Notes on CS-3372:

Formal Languages and the Theory of Computation

Instructor: PGS. TS. Nguyễn Đình Hân

Troy University Resources

1 Ôtômat và ngôn ngữ chính quy (Automata and Regular Languages)

Một *bảng chữ cái* (alphabet) Σ là tập hữu hạn, không rỗng các ký hiệu. Từ các ký hiệu đó, ta tạo ra các *xâu* hay *chuỗi* (string). Một *ngôn ngữ hình thức* (formal language) L trên Σ là một tập con của Σ^*, tức tập những xâu được xét. Khi đã mô tả L, ta cần kiểm tra một xâu cụ thể có thuộc L hay không.

Do đó câu hỏi quan trọng là:

Làm thế nào xây dựng một "máy" có thể xác định một xâu có thuộc một ngôn ngữ hay không?

Đó là bài toán mà *ôtômat* (automaton, số nhiều: automata) được dùng để nghiên cứu.

- Một xâu ω có thể thuộc hoặc không thuộc L; máy cần *chấp nhận* (accept) hoặc *từ chối* (reject) xâu đó.
- Máy *nhận* (recognize) ngôn ngữ L khi nó chấp nhận đúng những xâu thuộc L.

1.1 Ký hiệu và cách đọc

Trước khi xem định nghĩa máy, nên nhận diện các ký hiệu sẽ lặp lại trong công thức. Phần dưới là cách đọc, còn ý nghĩa cụ thể sẽ xuất hiện ở từng định nghĩa.

Tên đọc của các ký hiệu (tiếng Anh):

Σ = sigma, ω = omega, ε = epsilon, δ = delta, q_0 = q-zero, Σ^* = sigma-star.

Nắm được đám này thì đọc khoảng 80–90% công thức DFA đầu chương sẽ đỡ vướng hẳn.

1.2 x^k nghĩa là gì?

Khi làm việc với xâu, ta còn dùng ký hiệu lũy thừa để viết ngắn phép nối lặp lại. Đây không phải phép lũy thừa số học.

Với xâu (string),

$$x^k$$

nghĩa là nối xâu x với chính nó k lần. Chẳng hạn, nếu $x = ab$ thì $x^3 = ababab$; còn $x^0 = \varepsilon$ là chuỗi rỗng.

1.3 Định nghĩa ôtômat hữu hạn đơn định (DFA) bằng bộ 5 thành phần

Đây là công thức phải thuộc:

$$M = (Q, \Sigma, \delta, q_0, F).$$

DFA (deterministic finite automaton): năm thành phần

Q	Tập hữu hạn các trạng thái (states).
Σ	Bảng chữ cái đầu vào (input alphabet).
δ	Hàm chuyển trạng thái (transition function) $\delta : Q \times \Sigma \to Q$.
q_0	Trạng thái bắt đầu, $q_0 \in Q$.
F	Tập các trạng thái chấp nhận (accepting states), $F \subseteq Q$.

Slide định nghĩa DFA bằng đúng 5 thành phần này. DFA đọc từng ký hiệu đầu vào, chuyển trạng thái theo δ, rồi kiểm tra xem trạng thái cuối có thuộc F hay không.

Điểm dễ nhầm

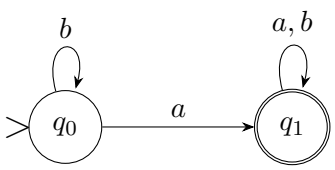
Đơn định (deterministic) không có nghĩa là mỗi trạng thái chỉ có một mũi tên đi ra. Với *mỗi cặp* (q, a), DFA phải xác định **đúng một** trạng thái kế tiếp $\delta(q, a)$.

Q**:** tập hữu hạn các trạng thái.

Ví dụ:

$$Q = \{q_0, q_1\}.$$

Ví dụ DFA nhỏ: Xét $\Sigma = \{a, b\}$ và ngôn ngữ gồm các xâu có ít nhất một ký tự a. DFA có thể dùng hai trạng thái: q_0 (chưa thấy a) và q_1 (đã thấy a).



