BỘ GIAO THÔNG VẬN TẢI
TRƯỜNG ĐẠI HỌC HÀNG HẢI
BỘ MÔN: KHOA HỌC MÁY TÍNH
KHOA: CÔNG NGHỆ THÔNG TIN

BÀI GIẢNG

NGÔN NGỮ HÌNH THỨC VÀ ÔTÔMAT

TÊN HỌC PHẦN : **Ngôn ngữ hình thức và Ôtômat**
MÃ HỌC PHẦN : 17204
TRÌNH ĐỘ ĐÀO TẠO : **ĐẠI HỌC CHÍNH QUY**
DÙNG CHO SV NGÀNH : **CÔNG NGHỆ THÔNG TIN**

HẢI PHÒNG - 2008

ĐỀ CƯƠNG CHI TIẾT

Tên học phần: Ngôn ngữ hình thức và Ôtômat **Loại học phần:** 1
Bộ môn phụ trách giảng dạy: Khoa học Máy tính **Khoa phụ trách:** CNTT
Mã học phần: 17204 **Tổng số TC**: 2

TS tiết	Lý thuyết	Thực hành/Xemina	Tự học	Bài tập lớn	Đồ án môn học
45	45	0	0	0	0

Điều kiện tiên quyết:

Sinh viên phải học xong môn học toán rời rạc.

Mục tiêu của học phần:

- Cung cấp các kiến thức cơ bản về ngôn ngữ, văn phạm và otomat.
- Cơ bản về chương trình dịch.

Nội dung chủ yếu

Gồm các phần:

- Văn phạm và ngôn ngữ
- Ngôn ngữ chính quy và otomat đẩy xuống
- Ngôn ngữ phi ngữ cảnh và otomat đẩy xuống
- Cơ bản về chương trình dịch

Nội dung chi tiết của học phần:

TÊN CHƯƠNG MỤC	PHÂN PHỐI SỐ TIẾT				
	TS	**LT**	**TH/Xemina**	**BT**	**KT**
MỞ ĐẦU					
Chương I. Văn phạm và ngôn ngữ.	**05**	**04**	**01**		
1.1. Bảng chữ cái, từ và ngôn ngữ 1.2. Tích ghép, phép chia, phép soi gương 1.3. Các phép toán trên ngôn ngữ 1.4. Văn phạm 1.5. Các ví dụ về văn phạm					
Chương II. Ngôn ngữ chính quy và otomat hữu hạn	**16**	**12**	**03**		**01**
2.1. Nguồn và ngôn ngữ được sinh bởi nguồn 2.2. Các phép toán trên nguồn 2.3. Otomat hữu hạn không lối ra và ngôn ngữ được đoán nhận bởi otomat hữu hạn không lối ra 2.4. Sự tương đương của nguồn và Otomat hữu hạn không lối ra 2.5. Sự tương đương của nguồn và văn phạm chính quy 2.6. Sự tương đương của nguồn và biểu thức chính quy 2.7. Bài tập tổng hợp 2.8. Tính đóng của lớp ngôn ngữ chính quy 2.9. Điều kiện cần của ngôn ngữ chính quy 2.10. Điều kiện cần và đủ của lớp ngôn ngữ chính quy 2.11. Otomat hữu hạn có lối ra 2.12. Ngôn ngữ chính quy					

TÊN CHƯƠNG MỤC	PHÂN PHỐI SỐ TIẾT				
	TS	**LT**	**TH/Xemina**	**BT**	**KT**
Chương III. Ngôn ngữ phi ngữ cảnh và otomat đẩy xuống.	**09**	**06**	02		**01**
3.1. Cây và phép thế cây 3.2. Dạng chuẩn Chomsky 3.3. Cây dẫn xuất và các tính chất 3.8. Khái niệm về Otomat đẩy xuống					
Chương IV: Cơ bản về chương trình dịch	**15**	**12**	02		01
4.1.Giới thiệu chương trình dịch 4.2.Chương trình dịch 4.2.1. Định nghĩa chương trình dịch 4.2.2. Cấu trúc của chương trình dịch 4.2.3. Cấu trúc tĩnh (cấu trúc logic) 4.2.4. Cấu trúc động 4.2.5. Vị trí của chương trình dịch trong hệ thống dịch 4.3.Sự cần thiết nghiên cứu chương trình dịch 4.4. Bộ phân tích cú pháp					

