Mục lục

CHƯƠNG 0

Một số kiến thức Toán học cơ sở

I. Lôgich

I.1. Khái niệm lôgích

Lôgích được sử dụng khi thực hiện các phép suy luận toán học (reasonin). Người ta phân biệt hai mặt của lôgích :

- Mặt cú pháp (syntaxe) chỉ ra các thao tác hình thức (formal manipulation) trên các ký hiệu.
- Mặt ngữ nghĩa (semantic) cho biết ý nghĩa sử dụng (meaning) khi sắp đặt các ký hiệu.

Từ đó người ta xây dựng các mô hình lôgích (lôgích model) dựa trên một ngôn ngữ các ký hiệu và một số các quy tắc thao tác, hay các luật.

Chẳng hạn về mặt cú pháp, *3+4* là một biểu thức toán học, về mặt ngữ nghĩa, đó là một phép cộng cho kết quả là *7*.

Ví dụ một chương trình máy tính là một dãy các ký hiệu, một ngôn ngữ lập trình là một dãy các quy tắc cú pháp cho phép sắp đặt các ký hiệu này. Một chương trình phải tuân thủ theo quy tắc cú pháp và phải có một nghĩa sử dụng nhất định (ngữ nghĩa) để giải bài toán.

Cơ sở để xây dựng môn học lôgích là *mệnh đề* (proposition). Mệnh đề lôgích là một phát biểu (câu) nào đó, xét trong một hoàn cảnh thời gian và không gian nào đó, chỉ nhận một trong hai giá trị *đúng* (true) hoặc *sai* (false), mà không thể vừa đúng vừa sai. Giá trị đúng sai được gọi là các *chân giá trị* (truth value).

Ví dụ I.1 :

Mệnh đề lôgích	Giải thích
$y > x + 1$	Tuỳ theo giá trị của x và y mà có giá trị đúng hoặc sai. chẳng hạn $x=1$ và $y=3$ thì có giá trị đúng
Hôm nay trời mưa !	Đúng nếu thời điểm nói trời mưa, sai nếu không phải.
$2 + 3 = 5$	Luôn luôn có giá trị đúng.
Luânđô là thủ đô của nước Đức	Luôn luôn có giá trị sai.
Hôm nay là ngày mấy ?	Câu hỏi không phải là một mệnh đề.
Mời anh vào đây !	Câu mệnh lệnh cũng không phải là một mệnh đề, v.v...

Các phép tính lôgích

Cho trước các mệnh đề lôgích p, q, r có thể nhận giá trị đúng hoặc sai, ta có các phép tính lôgích như sau :

Phép *phủ định* (not) hay phép đối của p, ký hiệu $\neg p$, có giá trị sai nếu p đúng và có giá trị đúng nếu.

Phép *và* (and) hay nhân lôgích của p và q, ký hiệu $p \wedge q$, có giá trị đúng khi và chỉ khi cả p và cả q đều có giá trị đúng.

Phép *hoặc* (or) hay cộng lôgích của p và q, ký hiệu $p \vee q$, có giá trị sai khi và chỉ khi cả p và cả q đều có giá trị sai.

Phép *kéo theo* (implication) hay phép suy ra, ký hiệu $p \Rightarrow q$, chỉ có giá trị sai khi p đúng và q sai, còn lại đều đúng.

Phép *tương đương* (equivalence) của p và q, ký hiệu $p \Leftrightarrow q$, có giá trị đúng khi cả p và q đều đúng hoặc đều sai, có giá trị sai khi hoặc p sai, q đúng, hoặc p đúng, q sai.

Biểu diễn quy ước giá trị đúng là 1, giá trị sai là 0, ta có bảng chân giá trị của các phép tính lôgích như sau :

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \Rightarrow q$	$p \Leftrightarrow q$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

I.2. Các tính chất

Cho trước các mệnh đề lôgích p, q, r, ta có các tính chất như sau :

$\neg(\neg p) = p$

$p \wedge 1 = p$ $\quad p \wedge 0 = 0$ $\quad p \wedge \neg p = 0$

$p \wedge q = q \wedge p$ $\quad p \wedge (q \wedge r) = (q \wedge p) \wedge r = p \wedge q \wedge r$

$p \vee 1 = 1$ $\quad p \vee 0 = p$ $\quad p \vee \neg p = 1$

$p \vee q = q \vee p$ $\quad p \vee (q \vee r) = (q \vee p) \vee r = p \vee q \vee r$

Định luật De Morgan :

$\neg (p \vee q) = \neg p \wedge \neg q$

$\neg (p \wedge q) = \neg p \vee \neg q$

Tính chất phân phối :

$p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$

$p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r)$

Chuyển đổi các phép kéo theo hoặc tương đương :

$p \Leftrightarrow q = (p \Rightarrow q) \wedge (q \Rightarrow p)$

$p \Rightarrow q = \neg p \vee q = (\neg q) \Rightarrow (\neg p)$

$(\neg p) \Rightarrow 0 = p$

Trong phép kéo theo $p \Rightarrow q$, ta gọi p là giả thiết, q là kết luận.

Phép $q \Rightarrow p$ được gọi là phép đảo của $p \Rightarrow q$. Ta có thể diễn tả phép kéo theo $p \Rightarrow q$ bằng các cách sau :

- Muốn có q, cần có p là đủ, hoặc nếu p thì q.

- p là điều kiện đủ để có q, q là điều kiện cần để có p.

Phép tương đương $p \Leftrightarrow q$ có thể diễn tả như sau :

- Muốn có p, cần và đủ phải có q, hoặc p là điều kiện cần và đủ để có q.
- p khi và chỉ khi q, hoặc p nếu và chỉ nếu q.

Phép suy luận phản chứng

Để chứng minh mệnh đề $p \Rightarrow q$ là đúng, ta giả thiết rằng p đúng, q sai, sau đó cần chứng minh rằng điều này dẫn đến mâu thuẫn. Bởi vì ta đã chứng minh được mệnh đề $(p \wedge (\neg q)$ sai, tức là mệnh đề $(\neg p \vee q)$ đúng, tức là $p \Rightarrow q$ đúng.

I.3. Biểu thức lôgích

Kết hợp các mệnh đề lôgích và các phép toán lôgích một cách tuỳ ý, ta nhận được các biểu thức lôgích. Thứ tự thực hiện các phép tính lôgích theo độ ưu tiên lần lượt là phép phủ định, phép và, phép hoặc, phép kéo theo và cuối cùng là phép tương đương. Có thể thêm các cặp dấu ngoặc () vào một biểu thức lôgích để thay đổi thứ tự thực hiện các phép tính hoặc để muốn dễ đọc. Nếu một biểu thức lôgích có chứa các cặp dấu ngoặc lồng nhau thì thứ tự thực hiện là từ trong ra ngoài, từ trái qua phải.

Để tính giá trị của biểu thức, người ta thường lập bảng chân lý.

Ví dụ I.2 :

Tính giá trị biểu thức $E(p, q) = (p \wedge \neg q) \vee (\neg p \wedge q)$:

p	q	$\neg p$	$\neg q$	$p \wedge \neg q$	$\neg p \wedge q$	$(p \wedge \neg q) \vee (\neg p \wedge q)$
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	1	0	1
1	1	0	0	0	0	0

Kết quả của biểu thức là cột cuối cùng, chẳng hạn $E(0, 0) = 0$, $E(0, 1) = 1$, v.v...

II. Tập hợp

II.1. Biểu diễn tập hợp

Tập hợp (set) là một nhóm hay một bộ sưu tập các đối tượng[1] phân biệt, chúng được gọi là các *phần tử* (elements) của tập hợp. Do các phần tử của một tập hợp không được sắp xếp thứ tự nên người ta không nói đến phần tử thứ nhất, phần tử thứ hai, v.v...

Trong hoàn cảnh đang xét nào đó, tập hợp tất cả các phần tử có cùng bản chất được gọi là *tập hợp vũ trụ* (universal set), giả sử là tập hợp U. Khi cho trước một tập hợp Snào đo, người ta xem rằng các phần tử thuộc S cũng đều thuộc U. Trong giáo trình, các tập hợp được đặt tên bằng các chữ cái hoa S, A, B,..., các phần tử được đặt tên bằng các chữ cái thường *a*, *b*, x, y,... Các phần tử thuộc một tập hợp được đặt trong một cặp dấu ngoặc { }. Chẳng hạn sau đây là các tập hợp :

A = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

B = { ♣, ♦, ♥, ♠ }

Tập hợp không chứa phần tử nào gọi là *tập hợp rỗng* (empty set), ký hiệu ∅. Người ta viết A = ∅ hay A = { }. Một tập hợp chỉ có một phần tử duy nhất x thì x được gọi là đơn tử và ký hiệu { x }.

Người ta gọi *bản số* (cardinality) của một tập hợp là số phần tử thuộc tập hợp đó, ký hiệu |A | hoặc card(A). Chẳng hạn các tập hợp trên có |A| = 10, |B| = 4. Bản số của một tập hợp rỗng bằng 0.

Cho tập hợp A gồm các phần tử x. Ta nói phần tử x thuộc A, ký hiệu $x \in A$, phần tử x không thuộc A, ký hiệu $x \notin A$, dĩ nhiên $x \in U$.

Khái niệm về lượng tử

Lượng tử phổ cập ∀ đọc là «với mọi» hay «với mỗi».

Lượng tử tồn tại ∃ đọc là «tồn tại ít nhất một phần tử».

Ký hiệu ∃! có nghĩa là « tồn tại một và chỉ một phần tử».

Tên biến (đối tượng) do một lượng tử tác động đến có thể lấy bất kỳ :

$\forall x, x \in A, P(x) \Leftrightarrow \forall y, y \in A, P(y)$ mọi phần tử của A có tính chất P

$\exists x, x \in A, P(x) \Leftrightarrow \exists y, y \in A, P(y)$ tồn tại một phần tử của A có tính chất P

Phép phủ định một mệnh đề lượng tử hoá như sau :

$\neg (\forall x, x \in A, P(x)) \Leftrightarrow \exists x, x \in A, \neg P(x)$

$\neg (\exists x, x \in A, P(x)) \Leftrightarrow \forall x, x \in A, \neg P(x)$

Để biểu diễn một tập hợp A gồm một số hữu hạn phần tử, ta có thể liệt kê hết các phần tử của A như ví dụ trên đây. Trong trường hợp A có vô hạn phần tử, người ta không thể liệt kê hết các phần tử của A, mà dùng cách biểu diễn tính chất (property) của các phần tử, có dạng :

A = { x | P(x) } là tập hợp các phần tử x sao cho x thoả mãn tính chất P.

Ví dụ II.1 :

---

[1] Khái niệm đối tượng có tính trực giác, do nhà Tóan học Đức G. Cantor đưa ra từ năm 1985. Lý thuyết tập hợp đã dẫn đến những nghịch lý tóan học (paradox) hay mâu thuẫn lôgích được nhà triết học người Anh B. Russell chỉ ra năm 1902. Ví dụ

M = { i | i nguyên dương và ∃j nguyên dương sao cho i = 2 * j}

hay có thể viết gọn hơn :

M = { i | i ∈ N và ∃j ∈ N sao cho i = 2 * j }.

II.2. Quan hệ giữa các tập hợp

Cho A, B là các tập hợp (có các phần tử thuộc một tập hợp vũ trụ U nào đó). Người ta xây dựng các phép toán trên tập hợp được như sau :

$A \subseteq B$ A *nằm trong* B, hay A là một *bộ phận* của B, hay A là *tập hợp con* (subset) của B, nếu thoả mãn $\forall x (x \in A \Rightarrow x \in B)$.

Khi A nằm hoàn toàn trong B, người ta viết $A \subset B$.

Trường hợp ngược lại, người ta viết $A \supseteq B$, hay $A \supset B$.

$A = B$ A bằng B nếu A và B có cùng các phần tử như nhau :
$\forall x (x \in A \Rightarrow x \in B)$ và $\forall x (x \in B \Rightarrow x \in A)$, hay $A \subseteq B$ và $B \subseteq A$.

$A \neq B$ A khác B nếu A và B rời nhau hoặc không có cùng các phần tử.

II.3. Các phép toán trên tập hợp

Cho A, B là các tập hợp. Người ta xây dựng các phép toán như sau :

Ký hiệu	*Ý nghĩa*
$A \cup B$	hợp của A và B, là tập hợp $\{x \mid x \in A \vee x \in B\}$
$A \cap B$	giao của A và B, là tập hợp $\{x \mid x \in A \wedge x \in B\}$

Nếu |A|= m, |B| = n, thì $|A \cup B| = m + n - |A \cap B|$ phần tử.

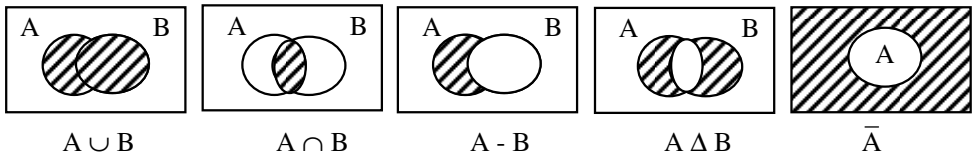
A - B	hay A\B, hiệu của A và B, là tập hợp $\{x \mid x \in A \wedge x \in B\}$
$A \Delta B$	hiệu đối xứng của A và B, là tập hợp A-B ∪ B-A
$\overline{A}$	bù của tập hợp A là tập hợp $\{x \mid x \in U \wedge x \notin A\}$, hay U - A, với U là tập hợp vũ trụ.
$A \times B$	tích Đêcac (Cartesian product) của A và B, gồm các cặp phần tử có thứ tự (*a*, *b*), $a \in A$ và $b \in B$.

Nếu |A|= m, |B| = n, thì $|A \times B| = m \times n$ phần tử.

2^A hay $\wp(A)$, là tập lũy thừa của A, hay tập các tập hợp con của A.
Ta có $|2^A| = 2^{|A|} = 2^m$, nếu |A|= m.

Chú ý $\wp(\varnothing) = \{ \varnothing \}$, $\wp(\{\varnothing\}) = \{ \varnothing, \{ \varnothing \} \}$.

Các phép toán ∪, ∩, hiệu, hiệu đối xứng và bù trên tập hợp được minh hoạ bởi các giản đồ Venn như hình dưới đây.



Hình II.1 Giản đồ Venn biểu diễn các phép toán cơ bản trên tập hợp.
Chú ý hình chữ nhật biểu diễn tập hợp vũ trụ.

Ví dụ II.2 :

Cho A = {*a*, *b*}, B = {*b*, c} với U là tập hợp các chữ cái tiếng Anh *a*..z. Khi đó :

$A \cup B = \{a, b, c\}$
$A \cap B = \{b\}$
$A - B = \{a\}$
$A \Delta B = \{a, c\}$
$\overline{A} = \{ c..z \}$
$A \times B = \{(a, b), (a, c), (b, b), (b, c)\}$
$2^A = \{\varnothing, \{a\}, \{b\}, \{a, b\}\}$

Một số tính chất của các phép toán trên tập hợp

Cho A, B và C là *ba* tập hợp.

Tính chất giao hoán (commutative) :

$A \cup B = B \cup A$ $A \cap B = B \cap A$

Tính chất kết hợp (associative) :

$(A \cup B) \cup C = A \cup (B \cup C)$ $(A \cap B) \cap C = A \cap (B \cap C)$

Tính chất phân phối (distributive) :

$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ $(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$

$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ $(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$

Luật DeMorgan :

$\overline{A \cup B} = \overline{B} \cap \overline{A}$ $\overline{A \cap B} = \overline{B} \cup \overline{A}$

II.4. Ánh xạ

Cho A, B là hai tập hợp và tích Đêcac $A \times B$. Xét f là một tập hợp con, $f \neq \varnothing$, của tích Đêcac $A \times B$, khi đó f được gọi là một *ánh xạ* (mapping), hay còn được gọi là một *đồ thị hàm*, từ A vào B như sau :

$$f : A \to B$$

Ta viết : $y = f(x)$, $y \in B$, $x \in A$, y là ảnh của x. Từ ánh xạ f, ta có thể xây dựng được quan hệ R(x, y) giữa các phần tử $x \in A$ và $y \in B$, sao cho :

$R(x, y) = \{ (x, y) \mid x \in A \Rightarrow \exists! \, y \in B, y = f(x) \}$

Anh xạ f được gọi là :

- *Toàn ánh,* hay ánh xạ lên (surjection), nếu $f(x) = y$ tức là :
  $\forall y \in B \Rightarrow \exists x \in A$ sao cho $y = f(x)$
- *Đơn ánh* hay phép nhúng (injection), nếu $f(x) = f(x') \Rightarrow x = x'$
- *Song ánh* hay xánh xạ 1 - 1, là phép đặt lên (bijection) nếu f vừa là toàn ánh, vừa là đơn ánh.

II.5. Tính đếm được của các tập hợp vô hạn

Cho A là một tập hợp. Nếu A có hữu hạn phần tử thì A được gọi là *đếm được* hay liệt kê được (enumerable).

Cho A, B là hai tập hợp đếm được. Ta nói A và B là có cùng bản số nếu $\exists$ song ánh giữa chúng : $card(A) = card(B)$. Nếu A là tập hợp con thực sự của B thì ta có $card(A) < card(B) \leq K < \infty$, với K là một hằng số nào đó.

Trong trường hợp A có vô hạn phần tử thì có thể xảy ra A đếm được hoặc không đếm được. Để chứng minh một tập hợp đã cho là vô hạn đếm được, chỉ cần xây dựng song ánh giữa A và tập hợp các số tự nhiên N.

Ví dụ : Cho A = {i ∈ N | i chẵn}, B = N. Khi đó, card(A) = card(B) vì tồn tại ánh xạ 1-1 giữa A và B.

Trong trường hợp này, để xác định xem khi nào thì hai tập hợp có cùng kích thước (có cùng số phần tử), người ta đặt tương ứng từng cặp phần tử của hai tập hợp này.

Nói cách khác, tồn tại một song ánh giữa hai tập hợp (hay ánh xạ 1.1).

Ví dụ 1.7 :

Các tập hợp { 0, 1, 2, 3 }, { *a*, *b*, c, d } và { α, β, γ, δ } có cùng kích thước. Chẳng hạn có thể đặt tương ứng từng cặp phần tử như sau :

{ (0, α), (1, β), (2, γ), (3, δ) }.

Một tập hợp vô hạn là liệt kê được và đếm được nếu tồn tại một song ánh giữa tập hợp này và tập hợp các số tự nhiên.

Bản số của các tập hợp đếm được thường được ký hiệu $\aleph$ ($\aleph$ là chữ cái đầu của bảng chữ Do thái, đọc là *aleph*).

Ví dụ 1.8 :

Tập hợp các số chẵn là đếm được nhờ có song ánh :

{ (0, 0), (2, 1), (4, 2), (6, 3), . . . }

có thể suy ra rằng mọi tập hợp con vô hạn các số tự nhiên là đếm được.

1. Các số hữu tỉ là đếm được. Thực vậy, mỗi số hữu tỉ được viết dưới dạng $\frac{a}{b}$ với $b \neq 0$ và giữa *a* và *b* không có ước số chung. Ta sẽ phân loại các số như vậy theo thứ tự của tổng *a*+*b* tăng dần. Ta có song ánh :

$$\{ (\frac{0}{1}, 0), (\frac{1}{1}, 1), (\frac{1}{2}, 2), (\frac{2}{1}, 3), (\frac{1}{3}, 4), (\frac{3}{1}, 5), \ldots \}$$

Tập hợp các câu trên bảng chữ cái { *a*, *b* } là đếm được. Để nhận được phép song ánh, người ta phân loại các câu theo thứ tự tăng dần của độ dài. Với các câu có cùng độ dài, người ta săp xếp chúng theo thứ tự từ vựng (theo thứ tự trong từ điển). Ta có song ánh : { (ε, o), (*a*, 1), (*b*, 2), (aa, 3), (*ab*, 4), (*ba*, 5), (bb, 6), . . . }.

2. Các biểu thức chính qui là đếm được. Thật vậy, chúng là các xâu ký tự trên một bảng chữ hữu hạn. Theo ví dụ 3 ở trên, tập hợp các câu trên một bảng chữ là đếm được. Các biểu thức chính qui là một tập hợp con vô hạn của các xâu ký tự và chúng cũng là đếm được theo lý luận ở 1.

Từ những ví dụ trên ta có thể tự đặt câu hỏi : có phải mọi tập hợp vô hạn đều có bản số $\aleph_o$? Không phải vì có những tập hợp vô hạn có bản số lớn hơn $\aleph_o$, chẳng hạn tập hợp tất cả các tập hợp con của một tập hợp đếm được. Ta có định lý sau :

Định lý 1.2 :

Tập hợp tất cả các tập hợp con của một tập hợp đếm được cho trước là không đếm được.

Chứng minh : Ta sử dụng kỹ thuật chéo hóa (Diagonalisation) như sau :

Giả sử cho A là một tập hợp đếm được :

A = { a_0, a_1, a_2, ... }

Gọi S là tập hợp tất cả tập hợp con của A. Giả sử rằng S đếm được và do đó,

S = { s_0, s_1, s_2, ... }

ta xây dựng bảng vô hạn như hình dưới đây. Bảng chỉ ra những phần tử nào của A thì thuộc về mỗi phần tử của S. Mỗi hàng của bãng tương ứng với một phần tử của S và gồm một dấu chéo (×) tại cột ttương ứng với a_i nếu $a_i \in s_j$.

	a_0	a_1	a_2	a_3	a_4	...
s_0	×	×		×		
s_1	×	□		×		
s_2		×	×		×	
s_3	×		×	□		
s_4		×		×	□	
...						

Như vậy, bảng này chỉ là sự biểu diễn đồ thị của nội dung các tập hợp s_j.

Bây giờ xét tập hợp $D = \{ a_i \mid a_i \notin s_j \}$

Tập hợp này được định nghĩa bởi ký hiệu nằm trên đường chéo chính của bảng. Dễ thấy rằng đây là tập hợp con của A : nó chứa mọi phần tử a_i của A sao cho ký hiệu □ nằm tại giao điểm của đường chéo chính của bảng với cột a_i.

Như vậy tập hợp D tồn tại nhưng nó không thể là một trong những tập hợp s_i. Thật vậy, giả sử rằng $D = s_k$. Điều này là không thể vì $a_k \in D$ nếu và chỉ nếu $a_k \notin s_k$. Như vậy nảy sinh mâu thuẫn. Điều này cũng mâu thuẫn với giã thiết ở đầu bước chứng minh là tập hợp các tập hợp con của A là đếm được □.

III. Các quan hệ trên tập hợp

III.1. Khái niệm

Cho A, B là hai tập hợp không nhất thiết khác nhau, một *quan hệ* R (hai ngôi) giữa A và B là tập hợp các cặp (a, b), $a \in$ A, $b \in$ B. Người ta viết :

$(a, b) \in$ R, hay a R b.

Nếu A = B, ta nói đó là quan hệ trên tập hợp A. Cho R là quan hệ trên A, lúc đó quan hệ R có các tính chất sau :

- *Phản xạ* (reflection), nếu $\forall a \in$ A : a R a.
- *Bất phản xạ* (non-reflection), nếu $\forall a \in$ A : a R a sai.
- *Truyền ứng*, hay *bắc cầu* (transitive), nếu a R b và b R c $\Rightarrow a$ R c.
- *Đối xứng* (symmetry), nếu a R $b \Rightarrow b$ R a.
- *Phản đối xứng* (non-symmetrical), nếu a R b kéo theo b R a sai.

Chú ý rằng mọi quan hệ phản đối xứng đều phải là bất phản xạ.

Ví dụ 6.2 :

Cho tập hợp các số tự nhiên N. Các quan hệ :

- bằng nhau (=) có các tính chất phản xạ, đối xứng và bắc cầu.
- nhỏ hơn (<) có các tính chất bắc cầu và phản đối xứng (và bất phản xạ).

III.2. Các quan hệ tương đương

Quan hệ R trên tập hợp A được gọi là *tương đương* nếu R có các tính chất phản xạ, đối xứng và bắc cầu.

Tính chất của quan hệ tương đương : Nếu R tương đương trên A, thì R sẽ phân hoạch tập hợp A thành các lớp tương đương không rỗng và rời nhau :

$$A = A_1 \cup A_2 \cup ...$$

trong đó, $\forall i, j, i \neq j$:

1. $A_i \cap A_j = \varnothing$
2. $\forall a, b \in A_i$: a R b đúng.
3. $\forall a \in A_i, \forall b \in A_j$, A R b sai.

Ví dụ 6.3 :

Cho A = N . Xét quan hệ tương đương là đồng dư modulo p, với $P \in \mathbb{Z}$

Ta viết $i \equiv_p j$ hay $i \equiv j \bmod p$ nếu $i, j \in \mathbb{Z}$sao cho i - j chia hết cho p.

Dễ thấy rằng quan hệ đồng dư có tính chất phản xạ, bắc cầu và đối xứng. Ta xây dựng được p lớp đồng dư modulô p như sau :

{ ... - p, 0, p, 2p, ... }

{ ... - (p-1), 1, p + 1, 2p + 1, ... }

{ ... - 1, p - 1, 2p - 1, 3p - 1, ... }

III.3. Bao đóng của quan hệ

Cho tập W gồm các tính chất nào đó của quan hệ R. Ta gọi *bao đóng* W của quan hệ R là quan hệ bé nhất R' bao gồm R và có tính chất trong W.

Ví dụ 6.4 : Bao đóng truyền ứng của R, trở bởi R^+ được định nghĩa một cách đệ quy (recursive definiting) như sau :

1. Nếu $(a, b) \in$ R thì $(a, b) \in R^+$.
2. Nếu $(a, b) \in R^+$ và $(b, c) \in$ R thì $(a, c) \in R^+$.
3. Không còn cặp nào khác trong R^+ theo cách khác nhờ khái niệm lũy thừa R^i được định nghĩa đệ quy như sau :
   - a R^1 B khi và chỉ khi a R b
   - a R^i B khi và chỉ khi $\exists$ c sao cho a R c và c R^{i-1} b với i > 1.

Tập R^+ được xác định như sau :

$$R^+ = R^1 \cup R^2 \cup ...$$

ta có thể nói a R^+ b khi và chỉ khi a R^i b, $\forall i \geq 1$.

Ngoài ra, người ta định nghĩa a R^0 b khi và chỉ khi $a = b$. Do vậy bao đóng phản xạ và bắc cầu của R, trở bởi R^*, là $R^0 \cup R^+$.

Ví du 6.5 :

Cho R = { $(a, b), (b, b), (b, c)$} là quan hệ trên tập hợp A = {$a, b,$ c}.

Khi đó :

$R^+ = \{(a, b), (b, b), (b, c), (a, c)\}$

$R^* = \{(a, a), a, b), (a, c), (b,b), (b, c), (c, c)\}$

IV. Chứng minh quy nạp

Giả sử cần chứng minh mệnh đề P(n) đã cho đúng với mọi số nguyên không âm n, ta sử dụng nguyên lý quy nạp toán học (Mathematical Induction) qua hai bước như sau :

(1) Chứng minh P(0) đúng (thử với giá trị n = 0).

(2) Nếu P(n-1) đúng kéo theo P(n) đúng với $n \geq 1$.

Bước (1) được gọi là cơ sở quy nạp.

Bước(2) được gọi là bước quy nạp, với P(n-1) là giả thiết quy nạp.

Ví dụ 6.6 : Phép quy nạp định nghĩa các số tự nhiên :

1. 0 là số tự nhiên, $0 \in N$

2. Nếu $n \in N$ thì $n + 1 \in N$

Ví dụ 6.7 : Chứng minh bằng quy nạp rằng :

$$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

(1) Cơ sở quy nạp : thay n = 0 trong vế phải, ta thấy cả hai vế đều bằng 0.

(2) Bước quy nạp : Thay n - 1 cho n trong vế phải để có giả thiết quy nạp để từ đó suy ra điều phải chứng minh.

$$\sum_{i=0}^{n-1} i^2 = \frac{(n-1)n(2n-1)}{6} \quad \Rightarrow \quad \sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

Ta thấy rằng $\sum_{i=0}^{n} i^2 = \sum_{i=0}^{n-1} i^2 + n^2$

Sử dụng giả thiết quy nạp. Ta phải chứng minh :

$$\frac{(n-1)n(2n-1)}{6} + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Đẳng thức sau cùng được kiểm chứng nhờ một vài biến đổi đại số đơn giản. Từ đó suy ra điều phải chứng minh.

V. Đồ thị và cây

V.1. Định nghĩa đồ thị

Một *đồ thị* (graph) được biểu diễn bởi bộ 3 G = (V, E, I), trong đó :

- V là tập hợp hữu hạn các nút (hay còn gọi là đỉnh).
- E là tập hợp hữu hạn các cung (hay còn gọi là cạnh) là các cặp nút.
- I là quan hệ giữa V và E, là tập con của tích Đêcac V × E × V, còn được gọi là *quan hệ tới*. Nếu V = ∅ thì đương nhiên E = ∅.

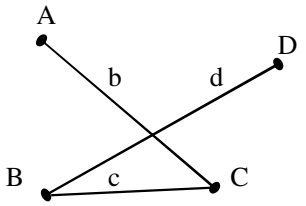
Ví dụ 6.8 : Cho đồ thị G như sau :

G : V = {A, B, C, D} tập các đỉnh

E = {*b*, c, d} tập các cạnh

I = { AbC, CcB, BdD } tập các quan hệ tới.

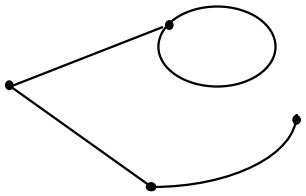
Đặc điểm của quan hệ tới : G = <V, E, I>



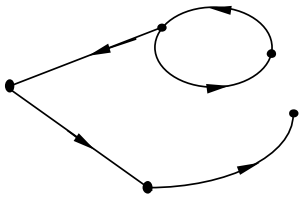
Nếu XeY phân biệt với XeY trong I thì G được gọi là *đồ thị có định hướng*, nếu không G được gọi là *đồ thị vô hướng*.

Cũng có thể định nghĩa G là đồ thị có định hướng nếu các cạnh của G đều có hướng (đi theo một chiều).

Ví dụ 6.9 :



Đồ thị vô hướng



Đồ thị có định hướng

Người ta thường định nghĩa đơn giản G = <V, E>.

Một đường đi (Path) trên đồ thị G là dãy các nút V_1, V_2, ..., V_k, trong đó k ≥ 1 sao cho ∀i, 1 ≤ i < k, ∃ một cạnh $e_i = (V_i, V_{i+1})$

Độ dài của đường đi là k-1. Nếu $V_1 = V_k$ thì đường đi được gọi là chu trình (Circuit).

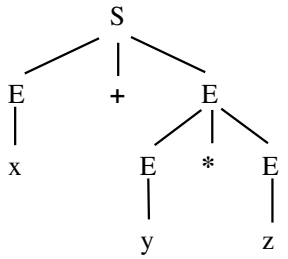
Nếu G là đồ thị có định hướng thì nếu V và W là các nút và V → W là một cung, V là nút trước của W, W là nút sau của V.

V.2. Cây (Tree)

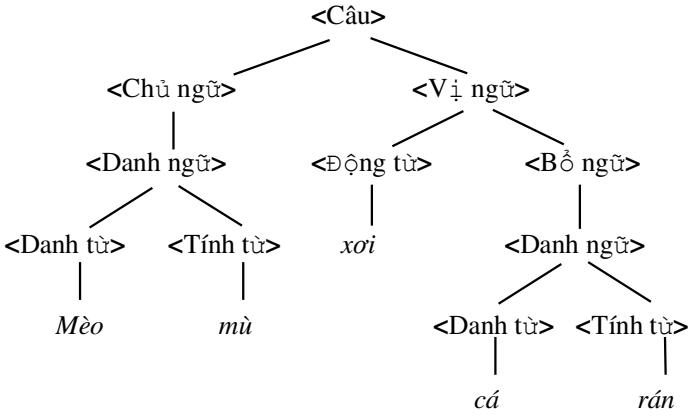
Cây là một đồ thị định hướng, trong đó mỗi đỉnh luôn được gọi là *nút,* có các tính chất sau đây :

1. Có một nút ở trên cùng, gọi là nút gốc (Root), không tồn tại nút trước (ở trên) của nó và có đường đi từ nút gốc tới tất cả các nút khác của cây.
2. Mỗi nút khác nút gốc có đúng một nút trước nó.
3. Các nút sau của một nút được sắp thứ tự (trái qua phải).

Ví dụ 6.9 : Cây sau đây biểu diễn câu x + y * z :



Ví dụ 6.10 : Cây sau đây biểu diễn câu "Mèo mù xơi cá rán" :



Trong cây thường dùng các khái niệm *nút cha* (trước), *nút con* (sau). Ví dụ nút <Chủ ngữ> là cha (trước) của nút <Danh ngữ>, nút <Danh từ> là con (sau) của nút<Danh ngữ>, v.v...

Một nút không con (nút cuối) gọi là *lá* (leaf), nút không là lá gọi là *nút trong*. Ví dụ các nút *Mèo, mù, xơi*, v.v... đều là các lá, các nút <Danh từ>, <Tính từ>, <Động từ>, v.v... đều là các nút trong.

Một cây được gọi là *cây nhị* phân (Binary Tree) nếu mỗi nút bất kỳ trừ lá có nhiều nhất là hai nút con. Ví dụ cây đã cho trong ví dụ 6.10 là cây nhị phân. Phép duyệt cây nhị phân là cách dò đến (đọc) lần lượt từng nút theo một thứ tự nào đó nhất quán.

Có 3 cách duyệt cây theo thứ tự lần lượt là :

Trái – gốc – phải. Ký hiệu TGP.

Gốc – trái – phải. Ký hiệu GTP.

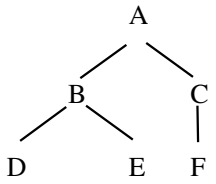
Phải – gốc – trái. Ký hiệu PGT.

Ví dụ 6.11 : Cho cây nhị phân và cách duyệt :

TGP : ((D B E) A (F C)) hay DBEAFC

GTP : (A (B D E) (C F)) hay ABDECF

PGT : ((F C) A (E B D)) hay FCAEBD



Bài tập

1. Tính giá trị các biểu thức logic (0 : **false** ; 1 : **true**) sau đây :

$x < 3 \wedge x+y = 8$	với	a. $x = 2 ; y = 6$ b. $x = -4 ; y = 1$ c. $x = 5 ; y = 3$
$-8 \le x \wedge x \le 7$	với	a. $x = 6$ b. $x = -9$
$(a \vee b) \wedge ((a \vee \neg b) \wedge c)$	với	a, b, c $\in$ { 0, 1} là các mệnh đề logic.