Nếu đọc `bbab`, máy đi từ q_0 sang q_1 khi gặp a và chấp nhận chuỗi. Đây là *ví dụ minh họa bổ sung*, độc lập với bảng NFA ở phần tiếp theo.

1.4 Chuyển ôtômat hữu hạn không đơn định (NFA) sang DFA

DFA có đúng một trạng thái kế tiếp cho mỗi cặp (trạng thái, ký hiệu), còn NFA (nondeterministic finite automaton) có thể có nhiều trạng thái kế tiếp, hoặc không có trạng thái nào. Để mô phỏng NFA bằng DFA, ta gom các trạng thái mà NFA *có thể đang ở* thành một trạng thái mới của DFA.

Khái quát cách chuyển NFA $\to$ DFA theo đúng kiểu thầy/Sipser như sau.

Giả sử NFA:

$$N = (Q, \Sigma, \delta, q_0, F).$$

Ta xây DFA:

$$D = (Q', \Sigma, \delta', q_0', F').$$

Năm bước của phương pháp xây dựng tập con (subset construction)
1. Tạo các tập trạng thái $\to$ **2.** tìm trạng thái bắt đầu $\to$ **3.** tính chuyển trạng thái $\to$ **4.** xác định trạng thái chấp nhận $\to$ **5.** giữ các trạng thái có thể đạt tới và vẽ DFA.

Troy University Resources

1. Tạo các trạng thái có thể có của DFA

Mỗi trạng thái của DFA là một tập các trạng thái của NFA:

$$Q' = \mathcal{P}(Q).$$

Ký hiệu $\mathcal{P}(Q)$ là *tập lũy thừa* (power set), tức tập tất cả các tập con của Q. Đây là tập trạng thái có thể xây dựng; khi vẽ DFA thực tế, ta chỉ cần giữ những trạng thái đạt tới được từ trạng thái bắt đầu.

Nếu NFA có n trạng thái thì DFA có tối đa 2^n trạng thái.

2. Xác định trạng thái bắt đầu mới

Nếu không có ε:

$$q_0' = \{q_0\}.$$

Nếu có phép chuyển ε thì lấy *bao đóng epsilon* (epsilon-closure):

$$q_0' = E(\{q_0\}).$$

Trong đó $E(R)$ là tập tất cả các trạng thái đạt tới được từ một trạng thái trong R qua 0 hoặc nhiều phép chuyển ε (không đọc ký hiệu đầu vào).

3. Tính hàm chuyển cho từng trạng thái mới

Đây là phần quan trọng nhất.

Với mỗi

$$R \in Q'$$

và mỗi

$$a \in \Sigma,$$

ta lấy tập trạng thái kế tiếp của mọi trạng thái nằm trong R, rồi hợp các tập đó lại.

Nếu không có epsilon:

$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a).$$

Nếu có epsilon:

$$\delta'(R, a) = E\left(\bigcup_{r \in R} \delta(r, a) \right).$$

Bảng chuyển trạng thái (transition table). Ví dụ dưới đây xét NFA có các trạng thái $\{1, 2, 3\}$. Mỗi hàng là một *tập trạng thái* của NFA; hai cột a, b cho biết DFA sẽ đi đến tập nào sau khi đọc ký hiệu tương ứng. Nếu liệt kê toàn bộ 2^3 tập con thì có 8 hàng:

Trạng thái DFA R	đọc a	đọc b
$\varnothing$	$\varnothing$	$\varnothing$
$\{1\}$	$\varnothing$	$\{2\}$
$\{2\}$	$\{2,3\}$	$\{3\}$
$\{3\}$	$\{1,3\}$	$\varnothing$
$\{1,2\}$	$\{2,3\}$	$\{2,3\}$
$\{1,3\}$	$\{1,3\}$	$\{2\}$
$\{2,3\}$	$\{1,2,3\}$	$\{3\}$
$\{1,2,3\}$	$\{1,2,3\}$	$\{2,3\}$

Đọc thử bảng ngay trên. Nếu trạng thái bắt đầu của NFA là 1, ta bắt đầu tại $\{1\}$. Chuỗi `ba` làm DFA đi theo $\{1\} \xrightarrow{b} \{2\} \xrightarrow{a} \{2,3\}$. Nếu tập trạng thái chấp nhận của NFA là $F = \{1\}$, đường đi này không được chấp nhận vì tập cuối không chứa 1. Ngược lại, `bba` cho $\{1\} \xrightarrow{b} \{2\} \xrightarrow{b} \{3\} \xrightarrow{a} \{1,3\}$, nên được chấp nhận.

