chuyển là ánh xạ : Q x Q thì M được gọi là ôtômát hữu hạn đơn định (deterministic finite automata - DFA),
  + Nếu hàm chuyển là ánh xạ : Q x 2Q thì M được gọi là ôtômát hữu hạn không đơn định (nondeterministic finite automata - NFA), và
* F Q là tập các trạng thái kết thúc.

***Định nghĩa 2.11***: Kích thước của một ôtômát hữu hạn trạng thái.

Kích thước của một ôtômát hữu hạn M = (Q, , , q0, F) là số trạng thái của M. Kí hiệu là |M|, trong đó |M| = |Q|.

***Định nghĩa 2.12***: Mô hình chính xác (Accurate Model) [6].

M là mô hình mô tả hành vi của một thành phần phần mềm C. Ta nói M là mô hình chính xác của C nếu \*, là một thực hiện được chính xác trên C. Khi đó là một vết của M.

**Chú ý 2.3**: Biểu diễn ôtômát hữu hạn.

Phương pháp thứ nhất là biểu diễn ôtômát hữu hạn bằng hàm chuyển trạng thái, đây là phương pháp biểu diễn ôtômát như đã mô tả trong định nghĩa 2.10.

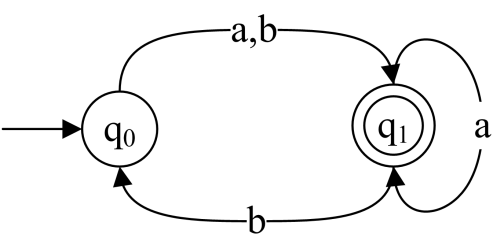
Phương pháp thứ 2 là biểu diễn bằng đồ thị. Hàm chuyển trạng thái được biểu diễn dưới dạng một đồ thị có hướng, trong đó, mỗi trạng thái là một đỉnh. Nếu ký hiệu vào là a∈Σ và từ trạng thái q ôtômát chuyển sang trạng thái p (p = (q, a)) thì sẽ có một cung từ đỉnh q đến đỉnh p với nhãn là a. Đỉnh ứng với trạng thái bắt đầu (q0) có mũi tên đến, đỉnh ứng với trạng thái kết thúc được biểu diễn bằng vòng tròn kép, các đỉnh còn lại biểu diễn bởi vòng tròn đơn.

**Ví dụ 2.8**: Biểu diễn ôtômát hữu hạn.

Hình 2.8 minh họa một cách biểu diễn ôtômát hữu hạn bằng đồ thị của ôtômát sau:

M = (Q, , , q0, F), trong đó:

* Q = {q0, q1},
* q0 là trạng thái khởi tạo,
* = {a, b},
* = {(q0, a, q1), (q0, b, q1), (q1, a, q1), (q1, b, q0)}, và
* = {q1}.



Hình 2.8: Biểu diễn ôtômát hữu hạn.

**Chú ý 2.4**: Cho một ôtômát hữu hạn trạng thái M và một chuỗi , ta ký hiệu (q,) để chỉ trạng thái của M sau khi đọc chuỗi bắt đầu từ trạng thái q. Một chuỗi được gọi là được đoán nhận bởi ôtômát hữu hạn M = (Q, , , q0, F) nếu (q0,) F. Ngôn ngữ đoán nhận bởi ôtômát M được định nghĩa L(M) = { **|** (q0,) F}.

Ta có thể mô tả hoạt động của ôtômát M khi cho một xâu vào = a1a2…an như sau:

Khi bắt đầu làm việc, ôtômát M ở trạng thái q0, dưới tác động của ký hiệu vào a1, M chuyển sang trạng thái mới (q0, a1) = q1 Q, khi M đang ở trạng thái q1, dưới tác động của ký hiệu vào a2, ôtômát chuyển từ trạng thái q1 sang trạng thái mới (q1, a2) = q2 Q. Quá trình này tiếp tục cho đến khi xảy ra một trong các tình huống sau:

Ôtômát đọc hết xâu vào = a1a2…an và (qn-1, an) = qn F, khi đó ta nói rằng ôtômát M đoán nhận xâu .

Ôtômát đọc hết xâu vào = a1a2…an và (qn-1, an) = qn F, khi đó ta nói rằng ôtômát M không đoán nhận xâu .

Ôtômát M đọc đến aj (j n) và (qj-1, aj) không xác định, khi đó ta nói rằng ôtômát M không đoán nhận xâu .

**Ví dụ 2.9**: Với ôtômát được minh họa trong hình 2.8, bằng việc kiểm tra quá trình đoán nhận xâu đầu vào như mô tả ở trên, ta dễ dàng thấy được xâu aaaaa L(M), nhưng chuỗi aaab L(M).

**Chú ý 2.5**: Một ôtômát hữu hạn trạng thái M được gọi là tiền tố đóng nếu ngôn ngữ được đoán nhận bởi nó L(M) là tiền tố đóng, tức là với mỗi chuỗi L(M) thì mọi tiền tố của cũng thuộc L(M).

**Chú ý 2.6**: Ôtômát M nhận được từ thuật toán học L\* [2] được trình bày trong chương 3 là ôtômát tối tiểu, đầy đủ và tiền tố đóng. Do đó nó sẽ chứa một số trạng thái không kết. Để nhận được LTS L từ ôtômát M chúng ta loại bỏ tất cả các trạng thái không kết và tất cả các quy tắc chuyển trạng thái đến trạng thái không kết đó. Chúng ta có thể định nghĩa cách chuyển một ôtômát M thành một LTS L như trong định nghĩa 2.13.

***Định nghĩa 2.13***: Với một ôtômát M = (Q {nas}, , , q0, F) ta xác định được hệ thống chuyển trạng thái được gán nhãn L tương ứng như sau:

L = ( Q, , ( Q x x Q ), q0)

Trong đó nas (non-acepting states) là các trạng thái không kết.

***Định nghĩa 2.14***: Ôtômát tối thiểu.

Ôtômát có số trạng thái ít nhất trong các ôtômát hữu hạn cùng đoán nhận ngôn ngữ L được gọi là ôtômát tối tiểu của ngôn ngữ L.

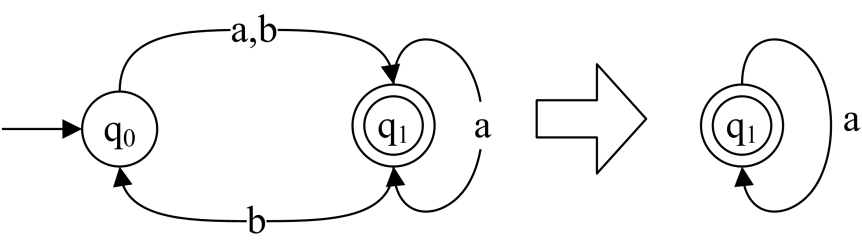
***Định nghĩa 2.15***: Trạng thái tương đương.

Hai trạng thái q1 và q2 được gọi là tương đương, ký hiệu q1 q2 nếu cả hai trạng thái (q1, x) và (q2, x) đều là những trạng thái kết thúc hoặc cả hai đều là không kết thúc với mọi x Σ\*.

***Định nghĩa 2.16***: Trạng thái k – tương đương.

Hai trạng thái q1 và q2 được gọi là k – tương đương, ký hiệu q1 k q2 nếu cả hai trạng thái (q1, x) và (q2, x) đều là những trạng thái kết thúc hoặc cả hai đều là không kết thúc với mọi x Σ\* có độ dài nhỏ hơn k ( k 0).

**Ví dụ 2.10**: Hình 2.9 minh họa cách chuyển ôtômát hữu hạn M thành LTS L.



Hình 2.9: Chuyển một ôtômát hữu hạn M thành LTS L.

**Chú ý 2.7**: Các quan hệ và k là các quan hệ tương đương (có tính chất phản xạ, đối xứng, bắc cầu). Các lớp tương đương của hai quan hệ này sẽ tạo ra các phân hoạch và k của Q. Các phần tử của k là các lớp k – tương đương.

***Định nghĩa 2.17***: Trạng thái (k+1) – tương đương.

Hai trạng thái q1 và q2 được gọi là (k+1) – tương đương nếu:

* Chúng là k – tương đương, và
* (q1, a) và (q2, a) cũng là k – tương đương với mọi a Σ.

# Chương 3: Các phương pháp sinh mô hình tự động cho thành phần phần mềm

Chương 2 đã trình bày một số kiến thức cơ bản về các phương pháp hình thức cho đặc tả phần mềm, máy hữu hạn trạng thái và ôtômát hữu hạn. Chương 3 sẽ trình bày về một số phương pháp sinh mô hình tự động cho thành phần phần mềm hiện nay.

Trong ngành công nghiệp phần mềm hiện đại, ngày càng có nhiều hệ thống phần mềm đồ sộ và phức tạp được xây dựng, một công ty bình thường khó lòng phát triển toàn bộ các thành phần của phần mềm được. Do đó, công nghệ phần mềm hướng thành phần đang trở thành một phần rất quan trọng của công nghiệp phần mềm nói chung. Tuy vậy, vấn đề lớn nhất là việc làm sao để đảm bảo các thành phần phát triển từ các bên khác nhau có thể cộng tác và làm việc tốt với nhau để đạt được mục tiêu của hệ thống. Giải pháp phổ biến ngày nay là giải pháp áp dụng các phương pháp kiểm chứng mô hình và kiểm thử tự động dựa trên mô hình. Tuy nhiên, các kỹ thuật này hiện nay đều đặt giả thiết là ta đã có mô hình phần mềm chính xác. Giả thiết này trên thực tế rất khó có được khi các thành phần phần mềm đều được phát triển bởi các bên thứ ba, ta không có mã nguồn, tài liệu đủ để xây dựng được mô hình đầy đủ của thành phần đó. Bài toán đặt ra là ta cần sinh được mô hình chính xác đặc tả hành vi của thành phần phần mềm đó. Trong số các phương pháp sinh mô hình cho thành phần phần mềm hiện nay, có hai phương pháp xây dựng mô hình chính xác của một thành phần phần mềm được đề cập đến trong [5] và [6]. Phương pháp đầu tiên [5] là xây dựng mô hình sử dụng thuật toán học tăng cường L\* được phát triển bởi Angluin [2]. Phương pháp thứ hai [6] là phương pháp xây dựng mô hình sử dụng thuật toán Thompson [3]. Tuy nhiên, hiện tại hai phương pháp này vẫn có một số điểm giới hạn nhất định, khó có thể thực thi hiệu quả với các hệ thống lớn như bị giới hạn bởi độ dài chuỗi hành vi của thành phần phần mềm, độ phức tạp lớn.

Phương pháp đầu tiên là phương pháp sinh mô hình tự động dựa vào thuật toán học L\*. Phương pháp này đặt giả thiết là đối với thành phần C, ta biết tất cả các chuỗi hành động có thể thực hiện được đều thỏa mãn một biểu thức chính quy cho trước và có độ dài không quá một giá trị biết trước MaxLength. Sau đó, phương pháp này sử dụng thuật toán học tăng cường L\* lần lượt truy vấn đối tượng Teacher để xây dựng bảng quan sát và các mô hình ứng viên ở mỗi bước của thuật toán. Khi thuật toán dừng, ta sẽ có một mô hình ứng viên phù hợp nhất đoán nhận các xâu thỏa mãn biểu thức chính quy và có độ dài không vượt quá MaxLength.

Phương pháp thứ hai là phương pháp sinh mô hình tự động cho thành phần phần mềm dựa vào thuật toán Thompson. Phương pháp này đặt giả thiết là thành phần C là một hộp đen có thể đáp ứng việc thử nghiệm thực hiện các chuỗi hành động cho trước và trả về kết quả là chuỗi hành động đó có thể thực hiện được trên thành phần C hay không. Vì phương pháp này yêu cầu phải thử nghiệm tất cả các hành động trong tập các chuỗi cho trước nên việc sinh mô hình bị giới hạn bởi độ dài MaxLength của các chuỗi có thể thử nghiệm được trên thành phần C. Sau khi có được tập các chuỗi hành động có thể thực hiện được trên thành phần C, phương pháp này xây dựng mô hình cho thành phần C bằng việc áp dụng thuật toán Thompson để xây dựng ôtômát hữu hạn mô tả chính xác cho thành phần C được đại diện bởi tập các chuỗi hành động được xây dựng ở bước trước đó.

## 3.1 Phương pháp sinh mô hình dựa trên thuật toán học L\*

Phương pháp xây dựng mô hình thứ nhất là phương pháp xây dựng mô hình dựa trên thuật toán học L\* [5]. L\* [2] là thuật toán học có thể học một ngôn ngữ không biết quy tắc để đưa ra một ôtômát hữu hạn đoán nhận nó. Ý tưởng chính của thuật toán này dựa trên định lý “Myhill – Nerode Theorem” [8] trong lý thuyết ngôn ngữ hình thức. Với mỗi tập có quy tắc U ⊆ Σ\*, tồn tại duy nhất một ôtômát tối tiểu mà tập các trạng thái của nó đẳng cấu với tập các lớp tương đương theo quan hệ sau: w ≈ w′ khi và chỉ khi u ⊆ Σ\*: wu U ⇔ w′u U. Do đó ý tưởng chính của L\* là học các lớp tương đương.

U là một ngôn ngữ không biết quy tắc trên bảng chữ cái Σ. Thuật toán học L\* sẽ học để thu được một ôtômát hữu hạn tối thiểu đoán nhận U, tức là L(M) = U. Để học được U, L\* sẽ phải tương tác với một đối tượng gọi là đối tượng giáo viên (Teacher). Teacher phải có khả năng trả lời hai loại câu hỏi sau:

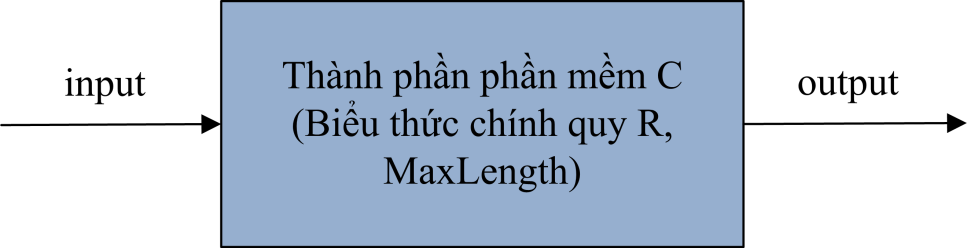
Loại 1 là câu hỏi truy vấn thành viên, với mỗi dẫn xuất Σ**\*** thì U không? Teacher sẽ trả lời *true* nếu U, trái lại Teacher sẽ trả lời *false*.

Loại 2 là câu hỏi kiểm tra ứng viên, với mỗi ứng viên M thì M có phải là ôtômát đoán nhận U không (tức là L(M) = U?). Nếu L(M) = U thì Teacher trả lời *true*, trái lại Teacher sẽ trả về một phản ví dụ phản ảnh sự khác biệt giữa L(M) và U.

Để làm được điều đó L\* duy trì một bảng T hai chiều gọi là bảng quan sát trong đó có các hàng là các tiền tố của các xâu đầu vào, các cột là các hậu tố của các xâu đầu vào. Các hàng của bảng T được chia làm hai loại, loại 1 là các hàng chứa các xâu đầu vào S, loại hai là chứa các xâu S.a với a Σ. Trong quá trình thực hiện thuật toán, L\* sẽ truy vấn đến đối tượng Teacher để cập nhật bảng T ở mỗi bước của thuật toán. Cho đến khi bảng T trở thành bảng đóng (closed) và nhất quán (consistent) thì L\* xây dựng một mô hình ứng viên M từ bảng quan sát T. Sau đó, L\* sử dụng câu hỏi kiểm tra ứng viên để kiểm tra M có phải mô hình chính xác của U hay không. Nếu Teacher trả lời là *true* thì thuật toán dừng, trái lại, L\* sẽ sử dụng phản ví dụ được trả về bởi Teacher để cập nhật lại bảng T và tiếp tục lặp lại quá trình trên.

### 3.1.1 Phương pháp sinh mô hình sử dụng thuật toán L\*

Trong phương pháp sinh mô hình tự động sử dụng thuật toán L\* [5], mỗi thành phần C của phần mềm được đặc tả bởi một biểu thức chính quy có được trong quá trình thiết kế hệ thống, từ biểu đồ tuần tự. Thành phần phần mềm đó có một giới hạn trên là MaxLength của độ dài tối đa của các chuỗi hành động có thể thực hiện được trên C. Khi đó, bài toán trở thành xây dựng mô hình chính xác cho thành phần phần mềm C có chuỗi các hành động được cho bởi một biểu thức chính quy và độ dài tối đa chuỗi các hành động có thể thực hiện được trên C. Thành phần C có thể được biểu diễn như trên hình 3.1 sau:



Hình 3.1: Mô hình thành phần phần mềm trong phương pháp sinh mô hình theo L\*.

Với thành phần phần mềm như hộp đen cho trên hình 3.1, phương pháp học L\* sẽ học để thu được mô hình ứng viên tối thiểu đoán nhận ngôn ngữ cho bởi biểu thức chính quy của thành phần phần mềm C. Để làm được việc này, L\* cần tương tác với một đối tượng Teacher. Teacher sử dụng thuật toán Vasilevskii – Chow (VC) [7] để trả lời hai loại câu hỏi như đã trình bày ở trên. Nếu ở bước lặp nào đó, ta có L(Mi) = L(C) thì thuật toán dừng lại và Mi sẽ là mô hình cần tìm và Teacher trả lại giá trị *true*, ngược lại Teacher sẽ trả về *false* và một chuỗi d phản ánh sự khác biệt của L(Mi) và L(C). Chuỗi này sẽ nằm trong L(Mi)\L(C) hoặc L(C)\L(Mi).

L\* duy trì một bảng T để ghi nhận mỗi chuỗi s Σ**\*** thì s có thuộc L(C) hay không bằng cách cho thực hiện chuỗi s trên C. Ở mỗi bước, L\* sử dụng bảng T để xây dựng môt mô hình ứng viên Mi và yêu cầu Teacher trả lời câu hỏi kiểm tra ứng viên. Nếu Teacher trả lời *true* thì thuật toán sẽ dừng. Trái lại L\* sử dụng giá trị d biểu diễn sự khác biệt giữa Mi và C được trả lại bởi Teacher để cập nhật lại bảng T.

***Định nghĩa 3.1***: Bảng quan sát T là một bộ gồm ba thành phần (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 giữa mô hình ứng viên Mi với thành phần C 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 L(C), trái lại T(s) = *false*.

***Định nghĩa 3.2***: Bảng quan sát *đóng* và *nhất quán*.

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

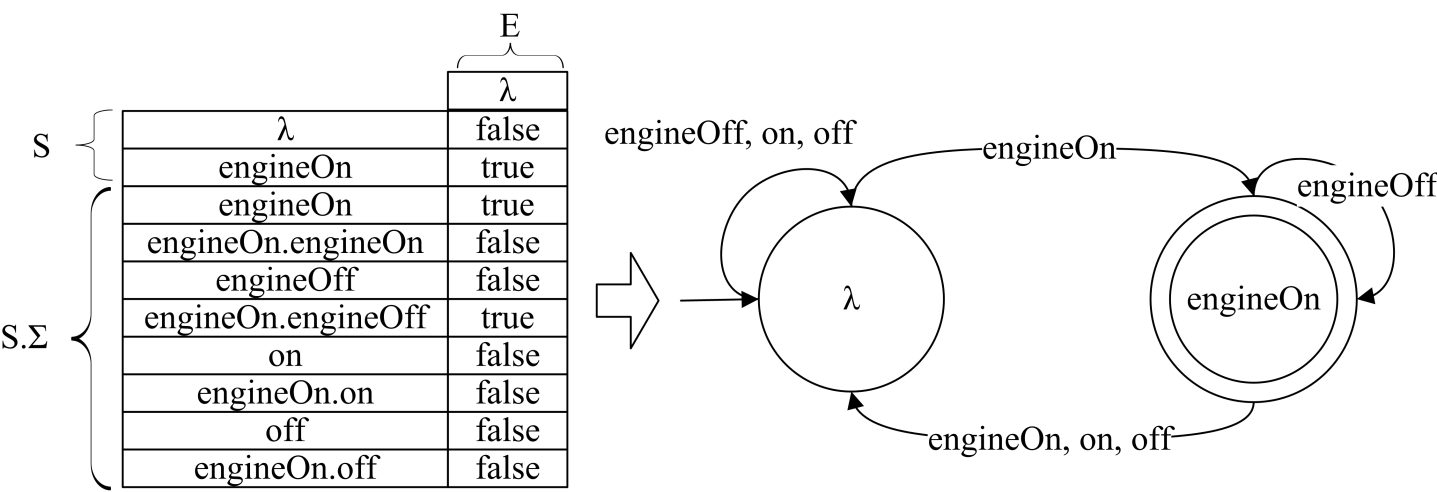
Một bảng quan sát được gọi là *nhất quán* nếu với s1 S, và s2 S mà có T(s1e)=T(s2e), E thì Σ, ta có T(s1ae)=T(s2ae).