2. Cho biết với những giá trị nào của *n* thì mệnh đề sau đây là đúng :

   $((n = 1)) \rightarrow (n = 2))$

   $((n = 1)) \leftrightarrow (n = 2))$

3. Bằng cách đặt tên cho các mệnh đề và sử dụng các phép nối lôgic, hãy chuyển các câu sau đây thành mệnh đề phức hợp :

   Hướng dẫn : Câu «*Tom đã lớn tuổi hoặc còn trẻ tuổi*» có hai mệnh đề :
   P = «*Tom đã lớn tuổi*» và *Q*= «*Tom còn trẻ tuổi*». Từ đó nhận được $P \vee Q$.

   1. *Con người ta đều phải chết*
   2. *Người ta không thể bơi lội khi chưa ngâm mình trong nước*
   3. *Con người ta còn trẻ khi tuổi dưới 18.*
   4. *Trong tháng 9, cu Tèo phải học suốt ngày, trừ phi nó không làm việc cả ngày.*
   5. *Không thể giảng dạy ở bậc đại học mà không có bằng đại học.*

4. Sử dụng bảng chân lý, hãy chứng minh các tính chất sau :

   1. $((F \rightarrow G) \vee (G \rightarrow H)) \rightarrow (F \rightarrow H)$
   2. $(F \leftrightarrow G) \leftrightarrow ((F \rightarrow G) \wedge (G \rightarrow F))$
   3. $(F \rightarrow G) \leftrightarrow (\neg F \vee G)$
   4. $(F \vee (G \wedge H)) \leftrightarrow ((F \vee G) \wedge (F \vee H))$
   5. $(F \wedge (G \vee H)) \leftrightarrow ((F \wedge G) \vee (F \wedge H))$
   6. $(\neg(F \vee G)) \leftrightarrow (\neg F \wedge \neg G))$
   7. $(\neg(F \wedge G)) \leftrightarrow (\neg F \vee \neg G))$
   8. $(F \rightarrow (G \rightarrow H)) \leftrightarrow ((F \wedge G) \rightarrow H)$
   9. $(F \rightarrow (G \wedge H)) \leftrightarrow ((F \rightarrow G) \wedge (F \rightarrow H))$
   10. $((F \wedge G) \rightarrow H) \leftrightarrow ((F \rightarrow H) \vee (G \rightarrow H)$
   11. $(F \rightarrow (G \vee H)) \leftrightarrow ((F \rightarrow G) \vee (F \rightarrow H))$
   12. $((F \vee G) \rightarrow H) \leftrightarrow ((F \rightarrow H) \wedge (G \rightarrow H))$
   13. $((F \leftrightarrow G) \leftrightarrow H) \leftrightarrow (F \leftrightarrow (G \leftrightarrow H))$

5. Hãy tìm các ví dụ minh hoạ các tính chất trong bài tập 4.

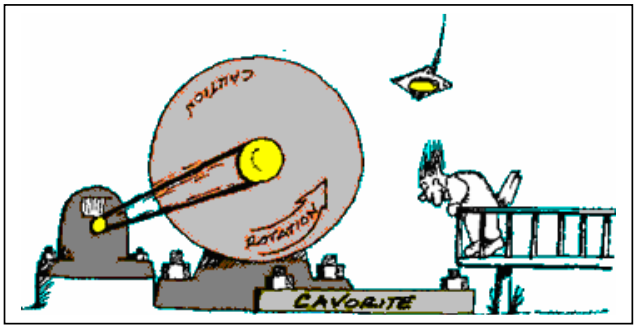
CHƯƠNG 1

Mở đầu

I. Cơ sở của môn học

Ngày nay, khi xem lại những bản vẽ thiết kế chuyển động vĩnh cửu (perpetual movement)[2] của các nhà phát minh vào những năm trước 1775 (thế kỷ thứ 18), chúng ta thấy chúng tỏ ra vô lý (và buồn cười). Vì rằng, những phát minh như vậy mâu thuẫn với những hiểu biết của chúng ta về Vật lý.

Chúng ta lập luận đơn giản như sau : từ những kiến thức khoa học cơ sở, có thể rút ra được những những cái có thể, hoặc không có thể thực hiện được trong thực tiễn (tính khả thi). Điều đó cho phép chúng ta đi đến kết luận mà không cần phải phân tích chi tiết mô hình của chuyển động vĩnh cửu đặt ra là có được hay không ?



Trong Tin học, chúng ta cũng thường gặp những chương trình đồ sộ với hàng ngàn dòng lệnh, chương trình chạy được, nhưng thực tế lại không giải quyết được những yêu cầu đòi hỏi, thế thì phải làm gì ?

- Thử tìm và sữa lỗi để có thể chạy đúng đắn ?
- Xem lại cách thiết kế chương trình và sử dụng một phương pháp khác để có thể giải quyết được vấn đề ?
- Kết luận rằng bài toán không thể giải được bởi bất kỳ một chương trình nào. Kết luận này được rút ra mà không xem xét chi tiết chương trình đã viết.

Mục đích của môn học là làm sao nhận biết được những trường hợp rơi vào điều thứ 3 ở trên. Đâu là giới hạn của Tin học ?

Ngành Vật lý đưa vào những giới hạn về những máy móc có thể được chế tạo, nhưng đó không là những giới hạn liên quan đến nội dung mà chúng ta quan tâm.

Chúng ta sẽ nghiên cứu những giới hạn về giải quyết các bài toán : bằng cách nào đó chỉ ra rằng một số bài toán là không giải được và sẽ không bao giờ giải được, dẫu rằng với sự tiến bộ của công nghệ Tin học trong tương lai.

---

2 Là chuyển động vĩnh viễn không bao giờ ngừng mà không cần tiêu tốn năng lượng (chúng không tồn tại vì mâu thuẫn với các định luật về nhiệt động lực học).

Ở đây, chúng ta căn cứ trên những nguyên lý cơ bản của Lôgích Toán đã được đề cập và phát triển từ những năm 1930, trước sự xuất hiện của máy tính điện tử (MTĐT). Những nguyên lý đó cho phép định nghĩa tường minh về phép chứng minh hình thức trong lý thuyết tính toán (theory of computation). Lý thuyết kinh điển về tính toán bắt đầu bằng các công trình của Godel, Tarski, Church, Post, Turing, Kleene... Lĩnh vực này được phát triển không ngừng và còn đang được tiếp tục hiện nay.

Như vậy, tồn tại các bài toán không giải được. Vấn đề là cần có một mô hình tính toán để thiết lập tính không giải được (non-resolvability). Tất nhiên, khi chứng minh tính giải được (resolvability) thì chỉ cần đưa ra một thủ tục cụ thể có hiệu qủa (hay thuật giải) theo trực giác.

Mô hình tính toán tổng quát được sử dụng tương đối rộng rãi là máy Turing. Các mô hình tính toán khác sau này là các thuật giải Markor, các hệ Post, các văn phạm và các hệ *L* (Lindermayer).

Để mô tả các mô hình tính toán, người ta sử dụng các công cụ là các ngôn ngữ hình thức – NNHT (formal languages). Lý thuyết tính toán còn liên quan đến các ôtômat hữu hạn (finite automaton). Đó là những mô hình tính toán dùng để đoán nhận các NNHT.

Một lĩnh vực khác của lý thuyết tính toán là độ phức tạp (complexity) của các bài toán giải được. Quan điểm nào (về thời gian chạy máy, về bộ nhớ cần sử dụng...) để nói rằng bài toán P1 là khó hơn bài toán P2 ? Bài toán nào là "bất trị" và hoàn toàn không thể khống chế được lượng thời gian để giải nó ?

Các chủ đề trung tâm của Tin học lý thuyết là :

- *Các NNHT và các ôtômat.*
- *Tính tính được* (computability)
- *Lý thuyết các hàm đệ quy* (theory of recursive functions).
- *Độ phức tạp tính toán* (computational complexity)
- *Mật mã học* (cryptologia)

Và một số hướng nghiên cứu mới trong lý thuyết tính toán.

II. Các khái niệm

Vấn đề cơ bản đặt ra là cần biết những bài toán nào thì giải được bởi một chương trình chạy trên một MTĐT. Có hai khái niệm cần xem xét :

- Khái niệm về bài toán.
- Khái niệm về chương trình chạy trên một MTĐT.

II.1. Khái niệm bài toán

Một bài toán là :

1. Mô tả cách biểu diễn (hữu hạn) các phần tử (hay dữ liệu) của một tập hợp hữu hạn hay vô hạn đếm được. Mỗi dữ liệu được gọi là một *thể nghiệm* (instance).
2. Một phát biểu liên quan đến các phần tử của tập hợp này, có thể là *đúng* (true), hoặc có thể là *sai* (false) tùy theo phần tử được chọn.

Ví dụ 1.1 : Bài toán «*xác định xem một số tự nhiên n là chẵn hay lẻ ?*».

Người ta thường kết hợp bài toán với ngôn ngữ, gọi là *ngôn ngữ đặc trưng* (characteristic language) của bài toán, được hợp thành từ tập hợp các câu biểu diễn một phần tử của tập hợp để từ đó, kết quả của bài toán là câu trả lời đúng. Tại mỗi trường hợp của bài toán, một câu hỏi đặt ra cho một phần tử nào đó sẽ có câu trả lời. Ví dụ đối với bài toán trên, câu trả lời «*số 35 chẵn* ?» là *sai*.

Khái niệm về bài toán độc lập với khái niệm về chương trình. Có thể viết một chương trình để giải quyết một bài toán, nhưng bài toán không được định nghĩa bởi chương trình. Mặt khác, nhiều chương trình có thể cùng giải một bài toán.

Để giải quyết một bài toán bằng chương trình, cần xét hết các trường hợp của bài toán, để chương trình có thể chạy được thông suốt từ đầu đến cuối.

Ví du 1.2 : Các trường hợp của bài toán ở ví dụ 1.1 (về các số tự nhiên) có thể được biểu diễn bởi các số nhị phân. Chẳng hạn, một chương trình giải quyết bài toán này có thể kiểm tra chữsố cuối cùng của số biểu diễn, số chẵn nếu chữ số cuối cùng là 0, số lẻ nếu là 1.

Ví du 1.3 :

1. Sắp xếp một mảng số theo thứ tự tăng dần là một bài toán.

- Xác định xem một chương trình Pascal có dừng hay không dù bất kỳ dữ liệu đưa vào như thế nào là một bài toán (bài toán dừng).
- Xác định xem một đa thức hệ số nguyên có các nghiệm nguyên hay không là một bài toán (bài toán thứ 10 của Hilbert).

Trong ví dụ này, câu 1 giải được bởi một chương trình thực hiện được trên MTĐT. Còn các câu 2 và 3 sẽ không giải quyết được như vậy.

Trong giáo trình, chúng ta sẽ nghiên cứu một lớp các bài toán có lời giải là *có* hoặc *không* (1 hoặc 0) gọi là lớp các *bài toán nhị phân* (bynary problem) vì có lời giải nhị phân.

Ví dụ, bài toán dừng có lời giải nhị phân. Người ta chứng minh được rằng lớp các bài toán không nhị phân có thể đưa về dạng nhị phân.

II.2. Khái niệm chương trình

Lời giải một bài toán có dạng một chương trình chạy được trên một MTĐT được gọi là một *thủ tục có hiệu quả* (effective procedure). Có sự khác nhau giữa một *thủ tục có hiệu quả* và một lời giải không có dạng chương trình.

Chẳng hạn một chương trình viết bằng ngôn ngữ Pascal là một thủ tục hiệu quả. Vì sao ? Vì chương trình này có thể biên dịch thành mã máy để chạy được với các số liệu dạng nhị phân.

Có thể giải thích một cách khác như sau :

Chương trình Pascal chứa mọi thông tin cần thiết để giải bài toán. Mỗi lần có đủ điều kiện để chạy chương trình, bài toán sẽ được giải quyết mà không cần bổ sung gì thêm, tự MTĐT thực hiện những lệnh đã có trong chương trình.

Để dễ hiểu khái niệm thủ tục hiệu quả, ta xét một thủ tục không hiệu quả. Ví dụ xét bài toán dừng, lời giải *xác định xem có phải chương trình không có vòng lặp vô hạn hoặc không có dãy các lời gọi đệ qui* là không hiệu quả.

Tuy nhiên từ lời giải này, câu hỏi đặt ra là làm sao biết được một vòng lặp, hoặc một dãy các lời gọi đệ qui, là vô hạn ?

Sẽ không có thủ tục hiệu quả để giải bài toán dừng.

Ví dụ 1.2 : Bài toán 3n+1 : chưa có câu trả lời về tính dừng của nó :

```
function threen(n: integer): integer;{ recursive }
begin
   if (n = 1) then 1
   else if odd(n) then threen(3*n+1)
        else threen(n div 2);
end;
```

Xác định xem một chương trình Pascal có phải là thủ tục hiệu quả không dẫn đến phải giải bài toán dừng. Nhưng bài toán dừng lại không giải được bằng một thủ tục hiệu quả. Sự luẩn quẩn này dẫn đến phải chứng minh tính không giải được của bài toán dừng.

Như vậy chúng ta đã sử dụng khái niệm ngôn ngữ lập trình để hình thức hóa khái niệm về thủ tục hiệu quả. Mặt khác, một ngôn ngữ lập trình chỉ định nghĩa một thủ tục hiệu quả bởi có sự can thiệp của một thủ tục diễn dịch (interpretation) hoặc biên dịch (compilation).

Sự can thiệp này làm phức tạp thêm vấn đề đang xét về thủ tục hiệu quả. Tuy nhiên sự tồn tại các thủ tục diễn dịch cho phép chạy các chương trình viết trên mọi ngôn ngữ lập trình thông dụng.

Để hình thức hóa các thủ tục hiệu quả, người ta sử dụng các ngôn ngữ lập trình có dạng đơn giản sao cho việc biên dịch chương trình là ngay lập tức. Sau khi biên dịch, chương trình ở dạng khả thi (executable program).

Ta gọi các kiểu chương trình có cơ cấu biên dịch ngay lập tức là những ôtômat. Ở đây ôtômat là một chương trình mà không phải là một cái máy để cho chương trình thực hiện trên đó. Mỗi lớp ôtômat (ngôn ngữ lập trình) sẽ có một cơ cấu xử lý rất đơn giản cho phép hiểu được cách thực hiện của ôtômat (chương trình).

II.3. Hình thức hóa các bài toán

II.3.1. Bảng chữ và câu

Để biểu diễn một chương trình, ta cần sử dụng một tập hợp các ký tượng (symbol), tập hợp này được gọi là bảng chữ (alphabet) và được biểu diễn bởi chữ cái Hy lạp Σ. Các ký tượng của bảng chữ thường được gọi là các *ký tự* (characters).
Định nghĩa 1.1 : Một *bảng chữ* là một tập hữu hạn các ký tự.

Mỗi phần tử cuả bảng chữ được đặt tương ứng với (hay được biểu diễn bởi) một ký tự, hay ký hiệu, nào đó. Kích thước của bảng chữ là số phần tử của bảng chữ đó.

Ví dụ 1.5 :

Với bảng chữ kích thước 3, có thể có các biểu diễn ký hiệu như sau :

{ *a*, *b*, c }, { α, β, γ }, { 1, 2, 3 }, hoặc { ♣, ♦, ♥ }.

Định nghĩa 1.2 :

Một câu (phrase, word), hay còn gọi là xâu (string), trên một bảng chữ nào đó là một dãy hữu hạn các phần tử của bảng chữ đó.

Ví dụ 1.3 :

a, *ab*, zt, computer là những câu trên bảng chữ { *a*....z },

4♣3♦2♠, 1234, ♣♠ là các câu trên bảng chữ { 0, ..., 7, ♠, ♣, ♦, ♥ }.

Độ dài của một câu là số ký tự có mặt trong câu. Độ dài câu là hữu hạn, nhưng không hạn chế là có bao nhiêu ký tự. Một câu có thể có 3 ký tự hoặc thậm chí có 10^{2534} ký tự.

Nếu câu được ký hiệu bởi *w* thì độ dài cuả câu được ký hiệu là |*w*| hay length(*w*). Độ dài câu có thể bằng không, trường hợp này được gọi là câu rỗng (empty word), ký hiệu là ε, hoặc e, hoặc λ hoặc ω.

Người ta gọi ε là câu *đơn vị* vì có εv = vε = v, với v là một câu bất kỳ.

Từ một câu có độ dài n, người ta có thể trích ra một ký tự nào đó có vị trí xác định trong phạm vi 1..n.

Ví dụ 1.4 :

Từ câu aaabbaabbba, có thể trích ra các ký tự :

w(1) = *a*, ..., *w*(4) = *b*, ..., *w*(11) = *a*.

Ta nói ghép tiếp của hai câu u và v là câu *w* = uv, nghĩa là câu *w* gồm hai phần, u là tiền tố (prefix) rồi đến v là hậu tố (postfix).

Đảo ngược một câu *w*, ký hiệu w^R, là câu *w* được viết theo thứ tự ngược lại.

Rõ ràng $\varepsilon^R = \varepsilon$.

II.3.2. Biểu diễn các bài toán

Ta có thể biểu diễn, hay mã hóa, các trường hợp của một bài toán đã cho bởi các câu. Điều này dễ hiểu vì rằng mọi dữ liệu sử dụng trong Tin học đều được biểu diễn bởi các xâu ký tự (trong máy là dãy các chữ số 0 và 1).

Chẳng hạn, một hình ảnh biểu diễn bởi một xâu ký tự là dãy các mật độ điểm ; một âm thanh cũng được biểu diễn bởi một dãy các số (số hoá). Thậm chí bài toán phát biểu bằng tiếng Anh, hay tiếng Pháp cũng đều là các xâu ký tự.

Cho một bài toán nhị phân mà các trường hợp của nó được mã hóa bởi các câu xây dựng từ bảng chữ Σ. Bảng chữ Σ có thể chia ra thành 3 tập hợp con như sau :

Các câu biểu diễn các trường hợp của bài toán với câu trả lời *có* (đúng).

Các câu biểu diễn các trường hợp bài toán với câu trả lời *không* (sai)..

Các câu không biểu diễn một trường hợp nào của bài toán.

Thông thường, người ta gộp hai trường hợp sau cùng thành một tập hợp.

II.3.3. Ngôn ngữ

Một bài toán có thể được đặc trưng bởi tập hợp các câu biểu diễn các trường hợp đúng. Tập hợp các câu tạo thành ngôn ngữ (language).

Định nghĩa 1.3 :

Một ngôn ngữ là tập hợp các câu được xây dựng trên cùng một bảng chữ.

Người ta cũng nói một bài toán được đặc trưng bởi ngôn ngữ gồm các câu đã được mã hóa và việc giải quyết một bài toán nhằm nhận biết các trường hợp tích cực từ các mã của chúng.

Chính vì vậy giáo trình này sẽ chỉ nghiên cứu các bài toán đoán nhận ngôn ngữ, nghĩa là xác định xem trong số các câu xây dựng trên một bảng chữ đã cho, những câu nào thì thuộc vào ngôn ngữ, còn những câu nào thì không ?

Như thế người ta sẽ không phân biệt giữa việc giải quyết một bài toán và việc nhận biết ngôn ngữ gồm mã các trường hợp tích cực của bài toán.

Ví dụ 1.5 :

{ ε, *a*, *b*, aaaa, bbbb, abbabbbb }, và ∅ (tập trống, không chứa bất kỳ câu nào – empty set) là các ngôn ngữ trên bảng chữ { *a*, *b* }.

Ngôn ngữ { 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, ... } trên bảng chữ { 0, 1 } chứa mọi số nhị phân. Tương tự, tập hợp mọi biểu diễn nhị phân các số chẵn cũng là một ngôn ngữ.

Ngôn ngữ ∅ khác với ngôn ngữ { ε } là ngôn ngữ chỉ chứa duy nhất một câu rỗng.

Tập hợp các câu là các chương trình viết trên ngôn ngữ Pascal (hay trên một ngôn ngữ lập trình bất kỳ nào đó) chạy thông suốt cũng là một ngôn ngữ.

Ta đưa vào *quy cách mô tả một số ngôn ngữ* (hay những ngôn ngữ nào đó) mà không phải mô tả cho mọi ngôn ngữ.

Làm thế nào để mô tả một ngôn ngữ ? Nếu ngôn ngữ là hữu hạn câu, chỉ cần liệt kê hết tất cả các câu thuộc ngôn ngữ. Nếu ngôn ngữ là vô hạn câu, ta không thể liệt kê hết.

Trong trường hợp này, có thể căn cứ trên sự biểu diễn trực tiếp của một số ngôn ngữ sơ cấp (chẳng hạn các ngôn ngữ chỉ chứa một câu duy nhất tạo từ một ký tự của bảng chữ). Từ đó, áp dụng các phép toán (trên các ngôn ngữ đã mô tả) để mô tả những ngôn ngữ phức tạp hơn.

III. Mô tả ngôn ngữ

III.1. Các phép toán trên ngôn ngữ

Cho hai ngôn ngữ L_1 và L_2. Hội của L_1 và L_2 là ngôn ngữ chứa các câu hoặc thuộc L_1 hoặc thuộc L_2. Một cách hình thức :

$L_1 \cup L_2 = \{ w \mid w \in L_1$ hoặc $w \in L_2 \}$

Ghép (hay tích) của L_1 và L_2 là ngôn ngữ chứa các câu được chép từ các câu của L_1, theo sau là các câu của L_2. Một cách hình thức :

$L_1.L_2 = \{ w \mid w = xy, x \in L_1$ và $y \in L_2 \}$

Bao đóng kleene (Kleene Closure) của L_1 là tập hợp các câu tạo ra bởi việc ghép hữu hạn các câu của L_1. Một cách hình thức :

$L_1^* = \{ w \mid \exists k \geq 0$ và $w_1, w_2, ..., w_k \in L_1$ sao cho $w = w_2 w_2 ... w_k \}$

Bù của một ngôn ngữ là tập hợp các câu không thuộc ngôn ngữ đã cho. Một cách hình thức :

$L_1 = \{ w \mid w \notin L_1 \}$

Tập hợp các *ngôn ngữ chính quy* (Regular Languages) được định nghĩa dựa trên các ngôn ngữ sơ cấp và các phép toán hội, ghép và đóng lặp *.

Định nghĩa 1.4 :

Tập hợp các ngôn ngữ chính quy $\Re$ trên bảng chữ Σ là tập hợp nhỏ nhất (chứa ít phần tử nhất) gồm các ngôn ngữ thõa mãn các điều kiện sau :

1. $\varnothing \in \Re$, $\{ \varepsilon \} \in \Re$
2. $\{ a \} \in \Re$ với $\forall a \in \Sigma$
3. Nếu $A, B \in \Re$, thì $A \cup B$, $A.B$ và $A^* \in \Re$.

Trong định nghĩa trên, rõ ràng $\Re$ là tập hợp bé nhất thỏa mãn các điều kiện đã nêu, nghĩa là chỉ có những tập hợp xây dựng từ các tập hợp sơ cấp $\varnothing$, $\{ \varepsilon \}$ và $\{ a \}$ bởi các phép hội, ghép và bao đóng lặp là có mặt trong $\Re$.

III.2. Biểu thức chính qui

Người ta sử dụng các *biểu thức chính quy* (Regular Expressions) để chứng minh các ngôn ngữ chính qui.

Có thể hình dung một cách đơn giản là làm sao nhận được tập hợp chính qui $\Re$ đã nêu từ các tập hợp chính qui sơ cấp.

Định nghĩa 1.5 :

Các biểu thức chính qui trên bảng chữ Σ là các biểu thức được tạo thành theo các qui tắc sau :

1. $\varnothing$, ε và $\forall a \in \Sigma$ (các phần tử của Σ) đều là những biểu thức chính qui ;
2. Nếu α và β là hai biểu thức chính qui, thì $(\alpha \cup \beta)$, $(\alpha\beta)$, $(\alpha)^*$ đều là những biểu thức chính qui.

Chú ý 1 : Nếu không xảy ra nhầm lẫn khi viết một biểu thức chính qui, có thể bỏ qua các dấu ngoặc đơn và theo mức ưu tiên giảm dần. Chẳng hạn ta viết a^* thay vì viết $(a)^*$.

Chú ý 2 : Người ta cũng viết $a+b$ thay vì viết $a \cup b$.
Ví dụ, biểu thức $((0\ (1^*)) + 0)$ có thể viết $01^* + 0$.
Dễ thấy rằng $a\varepsilon = \varepsilon a = a$.

Định nghĩa 1.6 :

Ngôn ngữ $L(\xi)$ chỉ định bởi biểu thức chính qui ξ được định nghĩa như sau :

$L(\varnothing) = \varnothing$, $L(\varepsilon) = \{ \varepsilon \}$;

$L(a) = \{ a \}$ cho $\forall a \in \Sigma$;

$L((\alpha \cup \beta)) = L(\alpha) \cup L(\beta)$

$L((\alpha\beta)) = L(\alpha)L(\beta)$

$L((\alpha)^*) = L(\alpha)^*$

Ta thấy các biểu thức chính qui cũng tạo thành một ngôn ngữ vì chúng là những xâu ký tự trên bảng chữ Σ :

$\Sigma^* = \Sigma \cup \{), (, \varnothing, \cup, *, \varepsilon \}$

Trong định nghĩa 1.6, chúng ta thấy có sự "hiểu ngầm" vì cách sử dụng một số ký hiệu. Thực vậy, tùy theo vị trí xuất hiện, cùng một ký hiệu có thể có ý nghĩa khác nhau.

Chẳng hạn dấu "(" biểu diễn một ký hiệu của bảng chữ để xây dựng các biểu thức chính qui nhưng lại được sử dụng để bao đối số của toán hạng. Cũng như vậy, dấu ∪ vừa là ký hiệu của bảng chữ vừa là một toán tử.

Để tránh sự nhầm lẫn, có thể sử dụng các dấu khác như dấu "[" thay cho "(". Như thế, *L*, [, và] sẽ là các thành phần của một siêu ngôn ngữ (Metalanguage) dùng để mô tả một ngôn ngữ khác, ở đây là mô tả các ngôn ngữ chính qui.

Tuy nhiên, trong thực tế, người ta vẫn dễ dàng phân biệt đươc đâu là ký hiệu thuộc ngôn ngữ cần định nghĩa và đâu là ký hiệu thuộc siêu ngôn ngữ.

Như thế, người ta không cần sử dụng các ký hiệu khác với cách viết thông thường, như ở định nghĩa 1.6.

Định lý 1.1 :

Một ngôn ngữ là chính qui nếu và chỉ nếu ngôn ngữ đó được chỉ định bởi một biểu thức chính qui.

Chứng minh :

Ta sẽ chứng minh theo hai chiều phân biệt *nếu* "⇒" và *chỉ nếu* "⇐".

"⇒" : Nếu ngôn ngữ là chính qui, thì ngôn ngữ này được biểu diễn bởi một biểu thức chính qui. Cần chỉ ra rằng mọi ngôn ngữ chính qui đều được chư nh bởi một biểu thức chinh qui.

Theo định nghĩa các ngôn ngữ chính qui, tập hợp $\mathfrak{R}$ được xây dựng một cách qui nạp : một số tập hợp con trong $\mathfrak{R}$ được xây dựng từ các tập hợp sơ cấp. Cách chứng minh đó được gọi là qui nạp có cấu trúc[3] (Structural Induction) như sau :

Comment [none1]:

Để chứng minh một tính chất P : *tồn tại một biểu thức chính qui biểu diễn ngôn ngữ này* là đúng với mọi phần tử của $\mathfrak{R}$, ta lý luận như sau:

Chỉ ra ra rằng P đúng với mọi ngôn ngữ ∅, { ε } và các ngôn ngữ có dạng { *a* }.

Chỉ ra rằng nếu tính chất P là đúng cho các ngôn ngữ A và B, thì P cũng đúng cho ngôn ngữ A∪B, A.B và A^*.

Từ đây, dễ dàng chứng minh "⇒".

Tương tự có thể chứng minh cho trường hợp *chỉ nếu* "⇐" □.

Chú ý 1 :

Bao đóng L^* của *L* có thể viết $L^* = L^0 \cup L^1 \cup L^2 \cup ...$ Trong đó L^i cho bởi định nghĩa đệ qui :

$L^0 = \{ \varepsilon \}$

$L^i = LL^{i-1}$ với $i \geq 1$

Bao đóng dương $L^+ = L^1 \cup L^2 \cup L^3 \cup ...$

---

3 Chẳng hạn phép chứng minh qui nạp đã biết đối với các số tự nhiên :
1. 0 là một số tự nhiên.
2. Nếu n là một số tự nhiên, thì n+1 cũng là một số tự nhiên.

Như vậy $L^+ = LL^* = L^*L$ và $L^* = L^+ \cup \{ \varepsilon \}$

Chú ý 2 :

Để chứng minh rằng hai tập hợp A và B đã cho là bằng nhau, A = B, cần chỉ ra A⊂B và B⊂A.

Ví dụ 1.9 : Cho bảng chữ $\Sigma = \{ a_1, ..., a_n \}$

1. Tập hợp các câu xây dựng trên Σ được chỉ định bởi biểu thức chính qui $(a_1 \cup ... \cup a_n)^*$, hay còn được viết Σ^*.

2. Tập hợp các câu khác rỗng ($w \neq \varepsilon$) xây dựng trên bảng chữ Σ được biểu diễn bởi biểu thức chính qui $(a_1 \cup ... \cup a_n)\,(a_1 \cup ... \cup a_n)^*$.

   Người ta viết biểu thức này $\Sigma\Sigma^*$ hay Σ^+.

   Một cách tổng quát $\alpha^+ = \alpha\alpha^*$

3. Ngôn ngữ được chỉ định bởi biểu thức chính qui $(a \cup b)^* a (a \cup b)^*$ là ngôn ngữ gồm các câu được xây dựng từ *a* và *b* chứa ít nhất một *a*.

Ví dụ 1.6 : Chứng minh rằng $(a^*b)^* \cup (b^*a)^* = (a \cup b)^*$,

hay các biểu thức chính qui $(a^*b)^* \cup (b^*a)^*$ và $(a \cup b)^*$ cùng chỉ định một ngôn ngữ chính qui.

Lời giải như sau :

" ⊂" : Rõ ràng $(a^*b)^* \cup (b^*a)^* \subset (a \cup b)^*$ vì $(a \cup b)^*$ biểu diễn tập hợp các câu xây dựng từ *a* và *b*.

" ⊃" : Để chứng minh minh điều ngược lại, ta xét một câu :

$w = w_1 w_2 ... w_n \in (a \cup b)^*$.

Xảy ra bốn trường hợp sau :

1. $w = a^n$, do đó $w \in (\varepsilon a)^* \subset (b^*a)^*$
2. $w = b^n$, do đó $w \in (\varepsilon b)^* \subset (a^*b)^*$
3. *w* chứa *a* và *b* và kết thúc bởi *b*. Ta có :

$$\underbrace{a \ldots ab}_{a^*b}\ \underbrace{b \ldots b}_{(a^*b)^*}\ \underbrace{a \ldots ab}_{a^*b}\ \underbrace{b \ldots b}_{(a^*b)^*}$$

   Và do đó, *w* ∈ ngôn ngữ chỉ định bởi $(a^*b)^* \cup (b^*a)^*$.

4. *w* chứa *a* và *b* và kết thúc bởi *a*. Tương tự trường hợp 3, ta có :
   $w \in L((a^*b)^* \cup (b^*a)^*)$.

III.3. Các ngôn ngữ phi chính qui

Như đã thấy, các biểu thức chinh qui cho phép chỉ định một số ngôn ngữ, nhưng không phải mọi ngôn ngữ. Tồn tại những ngôn ngữ phi chính qui và không có đủ các biểu thức chính qui để biểu diễn mọi ngôn ngữ.

Điều này dễ hiểu vì số các ngôn ngữ và số các biểu thức chính qui là vô hạn.

Định lý 1.2 nêu lên rằng : mọi ngôn ngữ không thể là chinh qui, thật vậy :

- Tập hợp các ngôn ngữ và tập hợp các tập hợp con của một tập hợp đếm được (tập hợp các câu) sẽ là không đếm được.
- Tập hợp các ngôn ngữ chính qui là đếm được vì mỗi ngôn ngữ chính qui được biểu diễn bởi một biểu thức chính qui và tập hợp các biểu thức chính qui là đếm được.
- Sẽ có nhiều ngôn ngữ khác với ngôn ngữ chính qui.

Những kết quả trên có tầm vóc đại cương vì cho phép nhìn nhận sự tồn tại của những bài toán không giải được bởi một thủ tục hiệu quả. Thực vậy, số các bài toán bằng số các ngôn ngữ và như vậy sẽ là không đếm được.

Thế nhưng một thủ tục hiệu qua khi được biểu diễn bởi một câu hửu hạn, sẽ chỉ tồn tại một số đếm được các thủ tục hiệu quả. Sẽ không có một thủ tục hiệu quả cho mỗi bài toán.

III.4. Vấn đề biểu diễn ngôn ngữ

Như đã định nghĩa, một ngôn ngữ trên bảng chữ Σ là tập hợp con của Σ^+.

Vấn đề đặt ra là với ngôn ngữ *L*, làm sao có thể biểu diễn hết mọi câu của *L* ? Với các ngôn ngữ hữu hạn, chỉ việc liệt kê các câu (xem ví dụ 1.8). Với các ngôn ngữ vô hạn, ta không thể liệt kê hết các câu mà ta phải tìm cách biểu diễn hữu hạn.

Nếu như một ngôn ngữ *L* nào đó gồm các câu có một số tính chất nhất quán nào đó, ta có thể dùng các tân từ để biểu diễn.
Ví dụ 1.9 : Cho các ngôn ngữ :

$L_1 = \{ a^i \mid$ i là một số nguyên tố $\}$

$L_2 = \{ a^i b^j \mid i \geq j \geq 0 \}$

$L_3 = \{ w \in \{ ab \}^* \mid$ số chữ *a* bằng và số chữ *b* $\}$

Ta có thể dùng các biểu thức chính qui để biểu diễn ngôn ngữ, tuy nhiên, như đã thấy, còn có các ngôn ngữ không phải là chính qui.