Nhiệm vụ của sinh viên :

Tham dự các buổi thuyết trình của giáo viên, tự học, tự làm bài tập do giáo viên giao, tham dự các bài kiểm tra định kỳ và cuối kỳ.

Tài liệu học tập :

- Nguyễn Văn Ba, *Ngôn ngữ hình thức*, Trường Đại học Bách khoa Hà nội, 1997
- Phan Đình Diệu, *Lý thuyết otomat và thuật toán*, Nhà xuất bản Đại học và Trung học Chuyên nghiệp, 1971
- Đỗ Đức Giáo, Đặng Huy Ruận, *Ngôn ngữ hình thức*, Nhà xuất bản KHKT, 1991
- Phạm Hồng Nguyên, *Giáo trình chương trình dịch*, ĐH KHTN HN
- Nguyễn Văn Ba, *Phân tích cú pháp*, ĐH Bách khoa HN

Hình thức và tiêu chuẩn đánh giá sinh viên:

- Hình thức thi cuối kỳ : Thi viết.
- Sinh viên phải đảm bảo các điều kiện theo Quy chế của Nhà trường và của Bộ

Thang điểm: Thang điểm chữ A, B, C, D, F

Điểm đánh giá học phần: Z = 0,2X + 0,8Y.

Bài giảng này là tài liệu **chính thức và thống nhất** của Bộ môn Khoa học máy tính, Khoa Công nghệ thông tin và được dùng để giảng dạy cho sinh viên.

Ngày phê duyệt: / /2010

Trưởng Bộ môn: Thạc sỹ Nguyễn Hữu Tuân

MỤC LỤC

CHƯƠNG 1: VĂN PHẠM VÀ NGÔN NGỮ

Kiến thức cơ bản: Để tiếp thu tốt nội dung của chương này, sinh viên cần có một số các kiến thức liên quan về chuỗi, ký hiệu, từ trong các ngôn ngữ tự nhiên như tiếng Việt, tiếng Anh; cấu trúc cú pháp của các chương trình máy tính viết bằng một số ngôn ngữ lập trình cơ bản như Pascal, C…

1.1. Bảng chữ cái, từ, ngôn ngữ

Các ngôn ngữ lập trình (như Pascal, C, ...) lẫn ngôn ngữ tự nhiên (như tiếng Việt, tiếng Anh, ...) đều có thể xem như là tập hợp các câu theo một cấu trúc quy định nào đó. Câu của ngôn ngữ, trong tiếng Việt như "An là sinh viên giỏi" hay trong Pascal là một đoạn chương trình bắt đầu bằng từ khóa program cho đến dấu chấm câu kết thúc chương trình, đều là một chuỗi liên tiếp các từ, như “An”, “giỏi” hay “begin”, “if”, “x2”, “215”, tức các chuỗi hữu hạn các phần tử của một bộ chữ cái cơ sở nào đó. Ta có thể xem chúng như là các ký hiệu cơ bản của ngôn ngữ.

Từ nhận xét đó, ta dẫn tới một quan niệm hình thức về ngôn ngữ như sau (theo từ điển): Ngôn ngữ, một cách không chính xác là một hệ thống thích hợp cho việc biểu thị các ý nghĩ, các sự kiện hay các khái niệm, bao gồm một tập các ký hiệu và các quy tắc để vận dụng chúng.

Định nghĩa trên chỉ cung cấp một ý niệm trực quan về ngôn ngữ chứ không đủ là một định nghĩa chính xác để nghiên cứu về ngôn ngữ hình thức. Chúng ta bắt đầu xây dựng định nghĩa này bằng các khái niệm mà mọi ngôn ngữ đều đặt nền tảng trên đó.