Trong trường hợp bảng *đóng* ’ biểu diễn trạng thái kế tiếp của sau khi thực hiện một hành động *a,* mọi 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.

Với mỗi bảng *đóng* và *nhất quán* (S, E, T) ta luôn xây dựng được một DFA tương ứng D = (Q, D, , q0, F) như sau:

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

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



Hình 3.2: Xây dựng một ứng viên từ bảng quan sát *đóng* và *nhất quán*.

**Thuật toán 3.1**: Thuật toán L\*[2] học mô hình chính xác của thành phần phần mềm.

*Đầu vào*: Một thành phần phần mềm C được đặc tả bằng biểu thức chính quy và bảng quan sát rỗng (S, E, T).

*Đầu ra*: Mô hình chính xác của C.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 1 | Khởi tạo S và E là {λ}. | | 2 | Dùng câu hỏi truy vấn thành viên cho λ và mỗi a Σ. | | 3 | Tạo bảng quan sát đầu tiên (S, E, T). | | 4 |  | | 5 | **repeat** | | 6 | **while** (S, E, T) là chưa đóng và nhất quán **do** | | 7 | **if** (S, E, T) là chưa nhất quán **then** | | 8 | Tìm s1 và s2 trong S, a Σ, và e E sao cho | | 9 | row(s1) = row(s2) and T(s1.a.e) ≠ T(s2.a.e), | | 10 | Thêm a.e vào E. | | 11 | Mở rộng T thành (S ∪ S. Σ).E sử dụng câu hỏi truy vấn thành viên. | | 12 | **end if** | | 13 | **if** (S, E, T) là chưa đóng **then** | | 14 | Tìm s1 S và a Σ sao cho | | 15 | row(s1.a) khác với row(s) với mọi s S. | | 16 | Thêm s1.a vào S. | | 17 | Mở rộng T thành (S U S. Σ).E sử dụng câu hỏi truy vấn thành viên. | | 18 | **end if** | | 19 | **end while** | | 20 | Khi (S, E, T) là đóng và nhất quán, đặt M = M(S, E, T) là mô hình ứng viên. | | 21 | Sử dụng câu hỏi truy vấn kiểm tra thành viên, kiểm tra M. (sử dụng thuật toán VC được trình bày phía dưới). | | 22 | **if** Teacher trả lời bằng một phản ví dụ t **then** | | 23 | Thêm t và tất cả tiền tố của nó vào S. | | 24 | Và mở rộng T thành (S ∪ S. Σ).E sử dụng câu hỏi truy vấn thành viên. | | 25 | **end if** | | 26 | **until** Teacher trả lời true với mô hình ứng viên M. | | 27 | **dừng và trả lại** M. | |

Chi tiết của thuật toán 3.1 như sau:

Đầu tiên, thuật toán khởi tạo các giá trị S và E của bảng quan sát T là {λ} (dòng 1), sau đó thuật toán dùng câu hỏi truy vấn thành viên cho λ và với mỗi kí tự a thuộc bảng chữ cái (dòng 2) để cập nhật bảng quan sát T đầu tiên (dòng 3). Tại mỗi bước lặp của thuật toán (dòng 5), thuật toán kiểm tra bảng quan sát T xem bảng đã phải là đóng và nhất quán chưa (dòng 6). Nếu bảng chưa là nhất quán (dòng 7) thì thuật toán tìm hai giá trị s1 và s2 trong S, a thuộc bảng chữ cái, e thuộc E sao cho toàn bộ dòng T tương ứng với s1 và s2 là bằng nhau nhưng dòng T tương ứng s1.a và s2.a thì khác nhau (dòng 8, 9) và thêm a.e vào E (dòng 10). Sau đó thuật toán mở rộng bảng quan sát T sử dụng câu hỏi truy vấn thành viên (dòng 11). Nếu bảng là chưa đóng thì thuật toán tìm s1 thuộc S và a thuộc bảng chữ cái sao cho dòng T tương ứng với s1.a và s khác nhau với mọi s thuộc S (dòng 14,15). Sau đó, thuật toán thêm s1.a vào S và tiến hành mở rộng bảng quan sát T sử dụng câu hỏi truy vấn thành viên (dòng 16, 17). Bước kiểm tra bảng đóng và nhất quán được lặp lại cho đến khi bảng quan sát là đóng và nhất quán. Khi đó, thuật toán xây dựng mô hình ứng viên M và sử dụng câu hỏi kiểm tra thành viên để kiểm tra xem mô hình ứng viên vừa xây dựng đã phù hợp chưa (dòng 20, 21). Nếu mô hình ứng viên đó chưa phù hợp, đối tượng Teacher trả về một phản ví dụ (dòng 22) thì thuật toán thêm toàn bộ phản ví dụ và các tiền tố của nó vào thành phần S của bảng quan sát và mở rộng bảng quan sát thu được bằng câu hỏi truy vấn thành viên (dòng 23, 24). nếu mô hình ứng viên đã phù hợp thì thuật toán dừng lại và trả về mô hình ứng viên tương ứng (dòng 27).

**Độ phức tạp**: Độ phức tạp của thuật toán này là O(kn2 + nlogm), trong đó k là độ lớn của 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 Thuật toán Vasilevskii-Chow

Thuật toán VC kiểm tra mô hình ứng viên Mi do thuật toán L\* đưa ra xem đó có phải là mô hình chính xác của thành phần phần mềm đã cho hay không. Ý tưởng chính của thuật toán là kiểm tra toàn bộ các xâu s ∈ Σ\* thỏa mãn điều kiện cùng thuộc hoặc cùng không thuộc L(C) và L(M). Nếu điều kiện này được thỏa mãn thì VC xác nhận Mi là mô hình đặc tả chính xác hành động của thành phần C. Ngược lại, VC trả về phản ví dụ cung cấp cho L\* để sinh ra mô hình ứng viên Mi+1 tốt hơn. Phản ví dụ được cung cấp là minh chứng cụ thể cho sự khác biệt giữa ngôn ngữ của Mi và thành phần C.

**Thuật toán 3.2**: Thuật toán VC.

*Đầu vào*: S, E, C, n, thành phần phần mềm C. Trong đó: S, E là tập các tiền tố, hậu tố của bảng quan sát (S,E,T), n là giá trị cận trên kích thước của các chuỗi hành động của thành phần phần mềm C.

*Đầu ra*: true nếu mô hình ứng viên Mi thỏa mãn là mô hình chính xác cho hành động của C; phản ví dụ sxe nếu mô hình ứng viên Mi không thỏa mãn là mô hình chính xác cho hành động của C.

|  |
| --- |
| 1. t = |S|.  2. **for** k = 1 **to** n – t **do**  3. **for** mỗi chuỗi x có kích thước k, *s* S, *e* E **do**  4. **if** check(sxe) **then**  5. **return** sxe.  6. **end if**  7. **end for**  8. **end for**  9. **return** true. |

Chi tiết của thuật toán 3.2 như sau:

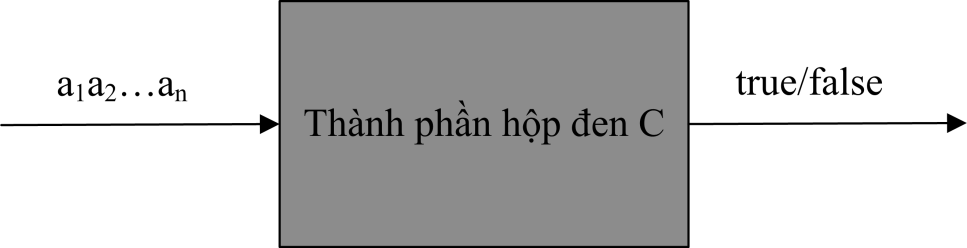
Ở mỗi bước lặp của thuật toán L\*, ta phải xây dựng một mô hình ứng viên Mi từ bảng quan sát (S,E,T). Sau đó mô hình này sẽ được kiểm tra tính tương thích giữa nó với thành phần phần mềm C. Để làm việc này, thuật toán VC sẽ kiểm tra tất cả các chuỗi s ∈ Σ\* có độ dài nhỏ hơn độ dài tối đa cho phép xem s có đều thuộc L(M) và L(C) hoặc s có đều không thuộc hai tập đó hay không. Nếu với mọi chuỗi s như vậy mà ta có s đều thuộc L(M) và L(C) hoặc s đều không thuộc hai tập đó thì ta nói Mi tương thích với thành phần phần mềm C hay nói cách khác Mi là mô hình chính xác của thành phần phần mềm C.

Để làm được điều này, ta xây dựng một hàm check (sxe) sao cho hàm này trả về *true* nếu s ∈ L(C) và s ∉ L(M) hoặc s ∈ L(M) nhưng s ∉ L(C). Nếu hàm check trả về *false* thì ta đã tìm được chuỗi sxe là chuỗi phản ví dụ để trả lại cho L\*.

**Độ phức tạp**: Độ phức tạp của thuật toán VC là O(t2kn-t+1) [7], trong đó, t là kích thước của S (tức là số trạng thái của Mi), k là kích thước của bảng chữ cái Σ và n là cận trên độ dài của các chuỗi hành động của C.

## 3.2 Phương pháp sinh mô hình sử dụng thuật toán Thompson

Phần này sẽ trình bày phương pháp sinh mô hình cho thành phần phần mềm sử dụng thuật toán Thompson [6]. Trong phương pháp này, mỗi thành phần phần mềm C được coi như một hộp đen, chúng ta hoàn toàn không biết được tất cả các trạng thái và các hành động có thể thực hiện được trên C. Giả thiết rằng mỗi thành phần C có những thông tin sau: Với mỗi chuỗi các hành động ta có thể kiểm tra được nó có phải là một thực hiện trên C hay không.



Hình 3.3: Mô hình giả thiết về thành phần phần mềm C.

Với mỗi chuỗi a1a2…an ta luôn kiểm tra được nó có phải là một thực hiện trên C hay không. Nếu chuỗi a1a2…an là thực hiện được trên C thì trả về giá trị *true* và được gọi là một chuỗi của thành phần C; trái lại trả về giá trị *false*. Ngoài ra, chúng ta cũng giả thiết rằng ta biết được độ dài lớn nhất của tập tất cả các chuỗi của thành phần C. Tập tất cả các chuỗi thực hiện được trên C gọi là ngôn ngữ đoán nhận bởi C, ký hiệu L(C). Tập tất cả các chuỗi hành động trên bảng chữ cái hữu hạn (kí hiệu Σ) của thành phần C được gọi là D={σ ∈ Σ | |σ| ≤ MaxLength}.

Với giả thiết như vậy, để thu được tập các chuỗi của thành phần C với độ dài nhỏ hơn hoặc bằng MaxLength, ta duyệt qua tất cả các thành phần của D để tính toán các phần tử của L(C). Với mỗi xâu σ trong tập D, nếu ta có thể thực hiện thử nghiệm thành công trên thành phần C thì σ được thêm vào L(C). Quá trình này được lặp lại với toàn bộ thành phần của D. Kết quả là chúng ta có được tập các hành động có thể thực hiện được trên thành phần C.

Với một thành phần C đã cho, giả sử σ = a1a2…an là một chuỗi của L(C), chuỗi này có thể được biểu diễn bởi một biểu thức chính quy mà được hình thành bằng việc ghép nối các biểu thức chính quy thành phần ai với i = . Biểu thức chính quy biểu diễn tập L(C) là hợp của các biểu thức chính quy biểu diễn từng thành phần của L. Đặt L = {σi, σ2, …, σm} là tập các xâu mà ở đó σi = ai1ai2…ain (n ≤ MaxLength), chúng ta kí hiệu Rσi là biểu thức chính quy đại diện cho ngôn ngữ chỉ chứa chuỗi σi. Cuối cùng, ta có biểu thức chính quy L biểu diễn tập các hành động có thể thực hiện được trên thành phần C như sau: RL = Rσ1 + Rσ2 + … + Rσm. Ta sử dụng thuật toán Thompson với đầu vào là RL để sinh ra mô hình cho thành phần phần mềm C ban đầu.

***Định nghĩa 3.3***: Các phép toán [1].

Giả sử Σ là bảng chữ cái hữu hạn không rỗng, Σ\* là tập tất cả các xâu (kể cả xâu rỗng) được xây dựng trên Σ.

Khi đó Σ+ = Σ\* là tập tất cả các xâu không rỗng trên Σ.

Tập E ⊆ Σ+ được gọi là một ngôn ngữ trên bảng chữ cái Σ.

Khi đó trên tập tất cả các ngôn ngữ ta định nghĩa các phép toán hợp, nhân, lặp như sau:

1. **Phép hợp**

Cho 2 ngôn ngữ E1, E2 trên tập Σ, ta định nghĩa phép hợp của E1và E2 như sau:

E1  E2 = { **|** E1 hoặc E2}.

1. **Phép nhân**

Cho 2 ngôn ngữ E1, E2 trên tập Σ, ta định nghĩa phép nhân E1với E2 như sau:

E1.E2 = { **|** E1 và E2}.

1. **Phép lặp**

Với ngôn ngữ E trên Σ ta định nghĩa phép lặp của E như sau:

E+ = E1 E2 E3 … = , trong đó En = EE…E (n lần).

***Định nghĩa 3.4:*** Ngôn ngữ chính quy [1].

* Các ngôn ngữ sơ cấp {, {ai} với ai là ngôn ngữ chính quy trên .
* Nếu E và F là 2 ngôn ngữ chính quy trên thì EF, E.F, E+ cũng là ngôn ngữ chính quy trên .
* Không có ngôn ngữ chính quy nào khác trên ngoài các ngôn ngữ đã được định nghĩa ở 2 bước trên.

***Định nghĩa 3.5***: Biểu thức chính quy [1].

Trên bảng chữ cái , ta định nghĩa biểu thức chính quy một cách đệ quy như sau:

* là biểu thức chính quy, biểu diễn ngôn ngữ rỗng.
* thì a là biểu thức chính quy, nó biểu diễn ngôn ngữ {a}.
* Nếu r, s là 2 biểu thức chính quy trên biểu diễn 2 ngôn ngữ R, S tương ứng, khi đó:
  + (r) (s) là biểu thức chính quy biểu diễn ngôn ngữ R S.
  + (r).(s) là biểu thức chính quy biểu diễn ngôn ngữ R.S.
  + (r)+ là biểu thức chính quy biểu diễn ngôn ngữ R+.

**Chú ý 3.1**:

Mọi ngôn ngữ chính quy trên đều nhận được từ các ngôn ngữ hữu hạn bằng cách áp dụng một số hữu hạn lần các phép toán hợp, nhân và lặp.

Một ngôn ngữ trên là chính quy khi và chỉ khi nó biểu diễn được bởi một biểu thức chính quy.

Một ngôn ngữ chính quy là vô hạn khi và chỉ khi biểu thức chính quy biểu diễn nó chứa phép toán lặp.

**Thuật toán 3.3**: Xây dựng ôtômát hữu hạn đoán nhận ngôn ngữ cho bởi biểu thức chính quy đại diện cho thành phần phần mềm C đã trình bày ở trên.

*Đầu vào*: Tập các dẫn xuất có độ dài không vượt quá MaxLength trên bảng chữ cái Σ = {a1, a2, …, an}.

L = {X1, X2, …, Xm} với **|** Xi **| MaxLength.**

Trong đó, mỗi Xi được xây dựng từ một số hữu hạn aj với phép toán nhân.

*Đầu ra*: Ôtômát hữu hạn M = (Q, , , q0, F) đoán nhận ngôn ngữ L.

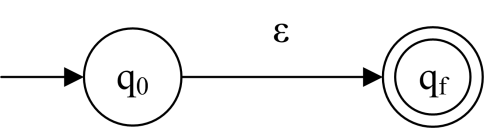
|  |
| --- |
| 1: Khởi tạo: Q = {q0}, F = {qf }, = .  2: **for each** (Xi ∈ L) **do**  3: Thêm qi1 vào Q.  4: **for** j = 1 **to** Xi.length **do**  5: Thêm luật (qij, aj, qij+1) vào .  6: Thêm qij+1 vào Q.  7: **end for**  8: Thêm luật (q0, , qi1) vào .  9: Thêm luật (qij+1, , qf) vào .  10: **end for** |

Chi tiết của thuật toán 3.3 như sau:

Đặt R = X1 X2 … Xm.

Mỗi Xi được xây dựng như trên là một biểu thức chính quy. Do đó, theo ***định nghĩa 3.5*** thì R cũng là một biểu thức chính quy. Ta sử dụng thuật toán Thompson để xây dựng ôtômát hữu hạn từ biểu thức chính quy R như sau:

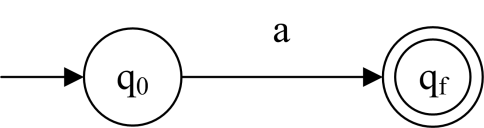
* Tách R thành các biểu thức chính quy thành phần r1, r2,…, rk. Sau đó, ta áp dụng luật 1 và luật 2 để xây dựng các ôtômát đoán nhận các ngôn ngữ sinh bởi các biểu thức chính quy thành phần L(r1), L(r2),…, L(rk).
* **Luật 1**: Đối với các ký hiệu rỗng, ta xây dựng ôtômát đoán nhận ngôn ngữ {} như sau:



Hình 3.4: Ôtômát tương ứng cho thành phần ε.

Trên hình 3.4 là một ôtômát tương ứng cho thành phần ε, trong đó q0 là trạng thái đầu, qf là trạng thái kết thúc.

* **Luật 2**: Đối với các ký hiệu a ∈ Σ ta xây dựng ôtômát đoán nhận ngôn ngữ {a} như sau:



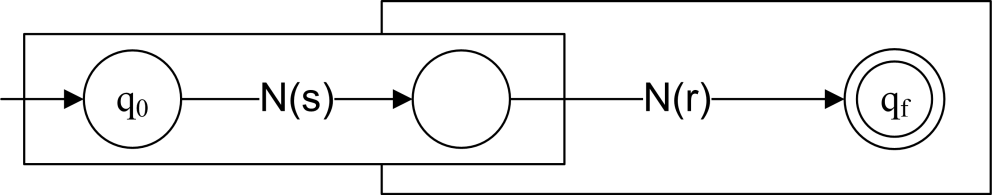
Hình 3.5: Ôtômát tương ứng cho thành phần a.

Trên hình 3.5 là một ôtômát tương ứng cho thành phần a, trong đó q0 là trạng thái đầu, qf là trạng thái kết thúc.

Ta sử dụng luật 3 để xây dựng ôtômát N đoán nhận ngôn ngữ L(R) như sau.

* **Luật 3**: Giả sử N(r) và N(s) là các ôtômát thành phần ứng với các biểu thức chính quy r và s. Khi đó:

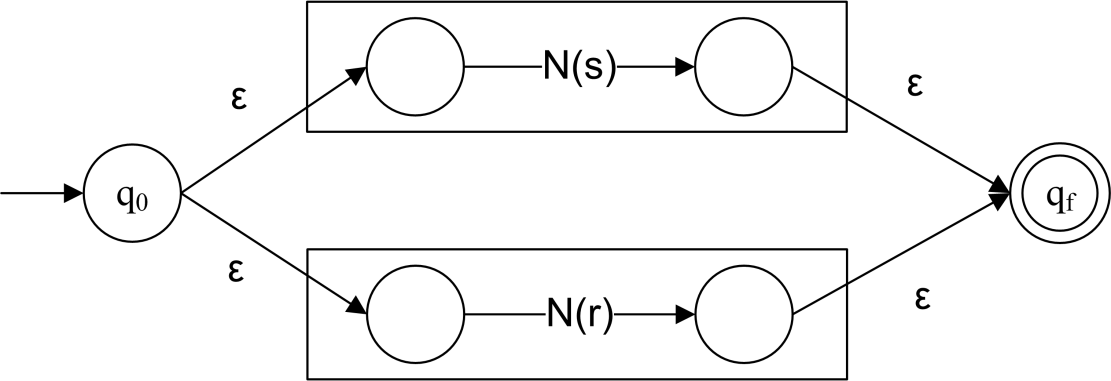
1. Với biểu thức chính quy dạng (s).(r) thì ta xây dựng ôtômát không đơn định N đoán nhận ngôn ngữ S.R như sau:



Hình 3.6: Ôtômát tương ứng cho thành phần (s).(r).

Trên hình 3.6 là ôtômát tương ứng cho thành phần (s).(r), trong đó: q0 là trạng thái ban đầu, qf là trạng thái kết thúc và N nhận được từ N(s) và N(r) bằng cách lấy trạng thái ban đầu của N(s) làm trạng thái ban đầu của N, trạng thái kết thúc của N(r) làm trạng thái kết thúc của N và đồng nhất trạng thái kết thúc của N(s) với trạng thái đầu của N(r).