Người ta dùng các ôtômat hay văn phạm để biểu diễn ngôn ngữ. Văn phạm sản sinh ra các câu của một ngôn ngữ. Còn ôtômat lại cho phép đoán nhận một câu bất kỳ nào đó có thuộc ngôn ngữ đang xét hay không ?

Ví dụ 1.10 : Cho *L* là ngôn ngữ trên $\{ a, b \}$ được định nghĩa như sau :

1. $\varepsilon \in L$.
2. Nếu $w \in L$ thì $awb \in L$.
3. *L* không còn câu nào khác nữa.

Cách định nghĩa trên cho ta qui luật sản sinh các câu của *L* như sau :

Từ (1), ta có câu ε, coi ε là *w*, từ (2), ta có câu $awb = a\varepsilon b = ab$.

Lại do (2), ta có aabb, aaabbb,... Cứ thế, ta có mọi câu của *L*.

Ở đây $w \notin \{ a, b \}$ đóng vai trò một câu trung gian. $L = \{ a^i b^i \mid \forall i \geq 0 \}$.

Ví dụ trên cho biết cách sãn sinh ra câu trên một ngôn ngữ thì ví dụ dưới đây lại cho ta cách đoán nhận một câu đã cho có thuộc ngôn ngữ đã cho hay không ?

Ví dụ 1.11 : Giả sử ngôn ngữ *L* được định nghĩa là tập các câu có thể thu gọn về câu rỗng bằng dãy các phép thay thế ẩn các xâu con *ab* bởi ε :

$$aabbab \Rightarrow abab \Rightarrow ab \Rightarrow \varepsilon$$

Như vậy, câu $w = aabbab \in L$.

Giả sử coi *a*, *b* lần lượt là cặp dấu ngoặc đơn (và) thì *L* là tập hợp các câu gồm các cặp dấu ngoặc đơn cân bằng nhau mà không cài nhau, gọi là các câu có ngoặc đơn cân bằng thu được từ một biểu thức toán học nào đó.

Ví dụ, trong biểu thức số học (3 x (x – y)) / (x + 1), nếu bỏ qua các ký hiệu toán tử và toán hạng, ta sẽ nhận được câu ngoặc đơn cân bằng (())(), tức là câu aabbab đã xét.

Bài tập chương 1

1. Cho các biểu thức chính qui r, s, t, trong đó nếu r = s thì có nghĩa $L(r) = L(s)$.
   Chứng minh các tinh chất sau :

1. $r + s = s + r$	6. $(\varepsilon + r)^* = r^*$	11. $\varnothing r = r\varnothing = \varnothing$
2. $r + (s + t) = (r + s) + t$	7. $(r^*)^* = r^*$	12. $\varnothing^* = \varepsilon$
3. $r (s + t) = = rs + rt$	8. $r + r = r$	13. $r + r^* = r^*$
4. $r\varepsilon = \varepsilon r = r$	9. $r(st) = (rs)t$	14. $(r^* s^*)^* = (r + s)^*$
5. $r + \varnothing = r$	10. $(r + s)t = rt + st$	

2. Tìm các biểu thức chính qui chỉ định phần bù của các ngôn ngữ sau :

   1. $(a \cup b)^* b$
   2. $((a \cup b)(a \cup b))^*$

   3. Cho ngôn ngữ *L* trên bảng chữ { *a*, *b* } được định nghĩa như sau :

   1. $\varepsilon \in L$
   2. Nếu $w \in L$ thì $awb \in L$
   3. Nếu $w \in L$ thì $bwa \in L$
   4. Nếu $w_1, w_2 \in L$ thì $w_1 w_2 \in L$

   Hãy chứng minh bằng quy nạp rằng ngôn ngữ *L* đã cho gồm mọi câu có số chữ *a* đúng bằng số chữ *b*, có thể viết $n_a(w) = n_b(w)$.

CHƯƠNG 2

Ôtômat hữu hạn

Chương này nghiên cứu một lớp các ôtômat hữu hạn (Finiste State Automata). Quan niệm về ôtômat hữu hạn giúp chúng ta hiểu sâu sắc hơn về khái niệm thủ tục hiệu quả, nhưng đây không phải là lựa chọn duy nhất để mô hình hóa khái niệm này.

Cơ chế hoạt động của ôtômat hữu hạn có thể được áp dụng để giải quyết một số bài toán trong Tin học, ví dụ bài toán tìm kiếm các chuỗi ký tự trong một văn bản. Để hình dung sự hoạt động của ôtômat, trước hết, ta hãy xét một mô hình tính toán mô phỏng một MTĐT để thực hiện các lệnh của một chương trình, ta gọi là máy RAM. Giả sử máy RAM gồm :

- Một bộ xử lý và các thanh ghi, trong đó có thanh đếm chương trình PC (Program Counter).
- Một bộ nhớ chứa chương trình và dữ liệu, hay gọi là RAM (Random Access Memory). Ở đây, có thể xem bộ nhớ không thể bị xoá, theo cách hoạt động của ROM (Read Only Memory).

Ta gọi nội dung của bộ nhớ và của các thanh ghi tại một thời điểm nào đó là một trạng thái (State). Việc thực hiện các lệnh của chương trình là sự chuyển đổi các trạng thái. Mỗi nội dung của bộ nhớ và của các thanh ghi xác định một trạng thái phân biệt. Sự chuyển trạng thái chỉ phụ thuộc vào chương trình và máy tính. Quá trình thực hiện chương trình là quá trình chuyển từ trạng thái đầu tiên đến trạng thái cuối cùng nhờ một hàm chuyển tiếp (Transition Function).

Quan niệm về trạng thái của một hệ thống Tin học, hay một hệ thống đang xét nào đó, là một quan niệm tổng quan và rất tiện dụng. Có thể hiểu đơn giản trạng thái của một hệ thống tại một thời điểm đã cho là tất cả thông tin cần thiết để đoán trước tiến triển của hệ thống trong tương lai.

Ví dụ trong một hệ thống động học, biết vị trí và vận tốc của mỗi phần tử đủ để xác định được cái gì sẽ xảy ra tiếp theo.

Chúng ta giả thiết rằng số các trạng thái là hữu hạn. Với mỗi trạng thái đầu, hoàn toàn có thể xây dựng được dãy các trạng thái tiếp theo.

Dãy này có thể vô hạn, nhưng vì số trạng thái là hữu hạn nên một trạng thái nào đó có thể xuất hiện đến lần thứ hai trong dãy. Từ thời điểm này, phần dãy giữa hai trạng thái vừa nói sẽ lặp lại một cách không đơn định và không nhất thiết phải tiếp tục tính toán từ lần xuất hiện lần thứ hai trở đi.

Chương trình trong máy tính đang xét có dạng một câu (chuỗi ký tự) được đọc từ ROM vào bộ nhớ trong (theo từng ký tự) nhờ một thiết bị đọc. Quy ước đọc ký tự tiếp theo của câu để xử lý như sau :

2. Khi thiết bị đọc đang đọc một ký tự thì tạm thời coi như máy tính bị hóc để chờ đọc xong.

- Mỗi lần xử lý một ký tự là mỗi lần chuyển tiếp một trạng thái. Nếu như phải có nhiều bước để xử lý một ký tự thì các bước này phải được thay thế bởi một bước duy nhất.
- Sau khi đọc xong ký tự cuối cùng thì máy dừng và cho kết quả.

Như vậy, một chương trình đang được thực hiện trên một máy tính nào đó sẽ được mô hình hoá bởi :

1. Một tập hợp hữu hạn các trạng thái.
2. Một hàm chuyển tiếp xác định trên tập hợp các trạng thái này.
3. Một trạng thái khởi đầu.

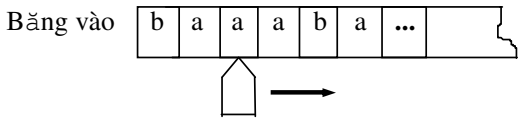
Ví dụ 2.1 :

Một máy tính có bộ nhớ 8 Mbytes và 16 thanh ghi 32 bit sẽ có :

$2^{2^{26}+2^9} = 2^{67109376}$ trạng thái khác nhau

$(2^{26} + 2^9 = 8 \text{ bit} \times 8 \times 2^{20} + 32 \text{ bit} \times 16)$.

Một ôtômat hữu hạn được mô tả bởi một đầu đọc và một câu nằm trên băng vào như hình dưới đây :



Đầu đọc di chuyển theo chiều mũi tên.

I. Ôtômat hữu hạn đơn định

I.1. Mô tả

Một ôtômat hữu hạn đơn định (DFA: Deterministic Finite State Automaton) gồm các phần tử :

- Băng vào (Input Tape) chứa câu cần xử lý gồm nhiều ô, mỗi ô chứa một ký tự. Một đầu đọc (Read Head) đọc lần lượt từng ký tự trong ô.
- Tập hợp các trạng thái trong đó có một trạng thái đầu (Initial State) và một số trạng thái cuối hay đạt được (Accepting States).
- Hàm chuyển tiếp chuyển ôtômat sang trạng thái tiếp theo từ trạng thái đang xét và ký tự vừa đọc được.

Hoạt động của ôtômat hữu hạn đơn định đối với câu vào như sau :

Đầu tiên, ôtômat ở trạng thái đầu, câu vào nằm trên băng vào và đầu đọc nằm ở vị trí trước ký tự đầu tiên của câu.

Tại mỗi thời điểm, ôtômat đọc một ký tự trên băng vào, hàm chuyển tiếp xác định trạng thái tiếp theo và đầu đọc dịch sang ký tự tiếp theo của băng.

Ôtômat dừng lại khi toàn bộ câu vào đã được đọc. Câu vào được thừa nhận (thuộc vào ngôn ngữ đang xét) nếu ôtômat đang ở trạng thái cuối.

I.2. Mô hình hóa

Một ôtômat hữu hạn đơn định được biểu diễn hình thức bởi bộ năm :

$$M = (Q, S, d, q_0, A)$$

trong đó :

Q tập hợp hữu hạn các trạng thái ;

Σ bảng chữ vào ;

δ $Q \times \Sigma \to Q$ là hàm chuyển tiếp ;

$q_0 \in Q$ là trạng đầu ;

$A \subseteq Q$ là tập hợp các trạng thái cuối.

Như vậy, ôtômat là loại máy dùng để đoán nhận một ngôn ngữ nào đó, đồng thời người ta cũng nói một ngôn ngữ nào đó được thừa nhận bởi ôtômat đã cho.

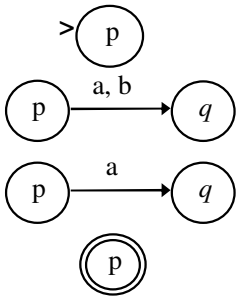
Chú ý rằng Q gồm các trạng thái ghi nhớ của ôtômat trong quá trình đoán nhận và khả năng ghi nhớ là hữu hạn. Mặt khác, hàm chuyển tiếp là toàn phần và đơn trị nên bước chuyển của ôtômat luôn được xác định một cách duy nhất.

Do hai đặc điểm trên mà ta gọi ôtômat là hữu hạn đơn định.

I.3. Biểu diễn ôtômat bởi sơ đồ

Cho ôtômat M = (Q, Σ, δ, q0, A). Người ta biểu diễn M như sau : mỗi trạng thái của M là một đỉnh của đồ thị, biểu diễn bởi một vòng tròn. Mỗi chuyển tiếp là một cung mũi tên có đánh nhãn là ký tự được đọc.

Nếu δ (p, σ) = *q* thì cung nối từ p sang *q* sẽ có nhãn là **σ, σ ∈ Σ**. Trạng thái đầu có gắn một dấu nhọn >. Các trạng thái cuối được biểu diễn bởi các vòng tròn kép. Sau đây là cách vẽ ôtômat :



chỉ rằng p là trạng thái đầu, p = q0

có nghĩa rằng (p, *a*, *q*) ∈ δ

có nghĩa rằng (p, *a*, *q*) ∈ δ và (p, *b*, *q*) ∈ δ

chỉ rằng p là trạng thái cuối, p ∈ A

Ví dụ 2.2 : Xét ôtômat hữu hạn đơn định M như sau :

S = { 0, 1 },

Q = { q_0, q_1, q_2, q_3 },

A = { q_0 }.

Hàm δ cho bởi bảng sau :

Trạng thái	*Ký tự đọc vào*	
	0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Cho câu vào *w* = 110101

Quá trình đoán nhận như sau (để đơn giản, chỉ vẽ tượng trưng đầu đọc) :

110101	1**1**0101	11**0**101	110**1**01	1101**0**1	11010**1**	110101
⇧	⇧	⇧	⇧	⇧	⇧	⇧
q_0	q_1	q_0	q_2	q_3	q_1	q_0 : *dừng*.

Vì $q_0 \in A$ nên *w* được thừa nhận. Ta thấy rằng mỗi trạng thái q_i ghi nhớ một tình trạng nhất định của câu vào đã đọc như sau :

q_0 : phần đã đọc gồm một số *chẵn* số 0 và một số *chẵn* số 1 ;

q_1 : phần đã đọc gồm một số *chẵn* số 0 và một số *lẻ* số 1 ;

q_2 : phần đã đọc gồm một số *lẻ* số 0 và một số *chẵn* số 1 ;

q_3 : phần đã đọc gồm một số *lẻ* số 0 và một số *lẻ* số 1 ;

Vì $q_0 \in A$ nên chỉ những câu có một số *chẵn* số 0 và một số *chẵn* số 1 mới được M thừa nhận.

Người ta đưa ra khái niệm *hình trạng* hay *hình trạng* (Configuration) là một cặp phần tử (*q*, *w*), trong đó $q \in Q$ và *w* là phần còn lại của câu vào :

$$(q, w) \in Q \times \Sigma^*.$$

Định nghĩa 2.1 :

Hình trạng (*q*, *w*) chuyển tiếp thành (*q*', *w*'), viết $(q, w) \vdash_M (q', w')$, nếu :

$w = \sigma w'$ ký hiệu đầu tiên của *w* là σ hay *w*' là *w* đã lấy đi ký tự đầu $\sigma \in S$.

$q' = \delta(q, \sigma)$ *q*' là kết quả của hàm chuyển tiếp từ *q* và từ σ.

Định nghĩa 2.2 :

$(q, w) \vdash_M^* (q', w')$ nếu :

$\exists k \geq 0$ và các hình trạng (q_i, w_i), $0 \leq i \leq k$ sao cho :

$(q, w) = (q_0, w_0)$

$(q', w') = (q_k, w_k)$

$\forall i, 0 \leq i < k, (q_i, w_i) \vdash_M (q_{i+1}, w_{i+1})$

Định nghĩa 2.3 :

Ôtômat M đoán nhận câu vào *w* bằng cách thực hiện dãy chuyển tiếp sau :

$$(q_0, w) \vdash_M (q_1, w_1) \vdash_M (q_2, w_2) \vdash_M \ ... \vdash_M (q_n, \varepsilon),$$

trong đó q_0 là trạng thái đầu, ε là câu rỗng.

Định nghĩa 2.4 :

Một câu *w* được thừa nhận bởi ôtômat M nếu $(q0, w) \vdash_M^* (q, \varepsilon)$, $q \in A$.

Ví dụ 2.3 : Cho M = $(q, \Sigma, \delta, q_0, A)$ với $\Sigma = \{ a, b \}$ và *w* = ababa.

Ôtômat thực hiện việc đoán nhận câu *w* như sau :

$(q_0, ababa) \vdash_M (q_1, baba) \vdash_M (q_2, aba) \vdash_M (q_3, ba) \vdash_M (q_4, a) \vdash_M (q_5, \varepsilon)$

Câu vào ababa được thừa nhận, nếu q_5 là trạng thái cuối, $q_5 \in A$.

Từ ví dụ 2.2, với *w*=110101, ta có các chuyển tiếp đoán nhận *w* như sau :

$(q_o, 110101) \vdash_M (q_1, 10101) \vdash_M (q_0, 0101) \vdash_M (q_2, 101)$

$\vdash_M (q_3, 01) \vdash_M (q_1, 1) \vdash_M (q_0, \varepsilon)$

$q_0 \in A$, vậy câu *w* được M thừa nhận.

Ngôn ngữ được thừa nhận bởi ôtômat M là tập hợp các câu được thừa nhận bởi M, ký hiệu *L*(M).

Định nghĩa 2.5 :

Ngôn ngữ được thừa nhận bởi M là tập hợp các câu thuộc *L*(M) sao cho :

$L(M) = \{ w \in \Sigma^* \mid (q_0, w) \vdash_M^* (q, \varepsilon)$ với $q \in A \}$

Ví dụ 2.4 :

Cho $M = (\{ q_0, q_1 \}, \{ a, b \}, q_0, \{ q_1 \})$,

trong đó hàm σ được cho như sau :

σ	*q*	δ (*q*, σ)
q_o	*a*	q_o
q_o	*b*	q_1
q_1	*a*	q_o
q_1	*b*	q_1

Có thể viết dạng hàm :

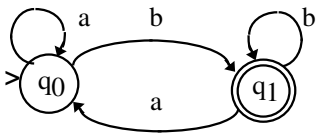
$\delta (q_o, a) = q_o$

$\delta (q_o, b) = q_1$

$\delta (q_1, a) = q_o$

$\delta (q_1, b) = q_1$

Sơ đồ :

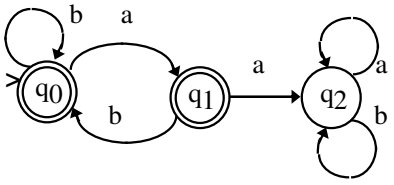


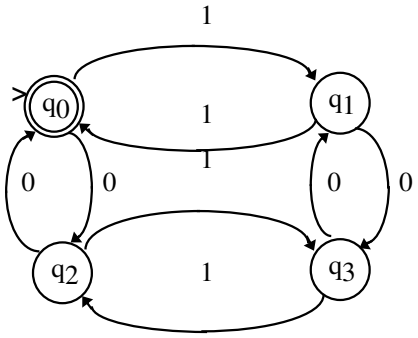
M thừa nhận các câu kết thúc bởi *b*. Ví dụ câu aaabbb = $a^3b^3 \in L(M)$.

Ví dụ 2.5 :

$M = (\{ q_0, q_1, q_2 \}, \{ a, b \}, q_0, \{ q_0, q_1 \})$ có sơ đồ cho ở hình dưới bên trái.

L(M) = { *w* | *w* không chứa hai chữ *a* liên tiếp }. Ví dụ câu bbaba ∈ *L*(M).





Ôtômat cho bởi ví dụ 2.2 ở trên có thể cho bởi sơ đồ ở hình bên phải.

II. Ôtômat hữu hạn không đơn định

II.1. Mô tả

Các ôtômat hữu hạn không đơn định hay không đơn định (NFA - Non-deterministic Finite-State Automata) là các ôtômat hữu hạn trong đó :

4. Nhiều chuyển tiếp ứng với cùng một ký tự cho mỗi trạng thái.
5. Tồn tại các chuyển tiếp cho câu rỗng (nghĩa là đầu đọc không tiến tới khi đọc câu vào).
6. Các chuyển tiếp ứng với các từ có độ dài lớn hơn 1 (nhóm các chuyển tiếp).

Một ôtômat hữu hạn không đơn định được định nghĩa hình thức giống ôtômat hữu hạn đơn định, sự khác nhau thể hiện ở *quan hệ chuyển tiếp* (transition relation) thay vì hàm chuyển tiếp.

Một cách hình thức, ôtômat hữu hạn không đơn định là bộ năm :

$M = (Q, \Sigma, \Delta, q_0, A)$ trong đó :

- Q là tập hợp các trạng thái
- Σ là bảng chữ
- $\Delta \subset (Q \times \Sigma^* \times Q)$ là quan hệ chuyển tiếp
- $q_0 \in Q$ là trạng thái đầu
- $A \subseteq Q$ là tập hợp trạng thái thừa nhận.

Với ôtômat hữu hạn không đơn định, quan hệ chuyển tiếp Δ thay thế hàm chuyển tiếp δ. Với mỗi trạng thái, có thể có nhiều trạng thái tiếp theo cho mỗi ký tự đọc vào, chẳng hạn $\delta(q, a)$ cho q_1 và q_2.

Người ta dùng bộ *ba* (p, σ, q) để nói rằng nếu ký tự đọc vào là σ thì có thể chuyển từ trạng thái p sang trạng thái *q*. Mặt khác, tại mỗi thời điểm đầu đọc có thể đọc một câu con u (gồm nhiều ký tự), $u \in \Sigma^*$.

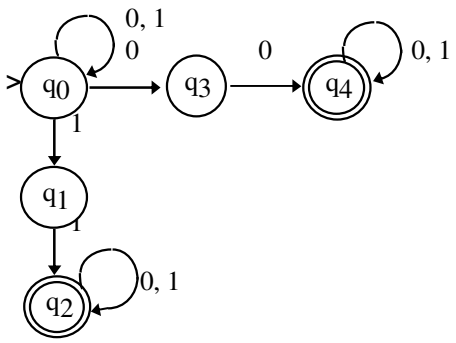
Sự chuyển tiếp cuả ôtômat không đơn định được định nghĩa giống như chuyển tiếp cuả ôtômat đơn định, nhưng lần này việc chuyển tiếp là không duy nhất.

Ôtômat không đơn định thừa nhận một câu vào khi có một chuyển tiếp dẫn đến một trạng thái cuối. Có thể có các chuyển tiếp không dẫn ôtômat đến trạng thái cuối, khi đó, ôtômat bị "hóc", hay không thừa nhận câu vào đã cho.

Ví dụ 2.6 :

$M = (\{ q_0, q_1, q_2, q_3, q_4 \}, \{ 0, 1 \}, \Delta, q_0, \{ q_2, q_4 \})$ với Δ được cho như sau :

	0	1
q_0	$\{ q_0, q_3 \}$	$\{ q_0, q_1 \}$
q_1	$\varnothing$	$\{ q_2 \}$
q_2	$\{ q_2 \}$	$\{ q_2 \}$
q_3	$\{ q_4 \}$	$\varnothing$
q_4	$\{ q_4 \}$	$\{ q_4 \}$



Cho $w = 01001$. Sự đoán nhận xảy ra như sau :

$(q_0, 01001) \vdash_{\mathbf{M}} (q_0, 1001) \vdash_{\mathbf{M}} (q_0, 001) \vdash_{\mathbf{M}} (q_3, 01) \vdash_{\mathbf{M}} (q_4, 1) \vdash_{\mathbf{M}} (q_4, \varepsilon)$

Sau đây chỉ là một quá trình trong số nhiều chuyển tiếp như sau :

$$\vdash_{\mathbf{M}} (q_0, 1) \vdash_{\mathbf{M}} (q_0, \varepsilon)$$
$$\vdash_{\mathbf{M}} (q_0, 01)$$
$$\vdash_{\mathbf{M}} (q_0, 001) \qquad \vdash_{\mathbf{M}} (q_3, 1) \quad \textbf{\textit{hóc !}}$$
$$\vdash_{\mathbf{M}} (q_0, 1001) \qquad \vdash_{\mathbf{M}} (q_3, 01) \vdash_{\mathbf{M}} (q_4, 1) \vdash_{\mathbf{M}} (q_4, \varepsilon)$$
$$(q_0, 01001) \qquad \vdash_{\mathbf{M}} (q_1, 001)$$
$$\vdash_{\mathbf{M}} (q_3, 1001)$$

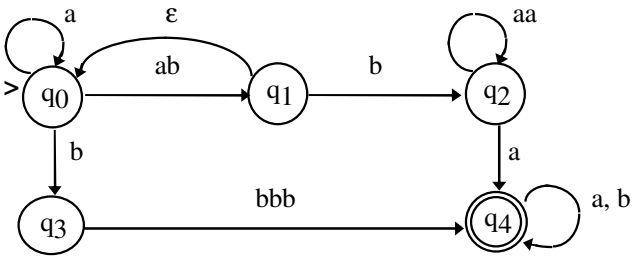
Câu w được thừa nhận vì ôtômat đọc hết câu và dẫn đến trạng thái cuối.

$L(\mathrm{M}) = (0+1)^*00(0+1)^* + (0+1)^*11(0+1)^*$.

Định nghĩa 2.6 : Nói rằng với ô M, $(q, w) \vdash_{\mathbf{M}} (qu', w')$, nếu :

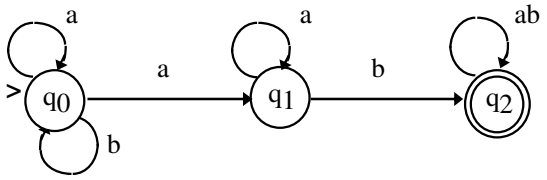
- $w = uw'$: câu w bắt đầu bởi tiền tố $u \in \Sigma^*$;
- $(q, u, q') \in \Delta$.

Ví dụ 2.7 : Ngôn ngữ được thừa nhận bởi ôtômat M cho bởi sơ đồ :



là : $L(M) = ((a \cup ab)^* bbbb\Sigma^*) \cup ((a \cup ab)^* abb(aa)^* a\Sigma^*)$.

Ví dụ 2.8 : Ngôn ngữ được thừa nhận bởi ôtômat M



là : $L(M) = \Sigma^* ab(ab)^*$, nghĩa là tập hợp các câu kết thúc ít nhất một lần lặp *ab*. Ví dụ, câu aaabab $\in L(M)$.

II.2. Khử bỏ tính không đơn định

Trong mục này, ta sẽ chứng minh rằng tính không đơn định không thêm khả năng của các ôtômat hữu hạn. Thực tế có thể thay thế một ôtômat hữu hạn không đơn định bởi một ôtômat hữu hạn đơn định tương đương. Hai ôtômat được gọi là tương đương nếu chúng cùng thừa nhận một ngôn ngữ.

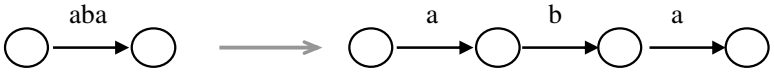
II.2.1. Nguyên tắc xây dựng

Có hai yếu tố phân biệt giữa ôtômat hữu hạn không đơn định (NFA) và một ôtômat hữu hạn đơn định (DFA) :

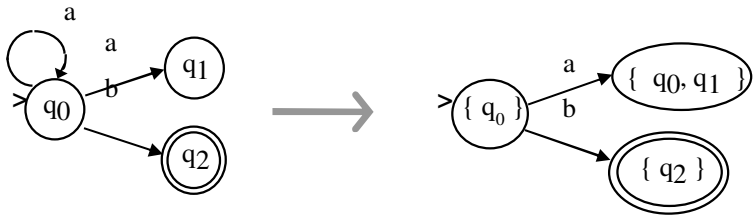
Tồn tại các chuyển tiếp trên các câu con có độ dài lớn hơn1.

Tồn tại các chuyển tiếp cho cùng một trạng thái và cùng một nội dung của băng vào (tính không đơn định).

Để xây dựng một DFA dựa vào một NFA, cần khử bỏ các dịch chyển trên các từ có độ dài >1. Cách xây dựng như sau :



Để loại bỏ tính không đơn định, vấn đề là xây dựng DFA sao cho tại mỗi thời điểm, lưu nhớ tất cả các trạng thái có thể có của NFA. Như vậy, mỗi trạng thái của DFA sẽ tương ứng với một tập hợp các trạng thái của NFA. Cách thức chuyển đổi như sau :



Với trạng thái q_0 và ký tự *a*, NFA có thể chuyển sang một trong hai trạng thái q_0 hoặc q_1. DFA chỉ có thể có một chuyển tiếp ứng với chữ *a*, và do vậy, sẽ chuyển sang trạng thái { q_0, q_1 } ứng với hai trạng thái q_0, q_1 của NFA.

II.2.2. Hình thức hóa việc xây dựng

Việc xây dựng gồm hai giai đoạn.

Giai đoạn 1 : Khử bỏ các chuyển tiếp có độ dài >1.

Giả sử cho NFA M=(Q, Σ, Δ, q_0, f).

Vấn đề là cần xây dựng M' = (Q', Σ', Δ', q_0, A) không chứa các chuyển tiếp có độ dài lớn hơn 1, nghĩa là ∀ (q, u, q') ∈ Δ', |u| ≤ 1. Tập hợp các trạng thái Q' và các chuyển tiếp Δ' của M' nhận được từ M như sau :

Đầu tiên Q' = Q và Δ' = Δ.

Với mỗi chuyển tiếp (q, u, q') ∈ Δ mà u = $\sigma_1\sigma_2$...., σ_k (k>1) :

- Loại bỏ chuyển tiếp này khỏi Δ' ;
- Thêm vào Q' các trạng thái mới $p_1, p_2, ..., p_{k-1}$;
- Thêm các chuyển tiếp mới (q, σ_1, p_1), (p_1, σ_2, p_2), . . ., (p_{k-1}, σ_k, q') vào D'.

Giai đoạn 2 : Khử bỏ tính không đơn định.

Cho M = (Q, Σ, Δ, q_0, A) mà mỗi chuyển tiếp (q, u, q') ∈ Δ, |u| ≤ 1, nghĩa là mỗi chuyển tiếp của NFA M có dạng hoặc (q, σ, q'), σ ∈ Σ, hoặc (q, ε, q'). Xây dựng DFA M' = (Q', Σ, δ', q_0', A') tương đương với M.

Việc xử lý các câu rỗng ε dạng (q, ε, q') sẽ gặp khó khăn. Thực tế, một DFA không có các chuyển tiếp cho câu ε. Vì vậy, cần nhóm tất cả các chuyển tiếp cho câu rỗng của M với một chuyển tiếp trên một phần tử cuả Σ.

Ta có định nghĩa sau đây :

Định nghĩa 2.8 :

Với mỗi trạng thái q của ôtômat M, E(q) là tập hợp các trạng thái có thể đạt được từ q bởi một dãy chuyển tiếp trên câu rỗng :

$E(q) = \{ p \in Q \mid (q, w) \vdash_{\mathbf{M}}^{*} (p, w) \}$.

Như vậy, với q đã cho, tập hợp E(q) gồm các trạng thái p mà trong sơ đồ biểu diễn ôtômat, tồn tại một đường đi từ q sang p có nhãn là câu rỗng ε.

Xây dựng các phần tử của M' như sau :

- Tập hợp các trạng thái Q' cuả M' là tập hợp của các tập con các trạng thái của M : Q = 2^Q.
- Mỗi trạng thái của DFA tương ứng với một tập hợp các trạng thái của NFA.
- Trạng thái đầu q_0' của M' phải biểu diễn các trạng thái mà từ đó, M có thể tiến hành đọc câu vào. Đó là các trạng thái đạt được từ trạng thái đầu q_0 của M bởi các chuyển tiếp trên câu rỗng. Như vậy q_0' = E(q_0).

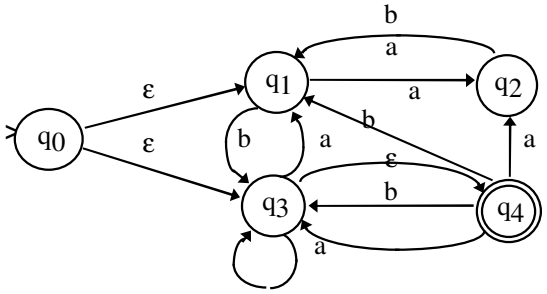
Hàm chuyển tiếp δ' phải nhóm hết các trạng thái của M. Với mỗi q ∈ M' (q là tập hợp các q_i ∈ M), hàm δ'(q, a) gồm tập hợp các trạng thái mà trong M có thể đạt được từ một trong các q_i ∈ q bởi một dãy các chuyển tiếp với chuyển tiếp đầu tiên có nhãn là a và các chuyển tiếp tiếp theo trên câu rỗng ε :

- $\delta'(q, a) = \cup \{ E(p) \mid \exists q \in q : (q, a, p) \in \Delta \}$

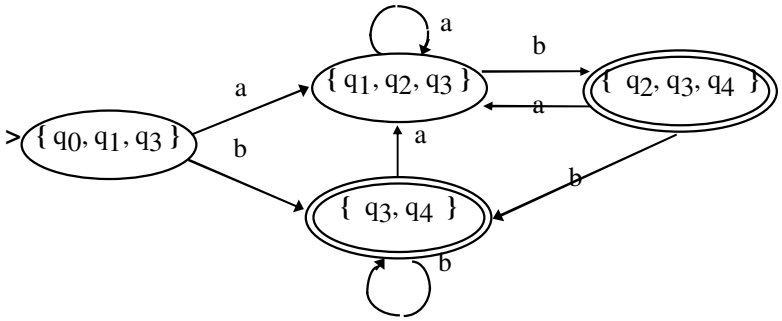
Trạng thái cuối của M' sẽ chứa một trạng thái cuối của M. Như vậy, nếu M' đang ở trạng thái q, M có thể đang ở một trạng thái bất kỳ $q \in q$. Định nghĩa A như sau :

$A' = \{ \boldsymbol{q} \in Q' \mid \boldsymbol{q} \cap A \neq \varnothing \}$.

Ví dụ 2.9 : Cho ôtômat hữu hạn không đơn định M như sau :



Ôtômat hữu hạn đơn định tương đương như sau :



Từ M, có thể xây dựng 2^5 trạng thái cho M**'**. Ở đây chỉ biểu diễn những trạng thái "đạt được" xuất phát từ trạng thái đầu của M**'**.

Như vậy, một trạng thái p là đạt được từ một trạng thái *q* nếu tồn tại một chuyển tiếp dẫn p đến *q*.

Định nghĩa 2.9 :

Một trạng thái p là đạt được từ *q* nếu tồn tại một câu *w* sao cho :

$$(q, w) \vdash_{\mathbf{M}}^{*} (p, \varepsilon).$$

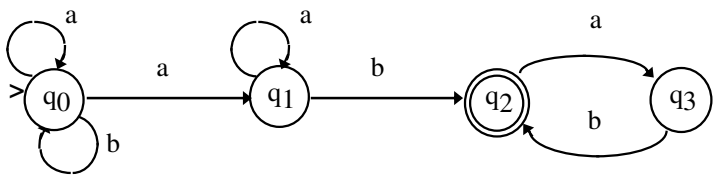
Trong sơ đồ biểu diễn ôtômat M, p đạt được từ *q* nếu tồn tại một đường đi từ *q* đến p. Một trạng thái của ôtômat không là đạt được từ trạng thái đầu sẽ không xuất hiện trong ôtômat.