4. Xác định các trạng thái chấp nhận mới

Một trạng thái của DFA là trạng thái chấp nhận nếu tập tương ứng chứa ít nhất một trạng thái chấp nhận của NFA:

$$F' = \{R \subseteq Q \mid R \cap F \neq \varnothing\}.$$

Ví dụ NFA có:

$$F = \{1\},$$

thì các trạng thái DFA

$$\{1\},\ \{1,2\},\ \{1,3\},\ \{1,2,3\}$$

đều là trạng thái chấp nhận.

5. Vẽ DFA và bỏ các trạng thái không đạt tới được

Sau khi có bảng:

Trạng thái DFA	a	b
R_1	$\cdots$	$\cdots$
R_2	$\cdots$	$\cdots$

thì vẽ sơ đồ trạng thái (state diagram).

Cuối cùng, trạng thái nào **không đạt tới được từ** q_0 thì có thể bỏ. Đây là bước loại các trạng thái không đạt tới được (unreachable states).

1.5 Hai chiều

Vì ta mô phỏng được NFA bằng DFA và chiều ngược lại thì trực tiếp (DFA cũng là một trường hợp đặc biệt của NFA), hai loại máy này nhận đúng cùng một lớp ngôn ngữ, dù số trạng thái có thể khác nhau.

Ngôn ngữ chính quy = những ngôn ngữ được ôtômat hữu hạn nhận

Troy University Resources

1.6 Chứng minh một ngôn ngữ không chính quy

Các phép dựng máy ở trên giúp chứng minh một ngôn ngữ là chính quy. Nhưng để chứng minh một ngôn ngữ *không chính quy*, ta thường dùng *bổ đề bơm* (pumping lemma) cho ngôn ngữ chính quy. Đây là **điều kiện cần**, không phải điều kiện đủ để kết luận một ngôn ngữ chính quy.

Nếu sử dụng bổ đề bơm để chứng minh một ngôn ngữ L không chính quy, cách lập luận thường theo khuôn sau:

Giả sử L chính quy $\rightarrow$ dùng bổ đề bơm $\rightarrow$ tạo mâu thuẫn

Bổ đề bơm (pumping lemma): ai chọn điều gì?

Bên quyết định	**Lựa chọn hoặc điều kiện**
Bổ đề bơm	Bảo đảm tồn tại *độ dài bơm* (pumping length) n.
Người chứng minh	Chọn $\omega \in L$, $\lvert\omega\rvert \geq n$.
Mọi cách phân tách hợp lệ	Xét *bất kỳ* $\omega = xyz$ với $\lvert xy\rvert \leq n$, $\lvert y\rvert > 0$.
Người chứng minh	Với từng cách phân tách ấy, tìm $i \geq 0$ làm $xy^iz \notin L$.

Bước 1 — Giả sử ngược lại rằng L chính quy

Viết:

$$\text{Suppose } L \text{ is regular.}$$

Khi đó, theo bổ đề bơm, tồn tại một độ dài bơm (pumping length):

$$n \geq 1$$

sao cho mọi xâu đủ dài trong L có cách phân tách thỏa các điều kiện của bổ đề.

Bước 2 — Chọn một xâu đặc biệt $\omega \in L$

Phải chọn:

$$|\omega| \geq n$$

và nên chọn xâu sao cho cấu trúc của nó phụ thuộc trực tiếp vào n.

Ví dụ với:

$$L = \{a^k b^k \mid k \geq 0\}$$

ta chọn:

$$\omega = a^n b^n$$

vì:

$$\omega \in L, \qquad |\omega| = 2n \geq n.$$

Bước 3 — Xét một cách phân tách (decomposition) bất kỳ

Bổ đề bơm bảo đảm tồn tại cách phân tách:

$$\omega = xyz$$

thỏa:

$$|xy| \leq n, \qquad y \neq \varepsilon.$$

Điểm rất quan trọng:

> Khi chứng minh không chính quy, không được tùy ý chọn một cách phân tách thuận tiện rồi chỉ xét cách đó.