1. Với biểu thức chính quy dạng (r) (s) thì ta xây dựng ôtômát không đơn định N đoán nhận ngôn ngữ RS như trên hình 3.7 sau:



Hình 3.7: Ôtômát tương ứng cho thành phần (s) ∪ (r).

Với q0 là trạng thái ban đầu, qf là trạng thái kết thúc và N nhận được bằng cách tổ hợp 2 ôtômát thành phần N(s) và N(r) theo sơ đồ trên.

Trong cài đặt, thay vì biểu diễn ôtômát dưới dạng đồ thị có hướng như trên ta biểu diễn dưới dạng hàm chuyển trạng thái. Với mỗi dẫn xuất Xi trong tập L ta lần lượt xây dựng các quy tắc chuyển trạng thái ứng với các hành động aj trong dẫn xuất Xi. Mỗi Xi được xây dựng theo thuật toán 3.3 đều có dạng Xi = ai1ai2..ain. Khi đó, ôtômát thành phần sinh dẫn xuất Xi được xây dựng theo luật 2 và luật 3.a. Biểu diễn bằng các quy tắc chuyển trạng thái, chúng ta xây dựng được các quy tắc tương ứng (qi1, ai1, qi2), (qi2, ai2, qi3), …, (qin, ain, qin+1). Ôtômát M được xây dựng bằng cách hợp các ôtômát thành phần lại theo luật 3.b. Khi biểu diễn bằng quy tắc chuyển trạng thái ta xây dựng các quy tắc tương ứng (q0, , qi1) và (qin+1, , qf) trong đó q0 và qf lần lượt là trạng thái bắt đầu và kết thúc. Tập trạng thái Q được xây dựng bao gồm tất cả các trạng thái qj ở trên.

**Độ phức tạp**: Tập D được xây dựng như trên D = {σ ∈ Σ \* **|** **|**σ**| *MaxLength***}.

Ta có: size(D) = n1 + n2 + .. + nmaxLength = n.(nmaxLength – 1)/(n - 1).

Trong đó n là số phần tử của bảng chữ cái .

Độ phức tạp về thời gian của việc kiểm tra một chuỗi σ có là dẫn xuất của thành phần C hay không là O(MaxLength). Do đó, độ phức tạp của toàn bộ quá trình sinh dẫn xuất cho thành phần C là O(MaxLength) \* O(nmaxLength) = O(MaxLength \* nmaxLength).

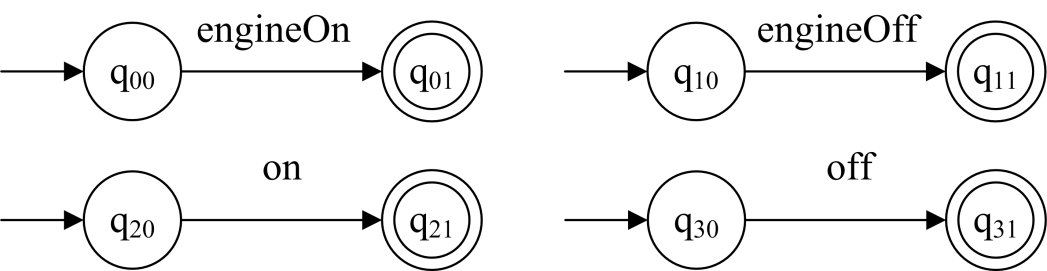
Ta lại có |L| ≤ |D| nên độ phức tạp của thuật toán sinh mô hình bằng thuật toán Thompson là nhỏ hơn độ phức tạp của quá trình sinh dẫn xuất. Vậy độ phức tạp của toàn bộ quá trình sinh mô hình cho thành phần phần mềm C là O(MaxLength \* nmaxLength).

**Ví dụ 3.2**: Cho tập dẫn xuất {engineOn, engineOn.engineOff, engineOn.engineOff.on, engineOn.engineOff.off} trên bảng chữ cái Σ = {engineOn, engineOff, on, off}.

Biểu thức chính quy L được xây dựng như sau:

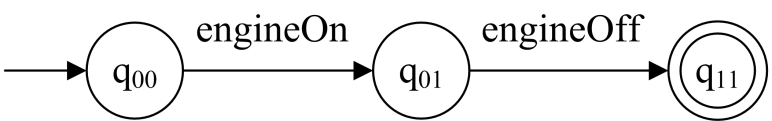
L=(engineOn)|(engineOn.engineOff)|(engineOn.engineOff.on)|(engineOn.engineOff.off).

Ta xây dựng các ôtômát thành phần đoán nhận các biểu thức chính quy cơ sở *engineOn*, *engineOff*, *on*, *off* tương ứng như trên hình 3.8 sau:



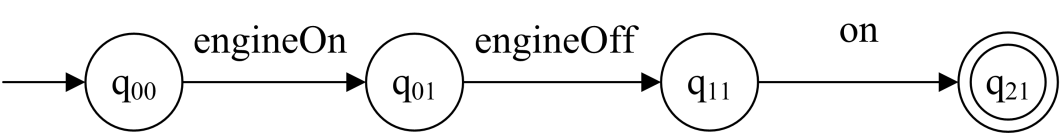
Hình 3.8: Ôtômát cho các biểu thức chính quy cơ sở.

Xây dựng ôtômát thành phần đoán nhận biểu thức chính quy *engineOn.engineOff* bằng cách lấy trạng thái đầu của *engineOn* làm trạng thái đầu của *engineOn.engineOff*, trạng thái kết của *engineOff* làm trạng thái kết của *engineOn.engineOff* và đồng nhất trạng thái kết của *engineOn* với trạng thái đầu của *engineOff* ta được ôtômát như trên hình 3.9 sau:

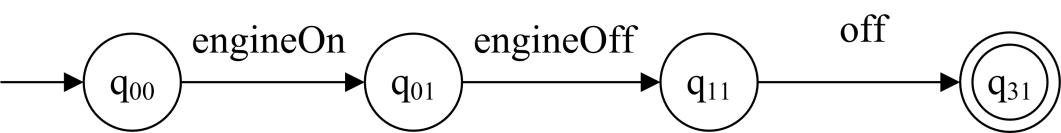


Hình 3.9: Ôtômát cho thành phần engineOn.engineOff.

Tương tự, ta xây dựng ôtômát cho các thành phần engineOn.engineOff.on và engineOn.engineOff.off như trên hình 3.10 và 3.11 sau:

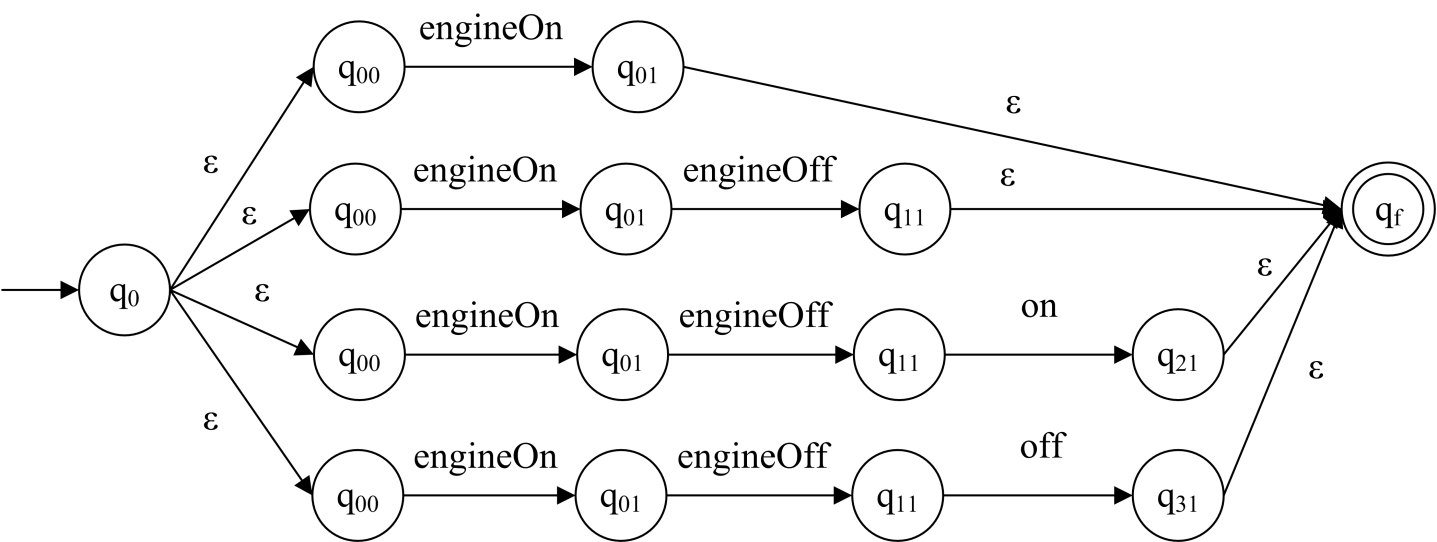


Hình 3.10: Ôtômát cho thành phần engineOn.engineOff.on.



Hình 3.11: Ôtômát cho thành phần engineOn.engineOff.off.

Ôtômát tương đương của L là hợp các ôtômát thành phần, ta có ôtômát cuối cùng như trên hình 3.12 sau:



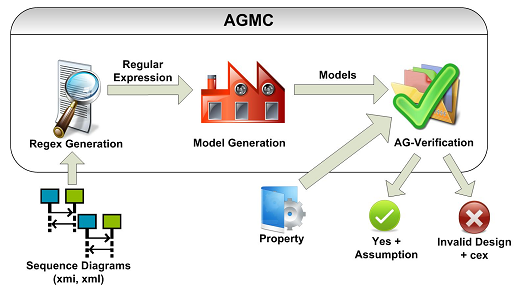
Hình 3.12: Ôtômát tương đương với L.

## 3.3 Hạn chế của các phương pháp sinh mô hình tự động theo thuật toán L\* và Thompson

Các phương pháp sinh mô hình tự động được trình bày trong phần 3.1 và 3.2 mặc dù đều có thể sinh được mô hình cho các thành phần phần mềm trong ngữ cảnh đặt ra của nghiên cứu, nhưng vẫn còn những hạn chế nhất định khiến cho việc áp dụng trong thực tế sẽ gặp nhiều khó khăn. Cụ thể, với phương pháp sử dụng thuật toán L\* [5], mặc dù có thể sinh được mô hình dựa vào thuật toán học L\* với đầu vào là một biểu thức chính quy các hành động của thành phần phần mềm, nhưng có hai hạn chế nổi bật của phương pháp này là hạn chế độ dài tối đa (MaxLength) của chuỗi các hành động thực hiện được trên phần mềm và độ phức tạp của thuật toán VC (O(t2kn-t+1)). Khi thực thi đối tượng Teacher để trả lời các câu hỏi truy vấn thành viên và kiểm tra ứng viên, chúng ta đã phải phân tích biểu thức chính quy đã cho để xây dựng được tập các chuỗi hành động của phần mềm. Trong khi biểu thức chính quy thì có thể biểu diễn chuỗi hành động có độ dài vô hạn, nhưng việc xây dựng các chuỗi này là hữu hạn, nên phương pháp này phải đặt giới hạn cận trên cho độ dài các chuỗi các hành động của thành phần phần mềm. Chính vì điểm này, mô hình thu được sau cùng của phương pháp sẽ có độ chính xác không cao nếu như độ dài tối đa giả thiết ban đầu là không đủ để biểu có thể lấy được toàn bộ các chuỗi hành động có thể thực hiện được trên thành phần phần mềm. Phương pháp này chỉ có thể xây dựng được mô hình đại diện cho tập các chuỗi với độ dài hữu hạn (MaxLength) các hành động được thực hiện thành phần phần mềm. Với phương pháp sử dụng thuật toán Thompson [6], trong ngữ cảnh của nghiên cứu, việc xây dựng được tập các chuỗi hành động có thể thực hiện được trên thành phần phần mềm sẽ bị giới hạn bởi độ dài tối đa các chuỗi hành động đó và độ phức tạp của quá trình này là lớn (O(MaxLength \* nmaxLength)). Sau đó, khi đã có được tập các chuỗi hành động này và đã có được biểu thức chính quy biểu diễn tập các chuỗi hành động đó, việc sử dụng thuật toán Thompson để xây dựng các ôtômát sinh ra các ôtômát khá cồng kềnh do có cả các chuyển trạng thái rỗng, các trạng thái không đến được hoặc không kết thúc. Điều này khiến cho thuật toán phải có thêm một bước loại bỏ các chuyển trạng thái rỗng, các trạng thái không kết thúc và các trạng thái không đến được trong quá trình tối ưu mô hình thu được. Điều này cũng làm giảm hiệu quả tính toán của thuật toán. Tuy nhiên, nếu thuật toán Thompson được sử dụng trong một ngữ cảnh khác là dùng để sinh mô hình cho thành phần phần mềm từ biểu thức chính quy có được trong quá trình thiết kế hoặc có được từ đặc tả của thành phần phần mềm mua được từ bên phát triển thứ ba thì phương pháp này sẽ không bị giới hạn bởi độ dài tối đa các chuỗi hành động có thể thực hiện được trên thành phần phần mềm. Tuy nhiên phương pháp này vẫn có hạn chế như đã nêu trên. Để giải quyết các hạn chế của các phương pháp sinh mô hình dựa vào thuật toán L\* và thuật toán Thompson như đã đề cập ở trên, chương 4 sẽ đề xuất sử dụng một phương pháp sinh mô hình cho thành phần phần mềm sử dụng thuật toán CNNFA.

# Chương 4: Nghiên cứu phương pháp sinh mô hình tự động cho thành phần phần mềm sử dụng thuật toán CNNFA

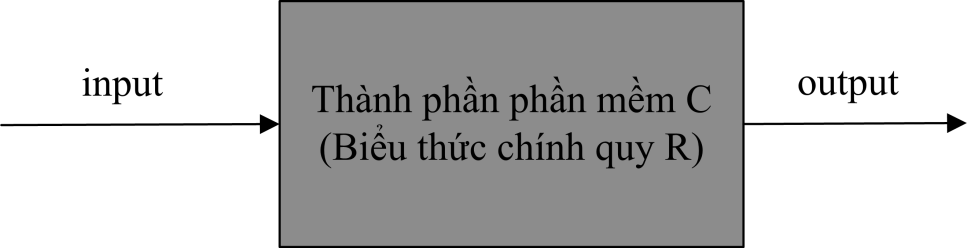
Trong ngành công nghiệp phần mềm ngày nay, người ta ngày càng có xu hướng hướng đến phương pháp để có thể đạt đến sự tự động hóa một số công đoạn trong quy trình sản xuất phần mềm từ giai đoạn phân tích, đặc tả yêu cầu, thiết kế, sinh mã nguồn, kiểm thử và kiểm tra tính đúng đắn với yêu cầu ban đầu. Trong nghiên cứu [5], các tác giả đã đưa ra một cách tiếp cận hướng đến tự động hóa từ khâu thiết kế, thông qua biểu đồ tuần tự, sinh mô hình tự động và kiểm chứng đảm bảo giả định. Đây là một bước tiến đầu tiên nhưng rất quan trọng cho việc tự động hóa này.



Hình 4.1: Kiến trúc kiểm chứng mô hình đảm bảo giả định [5].

Trong kiến trúc trên hình 4.1, ta thấy rằng, để có thể có được một kiến trúc có thể hoạt động được một cách thực tế và hiệu quả, ta cần có một thành phần sinh mô hình (Model Generation) hoạt động hiệu quả cho các thành phần phần mềm trong thực tế. Tuy vậy, trong phương pháp này, các tác giả sử dụng phương pháp sinh mô hình sử dụng L\* để sinh mô hình thành phần phần mềm từ biểu thức chính quy đã có được từ biểu đồ tuần tự ban đầu. Như ta đã trình bày trong chương 3, phương pháp sinh mô hình này còn có những giới hạn nhất định mà khó có thể thực hiện được trong thực tế như: còn phụ thuộc vào độ dài tối đa của chuỗi các hành vi của thành phần phần mềm, độ phức tạp tính toán lớn của thuật toán VC.

Chương 4 đề xuất áp dụng một phương pháp khác là phương pháp áp dụng thuật toán CNNFA [4], [9] cho việc sinh mô hình cho thành phần phần mềm nói trên để có thể áp dụng được trong các thành phần phần mềm trong thực tế với độ phức tạp tính toán cả về không gian và thời gian là nhỏ hơn so với các phương pháp được đề cập trong chương 3 về mặt lý thuyết. Trong phương pháp này, ta giả thiết thành phần phần mềm được mô tả bằng một biểu thức chính quy đầy đủ có được từ việc phân tích đầu ra của quá trình thiết kế là biểu đồ tuần tự được mô tả bởi các file xml, xmi như đề cập trong [5]. Đối với các thành phần cung cấp bởi bên phát triển thứ ba, nếu nhà cung cấp cung cấp cho ta đặc tả của thành phần phần mềm bằng một biểu thức chính quy các hành động được thực thi bởi thành phần phần mềm thì ta cũng có thể sử dụng phương pháp này để sinh mô hình cho thành phần phần mềm đó. Hình 4.2 mô tả mô hình thành phần phần mềm giả thiết cho nghiên cứu trong chương này.



Hình 4.2: Mô hình thành phần phần mềm cho bởi biểu thức chính quy.

Bài toán của ta trở thành sinh mô hình tự động cho thành phần phần mềm bằng việc phân tích biểu thức chính quy các hành động của thành phần phần mềm và sinh ra mô hình mô tả chính xác hành vi của thành phần phần mềm đó. Để có thể áp dụng thuật toán CNNFA để sinh ra mô hình cho thành phần phần mềm, phần 4.1 sẽ trình bày một số khái niệm, cấu trúc dữ liệu liên quan đến quá trình sinh mô hình. Sau đó, phần 4.2 đề cập các công thức được dùng để tính toán các biểu diễn CNNFA cho các biểu thức chính quy thành phần: λ, a ∈Σ, J|K, J.K, và J\*. Tiếp theo, phần 4.3 trình bày phép duyệt biểu thức chính quy và áp dụng các công thức có được trong phần 4.2 để xây dựng biểu diễn CNNFA cho ôtômát đuôi tương đương cho biểu thức chính quy ban đầu. Sau đó, phần 4.4 tính toán mô hình của thành phần phần mềm từ biểu diễn CNNFA đã có được từ phần 4.3. Mô hình thành phần phần mềm có được từ phần 4.4 được tối ưu hóa bằng các thuật toán được trình bày trong phần 4.5. Cuối cùng, phần 4.6 minh họa toàn bộ quá trình sinh mô hình cho thành phần phần mềm áp dụng thuật toán CNNFA bằng một ví dụ cụ thể.

## 4.1 Một số khái niệm liên quan

Phần 4.1 này trình bày một số khái niệm nhằm phục vụ cho việc biểu diễn biểu thức chính quy và ôtômát tương ứng của nó sau này.

***Định nghĩa 4.1***: Độ dài của biểu thức chính quy R, kí hiệu là |R|, là số lần xuất hiện của các kí tự bao gồm λ, kí tự trong bảng chữ cái, dấu ngoặc đơn, dấu sao và phép toán thay thế (|).

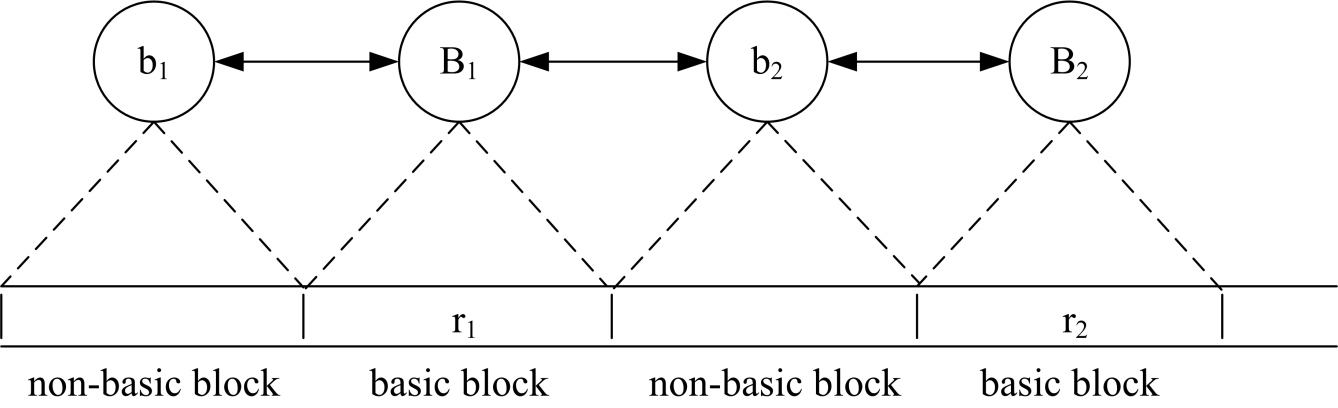
**Ví dụ 4.1**: Cho biểu thức chính quy (ab\*|(cd)), ta thấy rằng theo định nghĩa 4.1, độ dài của biểu thức chính quy đã cho là 10.