Như vậy, khi xây dựng DFA M**'** từ NFA M sẽ tồn tại những trạng thái không đạt được.Vấn đề đặt ra là làm thế nào đó để Q**'** chỉ chứa toàn các trạng thái đạt được ? Cách xây dựng là tạo ra lần lượt các phần tử của Q**'** như sau :

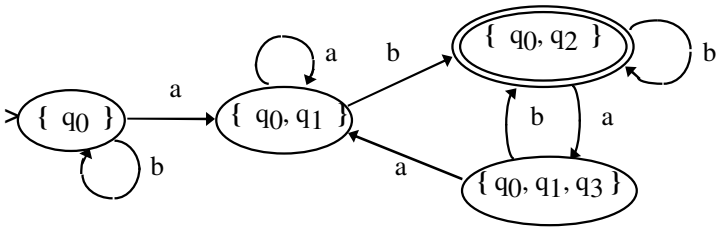
1. Lúc đầu, Q' chứa trạng thái đầu q0'.
2. Các bước sau đây được lặp lại cho đến khi không còn làm thay đổi Q**'** :
   a) Chọn một trạng thái $\boldsymbol{q} \in$ Q**'** sao cho bước *b* chưa được thực hiện.
   b) Với mỗi $a \in \Sigma$, tìm **p** sao cho **p** = δ**'** (***q***, *a*), **p** được thêm vào Q**'**.

Ví dụ 2.10 : Xây dựng DFA từ NFA M cho ở ví dụ 2.8 như sau :

Khử bỏ các chuyển tiếp có độ dài > 1 bằng cách thêm trạng thái mới q_3.



DFA đơn định tương đương :



II.2.3. Tính đúng đắn của phương pháp

Ta cần chứng minh rằng một câu được thừa nhận bởi NFA M thì cũng được thừa nhận bởi DFA M' và ngược lại. Có nghĩa rằng với mọi $w \in \Sigma^*$, nếu tồn tại một đoán nhận của M thừa nhận *w*, thì cũng tồn tại một đoán nhận của M' thừa nhận *w* và ngược lại.

Chứng minh "⇐"

Ta hãy xét một câu tùy ý $w = \boldsymbol{\delta_1\delta_2 \ldots \delta_k}$ và giả thiết rằng tồn tại một đoán nhận của M' cho *w*. Đó là dãy các chuyển tiếp của M' :

$$(E(q_0), w) \vdash_{\mathbf{M}'} (\mathbf{q}_1, w_1) \vdash_{\mathbf{M}'} (\mathbf{q}_2, w_2) \vdash_{\mathbf{M}'} \ldots \vdash_{\mathbf{M}'} (\mathbf{q}_k, \boldsymbol{\sigma}_k) \vdash_{\mathbf{M}'} (\mathbf{p}, \varepsilon)$$

Với $\mathbf{p} \in A$. Ta cần chỉ ra rằng tồn tại một đoán nhận của M cho *w*.

Theo cách xây dựng A', nếu $\mathbf{p} \in A'$ thì tồn tại $p \in Q$ sao cho đồng thời $p \in \mathbf{p}$ và $p \in A$. Hơn nữa vì $p \in \mathbf{p}$ và theo cách xây dựng hàm chuyển tiếp $\boldsymbol{\delta}$ thì :

$$(\mathbf{q_k}, \boldsymbol{\sigma_k}) \vdash_{\mathbf{M}'} (\mathbf{p}, \varepsilon)$$

Sẽ tồn tại $q_k \in Q$ và $r_k \in Q$ sao cho $q_k \in \mathbf{q}_k$, $p \in E(r_k)$ và $(q_k, \boldsymbol{\sigma}_k, r_k) \in \boldsymbol{\Delta}$.

Từ đó ta có : $(q_k, \boldsymbol{\sigma}_k) \vdash^*_{\mathbf{M}} (p, \varepsilon)$. Như vậy, tồn tại một đoán nhận của M, xuất phát từ trạng thái q_k, thừa nhận câu δ_k.

Bằng cách lý luận tương tự cho chuyển tiếp :

$(\mathbf{q}_{k-1}, \boldsymbol{\sigma}_{k-1}) \vdash_{\mathbf{M}'} (\mathbf{q}_k, \boldsymbol{\sigma}_k)$.

Ta suy ra rằng tồn tại các trạng thái $q_{k-1} \in Q$ và $r_{k-1} \in Q$ sao cho $q_{k-1} \in \mathbf{q}_{k-1}$, $q_k \in E(r_{k-1})$ và $(q_{k-1}, \sigma_{k-1}, r_{k-1}) \in \Delta$.

Cứ tiếp tục như vậy, cuối cùng ta đi đến kết luận rằng tồn tại một đoán nhận của M có dạng :

$$(q_0, w) \vdash_{\mathbf{M}}^{*} \cdots \vdash_{\mathbf{M}}^{*} (q_{k-1}, \sigma_{k-1}, \sigma_k) \vdash_{\mathbf{M}}^{*} (q_k, \sigma_k) \vdash_{\mathbf{M}}^{*} (p, \varepsilon)$$

Như vậy, $w \in L(M)$.

Chứng minh "⇒"

Bây giờ giả thiết rằng tồn tại một đoán nhận của M thừa nhận *w*. Có thể viết quá trình chuyển tiếp của M dưới dạng sau :

$$\underbrace{(q_0, w) \vdash_{\mathbf{M}} (q_{0_1}, w) \vdash_{\mathbf{M}} \cdots \vdash_{\mathbf{M}} (q_{0n_0}, w)}_{q_0, q_{0_1}, \cdots, q_{0n_0} \in E(q_0) = \mathbf{q}_0}$$

$$\vdash_{\mathbf{M}} \underbrace{(q_{1_1}, w_1) \vdash_{\mathbf{M}} (q_{1_2}, w_1) \vdash_{\mathbf{M}} \cdots \vdash_{\mathbf{M}} (q_{1n_1}, w_1)}_{q_{1_1}, q_{1_2}, \cdots, q_{1n_1} \in \delta(E(q_0), \sigma_1) = \mathbf{q}_1}$$

$$\vdash_{\mathbf{M}} \underbrace{(q_{2_1}, w_2) \vdash_{\mathbf{M}} (q_{2_2}, w_2) \vdash_{\mathbf{M}} \cdots}_{q_{2_1}, q_{2_2}, \cdots, q_{2n_2} \in \delta(\mathbf{q}_1, \sigma_2) = \mathbf{q}_2} \underbrace{\vdash_{\mathbf{M}} (p, \varepsilon)}_{p \in \delta(\mathbf{q}_k, \sigma_k) = \mathbf{p}}$$

Trong các dãy chuyển tiếp trên, ta đã nhóm mỗi chuyển tiếp trên một ký tự của Σ với tất cả các chuyển tiếp trên câu rỗng đi liền theo trực tiếp.

Như thế, ta đã cho M' bắt chước sự đoán nhận của M :

$$(E(q_0), w) \vdash_{\mathbf{M}}' (\mathbf{q}_1, w_1) \vdash_{\mathbf{M}}' (\mathbf{q}_2, w_2) \vdash_{\mathbf{M}}' \cdots \vdash_{\mathbf{M}}' (\mathbf{q}_k, \sigma_k) \vdash_{\mathbf{M}}' (\mathbf{p}, \varepsilon)$$

Định lý đã được chứng minh (qed).

II.3. Ôtômat hữu hạn và các biểu thức chính qui

Chúng ta đã nghiên cứu hai hình thức mô tả ngôn ngữ là các biểu thức chính qui và các ôtômat hữu hạn (đơn định và không đơn định). Hai hình thức này về bản chất là khác nhau.

Xuất phát từ các tập hợp cơ sở (gồm $\varnothing$, $\{ \varepsilon \}$ và $\{ a \}$, $a \in \Sigma$) và các phép toán trên các tập hợp này, *các biểu thức chính qui chỉ định các câu thuộc ngôn ngữ*. Còn *các ôtômat hữu hạn* thì lại *định nghĩa các ngôn ngữ bằng cách đoán nhận chúng*.

Vấn đề đặt ra là so sánh các ngôn ngữ được định nghĩa bởi hai hình thức trên. Định lý sau cho thấy một ngôn ngữ được chỉ định bởi một biểu thức chính quy thì cũng được thừa nhận bởi một ôtômat hữu hạn và ngược lại.

Định lý 2.2 :

Một ngôn ngữ là chính qui nếu và chỉ nếu ngôn ngữ đó được thừa nhận bởi một ôtômat hữu hạn.

Để chứng minh, cần sử dụng tính chất đã chưng minh trong định lý 1.1 : *một ngôn ngữ là chính qui khi và chỉ khi ngôn ngữ đó được chỉ định bởi một biểu thức chính qui.*

Mặt khác, theo định lý 2.1 thì chỉ cần làm việc với các ôtômat hữu hạn không đơn định.

Hai bổ đề sau đây cho phép chứng minh định lý 2.2 :

Bổ đề 2.1 : Nếu một ngôn ngữ được chỉ định bởi một biểu thức chính qui, thì ngôn ngữ đó được thừa nhận bởi một ôtômat hữu hạn không đơn định.

Bổ đề 2.2 : Một ngôn ngữ được thừa nhận bởi một ôtômat hữu hạn không đơn định là chính qui.

II.3.1. Xây dựng các ôtômat từ các biểu thức chính qui

Giả thiết rằng ngôn ngữ đang xét được chỉ định bởi một biểu thức chính qui, vấn đề là làm sao xây dựng được một ôtômat hữu hạn không đơn định thừa nhận ngôn ngữ này.

Sử dụng định nghĩa biểu thức chính qui, ta hãy chỉ ra rằng :

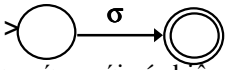
- Với mỗi biểu thức chính qui cơ sở ($\varnothing$, ε, $a \in \Sigma$), tồn tại một ôtômat hữu hạn thừa nhận ngôn ngữ do biểu thức đó chỉ định ;
- Với mỗi biểu thức chính qui dạng $(\alpha_1\alpha_2)$, $(\alpha_1 \cup \alpha_2)$, và $((\alpha_1)^*)$, tồn tại một ôtômat hữu hạn thừa nhận ngôn ngữ do biểu thức đó chỉ định, ôtômat này nhận được từ các ôtômat thừa nhận các ngôn ngữ chỉ định lần lượt bởi các biểu thức α_1 và α_2.

Trước hết, ta xây dựng các ôtômat thừa nhận các ngôn ngữ do các biểu thức chính qui cơ sở chỉ định. Ta xây dựng cụ thể cho $\varnothing$, ε và $\forall \sigma \in \Sigma$ như sau :

Với $\varnothing$, ôtômat sẽ là :

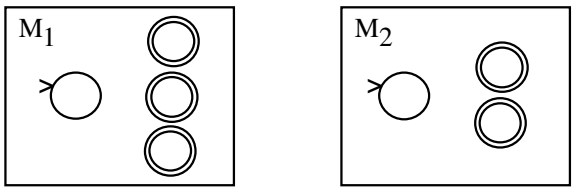
Với ε, ôtômat như sau :

Cuối cùng, ôtômat tương ứng với $\sigma \in \Sigma$ là :

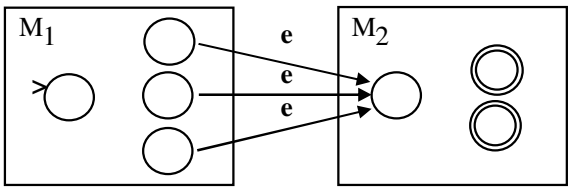


Bây giờ, giả sử đã xây dựng xong các ôtômat M_1 và M_2 tương ứng với các biểu thức chính qui α_1 và α_2. Để đơn giản cách trình bày, M_1 và M_2 sẽ chỉ gồm trạng thái đầu và các trạng thái cuối :

$M_1 = (Q_1, \Sigma, D_1, s_1, A_1)$ $M_2 = (Q_2, \Sigma, D_2, s_2, A_2)$



Trường hợp $\alpha = \alpha_1\alpha_2$:



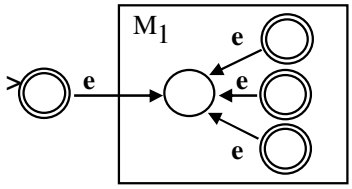
Ôtômat M tương ứng với biểu thức chính qui $\alpha_1\alpha_2$ nhận được bằng cách nối các trạng thái đạt được của M_1 tới trạng thái đầu của M_2 bởi các chuyển tiếp trên câu rỗng ε. Trạng thái đầu của M cũng là trạng thái đầu của A_1 và các trạng thái cuối của M cũng là trạng thái cuối của A_2.

Một cách hình thức, xây dựng ôtômat $M = (Q, \Sigma, \Delta, q_0, A)$, trong đó :

$$Q = Q_1 \cup Q_2 ; \quad \Delta = \Delta_1 \cup \Delta_2 \cup \{ (q, \varepsilon, s_2) \mid q \in A_1 \} ;$$
$$q_0 = s_1 ; \quad A = F_2.$$

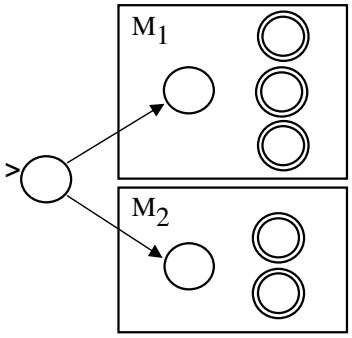
Trường hợp $\alpha = (\alpha_1)^*$:

Để có phép đóng lặp $(\alpha_1)^*$, xây dựng M từ M_1 bằng cách thêm vào M một trạng thái đầu mới. Trạng thái đầu q_0 này đồng thời cũng là trạng thái cuối vì có thể $\varepsilon \in L(\alpha_1{}^*)$. Có thể xem rằng $\varepsilon \notin L(M_1)$.



Trường hợp $\alpha = \alpha_1 \cup \alpha_2$:

Dễ thấy rằng việc sử dụng các ôtômat không đơn định chứng minh định lý 2.1 đơn giản hơn so với các ôtômat đơn định.



II.3.2. Xây dựng các ngôn ngữ chính quy từ các ôtômat

Cho ôtômat M, cần xây dựng một biểu thức chính quy chỉ định *L*(M). Lời giải có thể như sau :

1. Xem xét các đường đi từ trạng thái đầu đến một trong các trạng thái cuối của M.
2. Để nhận được một biểu thức chính quy tương ứng với mỗi một đường đi, cộng liên tiếp các nhãn của các chuyển tiếp. Các vòng lặp được biểu diễn bởi phép toán lặp (phép *).
3. Biểu thức chính quy mong muốn chính là hợp (Union) của các biểu thức chính quy nhận được từ 1. và 2.

Lời giải trên đây có vẻ chấp nhận được, tuy nhiên còn một vấn đề chưa rõ ràng là việc xử lý các vòng lặp. Cần phải xây dựng một biểu thức chính quy tương ứng với các đường đi nối hai trạng thái có vòng lặp.

Để giải quyết, chúng ta sử dụng phương pháp quy nạp.

Giả sử M có $Q = \{ q_1, q_2, ..., q_n \}$.

Để xây dựng tập hợp các câu cho phép đi từ trạng thái q_i đến một trạng thái q_j nào đó, ta hãy xét các câu cho phép chuyển từ q_i đến q_j, đầu tiên không có trạng thái trung gian, sau đó bởi chỉ có mỗi q_1, rồi bởi chỉ có q_1 và q_2, v.v...

Định nghĩa 2.9 :

Cho ôtômat M với $Q = \{ q_1, q_2, ..., q_n \}$ là tập hợp các trạng thái. Đặt R(i, j, k) là tập hợp các câu cho phép chuyển từ trạng thái q_i đến q_j bằng cách chỉ chuyển qua k-1 trạng thái $\{ q_1, q_2, ..., q_{k-1} \}$.

Bây giờ ta tìm cách định nghĩa R(i, j, k). Với k = 1, chỉ cần xét những câu xuất hiện trong các chuyển tiếp từ q_i đến q_j và câu rỗng ε nếu i = j.

Như vậy ta có:

$$R(i, j, 1) = \begin{cases} \{ w \mid (q_i, w, q_j) \in \Delta \} & \text{nếu } i \neq j \\ \{ \varepsilon \} \cup \{ w \mid (q_i, w, q_j) \in \Delta \} & \text{nếu } i = j \end{cases}$$

Với k>1, ta dùng phương pháp quy nạp.

Những câu xuất hiện trong các chuyển tiếp từ q_i đến q_j mà chỉ sử dụng các trạng thái trong số $\{ q_1, q_2, ..., q_k \}$ là :

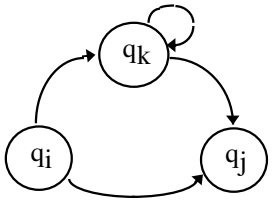
- Những câu tương ứng với chuyển tiếp từ q_i đến q_j mà chỉ sử dụng k–1 trạng thái $\{ q_1, q_2, ..., q_{k-1} \}$, tức là R(i, j, k).
- Những câu tương ứng với chuyển tiếp từ q_i đến q_k, rồi q_k đến q_k một số lần, và cuối cùng, từ q_k đến q_j, tất cả chỉ sử dụng $\{ q_1, q_2, ..., q_{k-1} \}$.

Ta có : $R(i, j, k+1) = R(i, j, k) \cup R(i, k, k)\, R(k, k, k)^* R(k, j, k)$.

Cuối cùng, ngôn ngữ thừa nhận bởi M là hợp của tất cả các tập hợp câu, xuất hiện trong các chuyển tiếp từ trạng thái đầu đến một trong những trạng thái cuối, bằng cách vượt qua tất cả các trạng thái của M.

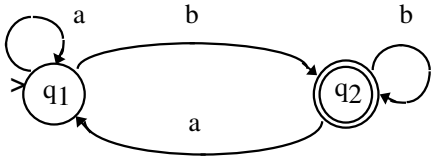
Giả thiết rằng trạng thái đầu là q_1 và M gồm n trạng thái, ta có :

$$L(M) = \bigcup_{q_j \in A} R(1, j, n+1).$$



Để định nghĩa các ngôn ngữ R(i, j, k), ở đây chỉ sử dụng những tập hợp hữu hạn câu, và các phép toán hợp, ghép (concatenation) và đóng lặp. Từ đó, nhận được ngôn ngữ chính quy *L*(M).

Ví dụ 2.11 : Cho ôtômat M như sau :



Với k = 1 và k = 2, các ngôn ngữ R(i, j, k) được định nghĩa như sau :

	k = 1	k = 2
R(1, 1, k)	$\varepsilon \cup a$	$(\varepsilon \cup a) \cup (\varepsilon \cup a)\,(\varepsilon \cup a)^*\,(\varepsilon \cup a)$
R(1, 2, k)	b	$b \cup (\varepsilon \cup a)\,(\varepsilon \cup a)^* b$
R(2, 1, k)	a	$a \cup a(\varepsilon \cup a)^*\,(\varepsilon \cup a)$
R(2, 2, k)	$\varepsilon \cup b$	$(\varepsilon \cup b) \cup a(\varepsilon \cup a)^*\, b$

Ngôn ngữ được thừa nhận bởi M là R(1, 2, 3) như sau :

$[b \cup (\varepsilon \cup a)\ (\varepsilon \cup a)^* b] \cup$

$[b \cup (\varepsilon \cup a)\ (\varepsilon \cup a)^* b]\ [(\varepsilon \cup b) \cup a(\varepsilon \cup a)^*\ b]^*\ [\ (\varepsilon \cup b) \cup a(\varepsilon \cup a)^*\ b].$

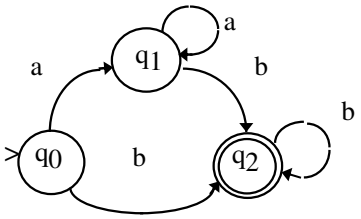
Bài tập chương 2

1. Xây dựng các NFA thừa nhận các ngôn ngữ sau đây :
- Biểu diễn nhị phân các số chẵn.
- Các câu trên bảng chữ **S = {** *a*, *b* **}** chứa aab hoặc aaab.

2. Xây dựng các DFA thừa nhận các ngôn ngữ đã cho trong bài tập 1 ở trên.

3. Xây dựng các ôtômat thừa nhận các biểu thức chính quy :
- $a^* b$.
- $\varepsilon \cup (a \cup aab)^*$.

4. Xây dựng các biểu thức chính quy từ ôtômat sau đây :



CHƯƠNG 3

Các văn phạm chính quy

I. Mở đầu

Chương này trình bày các văn phạm (Grammar) là tập hợp các quy tắc cho phép xây dựng các câu đúng của một ngôn ngữ.

Ví dụ 3.1 : Giả sử các câu tiếng Anh được xây dựng theo những quy tắc sau :

- Một *câu* (Phrase hay Sentence) gồm có *chủ từ* (Subject) và *động từ* (Verbe).
- Một *chủ từ* có thể là *He* hoặc *She*.
- Một *động từ* có thể là *sleep* hay *eat*.

Từ đó có thể xây dựng được các câu : *He sleep. He eat. She sleep. She eat.*

Như đã biết, trong các ngôn ngữ lập trình, người ta thường sử dụng dạng Backus-Naur (BNF – Backus Naur Form), hoặc dạng Backus-Naur mở rộng (EBNF – Extended Backus-Naur Form) để mô tả cấu trúc cú pháp (syntax structure), bao gồm một dãy các quy tắc, hay dạng thức.

Mỗi quy tắc gồm *vế trái*, *dấu định nghĩa* ::= (đọc *được định nghĩa bởi*) và *vế phải*. Vế trái là một ký hiệu phải được định nghĩa, còn vế phải là một dãy các ký hiệu, hoặc được thừa nhận, hoặc đã được định nghĩa từ trước đó, tuân theo một quy ước nào đó.

Ví dụ 3.1 : Văn phạm của một ngôn ngữ lập trình đơn giản dạng EBNF như sau :

```
<program>    ::= program <statement>* end
<statement>  ::= <assignment> | <loop>
<assignment> ::= <identifier> := <expression> ;
<loop>       := while <expression> do <statement>+ done
<expression> ::= <value> | <value> + <value> | <value> <= <value>
<value>      ::= <identifier> | <number>
<identifier> ::= <letter> | <identifier><letter> |
                          <identifier><digit>
<letter>     ::= 'A' | ... | 'Z' | 'a' | ... | 'z'
<digit>      ::= '0' | ... | '9'
<number>     ::= <digit> | <number><digit>
```

Một câu, tức là một chương trình đơn giản, chẳng hạn :

```
program
   n := 1 ;
   while n <= 10 do n := n + 1 ; done
end
```

được sản sinh từ văn phạm đã cho như sau :

```
<program>
program <statement>* end
program <statement> <statement> end
program <assignment> <loop> end
program <identifier> := <expression> ; while <expression> do
    <statement>+ done end
program n := <value> ; while <value> <= <value> do <statement> done end
program n := <number> ; while <identifier> <= <number> do <assignment>
    done end
program n := 1 ; while n <= 10 do <identifier> := <expression> ; done
    end
program n := 1 ; while n <= 10 do n := <value> + <value> ; done end
program n := 1 ; while n <= 10 do n := <identifier> + <number> ; done
    end
program n := 1 ; while n <= 10 do n := n + 1 ; done end
```

Dạng BNF cũng được sử dụng để mô tả một văn phạm ngôn ngữ tự nhiên theo nghĩa đã xét. Chẳng hạn với ví dụ 3.1, ta có thể viết :

<Câu> ::= <Chủ từ> <Động từ>
<Chủ từ> ::= He | She
<Động từ> ::= *Sleep* | *Eat*

Để nhận được câu *He sleep*, cần áp dụng lần lượt các quy tắc, bắt đầu từ <Câu> như sau :

<Câu>
<Chủ từ> <Động từ>
He sleep

Trong các NNLT, người ta sử dụng mô tả sản sinh (văn phạm) để viết một chương trình và mô tả phân tích (trình biên dịch) khi cần phân tích cú pháp (vào thời điểm biên dịch).

Trước đây, ta đã xét các ôtômát cho phép phân tích một ngôn ngữ bằng cách thừa nhận các phần tử (các câu) của ngôn ngữ. Như vậy cả hai phương pháp sử dụng các ôtômát để mô tả ngôn ngữ và sử dụng các văn phạm để xây dựng các câu đúng của một ngôn ngữ là bổ sung cho nhau và thông thường, người ta sử dụng cả hai dạng thức này.

Trong chương này, chúng ta sẽ trình bày cách sản sinh các ngôn ngữ chính quy (khác với biểu thức chính quy) và chỉ ra sự tương ứng giữa mô tả sản sinh và mô tả phân tích. Nghĩa là có thể biến đổi một mô tả phân tích (một ôtômát hữu hạn) thành một mô tả sản sinh và ngược lại.

Lý thuyết về các ngôn ngữ mà chúng ta nghiên cứu ở đây chưa đủ để mô tả đầy đủ các ngôn ngữ tự nhiên (chẳng hạn tiếng Anh, tiếng Pháp) nhưng ít ra cũng đầy đủ và được áp dụng rộng rãi cho việc mô tả và phân tích các ngôn ngữ lập trình.

II. Các văn phạm

II.1. Định nghĩa

Các quy tắc để sản sinh câu còn được gọi là các *quy tắc viết lại* (Rewrite Rules). Mỗi quy tắc chỉ ra một dãy các ký hiệu (Symbols) có thể được thay thế bởi một dãy các ký hiệu khác. Để nhận được các câu khác nhau, người ta xuất phát từ một ký tự đặc biệt, gọi là ký tự đầu (Start Symbol), rồi áp dụng lần lượt các quy tắc của văn phạm.

Định nghĩa 3.1 : Một văn phạm là một bộ bốn G = (V, ∑, R, S) trong đó :

- V là bảng chữ gồm một tập hợp hữu hạn các ký tự.
- $\sum \subseteq V$ là tập hợp các ký tự kết thúc, hay ký tự cuối (Terminal Symbols).
- $V - \sum$ là tập hợp các ký tự không kết thúc (Non-Terminal), hay còn gọi là các biến, được ký hiệu bởi N, các ký tự này chỉ xuất hiện trong quá trình sản sinh và không xuất hiện trong các câu đã được văn phạm sinh ra.
- $R \subseteq (V^+ \times V^*)$ là tập hữu hạn các quy tắc (Rules), hay còn gọi là các sản xuất (Productions), chính là các quy tắc viết lại vừa nói ở trên.

Ý nghĩa : Mỗi sản xuất dạng (a, b) cho biết phần tử bên trái ($a \in V^+$) của sản xuất được thay thế bởi phần tử bên phải ($b \in V^*$).

- $S \in V - \sum$ là ký tự đầu (Start Symbol).

Từ S, bắt đầu quá trình sản sinh ngôn ngữ bằng cách áp dụng các sản xuất. Sau đây là một số quy ước khi mô tả văn phạm G :

- Các ký tự thuộc $N = V - \sum$ được biểu diễn bởi A, B, C...
- Các ký tự thuộc ∑ được biểu diễn bởi *a*, *b*, c...
- Các quy tắc, hay sản xuất, viết $(\alpha, \beta) \in R$, được biểu diễn bởi :
  $\alpha \rightarrow \beta$ hay $\alpha \rightarrow_G \beta$
  nếu như muốn chỉ định đó là sản xuất thuộc văn phạm G.
- Ký tự đầu luôn luôn biểu diễn bởi S.
- Các câu rỗng được biểu diễn bởi ε.

Ví dụ 3.2 : Sau đây là một văn phạm G = (V, ∑, R, S) với :

V = { S, A, B, *a*, *b* }, ∑ = { *a*, *b* }, S là ký tự đầu.

$R = \{ S \rightarrow A, S \rightarrow B, B \rightarrow bB, B \rightarrow \varepsilon, A \rightarrow aA, A \rightarrow \varepsilon, \}$

Để cho gọn, khi mô tả một văn phạm, người ta thường nhóm các quy tắc có cùng ký tự bên trái với nhau, và thay vì liệt kê hết các thành phần của văn phạm, người ta chỉ liệt kê các quy tắc của văn phạm mà thôi.

Ví dụ văn phạm G đã cho được mô tả bởi :

$R = \{ S \rightarrow A \mid B, B \rightarrow bB \mid \varepsilon, A \rightarrow aA \mid \varepsilon \}$

Bằng cách áp dụng liên tiếp các sản xuất trong R, ta nhận được các câu sản sinh bởi văn phạm G. Ví dụ, *aaaa* là một câu do G sinh ra bằng cách áp dụng lần lượt các sản xuất :

S		
A	Áp dụng quy tắc	$S \rightarrow A$
aA	–	$A \rightarrow aA$
aaA	–	$A \rightarrow aA$
aaaA	–	$A \rightarrow aA$
aaaaA	–	$A \rightarrow aA$
aaaa	–	$A \rightarrow \varepsilon$

Định nghĩa 3.2 :

Cho văn phạm G = (V, ∑, R, S) và u∈ V^+, v∈ V^* được gọi là các dạng câu.

– Câu v được sản sinh từ u bằng một bước suy dẫn trong G (derives v from u in one step) được biểu diễn bởi :

$u \Rightarrow_G v$

nếu và chỉ nếu :

- u = xu'y u gồm 3 phần x, u' và y, các phần x và y có thể rỗng.
- v = xv'y v gồm 3 phần x, v' và y.
- u' → v' qui tắc (u', v') ∈ R.

– Câu v được sản sinh từ u bằng nhiều bước suy dẫn trong G (derives v from u in several steps) được biểu diễn bởi :

$u \overset{*}{\Rightarrow}_G v$

nếu và chỉ nếu ∃ k ≥ 0 và $v_0 ... v_k \in V^+$ sao cho :

- $u = v_0$
- $v = v_k$
- $v_i \Rightarrow G \quad v_{i+1}$ với ∀i, 0 < i ≤ k.

Các câu do văn phạm G sinh ra là các câu nhận được bởi quá trình suy dẫn từ ký tự đầu S cho đến khi nhận được câu gồm các ký tự kết thúc thuộc Σ, $w \in \Sigma^*$, và người ta viết :

$S \overset{*}{\Rightarrow}_G w.$

Ngôn ngữ *L* do văn phạm G sinh ra, viết *L*(G), là tập hợp các câu sao cho :

$L(G) = \{ w \in \Sigma^* \mid S \overset{*}{\Rightarrow}_G v \}.$

Ví dụ 3.3 :

Ngôn ngữ sinh ra bởi (language generated by) văn phạm đã cho trong ví dụ 3.2 ở trên gồm mọi câu hoặc chứa toàn chữ *a*, hoặc chứa toàn chữ *b*.

II.2. Phân cấp các loại văn phạm của Chomsky

Theo Chomsky, các văn phạm đươc chia ra thành bốn loại tùy theo tính chất của các sản xuất sử dụng trong văn phạm :

Loại 0 Không giới han về sản xuất (sản xuất có dạng bất kỳ).

Loại 1 Văn phạm cảm ngữ cảnh (Context Sensitive Grammar). Các sản xuất dạng α→β của văn phạm phải thỏa mản điều kiện : |α|≤|β|. Có nghĩa là thành phần bên phải của sản xuất (β) phải chứa ít nhất số ký tự bằng thánh phần bên trái (α). Đối vơí các câu rỗng, có thể sử dụng sản xuất : S → ε (S là ký tự đầu). Sản xuất này không chứa S ở bên phải.

Loại 2 Văn phạm phi ngữ cảnh (Context Free Grammar).
Các sản xuất có dạng A → β, trong đó α chỉ gồm duy nhất một ký tự không kết thúc A, A ∈ { V – Σ } và không có hạn chế gì về β.

Loại 3 Văn phạm chính qui (Regular Grammar). Mọi sản xuất của văn phạm là một trong hai dạng :

A → wB, hoặc

$$A \to w.$$

Trong đó, A và $B \in V - \Sigma$, $w \in \Sigma^*$.

Mọi sản xuất của văn phạm chính qui có vế trái α là một ký tự không kết thúc duy nhất, vế phải β gồm các ký tự kết thúc và tiếp theo sau là một ký tự không kết thúc (hoặc không có).

Tuy nhiên, có thể xây dựng một loại văn phạm chính qui mà các sản xuất có một trong hai dạng :

$A \to Bw$ và $A \to w$, với $A, B \in \{ V - \Sigma \}$, $w \in \Sigma^*$,

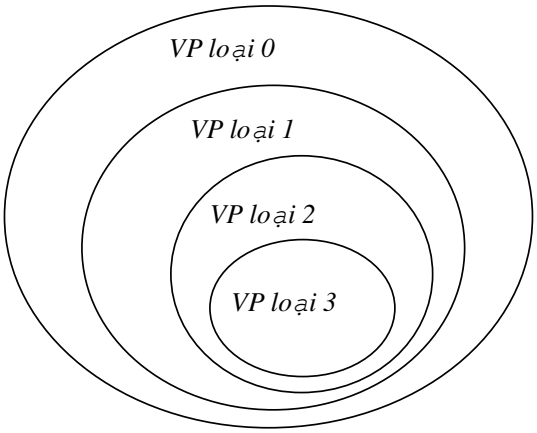
gọi là các văn phạm *tuyến tính trái* (Left Linear Grammars).

Như vậy, loại văn phạm với các sản xuất dạng $A \to wB$, $A \to w$ gọi là văn phạm *tuyến tính phải* (Right Linear Grammars).

Có thể chứng minh rằng mọi ngôn ngữ sản sinh bởi một văn phạm *tuyến tính phải* cũng có thể được sản sinh bỡi một văn phạm *tuyến tính trái* và ngược lại.

Quan hệ giữa các Loại văn phạm được biểu diễn như sau :

$$\text{Loại } 3 \subset \text{Loại } 2 \subset \text{Loại } 1 \subset \text{Loại } 0.$$



Các quan hệ bao hàm ở trên là thực sự, nghĩa là với $i > j$, thì tồn tại những ngôn ngữ có thể được sản sinh bởi một văn phạm loại j, nhưng không phải bởi một văn phạm loại i.

Theo phân cấp Chomsky, một văn phạm loại 3 thì cũng là văn phạm loại 0. Như vậy, nếu một ngôn ngữ được sản sinh bởi một văn phạm loại 3 thì ngôn ngữ đó cũng được sản sinh bởi một văn phạm loại 2 và suy luận tương tự cho các văn phạm loại 1 và loại 0.