1.1.1. Bảng chữ cái

Một dãy hữu hạn hoặc vô hạn các phần tử, kí hiệu Σ được gọi là một bảng chữ cái trong đó mỗi phần tử $a \in \Sigma$ được gọi là một kí hiệu (một chữ cái).

Ví dụ: $\Sigma = \{0,1\}$: Bảng chữ cái nhị phân

$\Sigma = \{0,1,\ldots,9\}$: Bảng chữ cái thập phân

1.1.2. Từ

- Một dãy kí hiệu $a_1, a_2, \ldots, a_n$ $(a_i, i=1 \div n)$ được gọi là một từ $\in \Sigma$.

Ví dụ : $\Sigma = \{0,1\}$

$a_1 = 0$, $a_2 = 1$, $a_3 = 00$, $a_4 = \wedge$,…

- Từ rỗng đượ kí hiệu là $\wedge$, mọi bảng chữ cái đều sinh ra từ rỗng.
- Tổng số các kí hiệu tạo nên từ được gọi là độ dài của từ.

Ví dụ : $\Sigma = \{a,b,c\}$

$a_1 = aabb$, $a_2 = ac$, $|a_1| = 4$, $|a_2| = 2$

- Độ dài của từ rỗng = 0.
- Nếu a là một từ thuộc bảng chữ cái Σ, Δ là một bảng chữ cái chứa bảng chữ cái Σ thì a ∈ Δ . Ví dụ : Σ = {0,1} , Δ = {0,1,2,…,9}

  $a_1 = 0110 \in \Sigma$, $\Sigma \in \Delta \Rightarrow a \in \Delta$
- Tập hợp tất cả các từ ∈ Σ kể cả từ rỗng kí hiệu là Σ^*.
- Tập hợp tất cả các từ ∈ Σ trừ rỗng kí hiệu là Σ^+. Vậy $\Sigma^+ = \Sigma^* \setminus \{\wedge\}$

1.1.3. Ngôn ngữ

Ngôn ngữ là một tập hợp các câu có một cấu trúc qui định nào đó. Một câu của ngôn ngữ là một dãy (hay xâu) các từ có sẵn được liệt kê trong một bảng chữ nào đó, như là các ký hiệu cơ bản của ngôn ngữ. Giả sử có bảng chữ cái Σ.

- Tập $A \in \Sigma^*$ được gọi là một ngôn ngữ sinh bởi Σ .

Ví dụ : Σ = {a,b,c}

A = {x ∈ Σ^*/x bắt đầu bởi kí hiệu a} được gọi là một ngôn ngữ ∈ Σ^*.

- Ngôn ngữ trống kí hiệu ϕ.
- Mọi bảng chữ cái Σ đều sinh ra ϕ.
- Ngôn ngữ trống và ngôn ngữ chứa từ rỗng là khác nhau.

1.2. Tích ghép, phép chia, phép soi gương

1.2.1. Tích ghép hai xâu

- Giả sử có bảng chữ cái Σ

Các từ $\alpha = a_1a_2a_3\ldots a_n$, $\beta = a_1a_2\ldots a_m$ ($a_i \in \Sigma$)

- Tích ghép của α và β được hình thành bằng cách ghép từ β vào sau từ α.

$\chi = \alpha.\beta = a_1a_2a_3\ldots a_na_1a_2\ldots a_m$

Ví dụ : Σ={0,1}

α = 011, β =1101

χ = 0111101

- Tính chất của tích ghép

+ Tích ghép không có tính chất giao hoán

$\alpha\beta \neq \beta\alpha$

+ Tích ghép có tính chất kết hợp

$(\alpha\beta)\chi = \alpha(\beta\chi)$

+ Tích ghép của một xâu với một xâu rỗng bằng chính nó

$\alpha.\wedge = \wedge.\alpha = \alpha$

Với tích ghép: $\chi=\alpha.\beta$

α : Tiếp đầu ngữ

β : Tiếp vị ngữ

1.2.2. Chia xâu

Giả sử có $z = x.y$ ($x,y,z \in \Sigma^*$)

+ Phép chia trái

Nếu ta thực hiện ngắt bỏ xâu x ra khỏi z có nghĩa là ta đã thực hiện phép chia trái xâu z cho x và kết quả là xâu y.