***Định nghĩa 4.2***: Khối cơ bản là một biểu thức chính quy con của R mà chứa tối thiểu một lần xuất hiện của các kí tự trong bảng chữ cái.

Các khối không cơ bản là các thành phần của biểu thức chính quy được chia tách bởi các khối cơ bản.

Mỗi một khối cơ bản có thêm hai thuộc tính gọi là *null* và *star*. Thuộc tính *null* trong khối cơ bản bằng {λ} khi và chỉ khi khối cơ bản đó chấp nhận chuỗi rỗng, ngược lại thuộc tính này bằng ∅. Thuộc tính *star* để chỉ có hay không có một kí tự “\*” trên biểu thức chính quy gắn với khối này.

**Ví dụ 4.2**: Khối cơ bản và khối không cơ bản của biểu thức chính quy.



Hình 4.3: Khối cơ bản và khối không cơ bản của biểu thức chính quy.

Hình 4.3 cho thấy quá trình phân tích biểu thức chính quy thành các khối cơ bản và không cơ bản đan xen lẫn nhau. Mỗi khối cơ bản B1, B2 chứa một biểu thức chính quy con tương đương là r1 và r2. Các khối không cơ bản b1, b2 được ngăn cách bởi các khối cơ bản B1 và B2.

***Định nghĩa 4.3***: NNFA (normal NFA) là một NFA mà tất cả các cạnh dẫn đến cùng một trạng thái có cùng nhãn. Ta sẽ đánh nhãn cho trạng thái thay vì cho cạnh.

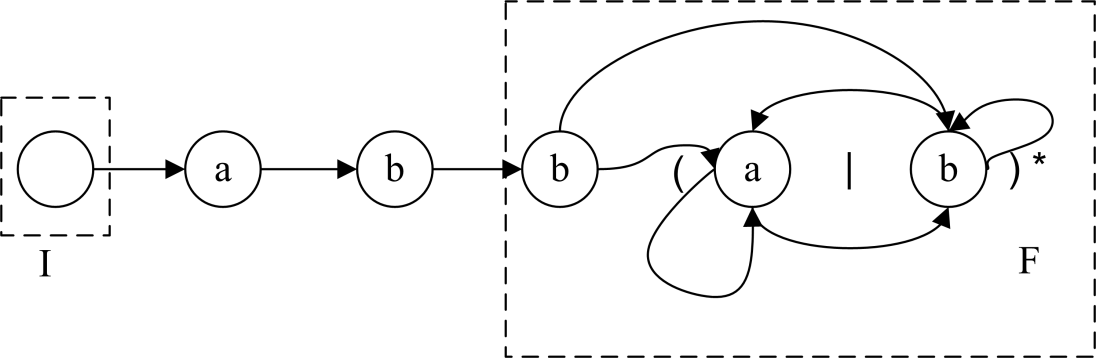
NNFA = (Σ, Q, δ, I, F, A), trong đó:

* Σ: Bảng chữ cái,
* Q: Tập trạng thái,
* δ ⊆ QxQ: Tập các cạnh chưa đánh nhãn,
* I ⊆ Q: Tập trạng thái bắt đầu,
* F ⊆ Q: Tập trạng thái kết thúc, và
* A: Q 🡪 Σ: Ánh xạ cho các trạng thái q ∈ Q thành một nhãn thuộc Σ.

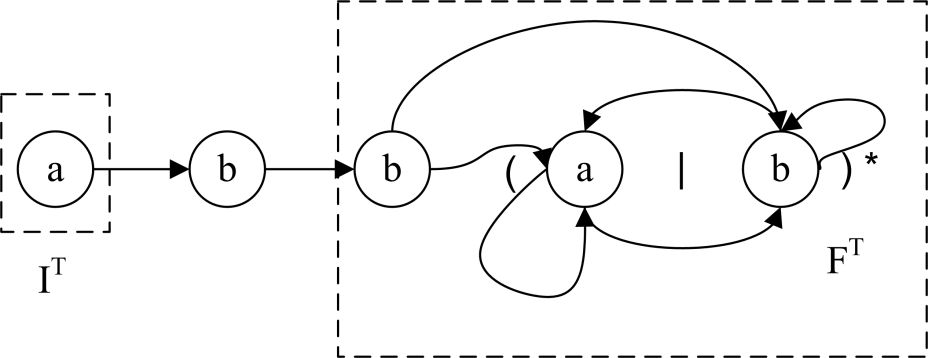
**Chú ý 4.1**: Một McNaughton/Yamada NNFA (viết tắt là MYNNFA) là một NNFA với một trạng thái bắt đầu với bậc vào là 0.

***Định nghĩa 4.4***: Ôtômát đuôi của một MYNNFA M= (Σ, Q, δ, I={q0}, F, A) là một NNFA MT = (ΣT, QT, δT, IT, FT, AT). Trong đó: ΣT = Σ, QT = Q – {q0}, δT = {[x,y] ∈ δ | x ≠ q0}, IT = {y : [q0,y] ∈ δ}, FT = F – {q0}, AT = A.

**Ví dụ 4.3**: Hình 4.4 và 4.5 minh họa một ví dụ về một ôtômát đuôi của MYNNFA.



Hình 4.4: MYNNFA tương đương của biểu thức chính quy abb(a|b)\*.



Hình 4.5: Ôtômát đuôi của MYNNFA tương đương với biểu thức chính quy abb(a|b)\*.

***Định nghĩa 4.5***: Cho hai tập hợp G và H, cặp của G và H được định nghĩa như sau:

pair(G,H) = {[G,H]}, nếu cả G và H đều khác rỗng; ngược lại, pair(G,H) = ∅.

**Chú ý 4.2**: Đặt nredM = - , ta gọi biểu diễn trễ của nó là *lazynred*. Tương tự, chúng ta cũng có biểu diễn trễ của δ được gọi là *lazyδ*. Cả *lazynred* và *lazyδ* đều là một danh sách các cặp của hai tập hợp. Chúng ta không tính toán ngay biểu thức trong suốt quá trình duyệt biểu thức chính quy mà chúng ta duy trì một cặp của hai tập hợp [, ] cho đến khi chúng ta có biểu diễn cuối cùng của ôtômát đuôi tương đương với biểu thức chính quy đã cho.

***Định nghĩa 4.6***: CNNFA.

Biểu diễn CNNFA của một ôtômát đuôi của ôtômát MYNNFA MT = (, , , , , ) là một biểu diễn nén được định nghĩa là bộ 7 thành phần như sau: PT = (, , , , , , ), trong đó:

* = ,
* = ,
* là biểu diễn trễ của nredM = - ,
* là biểu diễn trễ của ,
* = ,
* = , và
* = .

Từ một ôtômát đuôi của MYNNFA MT = (ΣT, QT, δT, IT, FT, AT) và *nullM* ta có thể xây dựng lại MYNNFA ban đầu M như sau:

|  |  |
| --- | --- |
| * Σ = ΣT, * Q = QT ∪{q0}, * δ = δT ∪ {[q0,y] | y ∈ IT}, * I = {q0}, * F = FT ∪ {q0}*nullM*, và * A=AT.   Trong đó, {q0}*nullM* = {q0} nếu *nullM* = {λ}, ngược lại {q0}*nullM* = ∅. | (4.1) |

**Chú ý 4.3**: Đôi khi, ta không viết Σ trong biểu diễn của NNFA, MYNNFA, ôtômát đuôi của MYNNFA và CNNFA khi nó đã quá rõ ràng. Đôi khi ta cho phép tập các nhãn A là không xác định với các trạng thái mà bậc vào là 0. Ví dụ trong trường hợp trạng thái bắt đầu của một MYNNFA.

Trong CNNFA, chúng ta biểu diễn các tập hợp *F* và *I* bằng việc sử dụng cấu trúc dữ liệu gọi là *F-Forest* và *I-Forest* như mô tả sau đây:

* Một tập được biểu diễn bằng một cây nhị phân.
* Các lá của cây được nối với nhau bằng danh sách liên kết đôi.
* Gốc của cây chứa con trỏ đến danh sách lá của cây.
* Hợp của hai tập hợp là một cây mới với gốc mới, hai tập đó trở thành cây con trái và cây con phải của cây mới. Phép hợp này có thể thực hiện được trong thời gian đơn vị đối với hai tập không giao nhau.

## 4.2 Sinh biểu diễn CNNFA cho các biểu thức chính quy thành phần

Bây giờ chúng ta có thể xây dựng biểu diễn CNNFA cho các ôtômát đuôi tương ứng với các biểu thức chính quy thành phần: λ, a, J|K, J.K, J\* như sau:

|  |  |
| --- | --- |
| = (=∅, = ∅, = ∅, = ∅, = ∅, = ∅),  nullλ = {λ}. | (4.2) |
| = (={q}, = pair(, ), = ∅, = {q}, ={q}, = {[q,a]}), nulla = ∅, trong đó a∈Σ và q là một trạng thái duy nhất. | (4.3) |

Sử dụng quy nạp, ta giả sử rằng J và K là hai biểu thức chính quy bất kỳ có biểu diễn CNNFA tương ứng là = (, , , ,, ) và = (, , , ,,), trong đó: và là không giao nhau, ta có các công thức sau đây:

|  |  |
| --- | --- |
| = ( = ∪ , = ∪ ∪ pair(, ) ∪ pair(, ), = ∪ , = ∪ , = ∪ , = ∪ ), nullJ|K = nullJ ∪ nullK. | (4.4) |
| = ( = ∪ , = pair(, ) ∪ nullK ∪ nullJ, = pair(, ) ∪ ∪ , = ∪ nullJ, = ∪ nullK, = ∪ ), nullJK = nullJnullK. | (4.5) |
| = ( = , = ∅, = ∪ , = , = , = ), nullJ\* = {λ}. | (4.6) |

Việc xây dựng biểu diễn CNNFA của ôtômát đuôi tương đương cho biểu thức chính quy *λ* trong công thức 4.2 là đơn giản. Công thức 4.3 cho chúng ta cách xây dựng biểu diễn CNNFA của ôtômát đuôi tương đương cho biểu thức chính quy *a*. Với a ∈ Σ, ta tạo mới một trạng thái duy nhất. Trong quá trình thực thi, ta đặt trạng thái đó là *q* và thêm số thứ tự của *a* vào *q* để làm cho nó trở thành trạng thái duy nhất. Sau đó, ta tạo một cây mới cho trạng thái mới tạo đó và thêm nó vào *F-Forest* và *I-Forest*. Tập chuyển trạng thái *δa* là rỗng nên trở thành một tập chỉ chứa một cặp của hai tập trạng thái là *pair(, )*. là rỗng. Cuối cùng, tập các nhãn là *{[q,a]}* ánh xạ trạng thái mới được tạo đó vào kí tự *a ∈ Σ* đang được phân tích. Cách thức tính toán các biểu diễn CNNFA của ôtômát đuôi cho biểu thức chính quy thành phần *J|K* với *J* và *K* là hai biểu thức chính quy thành phần không giao nhau được thể hiện trong biểu thức 4.4. Tập trạng thái là hợp của hai tập không giao nhau và . là hợp của bốn đại lượng: hai danh sách các cặp của hai tập hợp và , tập hợp bao gồm một cặp trạng thái là *pair(, )* và một tập hợp gồm một cặp trạng thái là *pair(, )*. là một danh sách mới của các cặp các tập hợp. Tương tự như , là hợp của hai tập hợp và . Đây cũng là danh sách mới của các cặp của các tập hợp. Tập các trạng thái bắt đầu là hợp của hai tập không giao nhau và . Đây là một cây mới trong *I-Forest* với gốc mới, cây con trái là và cây con phải là . Lá của hai cây con trái và phải này được nối với nhau bằng liên kết đôi. Tập các trạng thái kết thúc cũng là hợp của hai tập không giao nhau và . Đây cũng là một cây mới trong *F-Forest* với gốc mới, cây con trái là và cây con phải là . Lá của hai cây con trái và phải này cũng được nối vào nhau bằng liên kết đôi. Tập các nhãn là hợp của hai tập và . Công thức 4.5 cho ta cách tính toán biểu diễn CNNFA của ôtômát đuôi tương ứng với biểu thức chính quy thành phần *J.K*, trong đó *J* và *K* là hai biểu thức chính quy thành phần không giao nhau. Tập các trạng thái là hợp của hai tập trạng thái và . là hợp của ba thành phần: tập hợp gồm một cặp trạng thái *pair(, )*, *nullK*, và *nullJ*. Đây là một danh sách mới của các cặp của hai tập hợp. là hợp của ba thành phần: tập hợp gồm một cặp trạng thái *pair(, )*, và *.* Đây cũng là một danh sách các cặp của hai tập hợp. Tập các trạng thái bắt đầu là hợp của hai tập hợp: tập trạng thái bắt đầu , và *nullJ*. Tập trạng thái kết thúc là hợp của hai tập hợp: tập các trạng thái kết thúc và *nullK*. Tập các nhãn là hợp của hai tập các nhãn của hai thành phần: và . Cuối cùng, công thức 4.6 cho ta cách tính toán biểu diễn CNNFA của ôtômát đuôi cho biểu thức chính quy thành phần *J\**. Tập các trạng thái giống với tập trạng thái . Danh sách là rỗng. Danh sách là hợp của hai danh sách , và . Đây cũng là danh sách của các cặp của hai tập hợp. Tập trạng thái bắt đầu giống với tập trạng thái bắt đầu . Tập các trạng thái kết thúc giống với tập trạng thái kết thúc . Tập các nhãn giống với tập nhãn .

## 4.3 Phương pháp duyệt biểu thức chính quy

Phương pháp duyệt biểu thức chính quy sau [4] được sử dụng để duyệt một biểu thức chính quy thành các khối cơ bản và khối không cơ bản. Trong quá trình duyệt, thuật toán đồng thời xây dựng các biểu diễn CNNFA của các ôtômát đuôi cho mỗi khối cơ bản được tạo ra sử dụng các công thức từ 4.2 đến 4.6. Khi thuật toán dừng, nếu chỉ có một khối cơ bản còn lại thì biểu thức chính quy ban đầu là hợp lệ và chúng ta có biểu diễn CNNFA của ôtômát đuôi cuối cùng. Ngược lại thì biểu thức chính quy ban đầu là không hợp lệ.

**Bước 1**: Ở bước này, ta tạo một khối cơ bản mới và một khối không cơ bản mới mỗi khi không có gì rõ ràng để làm. Đầu tiên, ta duyệt kí tự ngoài cùng bên trái của R. Chúng ta duyệt từ trái sang phải cho đến khi gặp một kí tự trong bảng chữ cái hoặc một kí tự ngoặc đơn phải “)” mà khớp với kí tự ngoặc đơn trái tính “(“ từ vị trí bắt đầu duyệt. Nếu kí tự ngoặc đơn phải đó được tìm thấy, ta tiến hành bước 5. Nếu một kí tự thuộc bảng chữ cái được tìm thấy, giả sử là a, ta tạo một khối không cơ bản b để bao vùng vừa được duyệt (trừ a), tạo một khối cơ bản B để bao a. Biểu thức chính quy tương đương được gắn với B là a. Cả hai cờ *null* và *star* của B đều nhận giá trị *false*. Sau đó, ta thực hiện bước 2. Trong tình huống ta gặp kí tự kết thúc của R, nếu chỉ có một khối cơ bản bao phủ toàn bộ R thì quá trình duyệt kết thúc, ngược lại thì R không phải là một biểu thức chính quy hợp lệ.

Bảng 4.1: Các luật rút gọn được sử dụng ở bước 2.

|  |  |  |
| --- | --- | --- |
| **Trái** | **Phải** | **Hành động** |
| Không quan tâm | \* | Nếu cờ *star* của B là *false*, biểu thức chính quy tương đương của B trở thành (r)\*; ngược lại ta không làm gì cả. Cờ *star* của B trở thành *true*. |
| . | . | Thực hiện bước 3 |
| . | | | Thực hiện bước 3 |
| . | ) | Thực hiện bước 3 |
| | | | | Thực hiện bước 4 |
| | | ) | Thực hiện bước 4 |
| ( | ) | Không làm gì cả |
| | | . | Thực hiện bước 1 |
| Các tình huống khác | | Thực hiện bước 1 |

**Bước 2**: Mục tiêu của bước này là thực hiện việc rút gọn xung quanh của một khối cơ bản. Giả sử B là khối cơ bản đang được xem xét, r là biểu thức chính quy tương đương được gắn với B. Ta xem xét các phép toán bên trái và bên phải của B, và thực hiện càng nhiều phép rút gọn càng tốt như trong bảng 4.1.

**Bước 3**: Chúng ta duyệt R từ phải sang trái từ vị trí bên trái của B (ta coi các khối cơ bản như là một phần tử đơn lẻ) nhằm tìm được khối liên kết lớn nhất có dạng J1J2…JkB. Nếu gặp một dấu ngoặc phải, chúng ta chuyển vị trí đang duyệt (bỏ qua nội dung) sang vị trí dấu ngoặc đơn trái khớp với nó nhưng phải nằm cùng trong một khối không cơ bản. Trong quá trình duyệt sang trái, nếu một kí tự “|” hoặc một dấu ngoặc trái được tìm thấy, ta đã thấy tìm khối liên kết lớn nhất bao gồm B (và chứa B). Ta tạo một khối cơ bản mới B’ để bao khối liên kết này. Ta tạo một biểu thức chính quy tương đương cho B’ và xóa/thay đổi các khối mà có giao với khối B’ vừa tạo, cập nhật cờ *null* và *star* của B’.

**Bước 4**: Tương tự như bước 3, ta duyệt R từ phải qua trái từ vị trí bên phải cùng tính từ bên trái của B (coi các khối cơ bản như các phần tử đơn lẻ như trước) để tìm được các khối liên kết lớn nhất nối với nhau bởi dấu “|”. Tương tự như bước 3, ta duyệt R sang bên trái đến khi gặp một dấu ngoặc đơn trái mà khớp với dấu ngoặc đơn phải của B.

**Bước 5**: Trong bước này, ta rút gọn biểu thức chính quy con được bao bởi hai dấu ngoặc đơn. Phương pháp được dùng ở đây là giống với bước 4.

Với mọi biểu thức chính quy R với s lần xuất hiện của các kí tự trong bảng chữ cái, phương pháp duyệt này chuyển biểu thức R thành một biểu thức chính quy tương đương R’ dài O(s) trong thời gian O(|R|) và O(s) không gian phụ trợ [4].

## 4.4 Sinh mô hình cho thành phần phần mềm

Bây giờ ta đã có được biểu diễn CNNFA đầy đủ của ôtômát đuôi tương đương của biểu thức chính quy *R*, để có thể xây dựng được mô hình NFA của thành phần phần mềm đã cho, ta phải tính toán được tập hợp các chuyển trạng thái *δM* từ *lazyδM*. Đặt *V ⊆ Q*, ta cần tính được *δ(V,a)*, *∀a∈Σ*. Các công thức sau đây cho phép ta làm việc này:

|  |  |
| --- | --- |
| domain E = {x : ∃y | [x,y] ∈ E}. | (4.7) |
| E[S] = {y: ∃x ∈ S | [x,y] ∈E}.  E[S] được gọi là ảnh (image) của S. | (4.8) |

Đặt frontier(X) là tập các lá của một cây mà gốc là X. Ta tính δ(V, a) như sau:

|  |  |
| --- | --- |
| F\_domain(V) = {x ∈ domain lazyδ | V ∩ frontier(x) ≠ ∅}. | (4.9) |
| I\_image(V) = lazyδR[F\_domain(V)]. | (4.10) |

Cuối cùng, ta tính:

|  |  |
| --- | --- |
| δ R(V, Σ) = {z : ∃y ∈ I\_image(V) | z ∈ frontier(y)}. | (4.11) |
| Với mỗi a ∈ Σ, chúng ta tính: δR(V, a) = {q ∈ δ R(V, Σ) | A(q) = a}. | (4.12) |

Bây giờ ta đã có biểu diễn đầy đủ của ôtômát đuôi tương đương của *R*. Để sinh mô hình cho thành phần phần mềm đã cho, ta áp dụng công thức 4.1 để xây dựng mô hình NFA cho thành phần phần mềm từ đuôi của nó.

**Độ phức tạp tính toán:**

Biểu thức chính quy R có độ dài là r, và có s lần xuất hiện của các kí tự trong bảng chữ Σ, ta có thể tính toán CNNFA trong thời gian O(r) và không gian phụ trợ là O(s) [3].