Phải chỉ ra rằng với mọi cách phân tách hợp lệ, đều có thể chọn số lần bơm làm xâu rơi khỏi L. Thứ tự này quan trọng: người chứng minh chọn ω, còn cách phân tách x, y, z phải được xét *bất kỳ* miễn thỏa các điều kiện của lemma.

Với:

$$\omega = a^n b^n$$

vì:

$$|xy| \leq n$$

nên cả x và y đều nằm trong n chữ a đầu tiên. Vì y không rỗng, ta có thể viết $y = a^k$ với $k \geq 1$. Điều này giải thích vì sao chỉ cần xóa y là số chữ a và b không còn bằng nhau.

Bước 4 — Chọn một i để tạo xâu không thuộc ngôn ngữ

Bổ đề bơm yêu cầu:

$$xy^i z \in L$$

cho mọi

$$i \geq 0.$$

Vậy để tạo mâu thuẫn, ta chỉ cần tìm một i khiến:

$$xy^i z \notin L.$$

Thường dễ nhất là:

$$i = 0$$

hoặc:

$$i = 2.$$

Ví dụ chọn:

$$i = 0.$$

Ta bỏ y.

Ban đầu:

$$\omega = a^n b^n$$

mà:

$$y = a^k, \qquad k \geq 1,$$

nên sau khi bỏ y:

$$xy^0 z = a^{n-k} b^n.$$

Bây giờ:

$$n - k \neq n$$

nên số a khác số b.

Do đó:

$$xy^0 z \notin L.$$

Troy University Resources

Bước 5 — Kết luận mâu thuẫn

Nhưng bổ đề bơm nói rằng nếu L chính quy thì phải có:

$$xy^i z \in L$$

với mọi $i \geq 0$.

Ta vừa tìm được:

$$xy^0 z \notin L.$$

$\rightarrow$ mâu thuẫn.

Vậy giả thiết ban đầu sai:

L không phải ngôn ngữ chính quy.

1.7 Phép nối (concatenation) và biểu thức chính quy

Sau phần chứng minh *không chính quy*, ta quay lại cách tạo *ngôn ngữ chính quy*. Cần phân biệt phép toán trên *ngôn ngữ* và cú pháp *biểu thức chính quy* (regular expression) dùng để mô tả chúng.

Phép nối của hai ngôn ngữ

$$A \circ B = \{xy \mid x \in A,\ y \in B\}.$$

Nghĩa là: lấy một xâu x từ A, một xâu y từ B, rồi nối x với y.

Với:

$$A = \{good, bad\}, \qquad B = \{boy, girl\},$$

ta được:

$$A \circ B = \{goodboy, goodgirl, badboy, badgirl\}.$$

Chú ý:

$$A \circ B$$

là phép toán giữa **hai ngôn ngữ**.

Ví dụ:

$$0 \cup 1$$

biểu diễn:

$$\{0, 1\}.$$

Phép nối trong biểu thức chính quy

Cùng thao tác nối đó có thể viết trực tiếp trong biểu thức chính quy: nếu R_1, R_2 là các biểu thức chính quy:

$$R_1 \circ R_2$$

cũng là một biểu thức chính quy.

Ví dụ:

$$0 \circ 1$$

biểu diễn:

$$\{01\}.$$

Còn $\varnothing^*$

Dấu * (*phép đóng Kleene*, Kleene star) cho phép nối *0 hoặc nhiều* xâu thuộc ngôn ngữ ban đầu. Trường hợp dễ nhầm là phép đóng Kleene của ngôn ngữ rỗng.

Slide viết:

$$\varnothing^* = \{\varepsilon\}.$$

Phép đóng Kleene cho phép chọn 0 xâu để nối.