Kí hiệu $x\backslash^Z = y$

+ Phép chia phải

Nếu ta thực hiện ngắt xâu y ra khỏi xâu z có nghĩa là ta đã thực hiện phép chia phải xâu z cho xâu y và kết quả là xâu x.

Kí hiệu $^Z/y = x$

Ví dụ : $\Sigma = \{0,1\}$, x = 10, y = 11

$Z = x.y = 1011$

$x\backslash^Z = 11$

$^Z/y = 10$

1.2.3. Phép soi gương

+ Giả sử có bảng chữ cái Σ

+ Từ $\alpha = a_1a_2\ldots a_{m-1}a_m \in \Sigma^*$

+ Từ $\tilde{\alpha} = a_ma_{m-1}\ldots a_2a_1$ được gọi là từ soi gương của từ α hay còn gọi là từ ngược của từ α

$|\tilde{\alpha}| = |\alpha|$

1.3. Các phép toán trên ngôn ngữ

1.3.1. Phép hợp hai ngôn ngữ

Giả sử có bảng chữ cái Σ, A, B, C là các ngôn ngữ được sinh ra bởi Σ

$C = A \cup B = \{x \in \Sigma^*/x \in A \text{ hoặc } x \in B\}$

Ví dụ $\Sigma = \{a,b,c\}$

$A = \{a, b, ab, ac, cb\}$, $B = \{aa, bb, cc\}$

$C = A \cup B = \{a, b, ab, ac, cb, aa, bb, cc\}$

+ Tính chất

- Phép hợp có tính chất giao hoán $A \cup B = B \cup A$
- Có tính chất kết hợp

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

1.3.2. Phép giao hai ngôn ngữ

+ Giả sử có bảng chữ cái Σ, A, B, C là các ngôn ngữ sinh ra bởi Σ

$C = A \cap B = \{x \in \Sigma^* \mid x \in A \text{ và } x \in B\}$

Ví dụ : $\Sigma = \{a, b\}$

$A = \{a, ab, ac, cb\}$, $B = \{b, ab, cb, aa, bc\}$

$C = A \cap B = \{ab, cb\}$

+ Tính chất

- Tính kết hợp

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

- Tính giao hoán

$A \cap B = B \cap A$

- $A \cap \phi = \phi \cap A = \phi$

1.3.3. Phép tích ghép hai ngôn ngữ

+ Giả sử có bảng chữ cái Σ, A, B, C là các ngôn ngữ được sinh bởi từ Σ

$C=A.B=\{z \in \Sigma^* \mid z=x.y \mid x \in A, y \in B\}$

Ví dụ : $\Sigma = \{a, b, c\}$

$A = \{a, b, ab, ac\}$, $B\{c, cb\}$

$C = A.B = \{ac, bc, abc, acc, acb, bcb, abcb. accb\}$

+ Tính chất

- Không có tính chất giao hoán

$A.B \neq B.A$

- Có tính chất kết hợp

$A.(B.C) = (A.B).C$

- Có tính chất phân bố (đối với phép hợp)

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

- Tính chất lũy thừa $A.A.A\ldots A = A^n$

1.3.4. Phép hiệu

Giả sử có bảng chữ cái Σ, A, B, C là các ngôn ngữ sinh bởi Σ

$$C = A\backslash B = \{x \in \Sigma^* \mid x \in A, x \in B\}$$

được gọi là ngôn ngữ hiệu của hai ngôn ngữ A và B.