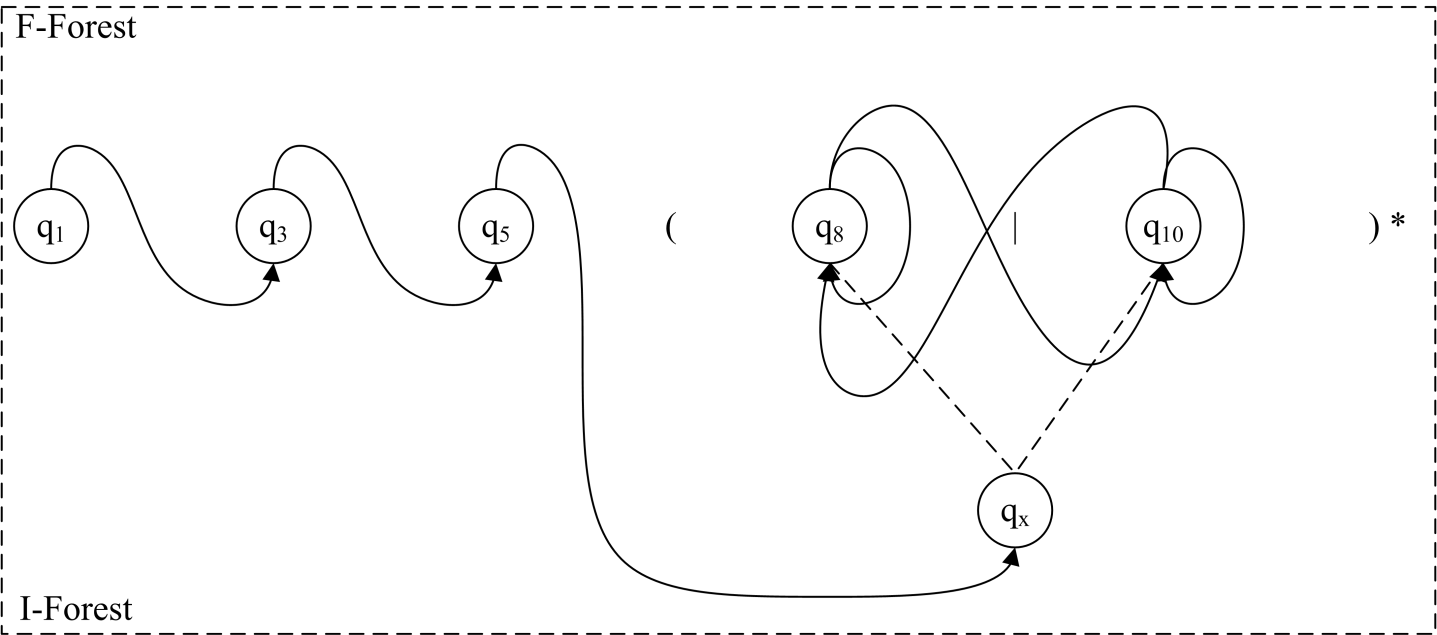
**Ví dụ 4.4**: Tính toán δ từ lazyδ. Hình 4.6 biểu diễn lazyδ của ôtômát đuôi tương đương của biểu thức chính quy abb(a|b)\*.

Ta có:

F\_Domain(F-Forest) = {q1, q3, q5, q8, q10}.

I\_Image(F-Forest) = {q3, q5, qx, q8, q10}.

Từ đó ta tính được = {(q1, b, q3), (q3, b, q5), (q5, b, q8), (q5, b, q10), (q8, b, q8), (q10, b, q10), (q10, b, q8), (q8, b, q10)}.



Hình 4.6: Tính toán δ từ lazyδ.

## 4.5 Tối ưu hóa mô hình

Ôtômát cuối cùng nhận được theo phương pháp CNNFA chỉ đơn định trong trường hợp mỗi kí tự trong bảng chữ cái chỉ xuất hiện đúng một lần trong biểu thức chính quy. Tuy nhiên, trong thực tế, điều này hiếm khi xảy ra. Do đó, ta cần thực hiện các công việc tối ưu hóa mô hình thu được để có thể có mô hình tốt nhất đặc tả cho thành phần phần mềm đã cho. Có hai việc cần làm là đơn định hóa NFA thành DFA và tối thiểu hóa DFA thu được.

**Thuật toán 4.1**: Đơn định hóa ôtômát NFA [6].

Với mỗi ôtômát không đơn định M = (Q, , , q0, F), chúng ta luôn xây dựng được một ôtômát đơn định M' = (Q’, , ’, q0’, F’) tương đương với M theo thuật toán sau:

* Q’ ⊆ 2Q,
* q0’ = {q0},
* F’ là tập tất cả các tập con của Q có chứa phần tử của F

F’ = { S Q | S F }, và

* ’ được xác định như sau: ’({q1,q2,.., qk}, a) = .

Thuật toán 4.1 nhận đầu vào và đầu ra như sau:

*Đầu vào*: M = (Q, Σ, , q0, F).

*Đầu ra*: M’ = (Q’, Σ, ’, q0’, F’) là ôtômát đơn định tương đương.

|  |
| --- |
| 1: Khởi tạo: q0’ = {q0}, Q’ = {q0’}.  2: **repeat**  3: lấy ’ Q’.  4: **if** q’ không được đánh dấu **then**  5: **for each** a ∈ Σ **do**  6: Tìm ’ = ’(’, a) = .  7: **if** p’ ∉ Q’ **then**  8: Thêm ’ vào Q’.  9: **end if**  10: **end for**  11: **end if**  12: đánh dấu q’.  13:**until** không còn phần tử mới được thêm vào Q’.  14:F' = { ’ Q’ | ’ F }.  15:return M’. |

Chi tiết của thuật toán 4.1 như sau:

Ban đầu, thuật toán thiết lập q0’ là {q0}, Q’ là {q0’} (dòng 1). Đặt q’ là một phần tử của Q’ (dòng 3). Nếu nó chưa được đánh dấu, thuật toán tìm những trạng thái mà được chuyển từ trạng thái q’ bằng các kí tự của bảng chữ cái Σ và thêm những trạng thái này vào Q’ nếu chúng chưa ở trong Q’ (từ dòng 4 đến dòng 11). Sau đó, thuật toán đánh dấu q’ (dòng 12). Quá trình từ dòng 3 đến dòng 12 được lặp lại cho đến khi không còn phần tử mới nào được thêm vào Q’. Khi đó, tập các trạng thái chấp nhận F’ được tính toán từ các phần tử của Q’ mà có chứa trạng thái chấp nhận của NFA ban đầu (dòng 14).

**Thuật toán 4.2**: Tối thiểu hóa ôtômát đơn định thu được [6].

Chúng ta sử dụng thuật toán Hopcroft để tối thiểu hóa DFA nhận được sau thuật toán 4.1 ở trên.

*Đầu vào*: ôtômát đơn định M = (Q, , , q0, F).

*Đầu ra*: ôtômát tối tiểu M’ = (Q’, , ’, q0’, F’) tương đương với M.

Ta nói rằng xâu x ∈ Σ\* phân biệt hai trạng thái p, q ∈ Q nếu hoặc δ(p,x) ∈ F và δ(q,x) ∉ F hoặc ngược lại. Trong trường hợp này, trạng thái p và q được gọi là có thể phân biệt được; ngược lại chúng được gọi là không thể phân biệt được hay tương đương. Để có thể lấy được M’, chúng ta tìm tất cả các lớp tương đương của các trạng thái không phân biệt được và đưa tất cả chúng vào một trạng thái của M’. Đặt [p] là một lớp tương đương. Để xác định phép chuyển trạng thái, chúng ta lấy bất kì một trạng thái p nào đó trong [p] và định nghĩa δ’([p], a) = [q], ở đó a là một kí tự thuộc bảng chữ cái của M và [q] là lớp tương đương chứa δ(p,a).

|  |
| --- |
| 1: Ban đầu, đánh dấu (p,q) ở đó p ∈ F và q ∉ F.  2: **repeat**  3: **for** mỗi cặp (p,q) chưa đánh dấu **do**  4: **for** mỗi kí tự a ∈ Σ **do**  5: **if** cặp δ(p,a), δ(q,a) được đánh dấu **then**  6: Đánh dấu cặp (p,q).  7: **end if**  8: **end for**  9: **end for**  10: **until** không có cặp mới nào được đánh dấu.  11: **for** mỗi q ∈ Q **do**  12: Tạo lớp tương đương [q] chứa các trạng thái không phân biệt được với q.  13: Thêm [q] vào Q’.  14: **if** [q] chứa trạng thái kết thúc của M **then**  15: Thêm [q] vào F’.  16: **end if**  17: **end for**  18: Đặt trạng thái khởi tạo q0’ = [q0].  19: **for** mỗi luật (q1, a, q2) **do**  20: **if** ([q1], a, [q2]) không ở trong δ’ **then**  21: Thêm ([q1], a, [q2]) vào δ’.  22: **end if**  23: **end for**  24: **return** M’. |

Chi tiết của thuật toán 4.2 như sau:

Ban đầu, thuật toán đánh dấu tất cả các cặp trạng thái có thể phân biệt được (dòng 1). Rõ ràng rằng nếu p ∈ F và q ∉ F thì p, q là phân biệt được nên thuật toán đánh dấu cặp p, q là phân biệt được. Với mỗi cặp p, q mà chưa được đánh dấu (dòng 3), nếu ta tìm thấy một kí tự a mà δ(p,a) và δ(q,a) là phân biệt được, ta cũng đánh dấu p, q là phân biệt được (dòng 4 đến dòng 8). Quá trình này được lặp lại cho đến khi không có cặp trạng thái nào được đánh dấu nữa. Sau đó, ta xây dựng DFA tối thiểu như sau. Tập trạng thái bao gồm tất cả các lớp tương đương được xác định ở phía trên (dòng 12 & 13). Tập các trạng thái chấp nhận là những lớp tương đương mà có chứa trạng thái chấp nhận của M (dòng 14 đến dòng 16). Trạng thái ban đầu là [q0] (dòng 18). Các luật chuyển trạng thái, là chuyển trạng thái giữa các lớp tương đương, được định nghĩa như ở phía trên (dòng 19 đến 23). DFA được sinh ra M’ là một ôtômát đơn định tối thiểu.

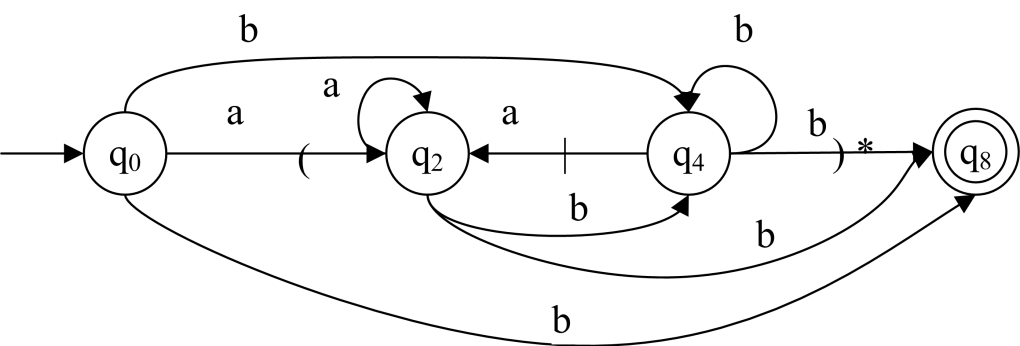
**Ví dụ 4.5**: Cho ôtômát tương đương của biểu thức chính quy (a|b)\*b như trong hình 4.7 sau:

1. *Đơn định hóa ôtômát M* (theo thuật toán 4.1).

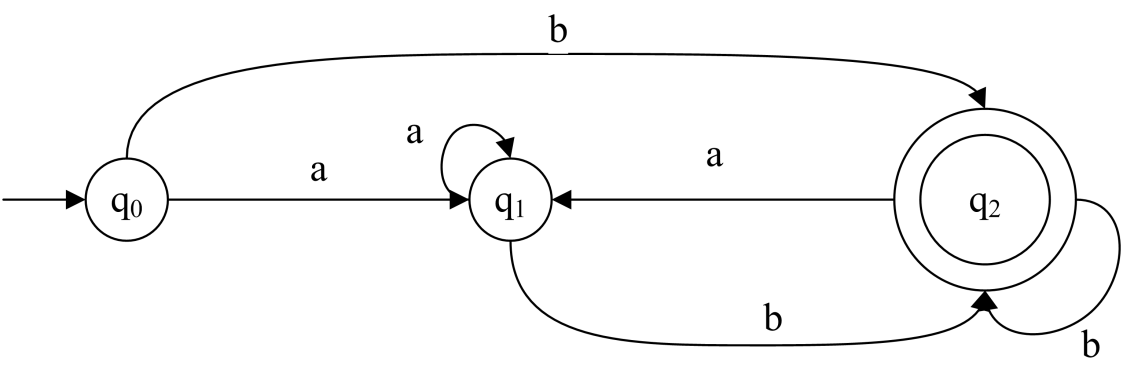
Xây dựng ôtômát M’ = (Q’, Σ, δ’, q0’, F’) đơn định, tương đương với ôtômát trên hình 4.7.

Chúng ta xây dựng các quy tắc chuyển trạng thái của M’ như sau:

Bắt đầu từ trạng thái khởi tạo {q0}, kí hiệu [q0]. Ta thấy từ [q0] có thể đến được [q2] qua cung có nhãn là a và có thể đến được [q4, q8] qua các cung có nhãn là b. Tập [q2], [q4, q8] được kết nạp vào tập trạng thái Q’ của ôtômát M’. Từ trạng thái [q2] có thể đến được [q2] với nhãn là a và đến được [q4, q8] qua các cung có nhãn là b. Từ tập trạng thái [q4, q8] ta có thể đến được [q2] qua các cung có nhãn là a và có thể đến được [q4, q8] qua các cung có nhãn là b. Kết thúc quá trình đơn định hóa M, ta có ôtômát M’ với tập trạng thái là {[q0], [q2], [q4, q8]}, tập trạng thái kết thúc là F’ = {[q4, q8]}. Ta có được đồ thị chuyển như trên hình 4.8.



Hình 4.7: Ôtômát M tương đương của biểu thức chính quy (a|b)\*b.

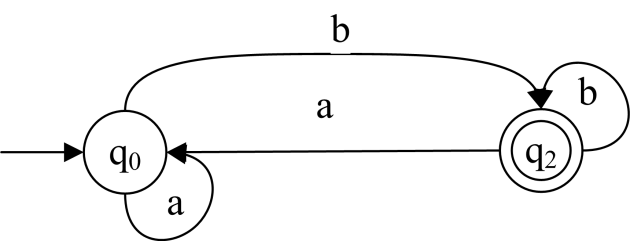


Hình 4.8: Ôtômát ở hình 4.7 sau khi đơn định hóa.

1. *Tối thiểu hóa* (Theo thuật toán 4.2).

Xây dựng ôtômát tối thiểu tương đương với ôtômát có đồ thị chuyển trạng thái như trên hình 4.8.

Theo thuật toán 4.2, đầu tiên, ta đánh dấu hai cặp trạng thái (q0, q2) và (q1, q2). Tiếp theo, ta thấy rằng chỉ còn một cặp trạng thái (q0, q1) là chưa được đánh dấu. Ta dễ dàng kiểm tra rằng không thể đánh dấu cặp trạng thái này với cả hai kí tự a và b của bảng chữ cái. Tiếp đến, ta xây dựng lớp tương đương [q0] của q0. Lớp này bao gồm {q0, q1}. Tương tự, ta có lớp tương đương [q2] bao gồm chỉ một trạng thái {q2}. Cuối cùng, ta có đồ thị chuyển trạng thái như hình 4.9 sau đây.



Hình 4.9: Ôtômát đơn định tối thiểu tương đương của biểu thức chính quy (a|b)\*b.

## 4.6 Ví dụ sinh mô hình cho thành phần phần mềm bằng thuật toán CNNFA

### 4.6.1 Xây dựng NFA bằng thuật toán CNNFA

Phần 4.6.1 này sẽ trình bày quá trình sinh mô hình cho thành phần phần mềm cụ thể bằng phương pháp CNNFA. Giả sử thành phần phần mềm được đặc tả bằng một biểu thức chính quy các hành động của nó như sau:

(engineOn)|(engineOn.engineOff)|(engineOn.engineOff.off)|(engineOn.engineOff.on)

Thuật toán CNNFA sẽ tiến hành duyệt biểu thức chính quy từ trái qua phải để phân tích các thành phần của biểu thức chính quy đã cho và đồng thời áp dụng các công thức từ 4.2 đến 4.6 để xây dựng các biểu diễn CNNFA cho các ôtômát đuôi tương ứng với các biểu thức chính quy thành phần. Trong phần mô tả dưới đây của ví dụ này, số thứ tự các kí tự trong biểu thức chính quy được tính từ 1.

Sau kí tự số 1 là “(“, kí tự trong bảng chữ cái đầu tiên mà thuật toán tìm được là R1 = “engineOn” ở vị trí số 2. Thuật toán sẽ xây dựng một biểu diễn CNNFA cho ôtômát đuôi = 2 tương ứng như sau:

* 2={q2},
* lazynredengineOn2 = pair(2, 2) = {(F1,I1)},
* lazyδengineOn2 = ∅),
* 2 = {q2} = {I1},
* 2 ={q2} = {F1},
* 2 ={[q2,engineOn]}, và
* nullengineOn2 = ∅.

I1 và F1 là một cây của rừng I\_Forest và F\_Forest tương ứng và đều là nút lá chứa trạng thái q2. Số 2 trong kí hiệu trạng thái được dùng trùng với số thứ tự trong biểu thức chính quy để phân biệt các trạng thái với nhau.

Qua kí tự “)” tại vị trí số 3 tiếp theo, thuật toán tìm được đến kí tự “|” ở vị trí số 4, nó sẽ duyệt ngược lại bên trái, nhưng biểu thức chính quy thành phần tối đa có thể tìm được vẫn chỉ là “engineOn” đã tính ở bước trước. Thuật toán tiếp tục duyệt biểu thức chính quy từ vị trí số 5 là kí tự “(“. Nó lại bắt gặp một kí tự “engineOn” trong bảng chữ cái ở vị trí số 6. Thuật toán lại xây dựng một biểu diễn CNNFA cho ôtômát đuôi tương ứng cho “engineOn” là 6 như sau:

* 6 = {q6},
* lazynredengineOn6 = pair(6, 6) = {(F2,I2)},
* lazyδengineOn6 = ∅,
* 6 = {q6} = {I2},
* 6 = {q6} = {F2},
* 6 ={[q6,engineOn]}, và
* nullengineOn6 = ∅.

I2 và F2 là một cây mới của rừng I\_Forest và F\_Forest tương ứng và đều là nút lá chứa trạng thái q6.

Tiếp theo, qua kí tự số 7 là “.”, thuật toán duyệt đến kí tự “engineOff” ở vị trí số 8 và xây dựng một biểu diễn CNNFA cho ôtômát đuôi tương ứng với “engineOff” là 8 như sau:

* 8 = {q8},
* lazynredengineOff8 = pair(8, 8) = {(F3,I3)},
* lazyδengineOff8 = ∅,
* 8 = {q8} = {I3},
* 8 = {q8} = {F3},
* 8 ={[q8,engineOn]}, và
* nullengineOff8 = ∅.

I3 và F3 là một cây mới của rừng I\_Forest và F\_Forest tương ứng và đều là nút lá chứa trạng thái q8.

Tiếp theo, thuật toán duyệt đến kí tự số 9 là “)” và kiểm tra kí tự bên trái của 8 vừa được tạo là “.” nên nó duyệt ngược từ phải qua trái và tìm thấy 6 ở vị trí số 6, thuật toán thực hiện phép ghép nối hai biểu diễn CNNFA này lại thành một biểu diễn CNNFA cho ôtômát đuôi tương ứng cho R2 = engineOn.engineOff theo công thức 4.5 như sau:

= 6\_8, trong đó:

* 6\_8 = 6 ∪ 8 = {q6, q8},
* lazynredengineOn.engineOff6\_8 = pair(8, 6) ∪ nullengineOff8lazynredengineOn6 ∪ nullengineOn6lazynredengineOff8 = {(F3, I2)},
* 6\_8 = pair(6, 8) ∪ 6 ∪ 8= {(F2,I3)},
* 6\_8 = 6 ∪ nullengineOn68 = {q6} = {I2},
* 6\_8 = 8 ∪ nullengineOff86 = {q8} = {F3},
* 6\_8 = 6 ∪ 8 = {(q6, engineOn), (q8, engineOff)}, và
* nullengineOn.engineOff6\_8 = nullengineOn6nullengineOff8 = ∅.

Tiếp theo, thuật toán duyệt đến kí tự số 10 là “|”, số 11 là “(“, “engineOn” ở số 12, “.” ở số 13, “engineOff” ở số 14. Thuật toán xây dựng các biểu diễn CNNFA cho các ôtômát đuôi tương ứng cho “engineOn”, “engineOff” là 12 và 14 như sau:

* 12 = {q12},
* lazynredengineOn12 = pair(12, 12) = {(F4,I4)},
* lazyδengineOn12 = ∅,
* 12 = {q12} = {I4},
* 12 ={q12} = {F4},
* 12 ={[q12,engineOn]}, và
* nullengineOn12 = ∅.