Sự bao hàm giữa 2 và 1 không được thật rõ ràng. Vấn đề là ở chỗ một văn phạm loại 2 có thể chứa các sản xuất dạng $A \to \varepsilon$, mà do vậy, điều kiện $|\alpha| \le |\beta|$ của các văn phạm loại 1 không thỏa mãn (tuy nhiên, với mọi loại sản xuất khác, điều kiện này lại thỏa mãn).

Bảng tóm tắt về tính tương đương giữa văn phạm và ôtômat của các lớp văn phạm theo Chomsky :

Loại	*Văn phạm*	*Dạng sản xuất*	*Ôtômat*	*Ví dụ*
0	Tự do	Không có hạn chế gì	Máy Turing	Ngôn ngữ tự nhiên
1	Cảm ngữ cảnh	$a \to b$, $\lvert a\rvert \le \lvert b\rvert$	Máy Turing với băng hữu hạn	$\{a^n b^n c^n \mid n \ge 1\}$
2	Phi ngữ cảnh	$A \to a$	Đẩy xuống	$\{a^n b^n \mid n \ge 1\}$
3	Chính quy	$A \to bB \mid a$	Hữu hạn	$\{0(10)^n \mid n \ge 0\}$

II.3. Các văn phạm chính qui

Không phải ngẫu nhiên mà các văn phạm loại 3 lại được gọi là văn phạm chính quy. Thực tế, các ngôn ngữ được sản sinh bởi văn phạm loại 3 đều là ngôn ngữ chính quy. Ta có định lý sau đây :

Định lý 3.1 :

Một ngôn ngữ là chính qui nếu và chỉ nếu ngôn ngữ đó được sản sinh bởi một văn phạm chính qui.

Chứng minh :

"$\Rightarrow$" *Nếu một ngôn ngữ là chính qui, ngôn ngữ đó được sinh bởi một văn phạm chính qui.*

Cho *L* là ngôn ngữ chính quy được thừa nhận bởi một ôtômat hữu hạn không đơn định M, *L* = *L*(M), trong đó :

$$M = (Q, \Sigma, \Delta, q0, A)$$

Sử dụng M để xây dựng một văn phạm chính quy $G = (V_G, \Sigma_G, S_G, R_G)$ sao cho *L*(G) = *L*. Các phần tử của G được định nghĩa theo M như sau :

- $\Sigma_G = \Sigma$ (các ký tự cuối của văn phạm G là bảng chữ của M),
- $V_G = Q \cup \Sigma$ (tồn tại một ký tự không kết thúc của G cho mỗi trạng thái của M),
- $S_G = q_0$ (ký tự đầu của G tương ứng với trạng thái đầu của M),
- $R_G = \{ A \to wB$ cho $\forall (A, w, B) \in \Delta$ và $A \to \varepsilon$ cho $\forall A \in A \}$.
  (Tồn tại một sản xuất cho mỗi chuyển tiếp $\in \Delta$ và một sản xuất rỗng cho mỗi trạng thái $\in A$).

Bây giờ chỉ còn chứng minh rằng, một câu được thừa nhận bởi M nếu và chỉ nếu nó được sản sinh bởi G.

Điều này là hiển nhiên khi tiến hành đồng thời giữa quá trình chuyển tiếp cùa M và quá trình sản sinh của G.

Thật vậy, cần chứng tỏ rằng :

$(q, w) \overset{*}{\underset{M}{\vdash}}$ (p, v) với $w = uv$ nếu và chỉ nếu $q \overset{*}{\underset{G}{\Rightarrow}}$ up

theo phương pháp quy nạp theo độ dài của các chuyển tiếp.

Theo định nghĩa của G, điều này đúng đối với các chuyển tiếp có độ dài 1. Giả sử điều này cũng đúng đối với các chuyển tiếp có độ dài là n, cần chứng minh điều này cũng đúng đối với các chuyển tiếp có độ dài là n+1.

Cuối cùng suy ra rằng :

$(q_0, w) \overset{*}{\underset{M}{\vdash}} (p, \varepsilon)$ với $p \in A$ nếu và chỉ nếu $q_0 \overset{*}{\underset{G}{\Rightarrow}} w$.

Từ đó suy ra điều phải chứng minh $L(G) = L(M)$.

"⇐" *Nếu một ngôn ngữ được sản sinh bởi một văn phạm chính quy, ngôn ngữ đó là chính quy.*

Ta sẽ chứng minh rằng một ngôn ngữ *L* được sinh ra bởi một văn phạm chính quy sẽ được thừa nhận bởi một ôtômát hữu hạn không đơn định.

Giả sữ $G = (V_G, \Sigma_G, S_G, R_G)$ là văn phạm sản sinh ra ngôn ngữ *L*. Ôtômat

$M = (Q, \Sigma, \Delta, q0, A)$ là NFA thừa nhận ngôn ngữ *L* được định nghĩa như sau :

- $Q = V_G - \Sigma_G \cup \{ f \}$ (các trạng thái cũa M là những ký hiệu không kết thúc của G cộng thêm một trạng thái mới *f*)
- $\Sigma = \Sigma_G$
- $q_0 = S_G$
- $A = \{ h \}$
- $\Delta = \{$ (A, w, B) với $\forall A \to wB \in R_G$ và
  (A, w, h) với $\forall A \to w \in R_G \}$.

Dễ thấy rằng ngôn ngữ sản sinh bởi G đồng nhất với ngôn ngữ được thừa nhận bởi M (cách chứng minh tương tự trường hợp trên) □.

III. Các ngôn ngữ chính quy

Cho đến lúc này, ta đã xét bốn công cụ khác nhau để làm việc với các ngôn ngữ chính quy :

1. Biểu thức chính quy.
2. Ôtômat hữu hạn không đơn định.
3. Ôtômat hữu hạn đơn định.
4. Văn phạm chính quy.

Trong mục này, ta tiếp tục xét các tính chất của ngôn ngữ chính quy.

III.1. Các tính chất của ngôn ngữ chính quy

Cho hai ngôn ngữ chính quy L_1 và L_2. Ta có các tính chất sau đây :

1. $L_1 \cup L_2$ cũng là ngôn ngữ chính quy.

Thật vậy, giả sử a_1 và a_2 là các biểu thức chính quy chỉ định các ngôn ngữ L_1, L_2 tương ứng.

Khi đó, biểu thức chính quy $a_1 \cup a_2$ chỉ định ngôn ngữ $L(a_1 \cup a_2) = L_1 \cup L_2$ cũng là chính quy (theo định lý 1.1).

2. $L_1.L_2$ và L_1^* cũng là những ngôn ngữ chính quy.

Cách chứng minh tương tự tính chất 1.

3. L_1^R cũng là ngôn ngữ chính quy.

Giả sử w^R là nghịch đảo của câu *w* (nhận được từ *w* bằng cách viết *w* từ phải qua trái, ví dụ nghịch đảo của *aab* là *baa*), khi đó ngôn ngữ :

$L_1^R = \{ w \mid w^R \in L_1 \}$ là ngôn ngữ chính quy.

Thật vậy, xuất phát từ ôtômat $M = (Q, Z, \Delta, q_0, A)$ thừa nhận L_1, có thể xây dựng ôtômat $M' = (Q, \Sigma, \Delta', s', A')$ thừa nhận L_1^R như sau :

$A' = \{ q_0 \}$ (trạng thái đầu trở thành trạng thái cuối duy nhất).

q_0' là trạng thái đầu mới được thêm vào.

$\Delta' = \{ (q, w^R, p) \mid (p, w, q) \in \Delta \} \cup \{ (q_0', e, q) \mid q \in A \}$.

Các chuyển tiếp của Δ' là các chuyển tiếp của M nhưng theo cách ngược lại, và thêm vào một chuyển tiếp mới từ trạng thái đầu vào mỗi trạng thái cuối của M.

4. Bù của L_1 cũng là ngôn ngữ chính quy.

Bù của L_1, ký hiệu $\overline{L_1} = \Sigma^* - L_1$, cũng là ngôn ngữ chính quy.

Thật vậy, có thể xây dựng ôtômat hữu hạn đơn định M' = (Q, ∑, d, q_0, A') thừa nhận $\overline{L_1}$ bằng cách hoán vị vai trò của các trạng thái đạt được và không đạt được của M, ở đây A' = Q - A.

Việc xây dựng trên không áp dụng được cho các ôtômat không đơn định.

5. $L_1 \cap L_2$ cũng là ngôn ngữ chính quy.

Dễ thấy rằng $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$.

Một khả năng khác là có thể xây dựng một ôtômat đơn định M = (Q, ∑, d, q_0, A) thừa nhận ngôn ngữ $L_1 \cap L_2$ xuất phát từ các ôtômat đơn định :

$M_1 = (Q_1, \Sigma, d_1, q_{01}, A_1)$ và $M_2 = (Q_2, \Sigma, d_2, q_{02}, A_2)$

thừa nhận lần lượt ngôn ngữ L_1 và L_2. Ôtômat M bắt chước (simule) thực hiện đồng thời hai ôtômat M_1 và M_2 và thừa nhận khi cả hai ôtômat M_1 và M_2 cùng thừa nhận.

Như vậy, mỗi trạng thái của M sẽ là một cặp trạng thái : một trạng thái của M_1 và một trạng thái của M_2. Ta có :

$Q = Q_1 \times Q_2$,

$\delta((q_1, q_2), s) = (p_1, p_2)$ nếu và chỉ nếu $\delta_1(q_1, s) = p_1$ và $\delta_2(q_2, s) = p_2$,

$q_0 = (q_{01} \cdot q_{02})$, $A = A_1 \times A_2$

Ví dụ 3.4 :

Ngôn ngữ $L_1 | L_2 = \{ x \mid \exists y \in L_2$ sao cho $xy \in L_1 \}$, được gọi là ngôn ngữ thương của L_1 với L_2, cũng là ngôn ngữ chính quy. Ngôn ngữ này gồm các câu của L_1 bị cắt bỏ hậu tố là một câu thuộc L_2.

Nếu $M_1 = (Q, \Sigma, \delta, q_0, A_1)$ là ôtômat thừa nhận ngôn ngữ L_1, khi đó $L_1 | L_2$ sẽ được thừa nhận bởi ôtômat :

$M = (Q, \Sigma, \delta, q_0, A)$ mà $A = \{ q, \in Q \mid L\ (Q, \Sigma, \delta, q, A_1) \cap L_2 \neq \varnothing \}$.

Ôtômat M đồng nhất với M, chỉ khác về các trạng thái đạt được. Đó là các trạng thái của M_1 mà từ đó có khả năng đạt được một trạng thái thừa nhận của M_1 bởi một câu của L_2.

III.2. Các thuật giải

Như đã biết, phần lớn các bài toán liên quan đến các ngôn ngữ chính quy đều có thể giải được bởi các thủ tục hiệu quả. Sau đây là một vài bài toán.

Cho trước ngôn ngữ chính quy *L* và một câu $w \in \Sigma^*$.

Bài toán 1 : Xác định xem có phải $w \in L$ hay không ?

Bài toán này là giải giải được bởi một thủ tục hiệu quả.

Thật vậy, chỉ cần xây dựng một ôtômat đơn định xuất phát từ sự mô tả của *L* và thực hiện ôtômat này trên câu *w* đã cho.

Bài toán 2 : Xác định xem có phải *L* là rỗng hay không ?

Bài toán xác định $L=\varnothing$ hay không là giải được bởi một thủ tục hiệu quả. Thật vậy, người ta xây dựng một ôtômat (đơn định hoặc không đơn định) thừa nhận L. Ngôn ngữ $L \neq \varnothing$ nếu, trong sơ đồ biểu diễn ôtômat này, tồn tại một đường đi giữa trạng thái đầu và một trạng thái kết thúc nào đó.

Bài toán 3 : Xác định xem có phải $L = \Sigma^*$?

Để xác định nếu một ngôn ngữ chính quy L chứa mọi câu (universal) thì chỉ cần kiểm tra nếu $L = \varnothing$?

Ta xây dựng một ôtômat thừa nhận ngôn ngữ bù của L và kiểm tra ôtômat này không thừa nhận một câu nào.

Bài toán 4 : L_1 và L_2 là các ngôn ngữ chính quy, xác định xem có phải $L_1 \subseteq L_2$?

Để kiểm tra rằng $L_1 \subseteq L_2$, cần kiểm tra $L_1 \cap L_2 = \varnothing$

Bài toán 5 : L_1 và L_2 là các ngôn ngữ chính quy, xác định xem có phải $L_1 = L_2$?

Tính bằng nhau của hai ngôn ngữ chính quy có thể đwọc kiểm tra bằng cách kiểm tra các điều kiện $L_1 \subseteq L_2$ và $L_2 \subseteq L_1$.

III.3. Nhận xét

Sau đây là một số nhận xét dễ dàng chứng minh.

Nhận xét 1 : Mọi ngôn ngữ hữu hạn (chứa hữu hạn câu) đều là chính quy.

Thật vậy, ngôn ngữ chứa hữu hạn câu $\{w_1, ..., w_k\}$ sẽ được chỉ định bởi biểu thức chính quy $w_1 \cup ... \cup w_k$ là hợp hữu hạn các câu của ngôn ngữ.

Nhận xét 2 : Một ngôn ngữ không chính quy phải chứa vô hạn câu.

Điều ngược lại không đứng. Chẳng hạn, ngôn ngữ $\sum^*$ là chính quy và chứa vô hạn các câu.

Nhận xét 3 : Một ngôn ngữ chứa vô hạn câu sẽ không có cận trên.

Nếu một ngôn ngữ chứa vô hạn câu, sẽ không có cận trên (borne) cho độ dài của các câu thuộc ngôn ngữ. Thật vậy, giả sử t là cận trên cho mọi độ dài của các câu thuộc ngôn ngữ. Giả sử bảng chữ $\sum$ để xây dựng ngôn ngữ này gồm k là một số hữu hạn ký tự. Như vậy, số các câu có độ dài nhỏ hơn t sẽ bị bao bởi :

$$
\begin{aligned}
&1 && \text{câu có độ dài 0}\\
+\ &k && \text{câu có độ dài 1}\\
+\ &k^2 && \text{câu có độ dài 2}\\
&\ldots\\
+\ &k^t && \text{câu có độ dài t}
\end{aligned}
$$

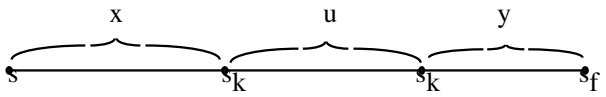
$= \dfrac{k^{t+1} - 1}{k - 1}$ không phải vô hạn (mâu thuẫn !).

Nhận xét 4 : Mọi ngôn ngữ chính quy được thừa nhận bởi một ôtômat hữu hạn chỉ gồm một số cố định trạng thái.

Nhận xét 5 :

Xét một ngôn ngữ chính quy vô hạn và một ôtômat gồm m trạng thái thừa nhận ngôn ngữ này.

Với mọi câu có độ dài lớn hơn m, việc thực hiện của ôtômat trên câu này phải vượt qua cùng một trạng thái s_k ít ra hai lần với một phần khác rỗng của câu ngăn cách hai nhịp vượt này.



Từ đó, nếu x, u và y là các câu đã đề cập ở trên, thì mọi câu có dạng xu^*y cũng được thừa nhận bởi ôtômát và là một phần của ngôn ngữ.

III.4. Định lí "bơm" (Pumping Theorem)

III.4.1. Phát biểu định lý "bơm"

Định lý 3.2 :

Giả sử *L* là ngôn ngữ chính qui vô hạn. Khi đó, tồn tại x, u, $y \in \Sigma^*$, với $u \neq \varepsilon$ sao cho $xu^*y \in L$, $\forall n \geq 0$.

Việc chứng minh định lí này căn cứ vào nhận xét ở mục (3.4.3). Ta xét một ôtômát với m trạng thái thừa nhận *L* và một câu *w* nào đó có độ dài lớn hơn m. Ta chia câu này thành *ba* phần x, u và y như trên, *w*=xuv. Các câu x, u, y có thể nhận được từ một câu nào đó bất kỳ đủ dài.

III.4.2. Phát triển định lý "bơm"

Định lý 3.3 :

Giả sử *L* là một ngôn ngữ chính qui vô hạn và giả sử $w \in L$ sao cho $|w| \geq |Q|$, Q là tập hợp các trạng thái của một ôtômát đơn định thừa nhận *L*, |Q| là bản số của Q. Khi đó sẽ $\exists$x, u, y, với $u \neq \varepsilon$ và $|xu| \leq |Q|$ sao cho xuy = *w* và, $\forall n$, $xu^n y \in L$.

III.4.3. Ứng dụng của định lí "bơm"

Ví dụ 3.5 :

Ngôn ngữ $a^n b^n$ (chứa các câu gồm một số chữ *a* rồi cùng một số như vậy chữ *b*) không phải chính qui.

Để chứng minh, ta sẽ chỉ ra rằng sẽ không thể tim thấy các câu x, u, y sao cho $xu^k y \in a^n b^n$, $\forall k$, và chỉ ra rằng mâu thuẫn với định lí (3.2).
Giả sử tồn tại x, u, y như vậy và xét u :

- $u \in a^*$: không thể được, vì nếu ta lặp u, áp dụng định lý 3.2, số chữ *a* sẽ không còn bằng số chữ *b*.
- $u \in b^*$: cũng không thể được với cách lập luận như trên.
- $u \in (a \cup b)^* - (a^* \cup b^*)$ cũng không thể được, vì rằng một dãy các *a* rồi một dãy các *b* trong u^k (k>1) và $xu^k y$ sẽ không thuộc ngôn ngữ $a^n b^n$.
- Ở đây :

$$(a \cup b)^* - (a^* \cup b^*) = (aa^* b(a \cup b)^*) \cup (bb^* a(a \cup b)^*)$$

biểu diễn tập hợp các câu tạo từ ít nhất một chữ *a* và ít nhất một chữ *b*, nghĩa là mọi câu ít ra chỉ chứa toàn chữ *a* hoặc toàn chữ *b*.

Ví dụ 3.6 :

Ngôn ngữ $L = a^{n^2}$ (chứa mọi câu trên bảng chữ { *a* } với độ dài n^2 là một số bình phương đúng) không là ngôn ngữ chính qui.

Để chứng minh, cần chứng tỏ rằng định lý 3.3 không thỏa mãn bởi ngôn ngữ này. Thật vậy, giả sử $m = |Q|$ là số trạng thái của một ôtômát thừa nhận ngôn ngữ *L*. Xét các câu a^{m^2}. Vì $m^2 \geq m$, từ định lý 3.3 suy ra rằng câu này có thể cắt thành *ba* phần x, u và y sao cho $|xu| \leq m$ và $xu^n y \in L, \forall n$.

Hiển nhiên ta có :

$x = a^p \qquad 0 < p < m - 1,$

$u = a^q \qquad 0 < q < m,$

$y = a^r \qquad r \geq 0.$

Từ các ràng buộc này, chỉ có thể $xu^2y \in L$. Vì rằng p+2q+r không phải là một bình phương đúng. Thật vậy :

$$m^2 < p + 2q + r \leq m^2+m < (m+1)^2 = m^2+2m+1.$$

IV. Ứng dụng các ngôn ngữ chính qui

Ví dụ 3.7 :

Tìm trong một tệp chương trình các tên (identifier) có hai ký tự đầu là *ab* & ký tự cuối là t.

Có thể xem nội dung của tệp như một chuỗi ký tự (một câu) trên bảng chữ Σ là tập các ký tự ASCII. Việc tìm kiếm các tên dẫn đến việc tìm các biểu thức chinh qui có dạng $ab\Sigma^*t$ trong câu biểu diễn nội dung cuả tệp.

Để giải quyết, sử dụng các tính chất của ngôn ngữ chính qui : tìm các xuất hiện của biểu thức chính qui α trong câu $w \in \Sigma^*$. Thực hiện như sau :

- Xét biểu thức chính qui $\beta = \Sigma^*\alpha$.
- Xây dựng một ôtômát không đơn định thừa nhận ngôn ngữ chỉ định bởi β bằng cách xây dựng như đã trình bày ở chương 2.
- Từ ôtômát này, xây dựng ôtômát đơn định Mβ theo cách xây dựng đã trình bày ở chương 2. Ôtômát này thừa nhận ngôn ngữ chỉ định bởi β.
- Thực hiện ôtômát Mβ trên câu *w*. Khi ôtômát Mβ ở trạng thái thừa nhận, ta sẽ tìm thấy xuất hiện của α trong câu *w*.

Bài tập chương 3

1. Mô tả các ngôn ngữ cho bởi các văn phạm sau đây :

$S \rightarrow aS \mid Sb \mid aSb \mid c$

$S \rightarrow SS \mid a \mid b$

$S \rightarrow aA \mid bB \mid c$ $\quad A \rightarrow Sa$ $\quad B \rightarrow Sb$

$S \rightarrow AB$ $\quad A \rightarrow Sc \mid a$ $\quad B \rightarrow dB \mid b$

$S \rightarrow SaS \mid b$

2. Mô tả văn phạm sản sinh các ngôn ngữ sau đây :

a) a^*b

b) $(a+b)(ab)^*(baab)^*$

CHƯƠNG 4

Ôtômat đẩy xuống và ngôn ngữ phi ngữ cảnh

Chúng ta đã thấy không tồn tại ôtômat hữu hạn thừa nhận ngôn ngữ a^nb^n. Tuy nhiên, một ngôn ngữ gồm các câu chứa cùng số chữ *a* và số chữ *b*, nhưng số này bị chặn trên, nghĩa là mỗi ngôn ngữ dạng :

$L_K = \{ a^nb^n \mid n \le K \}$,

thì sẽ được thừa nhận bởi một ôtômat hữu hạn. Chúng ta có nhận xét sau :

- Mỗi câu có dạng a^nb^n thuộc về một ngôn ngữ L_K và câu này có thể nhận biết bởi một ôtômat hữu hạn thừa nhận L_K.
- Không tồn tại ôtômat hữu hạn thừa nhận mọi câu a^nb^n, nghĩa là phép hợp vô hạn các ngôn ngữ L_K.

Như vậy, vấn đề đặt ra trong chương này là cần bổ sung khả năng vào lớp các ôtômat hữu hạn để chúng có thể thừa nhận ngôn ngữ a^nb^n.

I. Các ôtômat đẩy xuống

I.1. Mô tả

Một ôtômat đẩy xuống không đơn định (NSA : Non-deterministic Stack Auto-maton hay còn gọi NPA : Non-deterministic Pushdown Automaton) có một số phần tử tương tự ôtômat hữu hạn, nghĩa là cùng có :

- Một băng vào chứa câu sẽ được đoán nhận và một đầu đọc đọc lần lượt các ký tự của câu.
- Một tập hợp các trạng thái, trong đó có một trạng thái đầu và một số trạng thái cuối (trạng thái đạt được).
- Một quan hệ chuyển tiếp.

Ngoài ra, NSA còn có một danh sách đẩy xuống (stack) có thể chứa không hạn chế các ký tự nào đó. Lúc đầu, danh sách đẩy xuống (DSĐX) rỗng. NSA thực hiện việc thừa nhận câu vào tương tự một ôtômat hữu hạn không đơn định.

Tuy nhiên, tại mỗi thời điểm, NSA còn nhìn (xem xét) một phần của đỉnh danh sách và thay thế nó bởi một dãy các ký tự.

I.2. Mô tả hình thức

Một NSA được định nghĩa bởi bộ 7 : M = (Q, Σ, Γ, Δ, Z, q_0, A), trong đó :

- Q là tập hợp hữu hạn các trạng thái.
- Σ là bảng chữ vào (input alphabet).
- Γ là bộ chữ đẩy xuống (stack alphabet).
- $Z \in \Gamma$ là ký tự đầu của DSĐX (initial stack symbol).
- $q_0 \in Q$ là trạng thái đầu.
- $A \subseteq Q$ là tập các trạng thái cuối.
- $\Delta \subset ((Q \times \Sigma^* \times \Gamma^*) \times (Q \times \Gamma^*))$ là quan hệ chuyển tiếp (gồm một tập hợp hữu hạn các quan hệ).

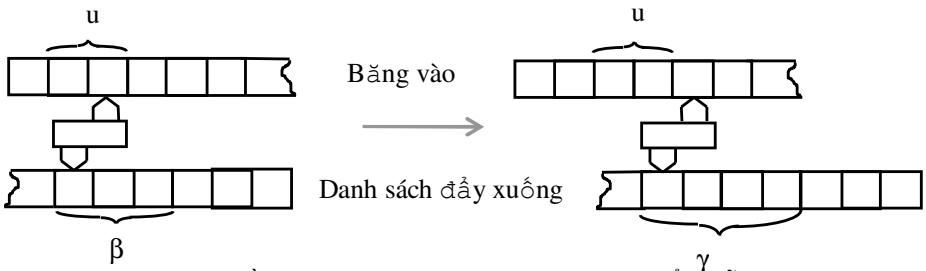
Bộ chữ đẩy xuống Γ chứa tập hợp các ký tự sẽ được đưa vào trong DSĐX. Không nhất thiết phân biệt Γ với bộ chữ vào Σ (có thể $\Gamma \cap \Sigma \neq \varnothing$). Ký tự Z là ký tự đầu hay nội dung ban đầu của DSĐX.

Các chuyển tiếp trong Δ giống như một ôtômat hữu hạn không đơn định có thêm sự hoạt động của DSĐX. Như vậy, chuyển tiếp :

$$((p, u, \beta), (q, \gamma)) \in \Delta$$

có nghĩa là ôtômat có thể chuyển từ trạng thái p sang trạng thái *q* sao cho câu vào bắt đầu bởi tiền tố u và dòng β đang nằm trên đỉnh của DSĐX.

Sau khi chuyển tiếp, đầu đọc đọc xong tiền tố u của dòng vào và thay thế đỉnh DSĐX β bởi dòng γ.



Trong hình trên, DSĐX nằm ngang, đỉnh ở bên trái cho phép biểu diễn sự đọc câu β và sự ghi câu γ. Câu β được đọc từ trái qua phải, ký tự đầu tiên của β nằm trên đỉnh của DSĐX. Câu γ được ghi vào DSĐX sao cho ký tự đầu tiên của γ sẽ nằm trên đỉnh của danh sách.

Ngôn ngữ được thừa nhận bởi NSA cũng được định nghĩa một cách hình thức tương tự các ôtômat hữu hạn trên cơ sở cấu hình chuyển tiếp (CHDC) một giai đoạn, CHDC nhiều giai đoạn và sự đoán nhận của NSA. Sự khác nhau cơ bản là sự có mặt của DSĐX trong CHDC.

CHDC của NSA là bộ *ba* gồm trạng thái của ôtômat, một phần của câu vào đang được xử lý và nội dung của DSĐX.

Một cách hình thức, đó là phần tử :

$$(q, u, \beta) \in Q \times \Sigma^* \times \Gamma^*$$

Định nghĩa 4.1 :

Cho ôtômat M, ta nói CHDC (*q*', *w*', α') nhận được từ CHDC (*q*, *w*, α). kí hiệu :

$(q, w, \alpha) \vdash_M (q', w', \alpha')$ nếu :

– $w = uw'$ (câu *w* bắt đầu tiền tố $u \in \Sigma$)

– $a = \beta\delta$

(trước khi chuyển tiếp, đỉnh của DSĐX chứa $\beta \in \Gamma$ nếu đọc từ trái qua phải).

– $a' = \gamma\delta$ (sau khi chuyển tiếp, phần β của DSĐX được thay thế bởi γ, ký tự đầu tiên của γ bây giờ nằm ở đỉnh của DSĐX)

– $(q, u, \beta), (q', \gamma)) \in \Delta$

- Cấu hình C' nhận được từ C qua nhiều giai đoạn, ký hiệu :

  $C \vdash_M^* C'$

  nếu $\exists k \geq 0$ và các cấu hình trung gian $C_0, C_1, C_2, ..., C_k$ sao cho :

- $C = C_0, C' = C_k, C_i \vdash_M C_{i+1}$ với $0 \leq i < k$.

Một đoán nhận (execution) của NSA trên câu $w \in S^*$ là dãy các cấu hình :

- $(q_0, w, Z) \vdash_M (q_1, w_1, \alpha_1) \vdash_M \ldots \vdash_M (q_n, \varepsilon, \gamma)$
- trong đó q_0 là trạng thái đầu, Z là ký tự đầu của DSĐX và ε là câu rỗng.

Lúc đầu, DSĐX chứa ký tự đầu Z và câu *w* cần xử lí đang nằm trên băng vào. Kết thúc đoán nhận, phần câu còn lại cần xử lí trở nên rỗng. Do chúng ta nghiên cứu các ôtômat không đơn định, nên có thể có nhiều phép đoán nhận phân biệt trên cùng một câu vào.

Quá trình đoán nhận một câu sẽ đạt đến các trạng thái cuối (kết thúc). Một câu *w* được thừa nhận bởi một ôtômat đẩy xuống : $M = (Q, \Sigma, \Gamma, \Delta, Z, q_0, A)$ nếu :

- $(q_0, w, Z) \vdash_M^* (p, \varepsilon, \gamma)$ với $p \in A$.

I.3. Một số ví dụ

Ví dụ 4.1 :

Ngôn ngữ $\{ a^n b^n \mid n \geq 0 \}$ được thừa nhận bởi $M = (Q, \Sigma, \Gamma, \Delta, Z, s, A)$ theo mô tả dưới đây :

$Q = \{ s, p, q \}, \Sigma = \{ a, b \}, \Gamma = \{ A \}, A = \{ q \}$,

Δ gồm các chuyển tiếp :

$(s, a, \varepsilon) \rightarrow (s, A)$ $\quad$ $(p, b, A) \rightarrow (p, \varepsilon)$

$(s, \varepsilon, Z) \rightarrow (q, \varepsilon)$ $\quad$ $(p, \varepsilon, Z) \rightarrow (q, \varepsilon)$

$(s, b, A) \rightarrow (p, \varepsilon)$

Ôtômat đạt tới trạng thái thừa nhận và DSĐX rỗng.

Ví dụ 4.2 :

Ngôn ngữ $\{ww^R\}$ được thừa nhận bởi ôtômat M theo mô tả dưới đây :

$Q = \{ s, p, q \}, \Sigma = \{ a, b \}, \Gamma = \{ A, B \}, A = \{ q \}$,

Δ chứa các chuyển tiếp :

(s, *a*, ε) → (s, A) (p, *a*, A) → (p, ε)

(s, *b*, ε) → (s, B) (p, *b*, B) → (p, ε)

(s, ε, ε) → (p, ε) (p, ε, z) → (*q*, ε)

Ngôn ngữ {wwR } được thừa nhận bởi ôtômat M đồng thời trong trạng thái cuối và DSĐX rỗng. Tuy nhiên, có thể xây dựng một ôtômat gồm chỉ hai trạng thái thừa nhận ngôn ngữ với DSĐX rỗng :

Q = { s, p }, ∑ = { *a*, *b* }, Γ = { A, B }, Δ gồm các chuyển tiếp :

(s, *a*, ε) → (ε, A) (p, *a*, A) → (p, ε)

(s, *b*, ε) → (s, B) (p, *b*, B) → (p, ε)

(s, ε, ε) → (p, ε) (p, ε, Z) → (*q*, ε)

II. Các ngôn ngữ phi ngữ cảnh

II.1. Định nghĩa

Ta đã xây dựng một văn phạm phi ngữ cảnh G = (V, ∑, R, S) gồm các sản xuất dạng A → β, với A ∈ (V- M) và không có hạn chế gì trên β.

Từ G, có thể định nghĩa ngôn ngữ phi ngữ cảnh.

Định nghĩa 4.2 :

Một ngôn ngữ là phi ngữ cảnh nếu tồn tại một văn phạm phi ngữ cảnh sản sinh ra ngôn ngữ này.

Ví dụ 4.3 :

- Ngôn ngữ $a^n b^n$, n (0 được sinh bởi văn phạm gồm các sản xuất :
  S → aSb | ε
  là phi ngữ cảnh.
- Ngôn ngữ các câu dạng ww^R sinh bởi văn phạm gồm các sản xuất :
  S → aSa | bSb | ε,
  là phi ngữ cảnh.
- Ngôn ngữ sinh bởi văn phạm gồm các sản xuất :
  S → aB | bA | e A → bAA | aS B → bS | aBB
  gồm các câu chứa cùng số chữ *a* và chữ *b* trong một thứ tự nào đó.

II.2. Quan hệ với các ôtômat đẩy xuống

Các ngôn ngữ thừa nhận bởi các ôtômat đẩy xuống có thể được sinh bởi các văn phạm phi ngữ cảnh. Thật vậy, ta có định lý sau đây :

Định lý 4.1 :

Một ngôn ngữ là phi ngữ cảnh nếu và chỉ nếu ngôn ngữ đó được thừa nhận bởi một ôtômat đẩy xuống.

Cách chứng minh giống định lý 3.1.

II.3. Tính chất của các ngôn ngữ phi ngữ cảnh

Ta đã có hai đặc trưng của các ngôn ngữ phi ngữ cảnh : Các ôtômat đẩy xuống và các văn phạm phi ngữ cảnh.

Giả sử L_1 và L_2 là hai ngôn ngữ phi ngữ cảnh, ta có các tính chất sau :

• Ngôn ngữ $L_1 \cup L_2$ là phi ngữ cảnh.

Thật vậy, nếu $G_1 = (V_1, \Sigma_1, R_1, S_1)$ và $G_2 = (V_2, \Sigma_2, R_2, S_2)$ là các văn phạm sinh ra lần lượt L_1 và L_2, ngôn ngữ $L_1 \cup L_2$ sinh bởi văn phạm phi ngữ cảnh G = (V, ∑, R, S) với :

- $V = V_1 \cup V_2 \cup \{ S \}$ (S là một ký tự mới thêm vào)
- $\Sigma = \Sigma_1 \cup \Sigma_2$.
- Ký tự đầu là S.
- $R = R_1 \cup R_2 \cup \{ S \to S_1, S \to S_2 \}$.

Ngôn ngữ $L_1.L_2$ là phi ngữ cảnh.