Ví dụ : $\Sigma = \{a, b, c\}$

$A = \{a, b, ac, bc, aa\}$, $B = \{b, bc, ab, bb\}$

$C = B\backslash A = \{b, ab, bb\}$

1.3.5. Phép lấy phần bù

Giả sử có bảng chữ cái Σ, A, B là các ngôn ngữ được sinh ra bởi Σ

$$A = C_B = \{ x \in \Sigma^* / x \notin B\}$$

được gọi là ngôn ngữ phần bù hay ngôn ngữ bù của ngôn ngữ B.

Ví dụ : Cho $B = \{a, bc\}$.

Khi đó ngôn ngữ bù của ngôn ngữ B sẽ là :

$$A = \{x \in \Sigma^* \mid x \neq a, x \neq bc\}.$$

1.3.6. Phép chia hai ngôn ngữ

- Giả sử có bảng chữ cái Σ, A, B là các ngôn ngữ được sinh ra bởi Σ.
- Tập các từ $C = \{z \in \Sigma^* \mid x \in A, y \in B, x=y.z\}$ được gọi là phép chia trái của ngôn ngữ A cho ngôn ngữ B và kí hiệu $B\backslash^A$.
- Tập các từ $C = \{z \in \Sigma^* \mid x \in A, y \in B, x = z.y\}$ được gọi là phép chia phải của ngôn ngữ A cho ngôn ngữ B và kí hiệu là A/ B.

Ví dụ : $\Sigma = \{a, b, c\}$

$A = \{a, bc, abc, aba\}$, $B = \{\wedge, b, ab, cbc\}$

$B\backslash^A = \{ a, bc, abc, aba, c\}$

$A/_B = \{a, bc, abc, aba\}$

1.3.7. Phép soi gương ngôn ngữ

- Giả sử có bảng chữ cái Σ ,A, B là các ngôn ngữ được sinh bởi Σ.

➢ $B = \tilde{A} = \{\tilde{x} \in \Sigma^* / \tilde{x}$ là từ soi gương của $x \in A\}$.

Ví dụ $\Sigma = \{0, 1\}$

$A = \{0, 1, 10, 110\}$

$B = \tilde{A} = \{0, 1, 01, 011\}$

1.3.8. Phép lặp

- Giả sử có bảng chữ cái Σ, A là ngôn ngữ được sinh bởi Σ.
- $A^* = \{\wedge\} \cup A \cup A \cup \ldots \cup A^n$ gọi là ngôn ngữ lặp của ngôn ngữ A.
- Tính chất :
  - $(A^*)^* = A^*$
  - $\{\wedge\}^* = \{\wedge\}$
  - $(\varnothing)^* = \{\wedge\} \cup \varnothing \cup \varnothing.\varnothing \cup \ldots = \{\wedge\}$
  - $(\varnothing)^+ = \varnothing \cup \varnothing.\varnothing \cup \ldots = \varnothing$

1.4. Văn phạm

Với mục đích sản sinh (hay đoán nhận) ngôn ngữ, văn phạm được dùng như một cách thức hiệu quả để biểu diễn ngôn ngữ.

1.4.1. Định nghĩa văn phạm cấu trúc (Grammar)

Theo từ điển, văn phạm, một cách không chính xác, là một tập các quy tắc về cấu tạo từ và các quy tắc về cách thức liên kết từ lại thành câu.