I4 và F4 là một cây mới của rừng I\_Forest và F\_Forest tương ứng và đều là nút lá chứa trạng thái q12.

* 14 = {q14},
* lazynredengineOff14 = pair(14, 14) = {(F5,I5)},
* lazyδengineOff14 = ∅,
* 14 = {q14} = {I5},
* 14 ={q14} = {F5},
* 14 ={[q14,engineOn]}, và
* nullengineOff14 = ∅.

I5 và F5 là một cây mới của rừng I\_Forest và F\_Forest tương ứng và đều là nút lá chứa trạng thái q14.

Tiếp theo, thuật toán duyệt đến kí tự số 15 là “.”, nó duyệt ngược trở lại và tìm thấy 12 và tính toán biểu diễn CNNFA cho ôtômát đuôi 12\_14 tương ứng với engineOn.engineOff như sau:

* 12\_14 = 12 ∪ 14 = {q12, q14},
* lazynredengineOn.engineOff12\_14 = pair(14, 12) ∪ nullengineOff14lazynredengineOn12 ∪ nullengineOn12lazynredengineOff14 = {(F5, I4)},
* 12\_14 = pair(12, 14) ∪ 12 ∪ 14 = {(F4,I5)},
* 12\_14 = 12 ∪ nullengineOn1214 = {q12} = {I4},
* 12\_14 = 14 ∪ nullengineOff1412 = {q14} = {F5},
* 12\_14 = 12 ∪ 14 = {(q12, engineOn), (q14, engineOff)}, và
* nullengineOn.engineOff12\_14 = nullengineOn12nullengineOff14 = ∅.

Tiếp theo, thuật toán duyệt đến kí tự số 16 là *off*, nó xây dựng một biểu diễn CNNFA cho ôtômát đuôi 16tương ứng với nó như sau:

* 16 = {q16},
* lazynredoff16 = pair(16, 16) = {(F6,I6)},
* lazyδoff16 = ∅,
* 16 = {q16} = {I6},
* 16 ={q16} = {F6},
* 16 ={[q16,off]}, và
* nulloff16 = ∅.

I6 và F6 là một cây mới của rừng I\_Forest và F\_Forest tương ứng và đều là nút lá chứa trạng thái q16.

Tiếp theo, thuật toán duyệt đến kí tự 17 là “)”, nó duyệt ngược trở lại và gặp 12\_14, khi đó, nó tính toán một biểu diễn CNNFA cho ôtômát đuôi tương ứng cho R3 = *engineOn*.*engineOff*.*off* là = 12\_16 như sau:

* 12\_16 = 12\_14 ∪ 16 = {q12, q14, q16},
* lazynredengineOn.engineOff.off12\_16 = pair(16, 12\_14) ∪ nulloff16lazynredengineOn.engineOff12\_14 ∪ nullengineOn.engineOff12\_14lazynredoff16 = {(F6, I4)},
* 12\_16 = pair(12\_14, 16) ∪ 12\_14 ∪ 16 = {(F5, I6), (F4, I5)},
* 12\_16 = 12\_14 ∪ nullengineOn.engineOff12\_1416 = {q12} = {I4},
* 12\_16 = 16 ∪ nulloff1612\_14 = {q16} = {F6},
* 12\_16 = 12\_14 ∪ 16 = {(q12, engineOn), (q14, engineOff), (q16, off)}, và
* nullengineOn.engineOff.off12\_16 = nullengineOn.engineOff12\_14nulloff16 = ∅.

Thuật toán tiếp tục duyệt qua kí tự số 18 là “|”, 19 là “(“, 20 là “engineOn”, 21 là “.” Và 22 là “engineOff” để xây dựng được hai biểu diễn CNNFA tương ứng cho ôtômát đuôi là 20và 22 tương ứng như sau:

* 20 = {q20},
* lazynredengineOn20 = pair(20, 20) = {(F7,I7)},
* lazyδengineOn20 = ∅,
* 20 = {q20} = {I7},
* 20 ={q20} = {F7},
* 20 ={[q20,engineOn]}, và
* nullengineOn20 = ∅.

I7 và F7 là một cây mới của rừng I\_Forest và F\_Forest tương ứng và đều là nút lá chứa trạng thái q20.

* 22 = {q22} = {I8},
* lazynredengineOff22 = pair(22, 22) = {(F8,I8)},
* lazyδengineOff22 = ∅,
* 22 = {q22} = {F8},
* 22 ={q22},
* 22 ={[q22,engineOn]}, và
* nullengineOff22 = ∅.

I8 và F8 là một cây mới của rừng I\_Forest và F\_Forest tương ứng và đều là nút lá chứa trạng thái q22.

Thuật toán duyệt đến kí tự số 23 là “.” và duyệt ngược trở lại để xây dựng được biểu diễn CNNFA tương ứng cho ôtômát đuôi cho *engineOn*.*engineOff* là 20\_22 như sau:

* 20\_22 = 20 ∪ 22 = {q20, q22},
* lazynredengineOn.engineOff20\_22 = pair(22, 20) ∪ nullengineOff22lazynredengineOn20 ∪ nullengineOn20lazynredengineOff22 = {(F8, I7)},
* 20\_22 = pair(20, 22) ∪ 20 ∪ 22 = {(F7,I8)},
* 20\_22 = 20 ∪ nullengineOn2022 = {q20} = {I7},
* 20\_22 = 22 ∪ nullengineOff2220 = {q22} = {I8},
* 20\_22 = 20 ∪ 22 = {(q20, engineOn), (q22, engineOff)}, và
* nullengineOn.engineOff20\_22 = nullengineOn20nullengineOff22 = ∅.

Tiếp theo, thuật toán duyệt đến kí tự số 24 là “on”, nó xây dựng một biểu diễn CNNFA tương ứng cho ôtômát đuôi tương ứng với nó là 24như sau:

* 24 = {q24},
* lazynredon24 = pair(24, 24) = {(F9,I9)},
* lazyδon24 = ∅,
* 24 = {q24} = {I9},
* 24 ={q24} = {F9},
* 24 ={[q24,on]}, và
* nullon16 = ∅.

I9 và F9 là một cây mới của rừng I\_Forest và F\_Forest tương ứng và đều là nút lá chứa trạng thái q24.

Tiếp theo, thuật toán duyệt đến kí tự số 25 là “)” và nó duyệt ngược lại và xây dựng được biểu diễn CNNFA tương ứng cho ôtômát đuôi tương ứng với R4 = *engineOn*.*engineOff*.*on* là = 20\_24 như sau:

* 20\_24 = 20\_22 ∪ 24 = {q20, q22, q24},
* lazynredengineOn.engineOff.on20\_24 = pair(24, 20\_22) ∪ nullon24lazynredengineOn.engineOff20\_22 ∪ nullengineOn.engineOff20\_22lazynredon24 = {(F9, I7)},
* 20\_24 = pair(20\_22, 24) ∪ 20\_22 ∪ 24 = {(F7, I8), (F8, I9)},
* 20\_24 = 20\_22 ∪ nullengineOn.engineOff20\_2224 = {q20} = {I7},
* 20\_24 = 24 ∪ nullon2420\_22 = {q24} = {F9},
* 20\_24 = 20\_22 ∪ 24 = {(q20, engineOn), (q22, engineOff), (q24, on)}, và
* nullengineOn.engineOff.on20\_24 = nullengineOn.engineOff20\_24nullon24 = ∅.

Đến đây thì đã duyệt hết biểu thức chính quy, ta duyệt ngược lại từ phải qua trái để xử lý các biểu thức chính quy con thành phần dạng R = R1.R2…Rn hoặc R1|R2|…|Rn. Đầu tiên, ta xây dựng được biểu diễn CNNFA cho ôtômát đuôi tương ứng với R3|R4 = (*engineOn*.*engineOff*.*off*)|(*engineOn*.*engineOff*.*on*) là như sau:

* = ∪ = {q12, q14, q16, q20, q22, q24},
* lazynredR3|R4 = {(F6,I4), (F9,I7), (F6,I7), (F9, I4)},
* lazyδR3|R4 = {(F4,I5), (F5,I6), (F7, I8), (F8,I9)},
* = ∪ = {I10},
* = ∪ = {F10},
* = ∪ = {(q12, engineOn), (q14, engineOff), (q16, off), (q20, engineOn), (q22, engineOff), (q24, on)}, và
* nullR3|R4 = nullR3 ∪ nullR4 = ∅.

Trong đó: I10 là một nút trong I\_Forest có hai con là: con trái = {I4}, con phải = {I7}. F10 là một nút trong F\_Forest có hai con là: con trái = {F6}, con phải = {F9}.

Tiếp đến, thuật toán sẽ tính toán biểu diễn CNNFA tương ứng cho ôtômát đuôi tương ứng với R2|R3|R4 là như sau:

* = ∪ = {q6, q8, q12, q14, q16, q20, q22, q24},
* lazynredR2|R3|R4 = {(F3, I2), (F6,I4), (F9,I7), (F6,I7), (F9, I4), (F3, I10), (F10, I2)},
* lazyδR2|R3|R4 = {(F4,I5), (F5,I6), (F7, I8), (F8,I9)},
* = ∪ = {I11},
* = ∪ = {F11},
* = ∪ = {(q6,engineOn), (q8, engineOff), (q12, engineOn), (q14, engineOff), (q16, off), (q20, engineOn), (q22, engineOff), (q24, on)}, và
* nullR2|R3|R4 = nullR2 ∪ nullR3|R4 = ∅.

Trong đó: I11 là một nút trong I\_Forest có hai con là: con trái = {I2}, con phải = {I10}. F10 là một nút trong F\_Forest có hai con là: con trái = {F3}, con phải = {F10}.

Tiếp đến, thuật toán tính toán biểu diễn CNNFA tương ứng cho ôtômát đuôi tương ứng với R = R1|R2|R3|R4 là = như sau:

* = ∪ = {q2, q6, q8, q12, q14, q16, q20, q22, q24},
* lazynredR1|R2|R3|R4 = {(F1, I1), (F3, I2), (F6,I4), (F9,I7), (F6,I7), (F9, I4), (F3, I10), (F10, I2), (F1, I11), (F11, I1)},
* lazyδR1|R2|R3|R4 = {(F2, I3), (F4,I5), (F5,I6), (F7, I8), (F8,I9)},
* = ∪ = {I12},
* = ∪ = {F12},
* = ∪ = {(q2, engineOn), (q6,engineOn), (q8, engineOff), (q12, engineOn), (q14, engineOff), (q16, off), (q20, engineOn), (q22, engineOff), (q24, on)}, và
* nullR1|R2|R3|R4 = nullR1 ∪ nullR2|R3|R4 = ∅.

Trong đó: I12 là một nút trong I\_Forest có hai con là: con trái = {I1}, con phải = {I11}. F12 là một nút trong F\_Forest có hai con là: con trái = {F1}, con phải = {F11}.

Ôtômát đuôi = có biểu diễn CNNFA (dạng nén) như trong hình 4.10.

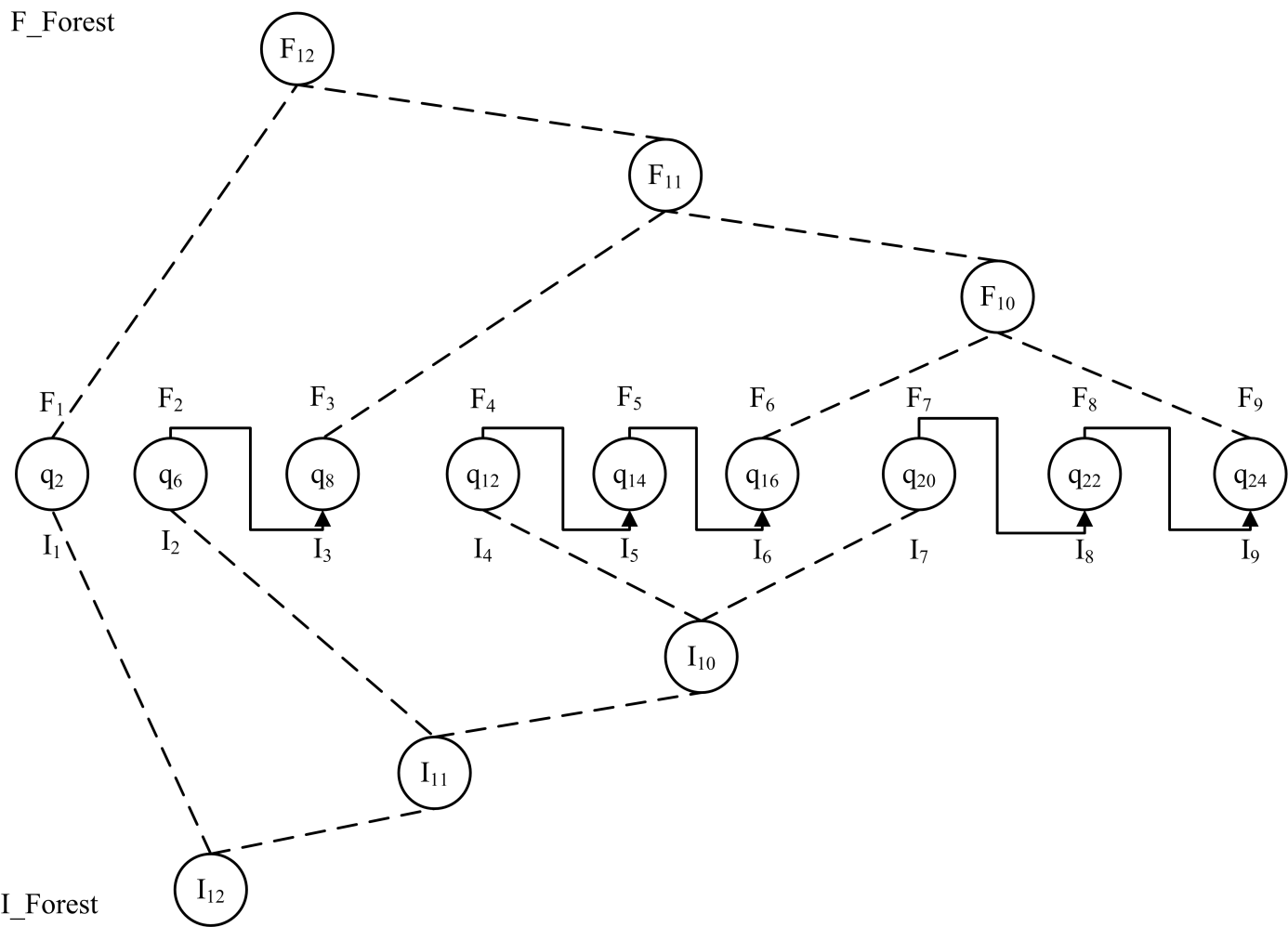
Bây giờ, ta sẽ tính toán δ từ lazyδ để thu được ôtômát đuôi cuối cùng. Ta có:

F\_Domain(F\_Forest) = {q6, q12, q14, q20, q22}.

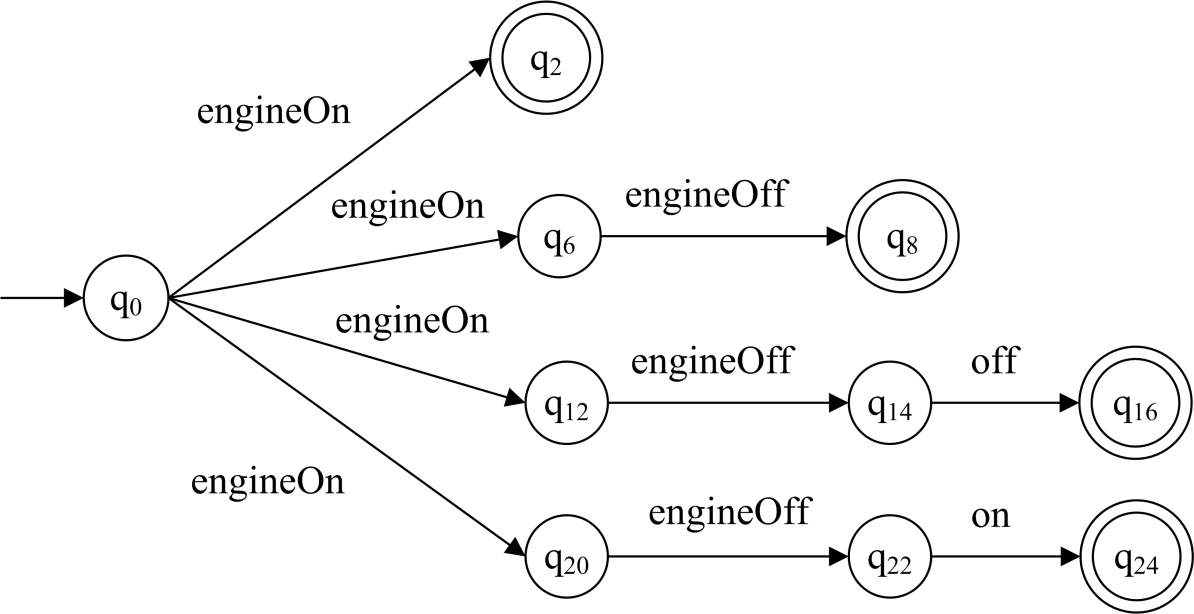
I\_Image(F\_Forest) = {q8, q14, q16, q22, q24}.

Từ đó ta tính được = {(q6, engineOff, q8), (q12, engineOff, q14), (q14, off, q16), (q20, engineOff, q22), (q22, on, q24)}.

Từ đó, ta có được ôtômát mô hình cuối cùng cần xây dựng như hình 4.11.



Hình 4.10: Ôtômát đuôi dạng nén theo thuật toán CNNFA.



Hình 4.11: Ôtômát không đơn định theo thuật toán CNNFA.

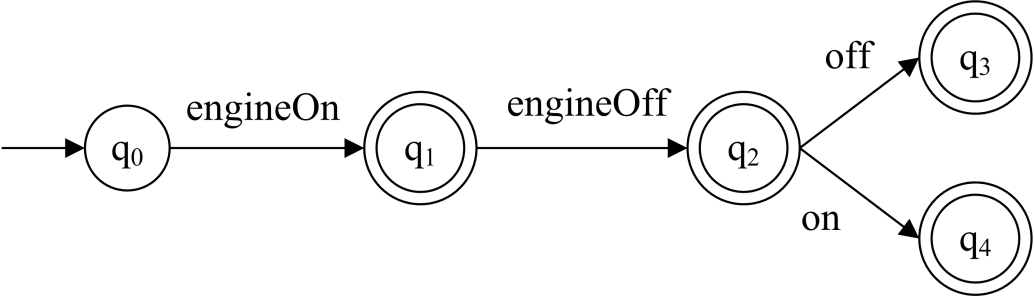
### 4.6.2 Đơn định hóa NFA

Xây dựng ôtômát M’ = (Q’, Σ, δ’, q0’, F’) đơn định, tương đương với ôtômát trên hình 4.11. Chúng ta xây dựng các quy tắc chuyển trạng thái của M’ như sau:

Bắt đầu từ trạng thái khởi tạo {q0}, kí hiệu [q0]. Ta thấy từ [q0] có thể đến được [q2, q6, q12,q20] qua cung có nhãn là *engineOn*. Tập [q2, q6, q12,q20] được kết nạp vào tập trạng thái Q’ của ôtômát M’. Từ trạng thái [q2, q6, q12,q20] có thể đến được [q8, q14, q22] với nhãn là *engineOff*. Từ tập trạng thái [q8, q14, q22] ta có thể đến được [q16] qua các cung có nhãn là *off* và có thể đến được [q24] qua các cung có nhãn là *on*. Kết thúc quá trình đơn định hóa M, ta có ôtômát M’ với tập trạng thái là {[q0], [q2, q6, q12,q20], [q8, q14, q22], [q16], [q24]}, tập trạng thái kết thúc là F’ = {[q2, q6, q12,q20], [q8, q14, q22], [q16], [q24]}. Ta có được đồ thị chuyển như hình 4.12.

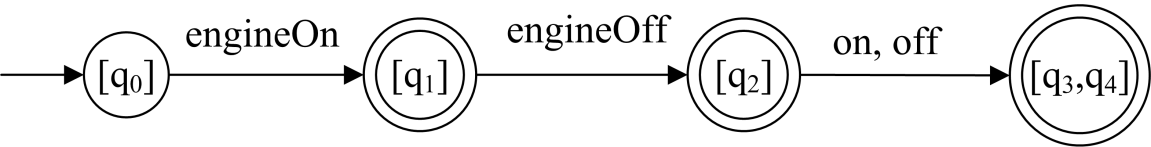
### 4.6.3 Tối thiểu hóa DFA

Xây dựng ôtômát tối thiểu tương đương với ôtômát có đồ thị chuyển trạng thái như trên hình 4.12.