Thật vậy, giống như trên, nếu G_1 và G_2 là các văn phạm phi ngữ cảnh sinh L_1 và L_2, văn phạm sinh ra ngôn ngữ L_1. L_2 là G = (V, ∑, R, S) với :

- $V = V_1 \cup V_2 \cup \{ S \}$
- $\Sigma = \Sigma_1 \cup \Sigma_2$
- Ký tự đầu là S
- $R = R_1 \cup R_2 \cup \{ S \to S_1. S_2 \}$

Ngôn ngữ L_1^* là phi ngữ cảnh. Một văn phạm phi ngữ cảnh sản sinh ra ngôn ngữ này là $G_1 = (V_1, \Sigma_1, R, S_1)$ với :

$R = R_1 \cup \{ S_1 \to \varepsilon, S_1 \to S_1 S_1 \}$

Ngôn ngữ $L_1 \cap L_2$ không hẳn PNC.

Nêú L_R là ngôn ngữ chính quy và L_2 là ngôn ngữ PNC thì ngôn ngữ $L_R \cap L_2$ là PNC. Thật vậy, giả sử :

$M_R = (Q_R, S_R, d_R, s_R, F_R)$ là ôtômat hữu hạn đơn định thừa nhận L_R và

$M_2 = (Q_2, S_2, G_2, D_2, Z_2, s_2, F_2)$ là ô xác định thừa nhận ngôn ngữ L_2.

Xây dựng ôtômat đẩy xuống M = (Q, S, G, D, Z, s, A) như sau :

$Q = Q_R \times Q_2$, $S = S_R \cup S_2$, $G = G_2$, $Z = Z_2$, $S = (s_R, s_2)$, $A = (F_R \times F_2)$,

$(((q_R, q_2), u, b), ((p_R, p_2), g)) \in D$ nếu và chỉ nếu :

$(q_R, u) \overset{*}{\underset{M}{\vdash}} (p_R, e)$ ôtômat M_R thừa nhận câu u và chuyển từ trạng thái q_R sang trạng thái p_R bởi một hoặc nhiều bước. Và :

$((q_2, u, b), (p_2, g)) \in D_2$ ôtômat đẩy xuống thừa nhận câu u và chuyển từ trạng thái q_2 sang trạng thái p_2, thay thế phần đỉnh *b* của danh sách bởi g.

Như vậy, ngôn ngữ $L_R \cap L_2$ được thừa nhận bởi ôtômat đẩy xuống $M_{\square}$.

Chú ý :

Cách xây dựng ôtômat đẩy xuống M ở trên tương tự cách xây dựng dùng để chứng minh giao của hai ngôn ngữ chính quy cũng là một ngôn ngữ chính quy. Thật vậy, nếu ta bỏ qua DSĐX của M_2 thì hai cách chứng minh là giống hệt nhau.

Chú ý rằng cách chứng minh trên không phù hợp với hai ôtômat đẩy xuống. Vì lúc này, mỗi ôtômat đều có riêng danh sách của mình và kết quả sẽ là một ôtômat đẩy xuống có hai danh sách. Một ôtômat đẩy xuống như vậy có thể thừa nhận một số ngôn ngữ nhưng những ngôn ngữ này lại không được thừa nhận bởi một ôtômat đẩy xuống một danh sách.

III. Làm việc với các ngôn ngữ PNC

Tương tự các ngôn ngữ chính quy, còn tồn tại các ngôn ngữ không phải PNC : tập hợp các ngôn ngữ PNC là đếm được (do mỗi ngôn ngữ PNC được biểu diễn bởi một dãy hữu hạn các ký tự) trong khi tập mọi ngôn ngữ là không đếm được.

Ví dụ 4.4 : Cho văn phạm với $\Sigma=\{a, b\}$ và R gồm các sản xuất :

1 2 3 4 (đánh số các sản xuất)

$S \rightarrow SS \mid aSa \mid bSb \mid \varepsilon$

Cho câu aabaab. Câu này có thể được sản sinh bởi các suy dẫn :

$$S \Rightarrow SS \Rightarrow aSaS \Rightarrow aaS \Rightarrow aabSb \Rightarrow aabaSab \Rightarrow aabaab$$

nhưng cũng có thể sản sinh bởi các suy dẫn :

$$S \Rightarrow SS \Rightarrow SbSb \Rightarrow SbaSab \Rightarrow Sbaab \Rightarrow aSabaab \Rightarrow aabaab$$

và ngoài ra còn 8 cách suy dẫn khác nữa.

Tuy nhiên, các cách suy dẫn này đều có một số bước giống nhau. Ở đây :

- Áp dụng một lần sản xuất 1 để nhận được SS.
- Áp dụng một lần sản xuất 2 và một lần sản xuất 4 cho ký tự không kết thúc đầu tiên của SS.
- Áp dụng một lần sản xuất 3 và một lần sản xuất 2 và cuối cùng một lần sản xuất 3 cho ký tự không kết thúc thứ hai của SS.

Sự khác nhau của 10 cách suy dẫn ra aabaab là thứ tự áp dụng các sản xuất của văn phạm đã cho.

Như vậy, tính chất có thứ tự khi áp dụng các sản xuất là không quan trọng và đó là đặc trưng của các ngôn ngữ phi ngữ cảnh. Một ký tự không kết thúc có thể được thay thế độc lập với dòng ký tự bao xung quanh (trước và sau) nó – không phụ thuộc vào "ngữ cảnh".

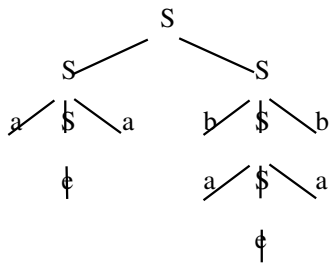
III.1. Khái niệm về cây phân tích

Người ta sử dụng cấu trúc cây để biểu diễn sự suy dẫn việc áp dụng trừu tượng

qua các quy tắc của văn phạm, gọi là cây phân tích (parse tree) hay cây suy dẫn (derivation tree).

Ví dụ 4.5 :

Sự suy dẫn câu aabaab sinh bởi văn phạm cho trong 4.3.1 có thể biểu diễn bởi cây phân tích ở hình bên :



Định nghĩa 4.3 :

Cây phân tích cho một văn phạm PNC G = (V, Σ, R, S) là cây mà mỗi nút được đánh nhãn bởi một ký hiệu V∪{ε} và thỏa mãn các điều kiện sau :

Rễ là ký tự xuất phát S.

Mỗi nút bên trong được đánh dấu bởi một ký hiệu không kết thúc (một phần tử của V – Σ).

Mỗi lá là một ký hiệu kết thúc (một phần tử của Σ) hay là một ký tự rỗng ε.

Với mỗi nút trong được đánh nhãn bởi một ký hiệu không kết thúc A và nếu các nhãn con trực tiếp của A là các nút $n_1, n_2, ..., n_k$ có các nhãn lần lượt tương ứng là $X_1, X_2, ..., X_k$ thì $A \rightarrow X_1X_2...X_k$ phải là một sản xuất thuộc G.

Nếu một nút có nhãn là ε thì nó là con trực tiếp duy nhất của cha nó (tránh có nhiều nút ε trong cây phân tích).

Một câu được sản sinh bởi cây phân tích bằng cách ghép tiếp (cộng) các lá của cây phân tích từ trái qua phải :

Định lý 4.2 :

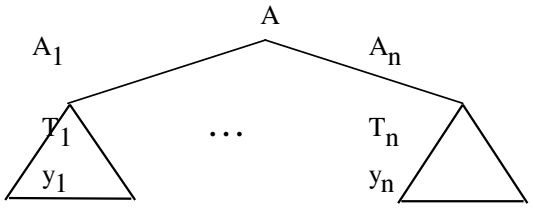
Cho văn phạm PNC G, một câu *w* được sinh bởi G, $S \overset{*}{\Rightarrow}_{\mathbf{G}} w$ và chỉ nếu tồn tại một cây phân tích của văn phạm G sinh ra *w*.

Chứng minh :

Ta sẽ chứng minh tính chất tổng quát sau đây : với mọi ký tự không kết thúc A, ta có $A \overset{*}{\Rightarrow}_{\mathbf{G}}$ u nếu và chỉ nếu tồn tại một cây con là cây phân tích có rễ là A sản sinh ra câu u.

Ta sẽ chứng minh bằng quy nạp theo cấu trúc của cây phân tích và độ dài của các suy dẫn. Giả sử rằng :

$A_1 \overset{*}{\Rightarrow} y_1, A_2 \overset{*}{\Rightarrow} y_2, . . ., A_n \overset{*}{\Rightarrow} y_n$



Cây phân tích cho $y_1y_2 ... y_n$

Khi đó, từ A có thể có các suy dẫn sau đây :

$$
\begin{aligned}
A &\Rightarrow A_1A_2 \ldots A_n \\
&\overset{*}{\Rightarrow} y_1A_2 \ldots A_n \\
&\overset{*}{\Rightarrow} y_1y_2 \ldots A_n \\
&\ldots \\
&\ldots \\
&\overset{*}{\Rightarrow} y_1y_2 \cdots y_n
\end{aligned}
$$

Chú ý rằng sự suy dẫn được duy trì theo cách chọn ký tự không kết thúc từ phía trái. Tuy nhiên vẫn không mất tính tổng quát nếu chọn từ phía phải hoặc chọn từ giữa ra.

Rõ ràng rằng có nhiều suy dẫn cho một câu ứng với một cây phân tích duy nhất. Tuy nhiên có thể có nhiều cây phân tích cùng suy dẫn cho một câu. Ví dụ 4.3.2 ở trên không phải ứng với trường hợp như vậy nhưng thực tế, tồn tại các văn phạm được gọi là *nhập nhằng* (ambiguous grammar). Khi đó, có thể có hai cây phân tích cùng suy dẫn cho một câu.

Một ngôn ngữ PNC có thể được sản sinh bởi nhiều văn phạm khác nhau.

Nếu các văn phạm này đều nhập nhằng, người ta nói rằng ngôn ngữ đã cho là *nhập nhằng cố hữu* (inherently ambiguous). Có thể chứng minh rằng tồn tại các ngôn ngữ nhập nhằng cố hữu.

Ta cần chứng minh rằng có thể loại bỏ các sản xuất dạng $A \to \varepsilon$ của một văn phạm phi ngữ cảnh $G = (V, \sum, R, S)$.

1. Nếu $\varepsilon \in L(G)$, ta thêm một ký tự đầu S' và các sản xuất $S' \to S \mid \varepsilon$.
2. Lặp lại các bước sau đây cho đến khi không còn các sản xuất dạng $A \to \varepsilon$:
   - Chọn một sản xuất dạng $A \to \varepsilon$ (khác với $S' \to \varepsilon$).
   - Loại bỏ sản xuất này khỏi R.
   - Với mỗi sản xuất $\alpha \to \beta \in R$ trong đó β chứa A, ta thêm vào R một sản xuất mới $\alpha \to \beta'$ với β' nhận được từ β bằng cách thay thế A bởi ε.

Một vấn đề khác cần xem xét là cách gọi văn phạm *phi ngữ cảnh* và văn phạm *cảm ngữ cảnh*. Một quy tắc dạng $A \to \beta$ được gọi là *phi ngữ cảnh* vì nó cho phép thay thế A bởi β độc lập với ngữ cảnh trong đó xuất hiện A (không liên quan gì đến những gì xung quanh A).

Trong khi đó, quy tắc : $aAb \to a\beta b$

không là "phi ngữ cảnh" vì nó chỉ cho phép thay thế A bởi *b* nếu như xung quanh A là *a* và *b*. Như vậy, tính áp dụng được của quy tắc này phụ thuộc vào ngữ cảnh trong đó xuất hiện A. Quy tắc này là *cảm* với *ngữ cảnh*.

III.2. Định lý "bơm"

Định lý "bơm" cho các ngôn ngữ PNC tương tự trường hợp của các ngôn ngữ chính quy : từ một câu có độ dài vừa đủ thuộc ngôn ngữ chính quy, có thể xây dựng một câu khác cũng thuộc ngôn ngữ bằng cách *lặp một dòng con* nào đó thuộc câu.

Tuy nhiên, định lý bơm cho các ngôn ngữ PNC khẳng định rằng xuất phát từ một câu có độ dài vừa đủ, có thể xây dựng một câu khác cũng thuộc ngôn ngữ bằng cách *lặp một hoặc hai dòng con* của câu.

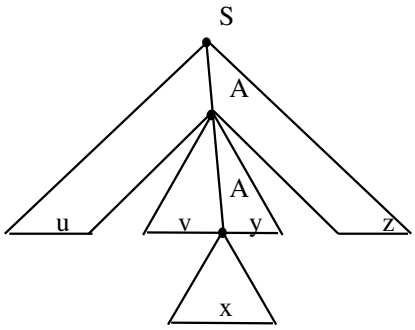
Định lý 4.3 :

Cho ngôn ngữ PNC *L*, tồn tại hằng K sao cho mọi câu $w \in L$ thỏa mãn điều kiện $|w| > K$ đều có thể viết dưới dạng uvxyz với $v \neq \varepsilon$ hoặc $y \neq \varepsilon$ và $uv^nxy^nz \in L$ cho mọi $n > 0$.

Chứng minh:

Cho G là văn phạm PNC sinh ra ngôn ngữ *L*. Để chứng minh ta chỉ cần chỉ ra rằng một cây phân tích tương ứng với một câu đủ dài phải chứa một đường đi mà trên đó, cùng một ký tự không kết thúc xuất hiện ít nhất hai lần. Phần cây phân tích gồm giữa hai phần câu (occurences) của cùng một ký hiệu không kết thúc có thể lặp tuỳ ý lần.

Để chứng minh đầy đủ ta cần chỉ ra cách chọn hằng K để làm xuất hiện tình huống ở hình bên.

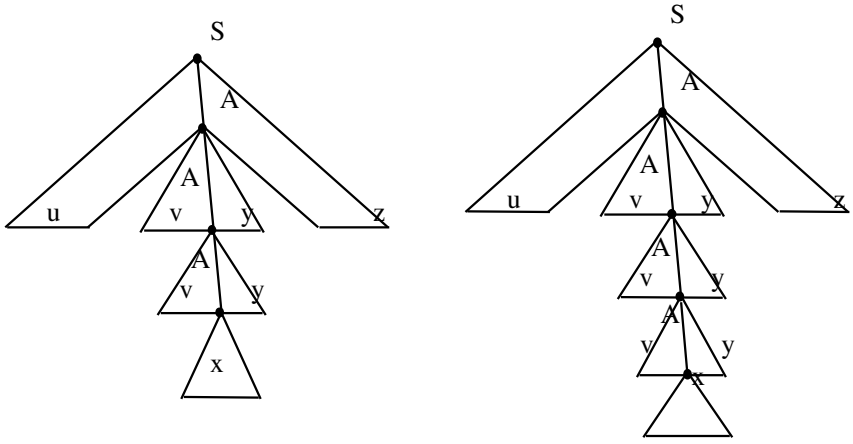


Giả thiết rằng các thành phần bên phải của các sản xuất của văn phạm G có không quá P phần tử, nghĩa là

$P = \max \{ | \alpha | , A \rightarrow \alpha \in R \}$.

Mỗi nút của một cây phân tích của G như vậy sẻ có tối đa p nút thừa kế. Độ dài tối đa của một câu được sinh ra bởi một cây phân tích có độ sâu m sẽ bằng số các lá tối đa của nó, giả sử P^m.

Do đó, nếu ta lấy $K = P^m$ mà $m = | \{ V- S\} |$, mọi cây phân tích sản sinh ra câu có độ dài lớn hơn K sẽ phải chứa ít nhất một đường đi dài hơn m và chứa tối thiểu hai lần cùng một ký tự không kết thúc.



Ở đây, có thể bảo đảm rằng cả hai câu v và y đều không đồng thời rỗng trên đường này.

Thật vậy, nếu cả hai câu v và y rỗng, phần cây phân tích giữa hai vế của ký tự không kết thúc A có thể bỏ đi mà không thay đổi câu đã sản sinh.

Nếu điều đó cũng xảy ra cho mọi đường đi có độ dài lớn hơn m, câu có thể được sản sinh bởi một cây phân tích có độ sâu lớn hơn m và không thể có một độ dài vượt quá $K = p^m$.

Chú ý :

Trong phát biểu định lý 4.2, ngôn ngữ *L* không được chỉ rõ ra là phải vô hạn. Liệu định lý có còn thỏa mãn không, nếu *L* hữu hạn ?

Điều này có thể suy ra ngay : Một ngôn ngữ PNC hữu hạn không thể có các câu có độ dài vượt quá p^m.

Vả lại, có thể thấy rằng định lý bơm cho các ngôn ngữ chính quy là trường hợp đặc biệt của các ngôn ngữ PNC, tương ứng với trường hợp hoặc v = e, hoặc y = ε.

III.3. Ap dụng định lý "bơm"

Định lý bơm cho phép chứng minh rằng một số ngôn ngữ không là PNC.

Ví dụ 4.6 : Ngôn ngữ $L = \{a^n b^n c^n\}$ không là PCN.

Để chứng minh, ta cần chỉ ra rằng không có khả năng tách một câu có dạng $a^n b^n c^n$ thành 5 phần u, v, x, y và z (với v và y không rỗng) sao cho với mỗi j > 0, câu $uv^j xy^j z \in L$ và như vậy mâu thuẫn với định lý bơm.

Giả sử rằng tồn tại một sự tách như vậy và ta sẽ xét khả năng khác nhau cho các câu v và y :

Cả hai câu v và y đều được tạo thành từ phép lặp của cùng một chữ, chẳng hạn $v \in a^*$ và $y \in b^*$. Trong trường hợp này, sự bằng nhau của số các chữ *a*, *b* và c không thể có sự lặp của v và y.

Các câu v và y được tạo thành từ các chữ khác nhau. Trong trường hợp này, các câu $uv^j xy^j z$ sẽ không còn có dạng $a^* b^* c^*$.

Chú ý rằng ngôn ngữ PNC $a^n b^n$ thỏa mãn định lý "bơm". Thật vậy, chỉ cần chọn cho v một dãy chữ *a* và cho y một dãy chữ *b* có cùng một độ dài như nhau.

Định lý "bơm" cũng cho phép chứng minh rằng các phép toán giao và bù không phải luôn luôn nói lên tính PNC của các ngôn ngữ. Ta có các tính chất sau đây :

Tồn tại hai ngôn ngữ PNC L_1 và L_2 sao cho $L_1 \cap L_2$ không là PNC. Thật vậy :

$L_1 = \{a^n b^n c^m\}$ và $L_2 = \{a^m b^n c^n\}$ đều là các ngôn ngữ PNC. Tuy nhiên :

$L_1 \cap L_2 = \{a^n b^n c^n\}$ không là PNC !

Bù của một ngôn ngữ PNC không phải luôn luôn PNC. Thật vậy, hợp của hai ngôn ngữ PNC là PNC. Như vậy nếu bù của một ngôn ngữ PNC luôn luôn là PNC, thì giao của hai ngôn ngữ PNC sẽ là PNC, vì rằng :

$L_1 \cap L_2 = L_1 \cup L_2$

Điều này mâu thuẫn với tính chất đã nêu ở 1.

III.4. Các thuật giải cho các ngôn ngữ PNC

Trong 3.4.2, ta thấy rằng phần lớn các bài toán liên quan đến các ngôn ngữ chính quy đều giải được bởi các thuật giải.

Với các ngôn ngữ PNC, vấn đề không còn đúng nữa. Ta xét ví dụ sau đây. Giả sử cho *L* là ngôn ngữ PNC (định nghĩa bởi một văn phạm PNC hoặc bởi một ôtômat đẩy xuống).

1. Cho câu *w*, tồn tại thuật giải xác định rằng $w \in L$ hay không.
2. Tồn tại thuật giải xác định rằng $L = \varnothing$ hay không.
3. Không tồn tại thuật giải xác định rằng $L = \Sigma^*$.

4. Nếu *L'* cũng là một ngôn ngữ PNC, khi đó, không tồn tại thuật giải xác định xem nếu $L \cap L' = \varnothing$ hay không.

Định lý 4.4 :

Cho văn phạm PNC G, tồn tại thuật giải để xác định xem một câu *w* đã cho có thuộc *L* (G) ?

Chứng minh :

Đầu tiên, ta xây dựng một ôtômat đẩy xuống thừa nhận *L*(G). Tiếp theo ta mô phỏng việc thực hiện của ôtômat này trên câu *w*. Tuy nhiên, ôtômat đẩy xuống xây dựng từ văn phạm PNC là không đơn định nên tồn tại nhiều cách thực hiện trên câu *w*.

Mặt khác, không giới hạn được số lần thực hiện vì ôtômat đẩy xuống có thể có các chuyển tiếp làm thay đổi DSĐX mà đầu đọc không hề chuyển tiếp trên câu vào *w*. Không bảo đảm được quy trình thực hiện là hữu hạn và không có giới hạn cho số bước chuyển tiếp (số bước thực hiện).

Vấn đề có thể giải được là giới hạn độ dài các chuyển tiếp, không phải trong ngữ cảnh các ôtômat mà trong ngữ cảnh các văn phạm. Một cách cụ thể, ta sẽ chỉ ra rằng nếu một câu có thể được sinh ra bởi một văn phạm, thì nó thể được sinh ra bởi một số giai đoạn giới hạn bởi một hàm theo độ dài của nó. Từ tính chất này, ta có thể xác định câu *w* được sinh ra bởi văn phạm G hay không bằng cách sử dụng thuật giải sau đây :

1. Tính số biên là số giai đoạn cần thiết để suy dẫn một câu có độ dài là độ dài của *w*. Giả sử biên đó là k.
2. Xây dựng một cách có hệ thống tất cả các chuyển tiếp có độ dài nhỏ hơn hoặc bằng k.

Số chuyển tiếp là hữu hạn vì rằng mỗi giai đoạn chuyển tiếp, sẽ có một số hữu hạn phép chọn có thể được cho giai đoạn tiếp theo : một lựa chọn bởi ký tự không kết thúc có thể được thay thế và một lựa chọn bởi sản xuất có thể được áp dụng cho ký tự không kết thúc đã thay thế.

3. Nếu một trong những suy dẫn này tạo sinh ra câu *w*, thì *w* được tạo sinh bởi văn phạm G. Nếu không phải như vậy, thì *w* không thể được sinh ra bởi văn phạm G và sẽ không thuộc *L*(G).

Vấn đề xảy ra là không có giới hạn cho độ dài các suy dẫn để tạo sinh một câu có độ dài đã cho.

Thật vậy, các sản xuất dạng $A \to B$ và $B \to A$ có thể dẫn đến các suy dẫn có độ dài tùy ý :

$$A \Rightarrow B \Rightarrow A \Rightarrow B \Rightarrow A \Rightarrow ...$$

Để khắc phục tình trạng này, cần chỉ ra rằng xuất phát từ một văn phạm PNC nào đó, có khả năng xây dựng một văn phạm PNC tương đương (sản sinh cùng một ngôn ngữ) mà mọi sản xuất có một trong hai dạng sau :

1. Dạng chuẩn Chomsky :

   $A \to BC \mid a$ với $A, B, C \in N$, $a \in S$.

2. Dạng chuẩn Greibach :

   $A \to aa$ với $A \in N$, $a \in S$, $a \in V^*$

Một ngoại lệ duy nhất là cho phép có mặt sản xuất $S \to \varepsilon$ với S là ký tự đầu và S không xuất hiện trong vế phải của bất kỳ một sản xuất nào.

Với văn phạm này, cần giới hạn độ dài các suy dẫn có thể được cho một câu *w*. Thật vậy, mỗi lần áp dụng một sản xuất dạng 2 sẽ làm tăng ít nhất một lượng là độ dài của chuỗi (gồm ký hiệu kết thúc và không kết thúc).

Như vậy, để sản sinh một câu có độ dài $|w|$, không thể áp dụng hơn $|w| - 1$ sản xuất dạng 2.

Trong trường hợp xấu nhất, chuỗi nhận được sau khi áp dụng các sản xuất dạng 2 sẽ chỉ gồm các ký hiệu không kết thúc để nhận được một câu gồm chỉ các ký hiệu kết thúc, cần phải áp dụng $| w |$ lần các sản xuất dạng 1. Với câu w, số sản xuất tối đa sử dụng, như vậy sẽ là giới hạn độ dài các suy dẫn, bằng $2 \times | w | - 1$.

Bây giờ chỉ cần chỉ ra rằng mọi văn phạm PNC có thể biểu diễn dưới dạng vừa trình bày ở trên. Trước hết, cần chỉ ra rằng có thể loại bỏ các sản xuất dạng $A \rightarrow \varepsilon$.

Thật vậy, với mỗi sản xuất dạng $A \rightarrow \varepsilon$ và mỗi sản xuất dạng $B \rightarrow vAu$, thêm vào sản xuất $B \rightarrow vu$. Sau đó loại bỏ các sản xuất dạng $A \rightarrow \varepsilon$.

Nếu trong số các sản xuất đã loại bỏ, tồn tại sản xuất rỗng $S \rightarrow \varepsilon$, đưa vào ký tự đầu mới S' và thêm vào các sản xuất $S' \rightarrow \varepsilon$ và sản xuất $S' \rightarrow a$ với mỗi sản xuất dạng $S \rightarrow \alpha$.

Để loại bỏ các sản xuất dạng $A \rightarrow B$, cần xác định các cặp ký tự không kết thúc A và B nếu $A \overset{*}{\Rightarrow} B$. Mỗi lần như vậy, với các sản xuất dạng $B \rightarrow u$ ($u \notin V - \sum$), thêm vào sản xuất $A \rightarrow u$. Từ đó có thể loại bỏ các sản xuất dạng $A \rightarrow B$.

Định lý 4.5 :

Cho văn phạm PNC G, tồn tại thuật giải xác định nếu $L(G) = \varnothing$.

Chứng minh :

Một văn phạm PNC sản sinh ít nhất một câu nếu tồn tại một cây phân tích cho nó.

Thật vậy, theo định nghĩa, các lá của một cây phân tích sẽ là những ký tự kết thúc (hoặc là câu rỗng ε) và như vậy, mọi cây phân tích sẽ sản sinh một câu.

Thuật giải sau đây mô tả việc tìm kiếm một cây phân tích :

1. Xây dựng cây phân tích bằng cách xét liên tiếp các cây có độ sâu tăng dần :
   - Bắt đầu từ cây có độ cao 0 gồm chỉ một nút duy nhất có nhãn chính là ký tự đầu S.
   - Các cây có độ sâu n + 1 nhận được từ cây có độ sâu n bằng cách áp dụng tất cả các sản xuất có thể được cho các lá có nhãn là những ký hiệu không kết thúc.
2. Ngay khi một trong những cây đang xét thực sự là một cây phân tích (mọi lá đều có nhãn là những ký hiệu kết thúc hoặc ε), việc tìm kiếm được dừng lại và người ta có thể nói rằng $L(G) \neq \varnothing$.

Điểm yếu duy nhất của phương pháp này là không biết được khi nào thì việc tìm kiếm dừng lại trong trường hợp không có câu nào được sản sinh bởi văn phạm G. Trong trường hợp này, chúng ta sẽ không bao giờ nhận được câu trả lời.

Để khắc phục điều này, ta sẽ chỉ ra rằng ta chỉ giới hạn việc tìm kiếm cho những cây phân tích có độ sâu trong phạm vi $| V - \sum |$.

Thật vậy, nếu tồn tại một câu sản sinh bởi G và cây phân tích cho câu này có độ sâu lớn hơn $| V - \sum |$, những đường đi của cây này có độ sâu vượt $| V - \sum |$ sẽ chứa một ký hiệu không kết thúc nào đó ít nhất 2 lần.

Như thế, người ta có thể loại bỏ một phần của cây phân tích gồm giữa hai ký hiệu không kết thúc giống nhau trên cùng một đường đi mà vẫn nhận được một cây phân tích mới.

Việc loại bỏ này có thể lặp lại cho đến khi không còn một đường đi nào trên cây phân tích chứa hai lần cùng một ký hiệu không kết thúc và như vậy, sẽ có độ sâu không vượt quá $| V - \sum |$.

Bởi vậy, nếu tồn tại một câu sản sinh bởi G, sẽ cũng tồn tại một câu sản sinh bởi G với cây phân tích có độ sâu không vượt quá $| V - \sum |$. Việc tìm kiếm một cây sản sinh ra một câu của $L(G)$ có thể được dừng lại ở những cây có độ sâu $| V - \sum |$.

Nếu không có một câu nào được sản sinh bởi một cây phân tích có độ sâu nhỏ hơn,

$L(G) = \varnothing$.

IV. Các ôtômat đẩy xuống đơn định

IV.1. Nguyên lý

Cho đến lúc này, ta mới chỉ xét những ôtômat đẩy xuống không đơn định. Bây giờ ta sẽ xét tới những ôtômat đẩy xuống đơn định. Lợi ích của các ôtômat đẩy xuống đơn định ở chỗ, không phải như các ôtômat đẩy xuống không đơn định, chúng biểu diễn một thủ tục tính toán dễ mô phỏng trực tiếp.

Chẳng hạn, nếu một ngôn ngữ PNC được thừa nhận bởi một ôtômat đẩy xuống đơn định mà mỗi đoán nhận đều hữu hạn, ôtômat này biểu diễn một thuật giải để nhận biết các câu xuất hiện trong ngôn ngữ này. Thuật giải này sẽ hiệu quả hơn nhiều lần thuật giải mà ta đã trình bày trong việc chứng minh định lý 4.4.

Một ôtômat đẩy xuống là đơn định nếu trong mỗi cấu hình chuyển tiếp, chỉ có duy nhất một chuyển tiếp có thể. Bây giờ ta sẽ hình thức hóa điều này.

IV.2. Hình thức hóa

Chúng ta bắt đầu bằng việc định nghĩa khái niệm các *chuyển tiếp tương thích*. Có thể thấy ngay là hai chuyển tiếp là tương thích nếu tồn tại một cấu hình mà hai chuyển tiếp này đều có thể xảy ra được.

Định nghĩa 4.5 :

Hai chuyển tiếp $((p_1, u_1, \beta_1), (q_1, \delta_1))$ và $((p_2, u_2, \beta_2), (q_2, \delta_2))$ được gọi là tương thích nếu :

- $p_1 = p_2$,
- u_1 và u_2 tương thích

  (nghĩa là u_1 là một tiền tố của u_2 hoặc u_2 là một tiền tố của u_1),
- β_1 và β_2 tương thích.

Một ôtômat đẩy xuống đơn định, đơn giản mà nói là một ôtômat đẩy xuống mà không có các chuyển tiếp tương thích phân biệt nhau.

Định nghĩa 4.6 :

Một ôtômat đẩy xuống là đơn định nếu với mỗi cặp chuyển tiếp tương thích, các chuyển tiếp này là đồng nhất với nhau.

Vấn đề đặt ra một cách tự nhiên là có phải mỗi ngôn ngữ PNC đều có thể được thừa nhận bởi một ôtômat đẩy xuống đơn định ? Câu trả lời là không thể. Tồn tại các ngôn ngữ PNC mà không được thừa nhận bởi một ôtômat đẩy xuống nào.

Các ngôn ngữ được thừa nhận bởi các ôtômat đẩy xuống đơn định lập nên một lớp con của các ngôn ngữ PNC. Chúng được gọi là các ngôn ngữ PNC đơn định (deterministic context free languages).

IV.3. Các ngôn ngữ PNC đơn định

Các ngôn ngữ PNC đơn định là những ngôn ngữ được thừa nhận bởi một ôtômat đẩy xuống đơn định.

Định nghĩa 4.7 :

Cho L là một ngôn ngữ được định nghĩa trên bảng chữ Σ, ngôn ngữ L là PNC đơn định nếu và chỉ nếu L được thừa nhận bởi một ôtômat đẩy xuống đơn định.

Ví dụ 4.7 :

Ngôn ngữ $L_1 = \{ wcw^R \mid w \in \{a, b\}^* \}$ là PNC đơn định.

Trong khi đó, ngôn ngữ $L_2 = \{ ww^R \mid w \in \{a, b\}^* \}$ chỉ là PNC, nhưng không PNC đơn định. Dễ thấy ngay rằng với L_2, cần phải xác định đâu là vị trí giữa của một câu để nhận biết L_2. Trong khi đó, với L_1 vị trí giữa được xác định bởi ký tự c.

IV.4. Tính chất của các ngôn ngữ PNC đơn định

Lớp các ngôn ngữ PNC đơn định có các tính chất khác với lớp các ngôn ngữ PNC. Nếu L_1 và L_2 là các ngôn ngữ PNC đơn định, ta có các tính chất sau đây :

1. **Ngôn ngữ $\Sigma^* - L_1$ là PNC đơn định.**

Dễ thấy ngay rằng để lấy bù một ngôn ngữ thừa nhận bởi một ôtômat đơn định, chỉ cần nghịch đảo các trạng thái đạt được và không đạt được. Thực tế, việc chứng minh tương đối phức tạp, vì rằng một ôtômat đẩy xuống có thể có các chuyển tiếp trên câu rỗng.

2. **Tồn tại các ngôn ngữ PNC không đơn định PNC**

Thật vậy nếu tất cả các ngôn ngữ PNC đều đơn định PNC, thì phần bù của một ngôn ngữ PNC sẽ luôn luôn PNC, điều này như đã biết là sai.

3. **Các ngôn ngữ L1 $\cup$ L2 và L1 $\cap$ L2 không phải luôn luôn đơn định PNC**

Đối với phép hội, điều đó dễ hiểu vì rằng, để xác định xem nếu một câu là trong $L_1 \cup L_2$, cần phải xem nếu câu đó là trong L_1 hoặc trong L_2, điều này không phải luôn luôn làm được một cách xác định.

Tính chất đối với phép giao được rút ra từ các tính chất của phép hội và phép lấy phần bù.

IV.5. Ứng dụng

Ứng dụng chính của các ngôn ngữ PNC là việc mô tả cú pháp của các ngôn ngữ lập trình và việc phân tích cú pháp tương ứng. Trong ứng dụng này, cú pháp của một ngôn ngữ lập trình được đưa ra bởi một văn phạm PNC. Các chương trình đúng đắn về cú pháp là những câu được sản sinh bởi văn phạm này.

Vấn đề phân tích cú pháp là việc xác định nếu một câu thuộc ngôn ngữ sản sinh bởi văn phạm PNC và thiết lập cách thức sản sinh ra câu này (nghĩa là dựa vào cây phân tích).