Dù $\varnothing$ không có phần tử nào, ta vẫn có trường hợp lấy 0 lần:

$$\varepsilon.$$

Nên:

$$\boxed{\varnothing^* = \{\varepsilon\}}.$$

1.8 Chuyển biểu thức chính quy (regex) sang NFA

Ta đã biết biểu thức chính quy mô tả một ngôn ngữ; bước tiếp theo là biến mô tả đó thành một máy có thể nhận các xâu thuộc ngôn ngữ ấy. Cách dựng NFA bắt đầu từ vài trường hợp đơn giản rồi ghép các máy nhỏ theo cấu trúc của biểu thức.

Có 6 trường hợp cơ bản trong cách dựng NFA từ biểu thức chính quy:

$$R = a,\ \varepsilon,\ \varnothing,\ R_1 \cup R_2,\ R_1R_2,\ R_1^*.$$

Trong đó 3 trường hợp đầu là **trường hợp cơ sở** (base cases), 3 trường hợp sau là **trường hợp đệ quy** (recursive cases) dựa trên việc ghép các máy nhỏ hơn. Khi ôn, nên nhớ theo hai nhóm này thay vì học thuộc sáu trường hợp rời rạc.

Tổng quát như sau:

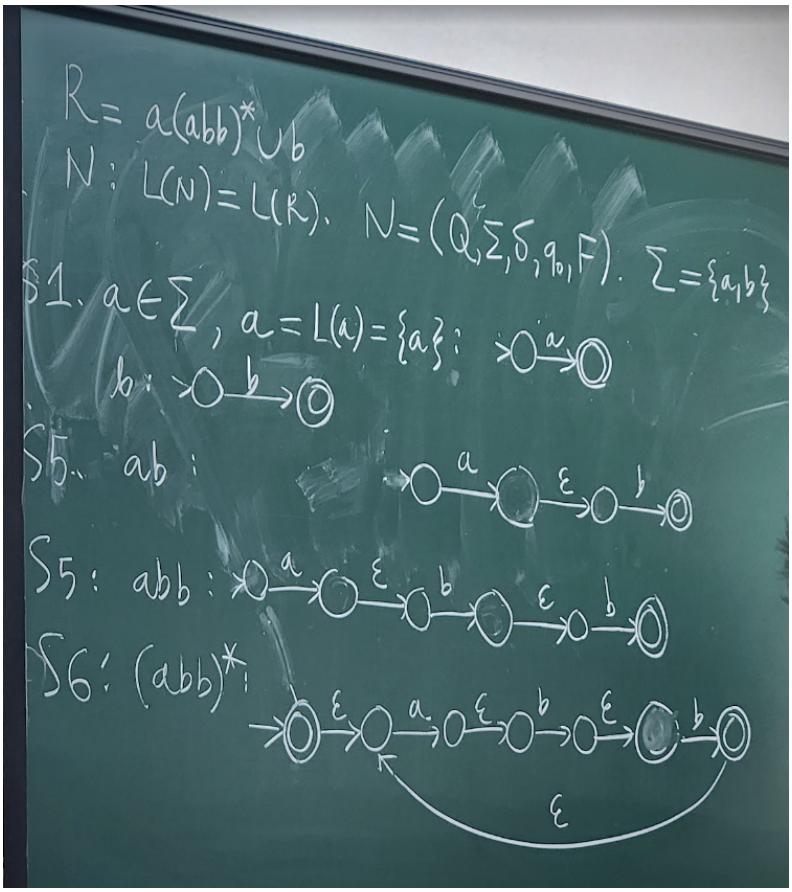
Regex	Cách dựng NFA
a	Một phép chuyển mang nhãn a
ε	Trạng thái bắt đầu cũng là trạng thái chấp nhận
$\varnothing$	Không có đường đi chấp nhận
$R_1 \cup R_2$	Phân nhánh bằng phép chuyển ε
R_1R_2	Nối NFA_1 sang NFA_2 bằng ε
R_1^*	Thêm đường lặp qua ε và đường chấp nhận ε

Ba phép dựng quan trọng: phép hợp (union), phép nối (concatenation) và phép lặp Kleene (star).

Ký hiệu thêm: R^+ là cách viết gọn của RR^*, tức có ít nhất một lần xuất hiện của R (khác R^*, vốn cho phép cả 0 lần).