Hình 4.12: Ôtômát sau khi đã đơn định hóa.

Theo thuật toán 4.2, đầu tiên, ta đánh dấu hai cặp trạng thái (q0, q1), (q0, q2), (q0, q3), (q0, q4). Tiếp theo, ta thấy rằng với kí tự *engineOn*, ta có δ(q0, engineOn) = q1, δ(q1, engineOn), δ(q2, engineOn), δ(q3, engineOn), δ(q4, engineOn) không tồn tại nên ta đánh dấu được cặp (q0, q1), (q0, q2), (q0, q3), (q0, q4) (các cặp này đã được đánh dấu). Với kí tự *engineOff*, ta có δ(q1, engineOff) = q2, δ(q2, engineOff), δ(q3, engineOff), δ(q4, engineOff) không xác định nên ta đánh dấu được (q1, q2), (q1, q3), (q1, q4),. Với kí tự *off*, ta có δ(q2, off) = q3, δ(q3, off), δ(q4, off) không xác định nên ta đánh dấu được (q2, q3), (q2, q4). Với kí tự *on*, ta có δ(q2, on) = q4, δ(q4, on), δ(q3, on) không xác định nên ta đánh dấu được (q2, q4), (q2, q3). Chỉ duy nhất cặp (q3, q4) không được đánh dấu. Tiếp đến, ta xây dựng lớp tương đương [q0] của q0. Lớp này bao gồm {q0}. Tương tự, ta có lớp tương đương [q1] bao gồm chỉ một trạng thái {q1}. Lớp tương đương [q2] chỉ gồm một trạng thái là {q2}. Lớp tương đương [q3] bao gồm hai trạng thái {q3, q4}. Cuối cùng, ta có đồ thị chuyển trạng thái như hình 4.13 sau đây.



Hình 4.13: Ôtômát đơn định, tối thiểu cuối cùng.

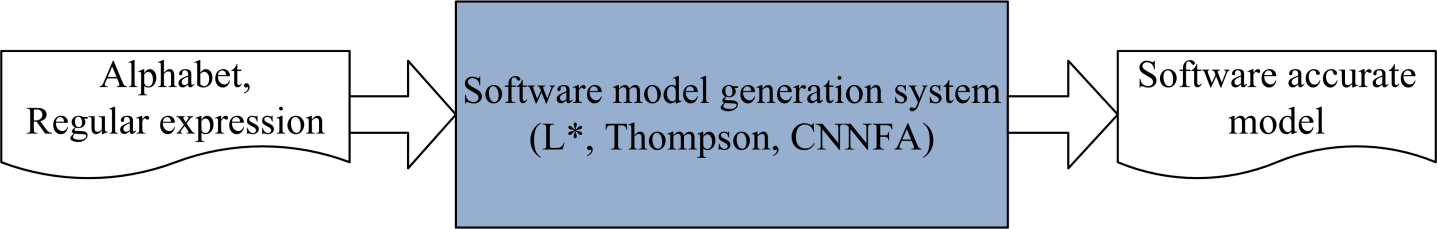
# Chương 5: Thực nghiệm

## 5.1 Công cụ sinh mô hình tự động cho phần mềm dựa trên thành phần

Chương 3 đã đề cập đến hai phương pháp sinh mô hình thành phần phần mềm tự động. Phương pháp thứ nhất sử dụng thuật toán học L\* để học một biểu thức chính quy và xây dựng được mô hình thành phần phần mềm đã cho. Trong phương pháp này, việc học mô hình ứng viên được thực hiện thông qua tương tác với đối tượng Teacher để kiểm tra sự tương thích của mô hình ứng viên với thành phần phần mềm. Một khó khăn của việc sinh mô hình dựa trên thuật toán học L\* là việc thực thi đối tượng Teacher thông qua thuật toán Vasilevskii – Chow có độ phức tạp rất lớn làm ảnh hưởng đáng kể đến hiệu năng của phương pháp. Phương pháp thứ hai là sử dụng thuật toán Thompson để xây dựng mô hình thành phần phần mềm từ biểu thức chính quy biểu diễn ngôn ngữ L gồm tập các dẫn xuất thực hiện được trên thành phần phần mềm đó.

Chương 4 đã đề xuất sử dụng một phương pháp khác cho việc sinh mô hình cho thành phần phần mềm từ biểu thức chính quy đặc tả của nó bằng việc sử dụng thuật toán CNNFA. Phương pháp này có độ phức tạp tính toán về thời gian và không gian là vượt trội so với hai phương pháp nói trên về mặt lý thuyết, tuy nhiên, khi thực hiện triển khai khi sử dụng C# thì cấu trúc dữ liệu của CNNFA làm cho phương pháp này sử dụng nhiều bộ nhớ hơn phương pháp Thompson đối với các mô hình lớn.

Công cụ sinh mô hình thành phần phần mềm sử dụng cả ba thuật toán L\*, Thompson và CNNFA được gọi là Software model generation system (gọi tắt là MG) nhằm mục đích so sánh hiệu năng của cả ba phương pháp với nhau cả về mặt thời gian thực hiện và bộ nhớ thực tế được cấp phát cho thuật toán. Công cụ được phát triển bằng Visual Studio 2010, C#.NET, chạy trên nền hệ điều hành Windows 7. Kiến trúc của công cụ được mô tả trong hình 5.1 sau đây:



Hình 5.1: Kiến trúc công cụ sinh mô hình thành phần phần mềm tự động.

Trên mỗi màn hình của một phương pháp sinh mô hình, có ba đầu vào và đầu ra cho mỗi phương pháp: file chứa biểu thức chính quy của thành phần phần mềm (Regular expression file), file chứa mô hình kết quả của thành phần phần mềm (Output file) và file chứa các thông số về hiệu năng của phương pháp (Benchmark file). Để chọn file khác cho mỗi file này, ta sử dụng nút “Browse” bên cạnh mỗi đầu vào. Đối với phương pháp CNNFA và L\*, file chứa biểu thức chính quy đầu vào được chia làm ba dòng. Dòng đầu tiên là một kí tự phân cách, dòng thứ hai là tập các kí tự của bảng chữ cái phân cách nhau bằng kí tự định nghĩa ở dòng thứ nhất, dòng thứ ba là biểu thức chính quy với các kí tự thuộc biểu thức chính quy này phân cách nhau bằng kí tự định nghĩa ở dòng thứ nhất. Riêng phương pháp L\* thì có thêm độ dài tối đa cho chuỗi các hành vi của thành phần phần mềm. Đối với file đầu vào của phương pháp Thompson, dòng đầu tiên là tập các kí tự của bảng chữ cái phân cách nhau bằng một dấu cách. Các dòng tiếp theo chứa tập các chuỗi hành động của thành phần phần mềm mà các hành động trong mỗi chuỗi cũng phân cách nhau bằng một dấu cách. Sau khi lựa chọn các file đầu vào và đầu ra cho mỗi phương pháp, nhấn “Generate model” để sinh mô hình cho thành phần phần mềm được mô tả trong file “Regular expression file”. Kiểm tra “Output file” và “Benchmark file” để kiểm tra mô hình đầu ra và các thông số hiệu năng của phương pháp.

## 5.2 Thực nghiệm

Để có thể so sánh được hiệu quả của ba phương pháp nêu trên, tôi đã thực hiện thử nghiệm với một số đầu vào khác nhau như: jiter-8-21, cruiseControl, jitter-16-101, sender-len18, sender-len9 (đây là tập dữ liệu thử nghiệm được dùng trong [6]) với môi trường thử nghiệm được trình bày trong bảng 5.1, và kết quả thử nghiệm được trình bày trong bảng 5.2 và 5.3.

Bảng 5.1: Môi trường thử nghiệm công cụ sinh mô hình thành phần phần mềm.

|  |  |
| --- | --- |
| Processor | Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz, 3300 Mhz, 2 Core(s), 4 Logical Processor(s) |
| Total Physical Memory | 3.40 GB |
| OS Name | Microsoft Windows 7 Ultimate |
| IDE | Visual Studio 2010 Ultimate |

Để có được một kết quả thử nghiệm thể hiện được tương quan về thời gian và dung lượng bộ nhớ đã được sử dụng cho mỗi phương pháp và các thông số khác của các mô hình sinh ra, tôi đã tiến hành thực nghiệm với 20 lần thử nghiệm trên từng phương pháp L\*, Thompson, CNNFA cho mỗi đầu vào. Bảng 5.2 trình bày một số thông số của các mô hình được sinh ra như: số lượng kí tự trong bảng chữ cái (|Σ|), số lượng xâu trong ngôn ngữ của ôtômát cần tính toán (|L|), độ dài tối đa của chuỗi hành động của thành phần phần mềm trong hai phương pháp L\* và Thompson (|σ|), kích thước của mô hình sinh ra ngay sau khi áp dụng các thuật toán L\*, Thompson và CNNFA (|M|), số lượng chuyển trạng thái trong từng mô hình tương ứng (|δ|), kích thước của mô hình cuối cùng trong từng phương pháp (|MR|), số lượng chuyển trạng thái trong từng mô hình tương ứng (|δR|). Kết quả thử nghiệm về thời gian và bộ nhớ của cả ba phương pháp được trình bày trong bảng 5.3. Phương pháp L\* và Thompson đều sử dụng việc loại bỏ các chuyển trạng thái rỗng, loại bỏ các trạng thái không đến được và trạng thái không kết thúc. Riêng phương pháp Thompson thì không sử dụng việc sinh tập các chuỗi hành vi của thành phần phần mềm mà chỉ sinh mô hình dựa trên tập các chuỗi hành vi đã có của thành phần phần mềm đã cho. Cả ba phương pháp này đều sử dụng việc đơn định hóa ôtômát hữu hạn không đơn định thu được và tối thiểu hóa ôtômát đơn định sau đó để có được mô hình thành phần phần mềm tối ưu. Để có thể đo được bộ nhớ đã sử dụng cho mỗi phương pháp, tôi sử dụng việc lấy mẫu cho số lượng bộ nhớ được cấp phát cho mỗi phương pháp trong suốt quá trình chạy với tốc độ lấy mẫu là 01 mili-giây. Dung lượng bộ nhớ đã được cấp phát cho mỗi phương pháp khi chạy là dung lượng lớn nhất đã được cấp phát trừ đi dung lượng bộ nhớ khi không có phương pháp nào chạy. Kết quả trung bình cho các phương pháp cả về thời gian và bộ nhớ cho thấy một tương quan chính xác giữa các phương pháp được sử dụng.

**Nhận xét**:

* + Phương pháp CNNFA tỏ ra vượt trội về mặt thời gian xử lý (trung bình của năm dữ liệu thử nghiệm lần lượt là: 00:00:00.048729135, 00:00:00.06638899, 00:00:42.16751797, 00:00:00.07410687, 00:00:00.005094653), đặc biệt là đối với dữ liệu lớn (thời gian xử lý trung bình trong trường hợp jiter-16-101 là: 00:00:42.16751797) . Thời gian xử lý trung bình của phương pháp L\* lần lượt là 00:00:00.13546443, Không đủ bộ nhớ, Không đủ bộ nhớ, Không đủ bộ nhớ, 00:00:00.06516444. Thời gian xử lý trung bình của phương pháp Thompson lần lượt là: 00:00:00.08898711, 00:00:00.07909826, 00:01:13.41504145, 00:00:00.23477519, 00:00:00.008008805.
  + Sinh mô hình phần mềm bằng phương pháp CNNFA không bị giới hạn bởi độ dài tối đa xâu các hành động như trong phương pháp L\* và phương pháp Thompson.
  + Về bộ nhớ thì cả CNNFA và Thompson đều nhỏ hơn rất nhiều so với L\*. Cụ thể là với phương pháp CNNFA, dung lượng bộ nhớ trung bình cần thiết lần lượt là: 568,926; 735,675; 4,226,692; 925,113; Quá nhanh để đo được bộ nhớ đã sử dụng. Với phương pháp L\*, dung lượng bộ nhớ trung bình cần thiết là: 5,963,545; Không đủ bộ nhớ; Không đủ bộ nhớ; Không đủ bộ nhớ; 4,477,565. Với phương pháp Thompson, dung lượng bộ nhớ trung bình cần thiết là: 771,630; 761,366; 2,025,680; 1,323,565; Quá nhanh để đo được bộ nhớ đã sử dụng. Đối với dữ liệu nhỏ thì Thompson sử dụng nhiều bộ nhớ hơn, nhưng khi dữ liệu lớn lên thì CNNFA sử dụng nhiều bộ nhớ hơn. Điều này có thể là do cấu trúc dữ liệu khá phức tạp của CNNFA nhằm đạt được thời gian xử lý tối ưu.

Bảng 5.2: Các thông số của kết quả thử nghiệm

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Hệ thống | |Σ| | |L| | Phương pháp L\* | | | | | Phương pháp Thompson | | | | | Phương pháp CNNFA | | | |
| |σ| | |M| | |δ| | |MR| | |δR| | |σ| | |M| | |δ| | |MR| | |δR| | |M| | |δ| | |MR| | |δR| |
| jiter-8-21 | 3 | 21 | 8 | 14 | 48 | 10 | 11 | 8 | 136 | 155 | 11 | 12 | 114 | 113 | 11 | 12 |
| cruiseControl | 11 | 19 | 9 | - | - | - | - | 9 | 133 | 150 | 11 | 17 | 113 | 112 | 11 | 17 |
| jiter-16-101 | 3 | 101 | 16 | - | - | - | - | 16 | 1290 | 1389 | 42 | 49 | 1188 | 1187 | 42 | 49 |
| sender-len18 | 3 | 18 | 18 | - | - | - | - | 18 | 191 | 207 | 19 | 18 | 172 | 171 | 19 | 18 |
| sender-len9 | 3 | 9 | 9 | 5 | 15 | 4 | 4 | 9 | 56 | 63 | 10 | 9 | 46 | 45 | 10 | 9 |

Bảng 5.3: Bảng kết quả thời gian và bộ nhớ thử nghiệm với L\*, Thompson, CNNFA.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| STT | Hệ thống | Lần thử | Phương pháp L\* | | Phương pháp Thompson | | Phương pháp CNNFA | |
|  |  |  | Thời gian (giờ:phút:giây) | Bộ nhớ sử dụng (bytes) | Thời gian (giờ:phút:giây) | Bộ nhớ sử dụng (bytes) | Thời gian (giờ:phút:giây) | Bộ nhớ sử dụng (bytes) |
| 1 | jiter-8-21 | 1 | 00:00:00.2008471 | 5532056 | 00:00:00.0856086 | 311296 | 00:00:00.0867612 | 637044 |
| 2 | 00:00:00.1513948 | 5535180 | 00:00:00.0893176 | 1428280 | 00:00:00.0532737 | 460276 |
| 3 | 00:00:00.1243328 | 7636664 | 00:00:00.0851745 | 434176 | 00:00:00.0451189 | 516096 |
| 4 | 00:00:00.1788430 | 6205596 | 00:00:00.1113994 | 548864 | 00:00:00.0439492 | 222248 |
| 5 | 00:00:00.1305010 | 7469408 | 00:00:00.0879666 | 589824 | 00:00:00.0434339 | 126944 |
| 6 | 00:00:00.1285320 | 5988932 | 00:00:00.0826442 | 524288 | 00:00:00.0448437 | 1014792 |
| 7 | 00:00:00.1258950 | 6227964 | 00:00:00.1008300 | 524284 | 00:00:00.0543772 | 230744 |
| 8 | 00:00:00.1312897 | 6187232 | 00:00:00.0859831 | 548864 | 00:00:00.0487682 | 455496 |
| 9 | 00:00:00.1252985 | 6693404 | 00:00:00.0828575 | 1260108 | 00:00:00.0436246 | 1525220 |
| 10 | 00:00:00.1290635 | 6970788 | 00:00:00.0826550 | 516096 | 00:00:00.0467924 | 720896 |
| 11 | 00:00:00.1278833 | 6090756 | 00:00:00.0838813 | 1460224 | 00:00:00.0448061 | 695172 |
| 12 | 00:00:00.1325794 | 4919040 | 00:00:00.0837047 | 524288 | 00:00:00.0480383 | 119916 |
| 13 | 00:00:00.1277091 | 6116928 | 00:00:00.0854684 | 573440 | 00:00:00.0452088 | 527284 |
| 14 | 00:00:00.1250917 | 5920944 | 00:00:00.1072488 | 491520 | 00:00:00.0455671 | 77656 |
| 15 | 00:00:00.1387444 | 7016420 | 00:00:00.0957336 | 499712 | 00:00:00.0457957 | 568124 |
| 16 | 00:00:00.1250979 | 5868660 | 00:00:00.0848955 | 1398264 | 00:00:00.0455167 | 294912 |
| 17 | 00:00:00.1249253 | 5161408 | 00:00:00.0845018 | 516096 | 00:00:00.0459736 | 1521112 |
| 18 | 00:00:00.1263270 | 4551928 | 00:00:00.0825599 | 1372680 | 00:00:00.0508339 | 465392 |
| 19 | 00:00:00.1275897 | 4846876 | 00:00:00.0882749 | 450560 | 00:00:00.0470300 | 215592 |
| 20 | 00:00:00.1273434 | 4330720 | 00:00:00.0890368 | 1459732 | 00:00:00.0448695 | 1051728 |
|  | Trung bình |  | 00:00:00.13546443 | 5,963,545 | 00:00:00.08898711 | 771,630 | 00:00:00.048729135 | 568,926 |
| 2 | cruiseControl | 1 | Không đủ bộ nhớ | Không đủ bộ nhớ | 00:00:00.0774924 | 385024 | 00:00:00.0666394 | 1073612 |
| 2 | 00:00:00.0763442 | 1517752 | 00:00:00.0649500 | 1130496 |
| 3 | 00:00:00.0740543 | 466944 | 00:00:00.0636612 | 139264 |
| 4 | 00:00:00.0761728 | 491520 | 00:00:00.0678824 | 1782320 |
| 5 | 00:00:00.0744881 | 499712 | 00:00:00.0648156 | 665436 |
| 6 | 00:00:00.0872392 | 401408 | 00:00:00.0634954 | 418888 |
| 7 | 00:00:00.0744449 | 466940 | 00:00:00.0644686 | 570140 |
| 8 | 00:00:00.0736780 | 458752 | 00:00:00.0691877 | 904640 |
| 9 | 00:00:00.0796669 | 1347984 | 00:00:00.0682973 | 793340 |
| 10 | 00:00:00.0759489 | 507904 | 00:00:00.0646113 | 163840 |
| 11 | 00:00:00.0991332 | 548864 | 00:00:00.0627500 | 465664 |
| 12 | 00:00:00.0789438 | 532480 | 00:00:00.0635492 | 564152 |
| 13 | 00:00:00.0747724 | 499712 | 00:00:00.0650738 | 1677004 |
| 14 | 00:00:00.0786238 | 1379112 | 00:00:00.0724809 | 350960 |
| 15 | 00:00:00.0808714 | 507904 | 00:00:00.0679169 | 405784 |
| 16 | 00:00:00.0974687 | 1409852 | 00:00:00.0774420 | 1301088 |
| 17 | 00:00:00.0752827 | 516096 | 00:00:00.0663731 | 163840 |
| 18 | 00:00:00.0737150 | 1381160 | 00:00:00.0643354 | 955188 |
| 19 | 00:00:00.0765700 | 442368 | 00:00:00.0629509 | 557056 |
| 20 | 00:00:00.0770545 | 1465828 | 00:00:00.0668987 | 630784 |
|  | Trung bình |  |  |  | 00:00:00.07909826 | 761,366 | 00:00:00.06638899 | 735,675 |
| 3 | jitter-16-101 | 1 | Không đủ bộ nhớ | Không đủ bộ nhớ | 00:01:14.0005629 | 1657468 | 00:00:43.6428225 | 5497708 |
| 2 | 00:01:12.2964041 | 1775604 | 00:00:41.5598194 | 2764668 |
| 3 | 00:01:16.1635906 | 1772940 | 00:00:41.4860294 | 5093816 |
| 4 | 00:01:19.2741980 | 2532776 | 00:00:41.4427161 | 5924248 |
| 5 | 00:01:14.5102285 | 1762464 | 00:00:42.4190050 | 2215156 |
| 6 | 00:01:16.2713128 | 1741996 | 00:00:43.0655291 | 5293680 |
| 7 | 00:01:17.4454949 | 3559904 | 00:00:41.5357459 | 5735688 |
| 8 | 00:01:11.5874292 | 1766900 | 00:00:45.1296372 | 2386616 |
| 9 | 00:01:12.9073685 | 1735492 | 00:00:42.7746007 | 5858808 |
| 10 | 00:01:12.5955386 | 2168472 | 00:00:42.2057743 | 4953496 |
| 11 | 00:01:14.2087711 | 1704728 | 00:00:41.1593824 | 2918536 |
| 12 | 00:01:12.1183115 | 1746312 | 00:00:46.1963192 | 6268000 |
| 13 | 00:01:11.9039033 | 2121076 | 00:00:41.3894212 | 2672384 |
| 14 | 00:01:11.4665072 | 1726600 | 00:00:41.2988785 | 3170120 |
| 15 | 00:01:11.7273517 | 3105600 | 00:00:41.2395097 | 4773824 |
| 16 | 00:01:12.9667289 | 2001376 | 00:00:41.3382874 | 3098920 |
| 17 | 00:01:11.6265070 | 1713472 | 00:00:41.3608864 | 3194852 |
| 18 | 00:01:11.7351488 | 1858296 | 00:00:41.3836496 | 6213280 |
| 19 | 00:01:11.8738907 | 1768012 | 00:00:41.3561536 | 3106944 |
| 20 | 00:01:11.6215807 | 2294116 | 00:00:41.3661918 | 3393088 |
|  | Trung bình |  |  |  | 00:01:13.41504145 | 2,025,680 | 00:00:42.16751797 | 4,226,692 |
| 4 | sender-len18 | 1 | Không đủ bộ nhớ | Không đủ bộ nhớ | 00:00:00.2163061 | 1309280 | 00:00:00.1289219 | 1497460 |
| 2 | 00:00:00.2702243 | 1419016 | 00:00:00.0681801 | 988116 |
| 3 | 00:00:00.2206555 | 1479452 | 00:00:00.0772504 | 1439880 |
| 4 | 00:00:00.2257345 | 1338388 | 00:00:00.0740552 | 651616 |
| 5 | 00:00:00.2431820 | 954504 | 00:00:00.0691594 | 709136 |
| 6 | 00:00:00.2529120 | 1378584 | 00:00:00.0682827 | 868612 |
| 7 | 00:00:00.2159808 | 1401040 | 00:00:00.0677272 | 1044652 |
| 8 | 00:00:00.2189866 | 1993752 | 00:00:00.0777120 | 526968 |
| 9 | 00:00:00.2175651 | 1391320 | 00:00:00.0714039 | 1021680 |
| 10 | 00:00:00.2944168 | 1394888 | 00:00:00.0708167 | 887300 |
| 11 | 00:00:00.2223508 | 1423200 | 00:00:00.0734481 | 770080 |
| 12 | 00:00:00.2148413 | 1105920 | 00:00:00.0689386 | 797872 |
| 13 | 00:00:00.2180250 | 1138688 | 00:00:00.0734376 | 979184 |
| 14 | 00:00:00.2183873 | 1179648 | 00:00:00.0699571 | 602596 |
| 15 | 00:00:00.2476984 | 1040384 | 00:00:00.0677599 | 1133372 |
| 16 | 00:00:00.2434594 | 1122304 | 00:00:00.0694583 | 1363388 |
| 17 | 00:00:00.2153271 | 1122304 | 00:00:00.0700952 | 942080 |
| 18 | 00:00:00.2263739 | 1212416 | 00:00:00.0730874 | 680800 |
| 19 | 00:00:00.2598716 | 1494716 | 00:00:00.0694567 | 724356 |
| 20 | 00:00:00.2532053 | 1571504 | 00:00:00.0735890 | 873112 |
|  | Trung bình |  |  |  | 00:00:00.23477519 | 1,323,565 | 00:00:00.07410687 | 925,113 |
| 5 | sender-len9 | 1 | 00:00:00.1416172 | 2146388 | 00:00:00.0072171 | Quá nhanh để đo được bộ nhớ đã sử dụng | 00:00:00.0044268 | Quá nhanh để đo được bộ nhớ đã sử dụng |
| 2 | 00:00:00.0612060 | 6146916 | 00:00:00.0075980 | 00:00:00.0063292 |
| 3 | 00:00:00.0581457 | 3150236 | 00:00:00.0104251 | 00:00:00.0045736 |
| 4 | 00:00:00.0582378 | 4425848 | 00:00:00.0075194 | 00:00:00.0063264 |
| 5 | 00:00:00.0596753 | 5438148 | 00:00:00.0079849 | 00:00:00.0041145 |
| 6 | 00:00:00.0585625 | 3420856 | 00:00:00.0074425 | 00:00:00.0041559 |
| 7 | 00:00:00.0915920 | 3718292 | 00:00:00.0085208 | 00:00:00.0070849 |
| 8 | 00:00:00.0631224 | 9594868 | 00:00:00.0073415 | 00:00:00.0051212 |
| 9 | 00:00:00.0605116 | 4530476 | 00:00:00.0077508 | 00:00:00.0055529 |
| 10 | 00:00:00.0561550 | 5608460 | 00:00:00.0073947 | 00:00:00.0050074 |
| 11 | 00:00:00.0588707 | 4592012 | 00:00:00.0101949 | 00:00:00.0064937 |
| 12 | 00:00:00.0617674 | 4872844 | 00:00:00.0075414 | 00:00:00.0040069 |
| 13 | 00:00:00.0616458 | 3972428 | 00:00:00.0072824 | 00:00:00.0050093 |
| 14 | 00:00:00.0574752 | 3457136 | 00:00:00.0074500 | 00:00:00.0040458 |
| 15 | 00:00:00.0602755 | 6097888 | 00:00:00.0072830 | 00:00:00.0040959 |
| 16 | 00:00:00.0561914 | 4493940 | 00:00:00.0087425 | 00:00:00.0045477 |
| 17 | 00:00:00.0575091 | 3925664 | 00:00:00.0080515 | 00:00:00.0044264 |
| 18 | 00:00:00.0629918 | 1928312 | 00:00:00.0076805 | 00:00:00.0045082 |
| 19 | 00:00:00.0576752 | 4456812 | 00:00:00.0078229 | 00:00:00.0038754 |
| 20 | 00:00:00.0600612 | 3573776 | 00:00:00.0089322 | 00:00:00.0075231 |
|  | Trung bình |  | 00:00:00.06516444 | 4,477,565 | 00:00:00.008008805 |  | 00:00:00.005094653 |  |