Để hiểu rõ hơn khái niệm này, ta xét ví dụ cây minh họa cấu trúc cú pháp của một câu đơn trong ngôn ngữ tiếng Việt "An là sinh viên giỏi" ở thí dụ 1.5 của chương 1. Xuất phát từ nút gốc theo dần đến nút lá, ta nhận thấy các từ ở những nút lá của cây như “An”, “sinh viên”, “giỏi”, … là những từ tạo thành câu được sản sinh. Ta gọi đó là các ký hiệu kết thúc bởi vì chúng không còn phát sinh thêm nút nào trên cây và câu được hoàn thành. Trái lại, các nút trong của cây như “câu đơn”, “chủ ngữ”, “danh từ”, … sẽ không có mặt trong dạng câu sản sinh, chúng chỉ giữ vai trò trung gian trong việc sinh chuỗi, dùng diễn tả cấu trúc câu. Ta gọi đó là các ký hiệu chưa kết thúc. Quá trình sản sinh câu như trên thực chất là sự diễn tả thông qua cấu trúc cây cho một quá trình phát sinh chuỗi. Các chuỗi được phát sinh bắt đầu từ một ký hiệu chưa kết thúc đặc biệt, sau mỗi bước thay thế một ký hiệu chưa kết thúc nào đó trong chuỗi thành một chuỗi lẫn lộn gồm các ký hiệu kết thúc và chưa, cho đến khi không còn một ký hiệu chưa kết thúc nào nữa thì hoàn thành. Quá trình này chính là phương thức phát sinh chuỗi của một văn phạm, được định nghĩa hình thức như sau:

Định nghĩa : Văn phạm cấu trúc G là một hệ thống gồm bốn thành phần xác định như sau **G (Σ,V, S, P)**, trong đó:

- **Σ: t**ập hợp các ký hiệu kết thúc (**t**erminal)
- **V** : tập hợp các biến (**v**ariables) hay các ký hiệu chưa kết thúc (non terminal)
- **P**: tập hữu hạn các quy tắc ngữ pháp được gọi là các luật sinh (**p**roduction.
- **S**: ký hiệu chưa kết thúc dùng làm ký hiệu bắt đầu (**s**tart)

Người ta thường dùng các chữ cái Latinh viết hoa (A, B, C, ...) để chỉ các ký hiệu trong tập biến V; các chữ cái Latinh đầu bảng viết thường (a, b, c, ...) dùng chỉ các ký hiệu kết thúc thuộc tập Σ.

Nhận xét : Bằng quy ước này chúng ta có thể suy ra các biến, các ký hiệu kết thúc và ký hiệu bắt đầu của văn phạm một cách xác định và duy nhất bằng cách xem xét các luật sinh. Vì vậy, để biểu diễn văn phạm, một cách đơn giản người ta chỉ cần liệt kê tập luật sinh của chúng.

1.4.2. Sự phân cấp Chomsky trên văn phạm

Bằng cách áp đặt một số quy tắc hạn chế trên các luật sinh, Noam Chomsky đề nghị một hệ thống phân loại các văn phạm dựa vào tính chất của các luật sinh. Hệ thống này cho phép xây dựng các bộ nhận dạng hiệu quả và tương thích với từng lớp văn phạm. Ta có 4 lớp văn phạm như sau :

1) Văn phạm loại 0: Một văn phạm không cần thỏa ràng buộc nào trên tập các luật sinh được gọi là văn phạm loại 0 hay còn được gọi là ***văn phạm không hạn chế*** (Unrestricted Grammar)

2) Văn phạm loại 1: Nếu văn phạm G là văn phạm các phép thế dạng $\alpha \rightarrow \beta$ và thỏa $|\alpha| \leq |\beta|$ thì G là văn phạm loại 1 hoặc còn được gọi là ***văn phạm cảm ngữ cảnh CSG*** **(C**ontext-**S**ensitive **G**rammar)

Ngôn ngữ của lớp văn phạm này được gọi là ngôn ngữ cảm ngữ cảnh (CSL)

3) Văn phạm loại 2: Nếu văn phạm G có các luật sinh dạng $A \rightarrow \alpha$ với A là một biến đơn và α là một chuỗi các ký hiệu thuộc $(V * T)^*$ thì G là văn phạm loại 2 hoặc còn được gọi là ***văn phạm phi ngữ cảnh CFG*** (**C**ontext-**F**ree **G**rammar)

Ngôn ngữ của lớp văn phạm này được gọi là ngôn ngữ phi ngữ cảnh (CFL)

4) Văn phạm loại 3: Nếu văn phạm G có mọi luật sinh dạng **tuyến tính phải** (rightlinear): A –> wB hoặc A –> w với A, B là các biến đơn và w là chuỗi ký hiệu kết thúc (có thể rỗng); hoặc có dạng **tuyến tính trái** (left-linear): A –> Bw hoặc A –> w thì G là văn phạm loại 3 hay còn được

gọi là ***văn phạm chính quy RG*** (**R**egular **G**rammar)

Ngôn ngữ của lớp văn phạm này được gọi là ngôn ngữ chính quy (RL)

Ký hiệu : L0, L1, L2, L3 là các lớp ngôn ngữ sinh ra bởi các văn phạm loại 0, 1, 2, 3 tương ứng.