Troy University Resources

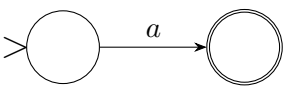
Ví dụ trên bảng. Hình dưới minh họa cách ghép các ôtômat con để đi từ biểu thức $R = a(abb)^* \cup b$ đến một NFA nhận cùng ngôn ngữ. Trước hết, ta xét trường hợp đơn giản nhất: ôtômat cho một ký tự a hoặc b.



Ghi chú trên bảng về dựng regex → NFA: $R = a(abb)^* \cup b$, $L(N) = L(R)$, $N = (Q, \Sigma, \delta, q_0, F)$, $\Sigma = \{a, b\}$; các bước cho a, b, ab, abb và $(abb)^*$.

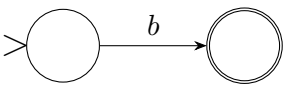
Bước 1: tạo ôtômat cho a

Vì $a \in \Sigma$, biểu thức chính quy a mô tả ngôn ngữ $L(a) = \{a\}$. NFA nhận ngôn ngữ này chỉ cần hai trạng thái và một phép chuyển mang nhãn a:



Đọc a thì đi tới trạng thái chấp nhận (accepting state).

Tương tự với b:



Các bước tiếp theo. Từ NFA cho a và b, ta ghép các máy trung gian *theo bảng dựng NFA phía trên*: nối để tạo ab rồi abb; dùng phép lặp Kleene cho $(abb)^*$; nối với a để được $a(abb)^*$; cuối cùng hợp với b để nhận $R = a(abb)^* \cup b$. Các phép ghép dùng chuyển ε theo bảng; hai sơ đồ trên chỉ minh họa trường hợp cơ sở, chưa phải NFA cuối cùng.

2 Ngôn ngữ phi ngữ cảnh (Context-Free Languages)

Các máy hữu hạn không đủ để ghi nhớ một số quan hệ phụ thuộc dài hạn, ví dụ số chữ a phải bằng số chữ b. Đây là lý do ta chuyển từ ngôn ngữ chính quy sang *ngôn ngữ phi ngữ cảnh* (context-free languages, CFL), một lớp ngôn ngữ rộng hơn.

Tức là:

$$\text{Regular Languages} \subsetneq \text{Context-Free Languages}.$$

Văn phạm phi ngữ cảnh có thể sinh cả những ngôn ngữ mà biểu thức chính quy không biểu diễn được, chẳng hạn $\{a^n b^n \mid n \geq 0\}$.

Ví dụ cực kỳ quan trọng:

$$L = \{a^n b^n : n \geq 0\}.$$

Văn phạm (grammar) là hệ quy tắc sinh; *ngôn ngữ* của văn phạm là tập tất cả các xâu chỉ gồm ký hiệu kết thúc có thể dẫn xuất từ ký hiệu bắt đầu.

> Trong CFG, ký hiệu $\Rightarrow$ biểu diễn *một bước dẫn xuất* (derivation), không phải phép chuyển trạng thái của ôtômat.

2.1 Định nghĩa văn phạm phi ngữ cảnh (CFG)

Để mô tả cách *sinh* xâu trong một ngôn ngữ phi ngữ cảnh, ta dùng *văn phạm phi ngữ cảnh* (context-free grammar, CFG). Văn phạm gồm các ký hiệu không kết thúc, ký hiệu kết thúc, ký hiệu bắt đầu và các luật sinh.

Slide định nghĩa:

$$G = (V, \Sigma, R, S).$$

Có 4 thành phần.

CFG (context-free grammar): bốn thành phần

V	Tập biến, còn gọi là các ký hiệu không kết thúc (nonterminals).
Σ	Tập ký hiệu kết thúc (terminals), tạo nên xâu đầu ra.
R	Tập luật sinh (production rules), ví dụ $S \to 0S1$.
S	Ký hiệu bắt đầu (start variable); $S \in V$.

V: các biến / ký hiệu không kết thúc

Ví dụ:

$$V = \{S, B\}.$$

Đây là các ký hiệu có thể tiếp tục được thay thế theo luật sinh.