* + Kích thước của những mô hình sinh bởi thuật toán L\* có kích thước nhỏ hơn những mô hình được sinh bởi hai thuật toán Thompson và CNNFA. Điều tương tự cũng đúng với các mô hình cuối cùng đối với cả ba phương pháp sinh mô hình. Trong phương pháp áp dụng thuật toán L\*, kích thước của mô hình được sinh bởi thuật toán L\*, số chuyển trạng thái của những mô hình đó, kích thước của mô hình tối ưu cuối cùng, số chuyển trạng thái của mô hình tối ưu cuối cùng cho hai hệ thống *jiter-8-21* và *sender-len9* lần lượt là: 14, 48, 10, 11 và 5, 15, 4, 4. Trong khi đó, các số liệu tương ứng cho phương pháp sử dụng thuật toán Thompson lần lượt là: 136, 155, 11, 12 và 56, 63, 10, 9. Các số liệu tương ứng cho phương pháp sử dụng thuật toán CNNFA lần lượt là: 114, 113, 11, 12 và 46, 45, 10, 9.

## 5.3 Ý nghĩa của công cụ thực nghiệm

Công cụ thực nghiệm đã triển khai được cả ba phương pháp sinh mô hình tự động cho thành phần mềm đã đề cập trong chương 3 và chương 4 của luận văn này. Công cụ này có ý nghĩa rất quan trọng trong việc sinh mô hình cho các thành phần phần mềm trong thực tế, là cơ sở cho việc kiểm chứng mô hình, kiểm chứng mô hình đảm bảo giả định được đề cập trong [5] và kiểm thử dựa trên mô hình nhằm nâng cao chất lượng phần mềm. Thay vì kiểm chứng toàn bộ phần mềm thì ta phân rã phần mềm thành nhiều thành phần và kiểm chứng riêng biệt các thành phần đó. Về mặt khoa học, công cụ cung cấp công cụ để so sánh các phương pháp sinh mô hình cho thành phần phần mềm hiện có được trình bày trong chương 3 với phương pháp sinh mô hình được đề xuất trong chương 4. Ngoài ra, công cụ còn có thể được dùng để cung cấp mô hình cho thành phần phần mềm phục vụ cho các nghiên cứu sau này. Từ đầu ra của công cụ này, ta có thể chuyển thành các LTS tương ứng làm đầu vào cho công cụ được đề cập trong [5] là công cụ sinh giả định được dùng trong phương pháp kiểm chứng đảm bảo giả định.

Ngoài ra, về mặt thực tế, trong nỗ lực của việc tự động hóa một số công đoạn từ đặc tả yêu cầu, thiết kế, sinh mã nguồn, kiểm thử tự động, công cụ này đóng vai trò như một mắt xích quan trọng trong chuỗi hành động trên thông qua việc sinh mô hình cho thành phần phần mềm thông qua biểu thức chính quy có được từ đầu ra của quá trình thiết kế là biểu đồ tuần tự. Thông qua công cụ này, ta có thể sinh mô hình thành phần phần mềm tương ứng và mô hình này là cơ sở cho việc kiểm thử tự động bằng kiểm chứng mô hình hoặc kiểm thử dựa trên mô hình xem thành phần phần mềm được thực thi cuối cùng có đáp ứng được mô hình thiết kế ban đầu hay không.

# Chương 6: KẾT LUẬN

Trong ngữ cảnh của ngành công nghiệp phần mềm hiện đại, khi công nghệ phần mềm hướng thành phần đóng vai trò quan trọng trong việc tạo ra các phần mềm lớn, có chất lượng cao thì mỗi thành phần mềm có thể coi như một hộp đen, có thể được phát triển bởi bên thứ ba, ta không biết được mã nguồn của nó. Tuy nhiên, nếu nhà cung cấp thành phần đó cung cấp một đặc tả cho thành phần đó dưới dạng một biểu thức chính quy các chuỗi các hành vi của thành phần phần mềm thì ta có thể xây dựng mô hình cho thành phần đó làm cơ sở cho việc kiểm chứng mô hình, kiểm thử tự động dựa trên mô hình cho toàn bộ phần mềm. Một ý nghĩa không kém phần quan trọng nữa của việc sinh mô hình tự động dựa vào đầu vào là biểu thức chính quy các chuỗi hành động của thành phần phần mềm là nó có thể đóng vai trò sinh ra các mô hình của thành phần phần mềm từ thiết kế nhằm tự động hóa một số công đoạn làm phần mềm từ thiết kế đến kiểm thử tự động.

Một số phương pháp sinh mô hình cho thành phần phần mềm hiện nay gồm có sinh mô hình theo phương pháp áp dụng thuật toán L\* và phương pháp áp dụng thuật toán Thompson. Trong phương pháp sinh mô hình sử dụng thuật toán L\* [5], ta sinh mô hình cho thành phần phần mềm thông qua phương pháp học biểu thức chính quy đầu vào để sinh mô hình ứng viên cho thành phần phần mềm thông qua việc tương tác với đối tượng Teacher. Ở mỗi bước lặp, thuật toán thực hiện truy vấn đến đối tượng Teacher để có được tập các chuỗi hành động đóng và nhất quán có thể thực hiện được trên thành phần phần mềm. Từ đó, ta có thể xây dựng mô hình ứng viên tương ứng cho thành phần phần mềm. Thông qua đối tượng Teacher, ta có thể kiểm tra mô hình ứng viên đó có phù hợp hay không, nếu không thì đối tượng Teacher trả lại một phản ví dụ cho thuật toán, và thuật toán lại lặp lại như trên, nếu đối tượng Teacher trả lời là mô hình ứng viên là phù hợp thì thuật toán dừng lại và ta có một mô hình chính xác cho thành phần phần mềm. Mô hình được sinh ra của phương pháp này có kích thước là nhỏ so với hai phương pháp áp dụng thuật toán Thompson và CNNFA. Phương pháp này mặc dù sinh được mô hình cho thành phần phần mềm, nhưng độ phức tạp của toàn bộ quá trình là lớn vì phụ thuộc vào thuật toán VC. Ngoài ra, thuật toán cũng sử dụng một khối lượng bộ nhớ rất lớn và do đó khó có thể thực hiện được cho các bài toán thực tế. Trong phương pháp sinh mô hình sử dụng thuật toán Thompson [6], ta có thể sinh được mô hình thành phần phần mềm thông qua đặc tả của thành phần đó là tập các chuỗi các hành vi có thể thực hiện được trên thành phần phần mềm đó. Việc thử nghiệm từng chuỗi hành động có thể thực hiện được trên thành phần phần mềm để có được danh sách chuỗi hành vi đó cũng khiến cho phương pháp này có một độ phức tạp rất lớn và khó thực hiện được trong thực tiễn. Tuy vậy, xét ở một góc độ khác, khi đầu vào của phương pháp sử dụng thuật toán Thompson là một biểu thức chính quy thì việc sinh mô hình theo phương pháp này cũng có những ưu điểm rất lớn so với phương pháp L\*.

Luận văn này đã đề xuất một phương pháp khác để sinh mô hình cho thành phần phần mềm khi đặc tả của nó là một biểu thức chính quy các hành động có thể thực hiện được trên thành phần phần mềm. Phương pháp này sử dụng thuật toán CNNFA để phân tích biểu thức chính quy đã cho và sinh mô hình thành phần phần mềm tương ứng. Cơ sở của phương pháp này là phương pháp chuyển biểu thức chính quy thành ôtômát hữu hạn không đơn định theo phương pháp McNaughton và Yamada. Phương pháp này sau đó được Chang cải tiến để đạt được hiệu quả tối đa khi phân tích biểu thức chính quy nhằm giảm thiểu các phép nhân tập hợp đắt đỏ khi sinh các ôtômát tương ứng cho các biểu thức chính quy thành phần. Thuật toán CNNFA sử dụng việc định giá trễ các biểu thức biểu diễn tập các chuyển trạng thái cho các ôtômát thành phần bằng việc biểu diễn chúng qua lazyδ. Chúng ta chỉ định giá biểu thức lazyδ này ở bước cuối cùng, khi cần có ôtômát hữu hạn cuối cùng. Ở các bước trung gian, ta đều biểu diễn ôtômát dưới dạng *nén* được gọi là CNNFA. Phương pháp này là phương pháp vượt trội so với các phương pháp L\* và Thompson về mặt lý thuyết về cả thời gian và không gian sử dụng. Tuy nhiên, thực nghiệm cho thấy rằng CNNFA vượt trội về mặt thời gian so với cả phương pháp L\* và Thompson khi cùng thử nghiệm với đầu vào là biểu thức chính quy đặc tả cho thành phần phần mềm. Còn về mặt không gian thì CNNFA cũng là vượt trội so với L\*, nhưng so với Thompson thì CNNFA chỉ sử dụng ít không gian bộ nhớ hơn khi làm việc với các biểu thức chính quy nhỏ, khi biểu thức chính quy lớn lên, không gian mà CNNFA sử dụng lại lớn hơn phương pháp Thompson. Điều này có thể là do cấu trúc dữ liệu đặc biệt của CNNFA, điều mà giúp cho CNNFA có một tốc độ tối ưu so với hai phương pháp còn lại nhưng lại khiến nó trả giá bằng việc sử dụng nhiều bộ nhớ hơn.

Về mặt thực nghiệm, công cụ MG dùng để sinh mô hình tự động cho thành phần phần mềm sử dụng cả ba phương pháp L\*, Thompson và CNNFA. Công cụ trên sinh một mô hình mô tả hành vi của một thành phần phần mềm dưới dạng một ôtômát hữu hạn là đặc tả hình thức cho thành phần đó. Mô hình thu được không những là cơ sở cho việc áp dụng các kỹ thuật kiểm chứng mô hình trong kiểm thử tự động mà còn là đầu vào cho việc sinh giả định hỗ trợ cho kỹ thuật kiểm chứng đảm bảo giả định hay kiểm thử dựa trên mô hình. Ngoài ra, công cụ này còn đóng vai trò to lớn trong việc tự động hóa một số công đoạn của việc phát triển phần mềm từ thiết kế, sinh mô hình tự động, sinh mã nguồn, kiểm thử tự động và cho các nghiên cứu sau này.

Luận văn đã trình bày các kiến thức cơ bản về đặc tả hình thức cho các thành phần phần mềm, đã tóm tắt được một số phương pháp sinh mô hình tự động cho thành phần phần mềm hiện nay là phương pháp sinh mô hình sử dụng thuật toán học L\* và thuật toán Thompson và đã đề xuất sử dụng một phương pháp sinh mô hình tự động mới dựa trên thuật toán CNNFA. Các phương pháp sinh mô hình dựa trên thuật toán L\*, Thompson, CNNFA đã được cài đặt nhằm so sánh các phương pháp với nhau và sinh được mô hình tự động là đầu vào cho các nghiên cứu khác cũng như cung cấp đầu vào cho việc kiểm chứng mô hình là một khâu quan trọng trong việc tự động hóa từ thiết kế đến kiểm thử tự động.

Hướng nghiên cứu tiếp theo của đề tài là nghiên cứu cách biểu diễn cấu trúc dữ liệu của CNNFA để nó có thể sử dụng ít không gian bộ nhớ hơn đối với các đầu vào lớn. Ngoài ra, ta cũng sẽ thực hiện các nghiên cứu liên quan đến việc tự động hóa từ các mô hình thiết kế phần mềm UML (Unified Modeling Language), các ngôn ngữ ràng buộc đối tượng OCL (Object Constraint Language) đến việc sinh ra các mô hình để có thể sử dụng cho kiểm thử dựa trên mô hình và kiểm chứng mô hình.

# TÀI LIỆU THAM KHẢO

**Tiếng Việt**

1. Đỗ Đức Giáo, Toán rời rạc ứng dụng trong tin học, Nhà xuất bản giáo dục Việt Nam, 2011.

**Tiếng Anh**

1. D. Angluin, “Learning regular sets from queries and counterexamples”, Information and Computation, 75(2), pp. 87-106, Nov. 1987.
2. Ken Thompson, "Regular expression search algorithm", Communications of the ACM 11(6), pp. 419–422, June. 1968.
3. C. Chang, “From regular expressions to DFA’s using compressed NFA’s”, Ph.D. Thesis, New York University, New York, 1992.
4. H. M. Duong, L. K. Trinh, and P. N. Hung, “An Assume-Guarantee Model Checker for Component-Based Systems”, The 10th IEEE-RIVF International Conference on Computing and Communication Technologies, 2013 (accepted).

<http://www.uet.vnu.edu.vn/~hungpn/GUI-AGTool/>

1. L. B. Cuong and P. N. Hung, “A Method for Generating Models of Black-box Components”, 4th International Conference on Knowledge and Systems Engineering (KSE 2012), IEEE Computer Society Press, pp. 177-222, 2012.
2. T. S. Chow, "Testing software design modeled by ﬁnite-state machine", IEEE Transactions on Software Engineering, vol. 4, no. 3, pp. 178–187, May. 1978.
3. A.Nerode, "Linear Automaton Transformations", Proceedings of the American Mathematical Society Vol. 9, No. 4, pp. 541-544, Aug. 1958.
4. C. Chang, R. Paige, “From regular expressions to DFA’s using compressed NFA’s”, Theoretical Computer Science 178, 1-36, 1997.
5. P. N. Hung, N. V. Ha, T. Aoki and T. Katayama, “On Optimization of Minimized Assumption Generation Method for Component-based Software Verification”, IEICE Trans. on Fundamentals, Special Issue on Software Reliability Engineering, Vol. E95-A, No.9, pp. 1451-1460, Sep. 2012.
6. D. Lorenzoli, L. Mariani and M. Pezz` e, “Automatic generation of software behavioral models”, ACM, Proceedings of the 30th international conference on Software engineering, pp. 501-510, 2008.
7. J.C. Corbett, M.B. Dwyer, J. Hatcliff, S. Laubach, C.S. Pasareanu, Robby and Hongjun Zheng, ”Bandera: extracting finite-state models from Java source code”, Software Engineering, Proceedings of the 2000 International Conference on, pp. 439-448d, 2000.
8. O. Tkachuk, M.B. Dwyer and C.S. Pasareanu, “Automated environment generation for software model checking”, Automated Software Engineering, Proceedings. 18th IEEE International Conference on, pp. 116-127, 2003.
9. A. Groce, D. Peled, and M. Y annak akis, “Black box checking”, J. Autom. Lang. Comb., pp. 225-246, Nov. 2001.
10. A. Groce, D. Peled, and M. Y annak akis, “Adaptive Model Checking”, Logic Journal of the IGPL, vol. 14, no. 5, pp. 729-744, Oct. 2006.

1. Đây là tên của một công ty phần mềm có website là: <http://www.devexpress.com/>. [↑](#footnote-ref-1)
2. Đây là tên của một công ty phần mềm có website là: <http://independentsoft.de/>. [↑](#footnote-ref-2)