Như đã thấy, vấn đề này đợc giải quyết bởi một thuật giải áp dụng cho bất kỳ một văn phạm PNC nào. Tuy nhiên, để nhận được các thuật giải phân tích một cách hiệu quả (áp dụng cho các chương trình rất dài), cần phải đưa vào các hạn chế đối với kiểu văn phạm PNC muốn

sử dụng. Một trong những hạn chế có hiệu quả là chỉ xét những văn phạm mô tả các ngôn ngữ PNC đơn định. Một họ các văn phạm chuyên được sử dụng cho các ngôn ngữ lập trình là họ các văn phạm LR.

Bài tập chương 4

1. Mô tả các ôtômat đẩy xuống thừa nhận các ngôn ngữ sau đây :

   a) $a^n b^n c^m$ *b*) $a^n b^m c^n$

2. Tìm văn phạm PNC sản sinh các ngôn ngữ sau đây :

   a) $a^n b^n c^m$ *b*) $a^n b^m c^n$

3. Chứng minh rằng ngôn ngữ $\{ a^i b^j c^k \mid i \neq j \text{ hoặc } i \neq k \}$ là PNC.
   Phần bù của ngôn ngữ này cũng là PNC ?
   Gợi ý : hội của các ngôn ngữ PNC cũng là PNC.

4. Chứng minh rằng ngôn ngữ $\{ a^n \mid n \text{ là số nguyên tố} \}$ không là PNC.

CHƯƠNG 5

Các máy Turing

Dẫu rằng được trang bị một bộ nhớ lớn tùy ý, các ô tô mat đẩy xuống vẫn không thể thừa nhận ngôn ngữ $a^n b^n c^n$.

Tuy nhiên, có thể tìm một thủ tục hiệu quả để thừa nhận ngôn ngữ này. Như chúng ta sẽ thấy, người ta không thể sử dụng các ôtômat đẩy xuống mà phải tìm kiếm các lớp ôtômat khác. Đó là các máy Turing[4].

Sự khác nhau căn bản giữa các ôtômat đẩy xuống và các máy Turing là các máy Turing có bộ nhớ lớn tùy ý và cách sử dụng bộ nhớ không giới hạn ở nguyên lý danh sách đẩy xuống (LIFO: Last In First Out).

I. Định nghĩa máy Turing

I.1. Mô tả máy Turing đơn định

Một máy Turing đơn định (Deterministic Turing Machine) gồm các phần tử như sau :

1. Một bộ nhớ vô hạn có dạng một băng vào (tape) được chia thành nhiều ô. Mỗi ô có thể chứa một ký tự thuộc một bảng chữ nào đó (tape alphabet). Băng chỉ có cận trái mà không có cận phải (cận phải kéo dài ra vô hạn).
2. Một đầu đọc (read head hay tape head) di chuyển trên băng.
3. Một tập hợp hữu hạn các trạng thái gồm một trạng thái đầu và một tập hợp các trạng thái thừa nhận (trạng thái cuối).
4. Một hàm chuyển tiếp cho phép ứng với mỗi trạng thái của máy và ký tự tại vị trí dưới đầu đọc thì xác định :
   - Trạng thái tiếp theo
   - Một ký tự trên băng tại vị trí dưới đầu đọc sẽ được đọc.
   - Một chiều di chuyển của đầu đọc.

Việc hoạt động của máy Turing được mô tả như sau :

1. Đầu tiên, câu vào nằm ở đầu băng. Tất cả những ô còn lại của băng chứa những ký hiệu đặc biệt, gọi là các ký hiệu trống (blank symbol). Đầu đọc nằm ở ô đầu tiên của băng và máy đang ở trạng thái đầu.
2. Với mỗi trạng thái thực hiện, máy sẽ :
   - Đọc ký hiệu nằm ở dưới đầu đọc,
   - Thay thế ký hiệu này bởi ký hiệu được cho trong hàm chuyển tiếp,

---

4 Alan Turung (1912 – 1954) : nhà Toán học người Anh, người đầu tiên đưa ra các ôtômat vào năm 1936.

– Di chuyển đầu đọc sang phải hoặc sang trái một ô theo chiều quy định của hàm chuyển tiếp.

– Thay đổi trạng thái như đã cho trong hàm chuyển tiếp.

Một câu được thừa nhận bởi máy Turing khi việc thực hiện đạt được trạng thái thừa nhận.

I.2. Định nghĩa hình thức

Một máy Turing được mô tả bởi bộ bảy M = (Q, **Γ, Σ, δ**, S, B, A), trong đó :

– Q là tập hữu hạn các trạng thái.

– **Γ** là bảng chữ băng (sử dụng trên dải).

– $\mathbf{\Sigma} \subseteq \mathbf{\Gamma}$ là bảng chữ vào (dùng cho các câu vào).

– $\mathbf{s} \in$ Q là trạng thái đầu.

– $A \subseteq Q$ là tập hợp các trạng thái thừa nhận.

– $B \in \mathbf{\Gamma} - \mathbf{\Sigma}$ là ký tự trống (thường ký hiệu).

– $\delta : Q \times \mathbf{\Gamma} \rightarrow Q \times \mathbf{\Gamma} \times \{ L, R \}$ là hàm chuyển tiếp, *L* để chỉ việc dịch đầu đọc sang trái (Left) và R để chỉ sang phải (Right).

Cũng như đối với các lớp ôtômat khác, ngôn ngữ thừa nhận bởi một máy Turing được xây dựng nhờ các khái niệm hình trạng và sử dụng chuyển đổi giữa các hình trạng. Một hình trạng chứa tất cả các thông tin cần thiết để tiếp tục việc đoán nhận, nghĩa là :

1. Trạng thái.
2. Nội dung của băng.
3. Vị trí của đầu đọc.

Sự khó khăn duy nhất trong việc biểu diễn các hình trạng của máy Turing là băng vô hạn. Nội dung của băng là một dãy vô hạn các ký hiệu của bảng chữ băng. Tuy nhiên, ở mọi thời điểm đoán nhận, chỉ duy nhất một phần hữu hạn của băng là có thể được sử dụng bởi máy Turing.

Thật vậy, đầu tiên, băng chứa câu vào mà độ dài của câu này là hữu hạn và không đổi. Sau đó, mỗi thay đổi trạng thái sẽ chuyển tiếp đầu đọc sang một ô mới.

Như vậy sau n giai đoạn, máy Turing vượt qua ít nhất n ô, số ô tối đa đạt được khi tất cả các chuyển tiếp đều hướng sang bên phải.

Bởi vậy, nội dung của băng có thể ở mọi thời điểm được định nghĩa bởi nội dung của các tiền tố hữu hạn, phần còn lại chỉ chứa các ký hiệu trống.

Các hình trạng của máy Turing liên quan đến *ba* yếu tố sau :

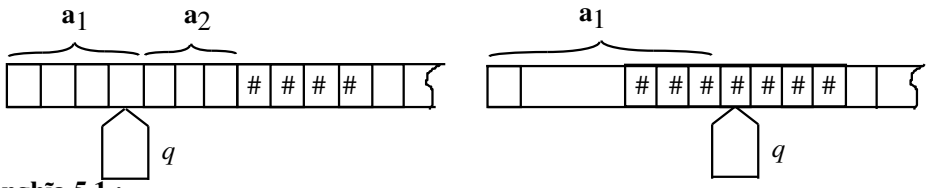
1. Trạng thái của máy.
2. Câu vào xuất hiện trên băng trước vị trí đầu đọc
3. Câu xuất hiện trên băng giữa vị trí đầu đọc và ký tự cuối cùng khác ký tự trống.

Một cách hình thức, một hình trạng là một phần tử của quan hệ :

$Q \times \Gamma^* \times (\varepsilon \cup \mathbf{\Gamma}^* (\Gamma - \{ B \}))$

Ví dụ 5.1 :

Các hình trạng tương ứng với các dãy là (q, α_1, α_2) và (q, α_1, ε) là :



Định nghĩa 5.1 :

Cho hình trạng (q, α_1, α_2). Ta viết hình trạng này dưới dạng (q, α_1, $b\boldsymbol{\alpha}'_2$) bằng cách lấy b = # trong trường hợp $\alpha_2 = \varepsilon$.

Các hình trạng chuyển tiếp xuất phát từ (q, α_1, α_2) sẽ được định nghĩa như sau :

- Nếu $\delta(q, b) = (q', b', R)$, ta có :

$$(q, \alpha_1, b\alpha'_2) \vdash_{\mathbf{M}} (q', \alpha_1 b', \alpha'_2)$$

- Nếu $\delta(q, b) = (q', b', L)$ và nếu $\alpha_1 \neq \varepsilon$ có dạng $\alpha'_1 a$, ta có :

$$(q, \alpha'_1 a, b\alpha'_2) \vdash_{\mathbf{M}} (q', \alpha'_1, ab'\alpha'_2)$$

Khái niệm về chuyển tiếp nhiều giai đoạn cũng được định nghĩa tương tự chương trước.

Định nghĩa 5.2 :

Hình trạng C chuyển tiếp sang hình trạng C' qua nhiều giai đoạn bởi máy Turing M, $C \vdash^*_{\mathbf{M}} C'$, nếu tồn tại $k \geq 0$ và các hình trạng trung gian $C_0, C_1, \ldots, C_k$ sao cho :

- $C = C_0$,
- $C' = C_k$,
- $C_i \vdash_{\mathbf{M}} C_{i+1}$ với $0 \leq i < k$.

Một câu được thừa nhận bởi một máy Turing nếu việc đoán nhận của máy Turing dẫn đến một hình trạng chứa trạng thái đạt được. Ta định nghĩa một ngôn ngữ thừa nhận bởi một máy Turing như sau :

Định nghĩa 5.3 :

Ngôn ngữ L (M) được thừa nhận bởi một máy Turing là tập hợp các câu w sao cho :

$$(q_0, \varepsilon, w) \vdash^*_{\mathbf{M}} (p, a_1, a_2), \text{ với } p \in A.$$

Ví dụ 5.2 : Cho máy Turing $M = (Q, \Gamma, \Sigma, \delta, q_0, B, A)$ với :

$Q = \{ q_0, q_1, q_2, q_3, q_4 \}$, $\Gamma = \{ a, b, X, Y, \# \}$, $\Sigma = \{ a, b \}$, $A = \{ q_4 \}$,

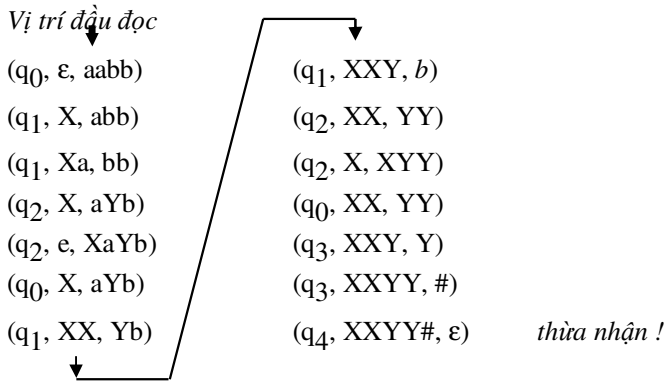
δ được cho bởi bảng dưới đây (dấu "–" chỉ ra rằng hàm chuyển tiếp không được định nghĩa cho các giá trị này).

	a	*b*	X	Y	#
q_0	(q_1, X, R)	—	—	(q_3, Y, R)	—
q_1	(q_1, a, R)	(q_2, Y, L)	—	(q_1, Y, R)	—
q_2	(q_1, a, L)	—	(q_0, X, R)	(q_2, Y, L)	—
q_3	—	—	—	(q_3, Y, R)	$(q_4, \#, R)$
q_4	—	—	—	—	—

Có suy ra rằng máy Turing đã cho thừa nhận ngôn ngữ $a^n b^n$. Thật vậy, máy sẽ thay thế lần lượt một cặp các ký hiệu *a* và *b* bởi X và Y.

Nếu tất cả các thay thế là có thể được và mỗi lần băng không còn chứa các ký hiệu *a* và *b*, câu vào sẽ được thừa nhận.

Ví dụ, dãy các hình trạng nhận được từ câu vào aabb như sau :



Để đơn giản, người ta viết các chuyển tiếp trên theo dạng chuỗi.

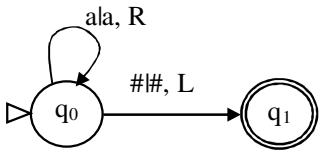
Các chuyển tiếp đoán nhận câu aabb lần lượt như sau :

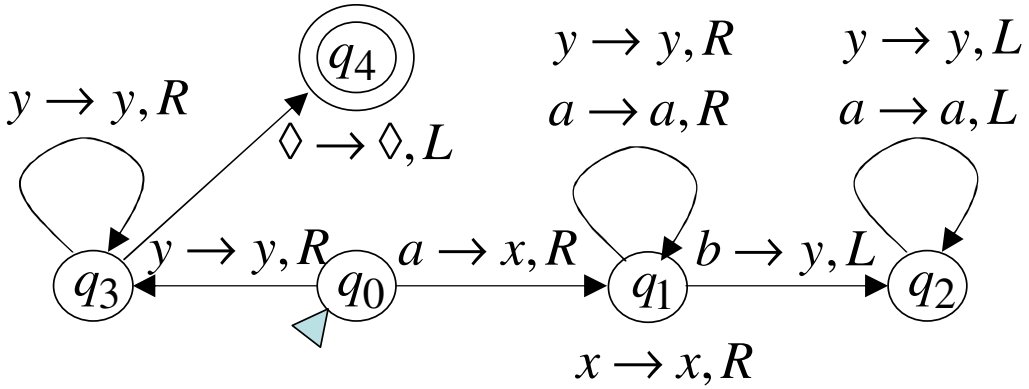
$q_0\underline{a}abb\#$	$q_0Xa\underline{Y}b\#$	$q_0XXY\underline{Y}\#$
$q_1Xabb\#$	$q_1XXYb\#$	$q_3XXYY\#$
$q_1Xa\underline{b}b\#$	$q_1XXY\underline{b}\#$	$q_3XXYY\underline{\#}\#$
$q_2XaYb\#$	$q_2XXYY\#$	$q_4XXYY\#\#$
$q_2Xa\underline{Y}b\#$	$q_2XXY\underline{Y}\#$	thừa nhận !

Tương tự, dãy các chuyển tiếp từ câu vào $a\underline{a}abbb$ được cho như sau :

$q_0aaabbb\#$	$q_2XaaYbb\#$	$q_2XXXYYY\#$
$q_1\underline{X}aabbb\#$	$q_0\underline{X}aaYbb\#$	$q_0XX\underline{X}YYY\#$
$q_1Xa\underline{a}bbb\#$	. . . . .	$q_3XXX\underline{Y}YY\#$
$q_1Xaa\underline{b}bb\#$	$q_1XXXYYb\#$	$q_3XXXY\underline{Y}Y\#$
$q_2XaaYbb\#$	$q_2XXXYYY\#$	$q_3XXXYYY\#$
$q_2XaaY\underline{b}b\#$	$q_2XXXYY\underline{Y}\#$	$q_4XXXYY\underline{Y}\#\#$
		thừa nhận !

Ví dụ 5.3 : Máy Turing thừa nhận ngôn ngữ chính quy aa* + b(a+b)* :





I.3. Ngôn ngữ thừa nhận được và ngôn ngữ xác định được

Môt máy Turing khi thừa nhận một ngôn ngữ lại không phải luôn luôn mô tả một thủ tục hiệu quả để nhận biết ngôn ngữ này. Thật vậy, xem xét dãy các hình trạng nhận được từ (q_0, ε, *w*) (dãy này là duy nhất, vì rằng ta đang xét các máy đơn định), nhiều trường hợp có thể xảy ra như sau :

1. Dãy các chuyển tiếp chứa một hình trạng có chứa trạng thái kết thúc. Trong trường hợp này, câu vào được thừa nhận ngay khi trong hình trạng đạt được trạng thái kết thúc.
2. Dãy các hình trạng được kết thúc bởi một hình trạng mà không còn hình trạng tiếp theo nào nữa, vì :

– Hàm chuyển tiếp không định nghĩa cho hình trạng này, hoặc

– Hàm chuyển tiếp đưa ra một sự dịch trái của đầu đọc mà hiện đầu đọc đang ở ô đầu tiên bên trái của băng.

3. Dãy các chuyển tiếp không bao giờ đạt đến trạng thái kết thúc và là vô hạn.

Trong hai trường hợp đầu, máy Turing cho phép xác định xem câu vào có thuộc ngôn ngữ đang xét hay không (thuộc ngôn ngữ trong trường hợp đầu, và không thuộc ngôn ngữ trong trường hợp thứ hai).

Trong trường hợp thứ *ba*, câu trả lời không bao giờ nhận được. Thật vậy, tại mọi thời điểm, ta không thể đảm bảo được rằng câu sẽ được thừa nhận, vì rằng không đảm bảo được máy Turing dừng hay không ?

Lúc này máy sẽ vượt qua nhiều hình trạng nhưng không đạt được trạng thái kết thúc và như vậy cũng không đạt được cho các hình trạng tiếp theo. Việc đoán nhận của máy là vô hạn và

máy sẽ thừa nhận một ngôn ngữ không được định nghĩa bởi một thủ tục hiệu quả (một thuật giải) để đoán nhận ngôn ngữ này.

Ta đưa vào một khái niệm mới : ngôn ngữ xác định được (decided language) bởi một máy Turing. Trước hết ta sẽ xem xét khái niệm đoán nhận của máy.

Định nghĩa 5.4 :

Việc đoán nhận (xử lý) (execution) của một máy Turing trên câu *w* là dãy cực đại các hình trạng :

$$(q_0, \varepsilon, w) \vdash_{\mathbf{M}} C_1 \vdash_{\mathbf{M}} C_2 \vdash_{\mathbf{M}} \cdots \vdash_{\mathbf{M}} C_k \vdash_{\mathbf{M}} \cdots$$

nghĩa là sao cho :

- Dãy hình trạng là vô hạn,
- Dãy tự kết thúc tại một hình trạng có chứa trạng thái kết thúc, hoặc
- Dãy tự kết thúc tại một hình trạng mà từ đó không còn hình trạng nào có thể chuyển đến.

Ngôn ngữ xác định được sẽ được định nghĩa như sau :

Định nghĩa 5.5 :

Một ngôn ngữ *L* là xác định được một máy Turing M nếu :

- M thừa nhận *L*,

- M không có các xử lý vô hạn.

Từ đó, một ngôn ngữ xác định được bởi một máy Turing có thể đoán nhận bởi một thủ tục hiệu quả. Trong mục 5.3, ta đã khẳng định rằng điều đó đúng và ngược lại cũng đúng. Vấn đề là khái niệm về ngôn ngữ xác định được có thể phù hợp với các kiểu ô tô mát đã xét.

Trước hết, đối với các ô tô mát đơn định, điều đó hiển nhiên vì rằng một ô tô mát khi xác định một ngôn ngữ đã biểu diễn một thủ tục hiệu quả. Vả lại, một ô tô mát không đơn định không biểu diễn thực sự một thuật giải vì rằng nó không thể chỉ ra một cách tường minh chuyển tiếp nào tiếp theo sẽ được chọn tại mỗi giai đoạn.

Ta có các nhận xét sau đây :

1. Với các ô hữu hạn đơn định, người ta không thể nói về ngôn ngữ xác định được (tính không xác định được). tuy nhiên, mọi ngôn ngữ thừa nhận được có thể xác định được bởi một ôtômat hữu hạn đơn định (định lý 2. 1).
2. Quan điểm về ngôn ngữ xác định được khong thể áp dụng cho các ôtomat đẩy xuống không đơn định. Tuy nhiên, định lý 4. 4 chỉ ra rằng tồn tại một thủ tục hiệu quả để nhận biết các ngôn ngữ thừa nhận được bởi các ôtômat đẩy xuống. Ta sẽ thấy rằng điều đó dẫn đến việc các ngôn ngữ có thể xác định được bởi một máy Turing
3. Một ôtômat đẩy xuống đơn định xác định một ngôn ngữ nếu nó không có đoán nhận vô hạn, tức là nếu ôtômat không chứa các vòng chuyển tiếp ε (các chuyển tiếp trên câu rỗng).

Tính xác định được của máy Turing có thể hiểu như sau :

Với mọi $(q, a) \in Q \times G$, tồn tại nhiều nhất một quy tắc $(q, a) \to (q', a', m)$, với $m = L$ hoặc R. Khi đó một hàm bộ phận $Q \times G \to Q \times G \times \{ L, R \}$ có thể tách thành *ba* hàm :

hàm "ký tự mới" $nc : Q \times G \to G$

hàm "di chuyển đầu đọc" $mh : Q \times G \to \{ L, R \}$

hàm "trạng thái mới" $ns : Q \times G \to Q$

Ba hàm này được xác định bởi $nc(q, a) = a'$, $mh(q, a) = m$ và $ns(q, a) = q'$ nếu và chỉ nếu $(q, a) \to (q', a', m)$ hay $qama' q' \in d$.

I.4. Các hàm tính được bởi máy Turing

Cho đến lúc này, ta mới chỉ xem xét các máy Turing như là các ôtômat đoán nhận một ngôn ngữ. Tuy nhiên một máy Turing có thể tính một hàm.

Đối số của hàm là câu vào và giá trị của hàm đơn giản là được ghi trên băng khi máy Turing kết thúc việc xử lý.

Định nghĩa 5. 9 :

Một máy Turing tính một hàm f : $S^* \rightarrow S^*$ nếu với một câu vào *w* bất kỳ, máy luôn luôn dừng trong một hình trạng mà f (*w*) có mặt ở trên băng.

Một hàm được gọi là tính được bởi máy Turing nếu tồn tại một máy Turing tính được nó. Chú ý rằng xác định một ngôn ngữ là tính một hàm trên khoảng xác đinh {0, 1}.

Ví dụ 1 :

Cho máy Turing M với G = { 0, 1, # }, Q = { q_1, q_2, q_3 }, A = { q_3 }

d : $q_1 0 R 0 q_1$ $q_2 0 L 1 q_3$

$q_1 1 R 1 q_1$ $q_2 1 L 0 q_2$

$q_1 \# L \# q_2$ $q_2 \# L 1 q_3$

Cho *w* = 10100, ta có đoán nhận $(e, q_1, 10100) \vdash_{\mathbf{M}}^{*} (1010, q_3, 1)$

Cho *w* = 100111, ta có đoán nhận $(e, q_1, 100111) \vdash_{\mathbf{M}}^{*} (10, q_3, 1000)$

Dễ dàng nhận thấy rằng, xuất phát từ hình trạng (e, q_1, w), sẽ tồn tại một đoán nhận của M dẫn đến hình trạng (a, q_1, b), sao cho nếu *w* biểu diễn nhị phân của số nguyên n thì câu ghép *ab* sẽ là biểu diễn nhị phân của số nguyên n + 1. Như vậy, M thực hiện việc tính hàm tính số nguyên kế tiếp dưới dạng biểu diễn nhị phân.

Ví dụ 2 :

Cho máy Turing M với G = { 1, # }, Q = { q_1, q_2, q_3, q_4 }, A = { q_4 }

d : $q_1 1 R \# q_2$ $q_3 1 R 1 q_3$

$q_2 1 R \# q_3$ $q_3 \# L 1 q_4$

Giả sử mỗi số nguyên n được biểu diễn bởi n + 1 số 1 (mục đích để phân biệt với các biểu diễn của số 0), chẳng hạn :

2 được biểu diễn bởi 111 3 số 1 liên tiếp

5 được biểu diễn bởi 111111 6 số 1 liên tiếp

Máy Turing đã cho có thể tính tổng của hai số nguyên m + n bằng cách viết m và n cách nhau một dấu #.

Cho *w* = 111#111111, ta có đoán nhận :

$(e, q_1, 111\#111111) \vdash_{\mathbf{M}}^{*} (\#\#11111111, q_4, \#)$

I.5. Các định nghĩa khác về máy Turing

Người ta có thể nghĩ ra nhiều kiểu máy Turing khác nhau. Chẳng hạn, có hai kiểu máy tương đối phổ biến cho phép xây dựng các lớp ngôn ngữ thừa nhận và xác định được.

I.5.1. Máy Turing loại một

Trong số các trạng thái của máy, chỉ có một trạng thái thừa nhận, gọi là trạng thái dừng (halt state).

Mặt khác, hàm dịch chuyển được định nghĩa tại mọi thời điểm. Tại trạng thái dừng, máy thừa nhận hoặc không thừa nhận một câu vào tùy theo ở đầu băng có chứa dấu hiệu "thừa nhận" (1) hay dấu hiệu "không thừa nhận" (0).

Máy thừa nhận một ngôn ngữ nếu với mọi câu của ngôn ngữ đã cho, việc đoán nhận đạt được trạng thái dừng, băng chứa số 1. Với các câu không thuộc ngôn ngữ, việc đoán nhận có thể không dừng (vô hạn) hoặc đạt được trạng thái dừng nhưng băng chứa số 0.

Máy xác định một ngôn ngữ nếu máy luôn luôn đạt được trạng thái dừng, băng chuá số 1 cho mọi câu thuộc ngôn ngữ, hoặc băng chuá số 0 với mọi câu không thuộc ngôn ngữ.

I.5.2. Máy Turing loại 2

Trong số các trạng thái của máy, người ta phân biệt hai trạng thái dừng là Q_Y và Q_N. Hàm chuyển tiếp được định nghĩa tại mọi thời điểm.

Máy thừa nhận một ngôn ngữ, nếu với mọi câu của ngôn ngữ việc đoán nhận của máy đạt được trạng thái dừng Q_Y. Với các câu không thuộc ngôn ngữ, việc đoán nhận có thể không bao giờ dừng (vô hạn) hoặc đạt được trạng thái dừng Q_N cho mọi câu không thuộc ngôn ngữ.

I.6. Các ngôn ngữ đệ quy và liệt kê đệ quy

Các ngôn ngữ xác định được bởi một máy Turing được gọi là đệ quy (recusive).

Các ngôn ngữ được thừa nhận bởi một máy Turing gọi là liệt kê đệ quy (recursively enumerable).

Từ đó ta có các định nghĩa sau :

Định nghĩa 5. 6 :

Một ngôn ngữ là đệ quy nếu nó được xác định bởi một máy Turing.

Định nghĩa 5. 7 :

Một ngôn ngữ là liệt kê đệ quy nếu nó được thừa nhận bởi một máy Turing.

I.7. Luận đề Turing-Church

Luận đề Turing-Church[5] được phát biểu như sau :

Các ngôn ngữ được nhận biết bởi một thủ tục hiệu quả là các ngôn ngữ xác định được bởi một máy Turing.

---

5 Alonzo Church : nhà Toán học người Mỹ đã nghiên cứu quan điểm về tính tính được (computability) từ việc tính hàm (functional calculus).

Thường thì người ta có thể phát biểu luận đề Turing - Chunch theo nghĩa của phép tính hàm :

Các hàm tính được bởi một thủ tục hiệu quả là các hàm tính đợc bởi một máy Turing.

Luận đề Turing-Church đóng một vai trò quan trọng trong việc nghiên cứu về lý thuyết tính toán được (computability).

Thật vậy, luận đề đưa ra lý lẽ để chứng minh rằng một số ngôn ngữ không thể được đoán nhận bởi một thủ tục hiệu quả : thực chất là sự hình thức hóa khái niệm tính toán.

Luận đề Turing-Church không phải là một định lý, nên không thể chứng minh được. Bởi vì Luận đề Turing-Church áp dụng mô hình lý thuyết là máy Turing được định nghĩa chặt chẽ để mô hình hoá quan niệm về thủ tục hiệu quả là khái niệm không được xác định rõ ràng.

Ta thấy rằng thuật giải được xác định bởi một máy Turing sẽ không được xác định bởi một thủ tục hiệu quả.

Thật vậy, dễ dàng mô phỏng tự hoạt động của một máy Turing nhờ một bút chì và tờ giấy hay hiện đại hơn, nhờ một chương trình chạy trên một máy tính cụ thể.

Vả lại, khó có thể phán định luận đề cho rằng mọi ngôn ngữ xác định bởi một thủ tục hiệu quả lại được xác định bởi một máy Turing.

Để làm được điều đó, lý luận chính được áp dụng ở đây là tất cả mọi sự mô hình hóa về khái niệm thủ tục hiệu quả đồng nhất với khái niệm ngôn ngữ xác định bởi một máy Turing.

Luận đề Turing-church được phán định như sau:

1. Sự mở rộng các máy Turing cũng như xây dựng các kiểu máy Turing khác nhau (nhất là các máy có bộ nhớ truy cập trực tiếp) không thể xác định được những ngôn ngữ nhờ máy Turing đã xác định.
2. Một mô hình hóa khác với khái niệm thủ tục hiệu quả (các hàm đệ quy chẳng hạn) cũng sẽ là đồng nhất với mô hình máy Turing.

II. Các kỹ thuật xây dựng máy Turing

II.1. Ghi nhớ ở bộ điều khiển hữu hạn

Có thể ghi nhớ một số hữu hạn thông tin nhờ các trạng thái của bộ điều khiển : xem mỗi trạng thái là một cặp phần tử. Phần tử đầu tham gia điều khiển, phần tử thứ hai là thông tin cần nhớ. Mặc dầu như vậy làm số trạng thái tăng lên, nhưng vẫn là hữu hạn.

Có thể xem mỗi trạng thái gồm bộ n phần tử, trong đó n-1 phần tử để ghi nhớ thông tin.

Ví dụ 5.3 :

Xây dựng máy Turing sao cho sau khi đọc ký tự đầu tiên thì ghi nhớ nó và sau đó kiểm tra rằng ký hiệu này không còn xuất hiện nơi nào khác trên băng. M = ({ 0, 1 }, Q, { 0, 1, B }, d, $[q_0, B]$, B, A)

trong đó Q = { q_0, q_1 } × { 0, 1, B } và A = { $[q_1, B]$ }.

Hàm d được cho như sau :

1. Từ trạng thái đầu $[q_0, B]$, M ghi nhớ ký hiệu đã đọc vào thành phần thứ hai của trạng thái.

d ($[q_0, B]$, 0) = ($[q_1, 0]$, 0, R)

d ($[q_0, B]$, 1) = ($[q_1, 1]$, 1, R)

2. Nếu ký hiệu đọc không giống với ký hiệu đã ghi nhớ thì cứ vượt qua bên phải.

d ([q_1, 0], 1) = ([q_1, 0], 1, R)

d ([q_1, 1], 0) = ([q_1, 1], 0, R)

3. M rơi vào trạng thái cuối [q_1, B] nếu gặp B.

d ([q_1, 0], B) = ([q_1, B], 0, *L*)

d ([q_1, 1], B) = ([q_1, B], 1, *L*)

Vậy M đã đọc hết câu vào và thừa nhận nó. Nếu gặp phải ký hiệu như đã ghi nhớ thì bị hóc vì (([q_1, 0], 0) và (([q_1, 1], 1) đều không xác định.

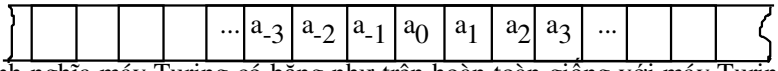
Ta có : $L(M) = \{ 01^n$ hoặc $10^n \}$

II.2. Mở rộng các máy Turing

Mục này trình bày hai mở rộng của các máy Turing và chứng minh rằng các ngôn ngữ được thừa nhận hay xác định bởi các máy mở rộng này cũng là được thừa nhận hay xác định bởi một máy Turing.

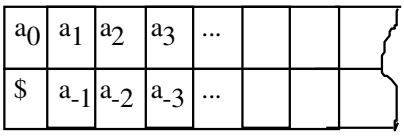
II.2.1. Băng vô hạn cả hai phía

Xét các máy Turing có băng vô hạn ở cả hai phía :



Cách định nghĩa máy Turing có băng như trên hoàn toàn giống với máy Turing cơ bản đã xét, trừ ngoại lệ là đầu đọc có thể di chuyển tùy ý về phía trái.

Ta cần chứng minh rằng mọi ngôn ngữ được thừa nhận hoặc được xác định bởi kiểu máy Turing mở rộng này cũng là ngôn ngữ được thừa nhận hoặc được xác định bởi máy Turing cơ bản có băng vô hạn về một phía. Ý tưởng để chứng minh là chuyển băng vô hạn cả hai phía thành một băng chỉ vô hạn về một phía bằng cách gập đôi băng lại. Kết quả sẽ là một băng mà mỗi ô chứa không phải một mà là hai ký hiệu.



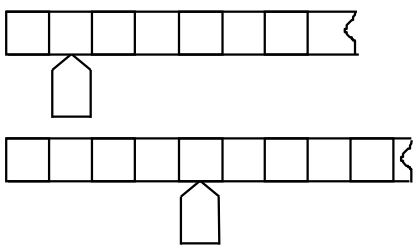
Mặc khác, cần chứng tỏ rằng mỗi cặp ký tự có thể được xét như một ký tự duy nhất của một bảng chữ lớn hơn và làm sao cho hàm chuyển tiếp của máy Turing cơ bản được thỏa mãn điều kiện chỉ có một phía băng vô hạn.

II.2.2. Máy Turing có nhiều băng

Xét các máy Turing có nhiều băng cùng lúc và có nhiều đầu đọc.

Loại máy này sẽ có hình trạng được đặc trưng bởi trạng thái, nội dung của mỗi băng và của mỗi đầu đọc.

Ta sẽ mô tả phỏng loại máy Turing này một lần nữa bởi một máy Turing cơ bản chỉ có một băng mà thôi.



Nguyên lý của việc mô phỏng là sử dụng các ký tự tổ hợp mô tả nội dung các băng khác nhau và vị trí của các đầu đọc.

Chẳng hạn, để mô phỏng một máy có hai băng, ta sử dụng một máy mà bảng chữ băng là bộ bốn được sử dụng để biểu diễn nội dung của mỗi băng, hai phần tử còn lại để biểu diễn các vị trí tương ứng của các đầu đọc (chẳng hạn giá trị 1 ở vị trí đầu đọc và 0 ở chỗ khác).

Để mô phỏng một giai đoạn thực hiện của máy hai băng, máy một băng phải :

- Tìm thấy vị trí các đầu đọc và xác định được các ký tự đã đọc.
- Thay đổi các ký tự này, di chuyển các đầu đọc và thay đổi trạng thái.