Chúng còn được gọi là: **ký hiệu không kết thúc** (nonterminals).

S: ký hiệu bắt đầu

Ký hiệu không kết thúc dùng để bắt đầu phép dẫn xuất.

Ví dụ:

$$S.$$

Troy University Resources

R: tập luật sinh

Là tập các luật sinh (production rules).

Ví dụ:

$$R = \{S \to 0S1,\ S \to B,\ B \to \#\}.$$

Σ: các ký hiệu kết thúc

$$\Sigma = \{0, 1, \#\}.$$

0, 1, # là các ký hiệu kết thúc vì chúng có thể xuất hiện trong xâu đầu ra và không được thay thế bằng luật sinh.

Các luật sinh của CFG chỉ thay thế ký hiệu không kết thúc, không thay thế ký hiệu kết thúc.

2.2 Dạng chuẩn Chomsky (Chomsky Normal Form, CNF)

Cùng một ngôn ngữ có thể được sinh bởi nhiều văn phạm khác nhau. *Dạng chuẩn Chomsky* (Chomsky Normal Form, CNF) chuẩn hóa dạng các luật sinh, hữu ích khi phân tích và thao tác với CFG.

Một CFG thông thường có thể có các luật sinh với nhiều dạng khác nhau:

$$S \to ASA, \qquad A \to B, \qquad B \to \varepsilon.$$

CNF giới hạn dạng luật sinh như sau:

Văn phạm ở dạng chuẩn Chomsky nếu các luật sinh có dạng:

$$A \to BC$$

hoặc:

$$A \to a.$$

Trong đó:

- A, B, C là các ký hiệu không kết thúc.
- a là một ký hiệu kết thúc.

Ngoại lệ:

$$S \to \varepsilon$$

được phép nếu ngôn ngữ chứa xâu rỗng ε. Theo quy ước CNF cho phép ngoại lệ này, ký hiệu bắt đầu S không xuất hiện ở vế phải của bất kỳ luật sinh nào.

2.3 Văn phạm phi ngữ cảnh và ôtômat đẩy xuống (CFG và PDA)

Nếu CFG dùng để *sinh* ngôn ngữ phi ngữ cảnh, thì loại máy nào có thể *nhận* lớp ngôn ngữ này?

Câu trả lời là:

Ôtômat đẩy xuống (pushdown automaton, PDA)

Và ý lớn nhất của toàn bộ Topic 2 là:

CFG ≡ PDA

Ở đây hiểu là văn phạm phi ngữ cảnh và *PDA không đơn định* (nondeterministic PDA) sinh/nhận cùng lớp ngôn ngữ phi ngữ cảnh, chứ không phải chúng có cấu trúc hoặc cách chạy giống nhau.

So sánh	**CFG**	**PDA không đơn định**
Cách nhìn	Sinh xâu bằng luật sinh	Nhận xâu bằng trạng thái và ngăn xếp
Cấu trúc chính	Ký hiệu không kết thúc, kết thúc, luật sinh	Trạng thái, phép chuyển, ngăn xếp
Kết quả	$L(G)$: tập xâu được sinh	$L(M)$: tập xâu được nhận

2.4 Định nghĩa PDA bằng bộ 6 thành phần

Điểm mới so với ôtômat hữu hạn là PDA có thêm *ngăn xếp* (stack) để lưu thông tin tạm thời. Vì vậy, ngoài trạng thái và ký hiệu đầu vào, phép chuyển còn có thể xét ký hiệu trên đỉnh ngăn xếp.

Slide định nghĩa:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F).$$

Có 6 thành phần.

PDA (pushdown automaton): sáu thành phần

Q	Tập hữu hạn các trạng thái.
Σ	Bảng chữ cái đầu vào.
Γ	Bảng chữ cái ngăn xếp (stack alphabet).
δ	Hàm chuyển trạng thái, xét đầu vào và đỉnh ngăn xếp.
q_0	Trạng thái bắt đầu.
F	Tập các trạng thái chấp nhận.