Ta có : $L3 \subset L2 \subset L1 \subset L0$ và các bao hàm thức này là nghiêm ngặt.

1.6. Các ví dụ về văn phạm

Cho bảng chữ cái $\sum = \{a_1, a_2, ..., a_n\}$. Xây dựng văn phạm sinh các ngôn ngữ sau:

VD 1. $L = \Sigma^*$

VD 2. $L = \Sigma^+$

VD 3. L = Các xâu có độ dài chẵn dương trên Σ

VD 4. L = Các xâu có độ dài chẵn trên Σ

VD 5. L = Các xâu có độ dài lẻ trên Σ

VD 6. Cho bảng chữ cái Σ $\{0,1,2\}$. ây dựng văn phạm G sinh ngôn ngữ L như sau:

L = gồm các từ bắt đầu bằng 011, chứa từ con 11211 và kết thúc bằng 1210

BÀI TẬP

Bài 1. Cho các ngôn ngữ sau: L_1 = {a, ab, abb, aabbb, abbaa, aaaabb, ababba}

$L_2 = \{\wedge, ab, bb, aba, bbb, aabb\}$

Tìm các ngôn ngữ hợp, giao, tích ghép, lặp của L_1và L_2. Tìm ngôn ngữ chia trái, chia phải của L_1 cho L_2. Tìm ngôn ngữ soi gương của L_1.

Bài 2. Cho bảng chữ cái Σ = {a, b, c}. Xây dựng văn phạm sinh ngôn ngữ

$L = \{a^n b^{2n+1} c^k \in \Sigma^* \mid n > 0, k \geq 0\}$

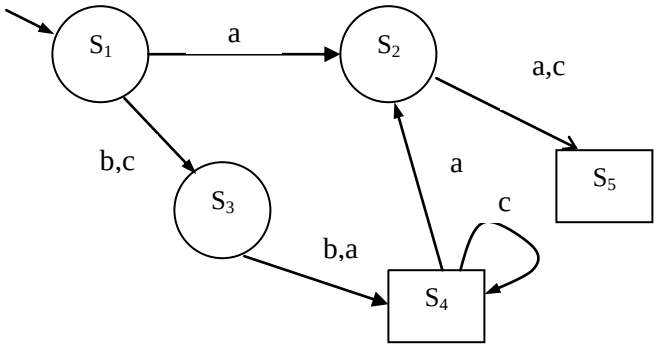
CHƯƠNG II: NGÔN NGỮ CHÍNH QUY VÀ OTOMAT HỮU HẠN

2.1. Nguồn và ngôn ngữ được sinh bởi nguồn

2.1.1. Định nghĩa: Nguồn là một đa đồ thị có hướng và có khuyên, có một đỉnh tách ra làm đỉnh vào, kí hiệu là($\rightarrow$O), một tập con các đỉnh mà mỗi đỉnh này được gọi là một đỉnh ra hay đỉnh kết và đặt trong một ô chữ nhật (□), đồng thời trên mỗi cung ghi một ký hiệu thuộc bảng chữ cái $\Sigma \cup \{\wedge\}$.

Cung trên đó ghi từ rỗng gọi là cung rỗng hay cung trống và thường người ta không ghi từ rỗng trên các cung này. Cung trên đó ghi ký hiệu thuộc Σ được gọi là cung cốt yếu. Đỉnh kết yếu là đỉnh có cung cốt yếu đi tới.