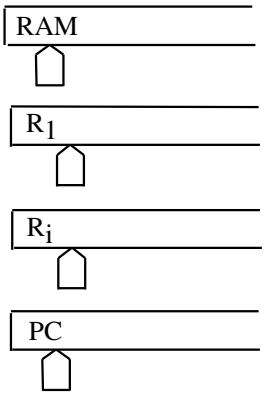
Việc xây dựng chính xác một máy Turing như vậy rất phức tạp, vấn đề là thuyết phục được tính khả thi của phương pháp.

II.2.3. Các máy Turing có bộ nhớ truy cập trực tiếp

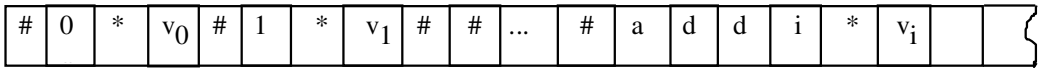
Ta hãy xem xét một máy nào đó giống hệt một máy tính thực và chỉ ra rằng kiểu máy này có thể được mô phỏng bởi một máy Turing.

Giả sử rằng đang mô phỏng có một bộ nhớ truy cập trực tiếp (RAM: random access memory) và một số thanh ghi, trong đó có một thanh đếm chương trình (PC: program counter).

Máy này có thể được mô phỏng bởi một máy Turing có nhiều băng. Một băng được dùng cho bộ nhớ và các băng khác dùng cho các thanh ghi.



Nội dung của bộ nhớ được biểu diễn bởi các cặp câu (địa chỉ, nội dung). Các phần tử của mỗi cặp như vậy được ngăn cách nhau bởi dấu * và các cặp liên tiếp nhau thì được đặt cách nhau bởi ký tự #.



Máy Turing sẽ mô phỏng máy có bộ nhớ RAM bằng cách lặp các chu kỳ như sau :

1. Chạy trên băng biểu diễn bộ nhớ RAM cho đến khi tìm thấy địa chỉ tương ứng với nội dung của thanh đếm chương trình PC.
2. Đọc và giải mã lệnh (decoding) tại địa chỉ này.
3. Đôi khi có thể tìm thấy các toán hạng của lệnh này.
4. Thực hiện lệnh. Có thể việc thực hiện này làm thay đổi bộ nhớ và/hoặc các thanh ghi.
5. Tăng thanh đếm chương trình (trừ trường hợp nếu lệnh là một lệnh rẽ nhánh) và chuyển qua chu kỳ tiếp theo.

Việc xây dựng đầy đủ và chi tiết sẽ rất nặng nề. Vấn đề quan trọng là có thể dẫn tới mục đích mong muốn.

III. Máy Turing không đơn định

III.1. Khái niệm

Các máy Turing đã xét đều đơn định theo nghĩa : xuất phát từ một hình trạng đã cho, có nhiều nhất một hình trạng sẽ được chuyển tới trong một giai đoạn. Vấn đề là hãy xem xét các ngôn ngữ được thừa nhận bởi các máy Turing sẽ thay đổi ra sao nếu như ta loại bỏ điều kiện vừa nói này.

Định nghĩa 5.8 :

Một máy Turing không đơn định đồng nhất với một máy Turing đơn định trừ ra cách xây dựng hàm chuyển tiếp được cho theo quan hệ sau :

$D : (Q \times G) \times (Q \times G \times \{ L, R \})$

Nghĩa là với một trạng thái và một hình trạng đã cho (định nghĩa không chỉ một bộ *ba* $(Q \times G \times \{ L, R \})$, mà tới một tập hợp các bộ *ba*. Dễ thay ngay là khi tiến hành, máy có thể chọn bất kỳ một bộ *ba* nào.

Một cách hình thức, một hình trạng đã được chuyển đổi từ một hình trạng khác khi trạng thái đó là một trong các bộ *ba* (q', x, X) sao cho :

$((q, a), (q', x, X)) \in D$.

Các khái niệm khác cũng định nghĩa tương tự như máy Turing đơn định. Tuy nhiên, một máy Turing không đơn định sẽ không có một đoán nhận duy nhất và chỉ cần một trong các đoán nhận có chứa một hình trạng mà trong đó trạng thái cuối đạt được để máy thừa nhận.

III.2. Khử bỏ tính không đơn định

Kết quả thật đáng ngạc nhiên vì tính không đơn định không làm cho các máy Turing mạnh hơn lên. Ta có định lý sau :

Định lý 5. 1 :

Mọi ngôn ngữ được thừa nhận bởi một máy Turing không đơn định thì cũng được thừa nhận bởi một máy Turing đơn định.

Chứng minh :

Định lý 5. 1 chỉ xét cho các ngôn ngữ được thừa nhận vì rằng quan niệm về ngôn ngữ được xác định không có nghĩa cho các máy không đơn định.

Việc chứng minh định lý dựa trên ý tưởng đơn giản sau : chỉ cần máy Turing đơn định mô phỏng tất cả các đoán nhận của máy Turing không đơn định. Ở đây tất cả các đoán nhận này không thể mô phỏng đồng thời (điều này đã được xét với các ôtômát hữu hạn trong việc chứng minh định lý 2. 1).

Thật vậy, để mô phỏng một đoán nhận, ta cần xét băng vào. Nhớ rằng băng này khó có thể được sử dụng bởi nhiều đoán nhận cùng lúc.

Lời giải chấp nhận được là nên mô phỏng từng đoán nhận một. Sẽ có một số đoán nhận là vô hạn. Nếu không may mắn như vậy, ta sẽ bắt đầu mô phỏng một đoán nhận vô hạn, lúc đó sẽ không bao giờ có thể phỏng một đoán nhận khác lại là đoán nhận thừa nhận câu vào.

Việc mô phỏng tất cả các đoán nhận có thể cần các tiền tố có độ dài tăng dần. Trước hết, ta mô phỏng các đoán nhận có độ dài tiền tố là 1, rồi độ dài tiền tố là 2, là 3, ... Khi một trong các

đoán nhận đạt đến trạng thái cuối (thừa nhận), câu vào sẽ được thừa nhận. Ta hình thức hóa ý trên như sau :

Từ máy Turing không đơn định đã cho, sẽ có một số tối đa (maximum) chọn lựa có thể được lấy từ quan hệ chuyển tiếp D.

Số đó là :

$$r = \max_{q \in Q,\ a \in D} | \{ ((q, a), (q', x, X)) \in D \} |$$

Giả sử rằng với mỗi cặp (q, a), ta định số các lựa chọn cho phép bởi quan hệ chuyển tiếp từ 1 đến r (đến một số nhỏ hơn r nếu có ít hơn r lựa chọn).

Lúc này, để mô tả các lựa chọn thực hiện trong một tiền tố đoán nhận có độ dài m, cần đưa ra một dãy gồm m số nhỏ hơn r (nếu việc lựa chọn đã chỉ ra không tương ứng với một chuyển tiếp hiện hữu, việc đoán nhận sẽ dừng lại mà không thừa nhận).

Như vậy, ta có thể xây dựng một máy Turing đơn định có *ba* băng để mô phỏng một máy Turing không đơn định sau :

1. Băng thứ nhất chứa câu vào không bị thay đổi (cho phép tìm lại câu vào khi bắt đầu một đoán nhận).
2. Băng thứ hai dùng để chứa các dãy số nhỏ hơn r.
3. Băng thứ *ba* dùng cho máy đơn định để mô phỏng chức năng của máy không đơn định.

Máy đơn định sẽ hình thành như sau :

1. Trên băng thứ hai, máy sản sinh ra tất cả các dãy hữu hạn số nhỏ hơn r. Các dãy này được sinh bởi thứ tự tăng dần của độ dài.
2. Với mỗi dãy, máy mô phỏng máy không đơn định bằng cách sử dụng các lựa chọn biểu diễn bởi dãy số trên.
3. Máy dừng và thừa nhận ngay khi sự mô phỏng của một đoán nhận của máy không đơn định đạt đến trạng thái thừa nhận.

III.3. Các máy Turing vạn năng

Một vấn đề thú vị là liệu có thể có một máy Turing mô phỏng được bất kỳ máy Turing nào ?

Một cách tường minh, ta muốn cung cấp cho một máy Turing M sự mô tả của một máy Turing M' bất kỳ nào đó sao cho với một câu vào *w* nào đó, máy Turing M' có thể mô phỏng sự đoán nhận của M trên *w*.

Một máy Turing như vậy sẽ là một sự nhại lại các máy Turing khác, và được gọi là máy Turing vạn năng (Universal Turing Machine).

IV. Máy Turing và văn phạm ngữ cảnh

IV.1. Định nghĩa

Văn phạm ngữ cảnh, hay còn được gọi là văn phạm ngữ cấu (văn phạm loại 0) là bộ 4 G = (S, D, P, S), trong đó :

- D là tập hữu hạn các biến hay các ký hiệu không kết thúc, còn được ký hiệu bởi N (Non-terminal symbols).
- S là tập hữu hạn các ký hiệu cuối, $D \cap S = \varnothing$.

– $S \in D$ là ký hiệu đầu.

– P là tập hợp các sản xuất có dạng $a \rightarrow b$, với $a, b \in (D \cup S)^*$. $a \neq e$ và a có thể chứa biến, ngoài ra không có hạn chế gì.

Từ P có thể xây dựng hệ thống viết lại (rewriting) và phép suy dẫn ra các dạng câu theo quan hệ $\Rightarrow$ và $\overset{*}{\Rightarrow}$ như đã xét.

Ngôn ngữ được sản sinh bởi văn phòng ngữ cảnh G là :

$L(G) = \{ w \mid w \in S^* \text{ và } S \overset{*}{\Rightarrow} w \}$

Ví dụ 5.4 :

Cho văn phạm ngữ cảnh sản sinh ngôn ngữ $L = \{a^i \mid i \text{ là lũy thừa dương của } 2\}$ như sau :

1. $S \rightarrow ACaB$
2. $Ca \rightarrow aaC$
3. $CB \rightarrow DB$
4. $CB \rightarrow E$
5. $aD \rightarrow Da$
6. $AD \rightarrow AC$
7. $aE \rightarrow Ea$
8. $AE \rightarrow e$

Trong văn phạm này, A và B có vai trò đánh dấu mút trái và mút phải của một dạng câu.

Biến C di chuyển từ trái qua phải vượt qua các ký tự a nằm giữa A & B, và gấp đôi số a đó lên theo sản xuất 2.

Khi C gặp nút phải B, biến C trở thành D theo sản xuất 3, hay C thành E, theo sản xuất 4.

Nếu D được chọn, thì D lộn về trái bởi 5, đến khi gặp A lại trở thành C, theo sản xuất 6. Từ đó lại tiếp tục chu trình đối với C.

Còn nếu E được chọn thì bởi sản xuất 4, biến B biến mất, tiếp đó, E lộn qua trái theo sản xuất 7, cho đến khi gặp mút trái A thì xóa A và biến mất theo 8.

Như vậy chỉ còn lại câu gồm 2^i ký tự a, $i > 0$.

Có thể chứng minh bằng quy nạp theo số bước trong suy dẫn rằng nếu sản xuất 4 chưa dùng đến thì mọi dạng câu trong suy dẫn ở một trong 3 dạng sau :

1.S

2. $A\,a^iCa^jB$, trong đó $i + 2j$ là lũy thừa dương của 2.

3.$A\,a^iDa^jB$, trong đó $i + j$ là lũy thừa dương của 2.

Khi áp dụng sản xuất 4 thì sẽ có dạng câu Aa^iE, trong đó i là lũy thừa dương của 2.

Tiếp đó chỉ có thể áp dụng i lần sản xuất 7 để đi tới dạng câu Aea^i.

Cuối cùng, với sản xuất 8 ta có dạng câu a^i với i là lũy thừa dương của 2.

IV.2. Sự tương đương giữa văn phạm ngữ cảnh và máy Turing

Định lý 5. 2.

Nếu $L = L(G)$, $G = (S, D, P, S)$ là văn phạm ngữ cảnh, thì $L = L(M)$ với M là một máy Turing nào đó.

Chứng minh :

Xây dựng một máy Turing M không đơn định hai băng thừa nhận L,

Băng 1 của M chứa câu vào w. Trên băng 2, M sản sinh các dạng câu a của G.

Lúc đầu a là S. Sau đó M lặp lại quá trình sau đây :

Chọn một cách không đơn định một vị trí i trên a ($1 \le i \le |a|$). Nghĩa là bắt đầu từ bên trái a, chọn một trong hai khả năng : hoặc chọn i là vị trí hiện hành, hoặc tiến qua phải rồi lặp lại quá trình.

Chọn một cách không đơn định một sản xuất $b \to g$ của G.

Nếu b xuất hiện trên a kể từ vị trí i, thay thế b bởi g. Nếu $|b| \ne |g|$ thì phải chuyển dời phần cuối của a để đủ chỗ cho sự thay thế.

1. So sánh dạng câu thu được trên băng 2 với w ở băng 1. Nếu giống nhau thì thừa nhận w, nếu không quay lên bước 1.

Có thể chứng minh được rằng tất cả chỉ và chỉ những câu của G xuất hiện trên băng 2 ở bước 4.

Vậy : $L(M) = L(G) = L$ (qed).

Định lý 5.3 :

Nếu $L = L(M)$ với một máy Turing nào đó thì $L = L(G)$ với một văn phạm ngữ cảnh G nào đó.

Chứng minh :

Giả sử L được thừa nhận bởi $M = (S, Q, T, G, d, q_0, B, A)$. Lập văn phạm ngữ cảnh G mà mỗi quá trình suy dẫn của M (một cách phi hình thức) diễn ra qua 3 giai đoạn như sau :

G sản sinh một cách ngẫu nhiên (không đơn định) một câu $w \in S^*$, sau đó w được sao ra thành 2 bản, một bản được giữ nguyên cho đến cuối, còn một bản kia sẽ biến đổi theo cách làm việc của M.

1. G mô phỏng quá trình làm việc của M trên câu w, bằng cách diễn lại đúng quá trình làm việc của hệ viết lại ngầm định của M.
2. Chỉ sau khi giai đoạn 2 kết thúc với sự xuất hiện một trạng thái $q \in A$ của M (tức câu w đã được M thừa nhận), lúc đó G bắt đầu "thu dọn" để chuyển dạng câu đã có trở về câu w, và như vậy w đã được sản sinh.

Một cách hình thức, $G = (S, D, P, S1)$, trong đó :

$D = ((S \cup \{ e \}) \times G) \cup \{ S_1, S_2, \# \}$

Các sản xuất trong P như sau :

1.1. $S_1 \to \#q_0 S_2\#$

1.2. $S_2 \to [a, a]S_2$ với $\forall a \in S$

1.3. $S_2 \to e$

Nếu $d(q, x) = (p, Y, R)$ với $p, q \in Q$ và $X, Y \in G$ thì thêm các sản xuất sau vào P :

2.1. $q[a, X][b, Z] \to [a, Y]p[b, Z]$ với $\forall a, b \in S \cup \{ e \}$ và $\forall Z \in G$.

$q[a, X]\# \to [a, Y]p[e, B]\#$ với $\forall a \in S \cup \{ e \}$

Nếu $d(q, X) = (p, Y, L)$ với $p, q \in Q$ và $X, Y \in G$, thì thêm các sản xuất sau vào P :

2.3. $[b, Z]q[a, X] \to q[b, Z][a, Y]$ với $\forall a, b \in S \cup \{ e \}$ và $\forall Z \in G$.

Nếu $q \in A$ thì với $\forall a \in S \cup \{ e \}$ và $\forall X \in G$, thêm các sản xuất sau đây vào P :

3.1. $[a, X]q \to qaq$

3.2. $q[a, X] \to qaq$

3.3. $q\# \to e$

3. 4. $\#q \to e$

3. 5. $q \to e$

Ap dụng các sản xuất 1.1, 1.2 và 1.3, ta có :

$S1 \overset{*}{\underset{G}{\Rightarrow}} \#q_0[a_1, a_1][a_2, a_2] \ldots [a_n, a_n]\#$

dạng câu trên đại diện cho một hình trạng đầu của M là $\#q_0a_1a_2 \ldots a_n\#$.

Từ đó, áp dụng các sản xuất 2.1, 2.2 và 2.3 là sự nhại lại các chuyển tiếp của M. Sự suy dẫn trong G tức là quá trình làm việc của M. Nếu M dẫn đến trạng thái cuối $q \in A$, tương ứng với câu $a_1a_2 \ldots a_n$ được thừa nhận, thì các sản xuất từ 3.1 đến 3.5 được áp dụng để nhận được câu $a_1a_2 \ldots a_n$.

Vậy : $S_1 \overset{*}{\underset{G}{\Rightarrow}} a_1a_2 \ldots a_n$

Cuối cùng chỉ cần chứng minh $L(M) \subseteq L(G)$ và $L(G) \subseteq L(M)$.

Từ đó dẫn đến $L(M) = L(G)$ (qed).

V. Ôtômat tuyến tính giới nội và văn phạm cảm ngữ cảnh

Ta xét một loại ôtômat không mạnh bằng máy Turing và văn phạm tương ứng với nó.

V.1. Ôtômat tuyến tính giới nội

Ôtômat tuyến tính giới nội (LBA - Linear Bounded Automation) là máy Turing không đơn định và không có khả năng nới rộng vùng làm việc ra khỏi mút trái và mút phải của câu vào.

LBA không dùng tới các ký tự trắng trên băng về cả hai phía, phía trái và phía phải, của câu vào, vì vậy LBA không cần dùng ký hiệu B.

Để nhận biết các giới hạn trái và phải của câu vào, LBA có hai ký hiệu đặc biệt là $\#_L$ và $\#_R$ để đánh dấu hai nút *L* và R.

Lúc bắt đầu, trên băng có dạng $\#_L w\#_R$, trong đó $w \in (S \cup \{\#_L, \#_R\})^*$ là câu cần đoán nhận.

Khi hoạt động, nếu đầu đọc của LBA tới ô chứa ký tự $\#_L$ hay $\#_R$ thì bước tiếp theo chỉ có thể là đổi trạng thái, chuyển đầu đọc trở vào phía trong (đầu đọc qua phải nếu gặp $\#_L$, qua trái nếu gặp $\#_R$), mà không làm thay đổi các ô $\#_L$ và $\#_R$.

Một cách hình thức, LBA là bộ 8 :

$M = (S, Q, G, d, q_0, \#_L, \#_R, A)$

trong đó Q, S, G, d, q_0, A, giống như máy Turing, còn $\#_L$, $\#_R$ được thêm vào.

Hàm d :

$d : Q \times G \to \wp(Q \times G \times \{L, R\})$

thỏa mãn điều kiện :

Nếu $(p, Y, E) \in d(q, \#_L)$ thì $Y = \#_L$ và $E = R$.

Nếu $(p, Y, E) \in d(q, \#_R)$ thì $Y = \#_R$ và $E = L$.

trong đó $p, q \in Q$.

Hệ viết lại, hay văn phạm, ngầm định của LBA có bộ chữ $V = Q \cup G \cup \{\#_L, \#_R\}$ và tập P các sản xuất có dạng :

$\forall$ p, $q \in$ Q, X, Y $\in$ G - $\{\#_L, \#_R\}$ và $\forall$ Z $\in$ G :

$qX \to Yp$ tương ứng với $(p, Y, R) \in d(q, X)$.

$ZqX \to pZY$ tương ứng với $(p, Y, L) \in d(q, X)$.

$q\#_L \to \#_L p$ tương ứng với $(p, \#_L, R) \in d(q, \#_L)$.

$Zq\#_R \to pZ\#_R$ tương ứng với $(p, \#_R, L) \in d(q, \#_R)$.

Với các quan hệ suy dẫn $\Rightarrow$ và $\overset{*}{\Rightarrow}$, ta định nghĩa ngôn ngữ thừa nhận bởi LBA M là :

$L(M) = \{w \mid w \in (S - \{\#_L, \#_R\})^*$ và

$\#Lq_0w\#_R \overset{*}{\Rightarrow} g_1pg_2$ với p $\in$ A và $g_1, g_2 \in G^*$ }

Chú ý : Theo các dạng sản xuất từ 1 đến 4 thì một sản xuất $a \to b$ trong hệ viết lại ngầm định của LBA luôn luôn có $|a| = |b|$. Trái lại, máy Turing có thể có các sản xuất dạng qX# $\to$ YpB# (nới rộng vùng làm việc qua phải) không thỏa mãn yêu cầu này.

V.2. Văn phạm cảm ngữ cảnh

Một văn phạm cảm ngữ cảnh (CNC) là một hệ thống G = (S, D, P, S), trong đó :

S là tập hợp hữu hạn các ký tự cuối (kết thúc).

D là tập hữu hạn các biến (không kết thúc).

- S là ký hiệu đầu.

P là tập hợp hữu hạn các sản xuất dạng $a \to b$ trong đó $a, b \in (D \cup S)^*$, a chứa biến và $|a| \leq |b|$.

Với các quan hệ suy dẫn $\Rightarrow$ và $\overset{*}{\Rightarrow}$, ta định nghĩa ngôn ngữ là do G sinh ra là :

$L(G) = \{w \mid w \in S^*$ và $S \overset{*}{\Rightarrow} w\}$

$L(G)$ được gọi là ngôn ngữ *cảm ngữ cảnh* (văn phạm loại 1). Thuật ngữ cảm ngữ cảnh bắt nguồn từ dạng chuẩn của một sản xuất của văn phạm tương ứng là :

$a_1Aa_2 \to a_1ba_2$

với $b \neq e$, cho thấy một biến A được thay thế bởi dạng câu b trong *ngữ cảnh* là a_1 và a_2.

Trái lại, trong các văn phạm phi ngữ cảnh, các sản xuất có dạng :

$A \to b$, với $(|b| \leq 0)$

thì biến A được thay thế không cần ngữ cảnh (thay thế A bởi b ở mọi nơi của dạng câu có xuất hiện A).

Ví dụ 5.5 :

Xét văn phạm CNC G = ({*a*, *b*, c}, {S, B, C}, P, S), trong đó P gồm :

1. S $\to$ aSBC
2. S $\to$ aBC
3. CB $\to$ BC
4. aB $\to$ *ab*
5. bB $\to$ bb
6. bC $\to$ bb
7. cC $\to$ cc

Dễ thấy rằng $L(G) = \{ a^nb^nc^n \mid n \geq 1 \}$.

Thật vậy, với các sản xuất 1 và 2 ta có $S \overset{*}{\Rightarrow} a^n(BC)^n$.

Với sản xuất 3, mọi B chạy lên trước mọi C : $a^n(BC) \overset{*}{\Rightarrow} a^nB^nC^n$. Các sản xuất 4 và 5 biến mọi B thành *b*, cuối cùng, các sản xuất 6 và 7 biến mọi C thành c.

Tóm lại :

$S \overset{*}{\Rightarrow} a^n(BC)^n \overset{*}{\Rightarrow} a^nB^nC^n \overset{*}{\Rightarrow} a^nb^nC^n \overset{*}{\Rightarrow} a^nb^nc^n$.

V.3. Sự tương đương giữa LBA và văn phạm CNC

Chú ý rằng LBA có thể thừa nhận câu rỗng e, còn văn phạm CNC thì không thể sinh ra câu rỗng e. Như vậy, trừ trường hợp e, ta sẽ thấy rằng LBA thừa nhận đúng các ngôn ngữ CNC.

Định lý 5. 4 :

Nếu *L* là ngôn ngữ CNC, thì *L* được thừa nhận bởi một LBA nào đó, $L = L$ (M).

Chứng minh :

Cách chứng minh tương tự cách chứng minh định lý 5.3, chỉ khác là M không cần băng thứ hai để sản sinh các dạng câu, bắt chước các suy dẫn của văn phạm, mà chỉ cần dùng băng thứ hai.

Cho G = (S, D, P, S) là văn phạm CNC. Ta xây dựng máy Turing M gồm 2 băng như sau :

- Băng 1 chứa câu vào *w* với các dấu mút trái $\#_L$ và mút phải $\#_R$ ở hai đầu băng.

- Băng 2 để sản sinh các dạng câu *a*.

Bắt đầu, nếu *w* = e thì M ngừng và không thừa nhận *w*. Nếu không, viết S ở băng 2, ngay dưới ký hiệu trái nhất của *a*.

Máy Turing M thực hiện như sau :

Chọn một cách không đơn định một câu con *b* của *a* trên băng 2 sao cho $a \rightarrow b$ là một sản xuất trong P.

Thay *b* bởi g và sau đó chuyển tiếp phần đuôi của *a* sao cho đủ chỗ. Tuy nhiên, nếu chuyển tiếp vượt qúa $\#_R$ thì ngừng và không thừa nhận.

Khi tới bước 3 thì băng 1 là $\#_L w\#_R$, còn ở băng 2 là *a*, mà :

$S \overset{*}{\underset{G}{\Rightarrow}} a$ và $|a| \le |w|$.

So sánh băng 1 và băng 2. Nếu $a = w$: ngừng và thừa nhận *w*. Nếu không, quay lại bước 1. Như vậy, khi M thừa nhận *w* thì $S \overset{*}{\underset{G}{\Rightarrow}} w$.

Ngược lại, nếu $S \overset{*}{\underset{G}{\Rightarrow}} w$ thì mọi dạng câu *a* xuất hiện trong suy dẫn đó đều thỏa mãn điều kiện $|a| \le |w|$. Bởi vì mọi sản xuất $b \rightarrow g$ trong G đều có $|b| \le |g|$.

Như vậy, M có thể thực hiện các suy dẫn trong băng 2, giữa hai mút $\#_L$ và $\#_R$, và thừa nhận câu vào *w*, hay những câu sinh ra bởi văn phạm G ($\forall\, w \in L$(G)) **qed**.

Định lý 5. 5 :

Nếu $L = L$ (M) với M là LBA thì $L - \{e\}$ là ngôn ngữ CNC.

Chứng minh :

Tương tự cách chứng minh định lý 5.4. Ta xây dựng văn phạm G làm việc qua 3 giai đoạn. Giả sử T = (S, Q, G, d, q_0, $\#_L$, $\#_R$, A) ta có :

Giai đoạn 1 Văn phạm sinh ra w là câu vào của M gồm hai bản với $\#_L$, $\#_R$ và q_0.

Giai đoạn 2 Văn phạm lặp lại các thao tác của hệ viết lại ngầm định của M.

Giai đoạn 3 Khi xuất hiện $q \in A$, thu về w. Lưu ý rằng trong hệ viết lại của T, các sản xuất $a \to b$ đều có $|a| \le |b|$, nên việc mô phỏng lại chúng là có thể được.

Để xóa đi các ký hiệu $\#_L$, $\#_R$ và q trong giai đoạn 3 mà không làm co ngắn câu, ta gắn chúng kề bên các ký tự của w mà không đứng rời ra như trước. Cụ thể, từ giai đoạn 1 có thể có sản xuất :

$$S_1 \to [a, \#_L q_0 a] S_2, \qquad S_1 \to [a, \#_L q_0 a \#_R]$$
$$S_2 \to [a, a] S_2, \qquad S2 \to [a, a\#_R]$$

với $\forall\, a \in S - \{\#_L, \#_R\}$.

Các sản xuất trong G ở giai đoạn 2 bắt chước y hệt hệ viết lại ngầm định của M.

Cuối cùng, ở giai đoạn 3, các sản xuất sau được sử dụng :

$$[a, a]b \to ab, \qquad b[a, a] \to ba$$

với $q \in A$, $\forall\, a \in S - \{\#_L, \#_R\}$ và $\forall\, a, b$.

Vậy, văn phạm được xác định là văn phạm CNC.

Bài tập chương 5

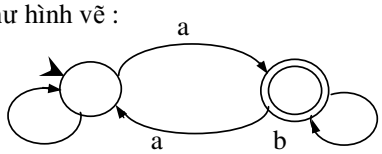
1. Xây dựng các máy Turing thừa nhận các ngôn ngữ sau đây :
   a) $\{ 0^n 1^n 0^n \mid n \ge 1 \}$
   b) $\{ ww^R \mid w \in (0+1)^* \}$
   c) Tập các câu thuộc $\{ 0, 1 \}^*$ có số các con số 0 đúng bằng số các con số 1.
2. Xây dựng các máy Turing tính các hàm sau đây :
   a) n^2 *b*) $n!$
3. Xây dựng các văn phạm ngữ cảnh cho các ngôn ngữ sau đây :
   a) $\{ ww \mid w \in (0+1)^* \}$ *b*) $\{ 0^k \mid k = i^2, i \ge 1 \}$

PHỤ LỤC

Một số đề thi

Đề số 1

Câu 1 : Cho ôtômat M như hình vẽ :



1. Xây dựng biểu thức chính quy r sao cho L(r) = L(M).
2. Xây dựng văn phạm chính quy G sao cho L(G) = L(M).
3. Xây dựng ôtômat M' đơn định tương đương với M, nghĩa là L(M') = L(M).
4. Xây dựng ôtômat M' thừa nhận ngôn ngữ bù của L(M), nghĩa là L(M') = S^* - L(M).
5. Xây dựng ôtômat M" sao cho L(M") = $(L(M))^R$
   nghĩa là L(M") là ngôn ngữ nghịch đảo của L(M).

Câu 2 : Cho văn phạm G gồm các sản xuất S → aSb | aSbT | e và T → Tc | c.

Vẽ cây suy dẫn cho câu $w \in L$(G) bất kỳ sao cho $|w| \geq 9$.

Đề số 2

Câu 1 : Cho biểu thức chính quy : r = $(b \cup b^*a)ba^*b$.

1. Xây dựng ôtômat M sao cho L(M) = L(r).
2. Xây dựng văn phạm chính quy G sao cho L(G) = L(M).
3. Xây dựng ôtômat M' đơn định tương đương với M, nghĩa là L(M') = L(M).
4. Xây dựng ôtômat M' thừa nhận ngôn ngữ bù của L(M), nghĩa là L(M') = S^* - L(M).
5. Xây dựng ôtômat M" thừa nhận ngôn ngữ nghịch đảo của L(M),
   nghĩa là L(M") = $(L(M))^R$.

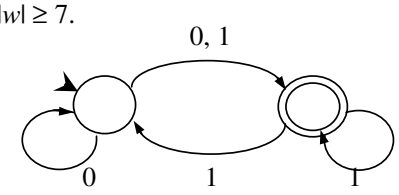
Câu 2 : Cho văn phạm G gồm các sản xuất :

S → S+T | T T → T*E | E E → (S) | c.

Vẽ cây suy dẫn cho câu $w \in L$(G) bất kỳ sao cho $|w| \geq 7$.

Đề số 3

Câu 1 : Cho ôtômat M như hình vẽ :



1. Xây dựng biểu thức chính quy r sao cho L(r) = L(M).
2. Xây dựng văn phạm G sao cho L(G) = L(M).
3. Xây dựng ôtômat M' đơn định tương đương với M, nghĩa là L(M') = L(M).
4. Xây dựng ôtômat M" sao cho L(M") = $(L(M))^R$
   nghĩa là L(M") là ngôn ngữ nghịch đảo của L(M).

Câu 2 : Cho văn phạm G gồm các sản xuất S → aSb | aSbT | e và T → Tc | c.
Vẽ cây suy dẫn cho câu $w \in L$(G) bất kỳ sao cho $|w| \geq 9$.

Đề số 4

Câu 1 : Cho biểu thức chính quy : r = baa*($ba \cup ab^*$).

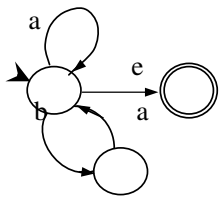
1. Xây dựng ôtômat M sao cho L(M) = L(r).
2. Xây dựng văn phạm chính quy G sao cho L(G) = L(M).
3. Xây dựng ôtômat M' thừa nhận ngôn ngữ bù của L(M), nghĩa là L(M') = S^* - L(M).
4. Xây dựng ôtômat M" thừa nhận ngôn ngữ nghịch đảo của L(M),
   nghĩa là L(M") = $(L(M))^R$.

Câu 2 : Cho văn phạm G gồm các sản xuất :

S → S+T | T T → T*E | E E → (S) | c.

Đề số 5

Câu 1 : Cho ôtômat M như hình vẽ :



1. Xây dựng biểu thức chính quy r sao cho L(r) = L(M).
2. Xây dựng ôtômat M' đơn định tương đương với M, nghĩa là L(M') = L(M).
3. Xây dựng ôtômat M" thừa nhận ngôn ngữ nghịch đảo của L(M),
   nghĩa là L(M") = $(L(M))^R$.

Xây dựng văn phạm G sao cho L(G) = L(M).

Tài liệu tham khảo

[1] A.V.Aho, J.D.Ulman, *The Theory of Parsing, Translation and Compiling*, Vol 1 : Parsing, Pretice Hall, 1972.

[2] J.-M.Autebert. *Calculabilité et décidabilité*. Masson Paris, 1992.

[3] J.-M.Autebert. *Théorie des langages et des automats*. Masson Paris, 1994.

[4] Nguyễn Văn Ba. *Ngôn ngữ hình thức*. Khoa Công nghệ Thông tin, trường Đại học Bách khoa Hà nội, 1993.

[5] Pierre BERLIOUX & M. LEVY. *Théorie des langages*. Notes de cours, ENSIMAG 1991.

[6] PierreBERLIOUX. *Calculabilité*. Notes de cours, ENSIMAG 1991.

[7] SM.D.Davis, E.J.Weyuker, *Computability, Complexity and languages*, Academic Press, 1983.

[8] J.E.Hopcroft, J.D.Ulman, *Formal Languages and Their Relation to Automata.* Addison - Wesley, 1969.

[9] J.E.Hopcroft, J.D.Ulman, *Introduction to Automata Theory, Languages and Computation*, Addison - Wesley, 1979.

[10] C.PAYAN, J.SIFAKIS. *Conception des systèmes logiques*. Notes de cours, ENSIMAG 1974.

[11] A.SALOMAA. *Nhập môn Tin học lý thuyết tính toán và các Ôtômat*. Nguyễn Xuân My và Phạm Trà Ân dịch. Nhà Xuất bản Khoa học và Kỹ thuật, Hà nội 1992.

[12] Nguyễn Thanh Tùng. *Lý thuyết ngôn ngữ hình thức và ôtômat*. Khoa Công nghệ Thông tin, trường Đại học Kỹ thuật – Đại học Quốc gia, thành phố HCM, 1993.

[13] H.S.Wilf, *Algorithmes et complexité*. Pretice Hall, 1989.
P.WOLPER. *Introduction à la calculabilité*. InterEditions, Paris 1991.