Định nghĩa bộ 6 thành phần này theo quy ước của Sipser. Một số giáo trình dùng bộ 7 thành phần khi quy định thêm ký hiệu đáy ngăn xếp ban đầu; hai cách định nghĩa không nên trộn lẫn khi làm bài.

Γ: bảng chữ cái ngăn xếp

Đây là cái mới.

Γ là tập các ký hiệu được phép lưu trong ngăn xếp.

Ví dụ:

$$\Gamma = \{a, b\}.$$

Không nhất thiết:

$$\Gamma = \Sigma.$$

Hai alphabet có thể khác nhau.

Ví dụ input là:

$$\Sigma = \{0, 1\}$$

nhưng ngăn xếp có thể dùng:

$$\Gamma = \{X, \$\}.$$

Troy University Resources

2.5 Phép chuyển trạng thái và cấu hình của PDA

$$\delta(\text{trạng thái},\ \text{ký hiệu vào},\ \text{đỉnh ngăn xếp})$$
$$\downarrow$$
$$(\text{trạng thái mới},\ \text{thao tác ngăn xếp}).$$

Một phép chuyển (transition) trong slide có dạng:

$$((p, a, \beta), (q, \gamma)).$$

Hiểu là: ở trạng thái p, đọc a (hoặc không đọc ký hiệu nếu $a = \varepsilon$), xử lý β ở đỉnh ngăn xếp, chuyển sang q và thay β bằng γ. Với dạng 6 thành phần nêu trên, $\beta, \gamma \in \Gamma \cup \{\varepsilon\}$.

Có ba thao tác ngăn xếp cơ bản: **push** (đẩy thêm ký hiệu lên đỉnh), **pop** (lấy ký hiệu khỏi đỉnh), và **không thay đổi** ngăn xếp. Cách viết $\beta \to \gamma$ trong mô hình ở đây biểu diễn việc đọc/xóa ký hiệu đỉnh β rồi ghi γ (mỗi bên có thể là ε). Nếu giáo trình dùng biến thể cho phép thay đỉnh bằng cả một xâu, cần đọc theo định nghĩa riêng của biến thể đó.

Hình dung nhanh: với ngăn xếp có đỉnh ở bên trái, từ `ba` có thể *push* `c` thành `cba`, hoặc *pop* `b` thành `a`. Nếu không sửa ngăn xếp thì vẫn là `ba`. Đây là ví dụ thao tác, không phải phép chuyển cụ thể của PDA đang xét.

Một *cấu hình* (configuration) có thể biểu diễn bằng bộ ba:

$$(\text{trạng thái},\ \text{đầu vào chưa đọc},\ \text{nội dung ngăn xếp}).$$

Ví dụ:

$$(q_1, bba, bba)$$

nghĩa là:

- máy đang ở trạng thái q_1;
- đầu vào chưa đọc là `bba`;
- nội dung ngăn xếp hiện tại là `bba`.

Tài liệu quy ước nội dung ngăn xếp được viết:

đỉnh ngăn xếp được viết bên trái (top-first)

Cho nên trong:

`bba`

ký tự `b` đầu tiên là **đỉnh ngăn xếp** (stack top).

3 Bản đồ khái niệm và thuật ngữ

Tóm lại, có hai cách nhìn tương ứng: *Biểu thức chính quy/DFA/NFA* dùng để mô tả hoặc nhận ngôn ngữ chính quy; còn *CFG/PDA* là cặp công cụ tương ứng cho ngôn ngữ phi ngữ cảnh. Ngôn ngữ chính quy là một lớp con thực sự của lớp ngôn ngữ phi ngữ cảnh.

Lớp ngôn ngữ	**Cách mô tả**	**Máy nhận**
Ngôn ngữ chính quy	Biểu thức chính quy (regex)	DFA hoặc NFA
Ngôn ngữ phi ngữ cảnh	Văn phạm phi ngữ cảnh (CFG)	PDA không đơn định

Ví dụ $L = \{a^n b^n \mid n \geq 0\}$ là ngôn ngữ phi ngữ cảnh nhưng không chính quy. Vì vậy, **mọi ngôn ngữ chính quy đều là ngôn ngữ phi ngữ cảnh, nhưng chiều ngược lại không đúng**.

Troy University Resources