Ví dụ về nguồn :



Hình 1.1. Nguồn I_1

2.1.2. Nguồn đơn định

Nguồn I được gọi là nguồn đơn định nếu nó không chứa cung rỗng và hai cung tùy ý xuất phát từ một đỉnh đều được ghi bằng hai ký hiệu khác nhau. Nguồn I_1 là nguồn đơn định.

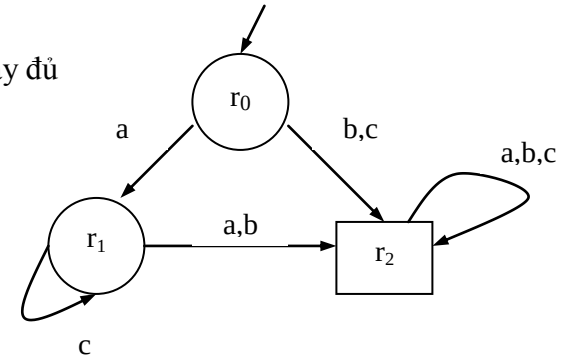
2.1.3. Nguồn đầy đủ

Nguồn đầy đủ là nguồn mà từ một đỉnh các cung đi ra phải chứa đầy đủ các ký hiệu $\in$ bảng chữ cái Σ.

2.1.4. Nguồn đơn định và đầy đủ

Là nguồn vừa đơn định vừa đầy đủ

Ví dụ :



Hình 1.2. Nguồn đơn định và đầy đủ I_2

2. 1. 5. Từ được sinh bởi nguồn

- Từ được sinh bởi nguồn là dãy các ký hiệu của bảng chữ cái Σ nằm trên các cung của nguồn đi từ đỉnh vào và đến một trong những đỉnh kết.

- Ngôn ngữ được sinh bởi nguồn I là tập hợp các từ được sinh bởi nguồn I, ký hiệu là $N(I) = \{x \in \Sigma^* | \ x \in N_I(v(I),s), \ s \in F(I)\}$.

Trong đó v(I) : đỉnh vào

F(I) : Tập đỉnh kết

- Theo nguồn trên ta xác định ngôn ngữ được sinh bởi nguồn I

$N_I(s_1,s_3) = \{b,c\}$

$N_I(s_1,s_4) = N_I(s_1,s_3).\{b,a\}.c^* = \{b,c\}.\{b,a\}.\{c^s | s = 0,1,2,\ldots\}$

$= \{bbc^s, bac^s, cbc^s, cac^s \mid s = 0,1,2\ldots\}$

$N_I(s_1,s_2) = \{a\} \cup N_I(s_1,s_4).\{a\} = \{a\}. \cup \{bbc^sa, bac^sa, cbc^sa, cac^sa\}$

$=\{bbc^sa, bac^sa, cbc^sa, cac^sa, a \mid s=0,1,2\ldots\}$

$N_I(s_1,s_5)=N_I(s_1,s_2).\{a,c\} =$

$bbc^saa, bac^saa, cbc^saa, cbc^saa, aa, bbc^sac, bac^sac, cbc^sac, cac^sac, ac | s=0,1,2\ldots\}$

$\Rightarrow$ Ngôn ngữ được sinh bởi nguồn I_1 là :

$N(I_1) = N_I(s_1,s_4) \cup N_I(s_1,s_5) = \{bbc^s, bac^s, cbc^s, cac^s \ | s = 0,1,2\ldots\} \cup bbc^taa, \ bac^taa, \ cbc^taa, \ cbc^taa, \ aa, \ bbc^tac, \ bac^tac, \ cbc^tac, \ cac^tac, \ ac \ | t = 0,1,2\ldots\} = \{aa, ac, bbc^s, bac^s, cbc^s, cac^s, bbc^taa, bac^taa, cbc^taa, cbc^taa, bbc^tac, bac^tac, cbc^tac, cac^tac \ | s, t = 0,1,2\ldots\}$

2.2. Các phép toán trên nguồn

2.2.1. Phép đơn định hóa

Giả sử nguồn I chưa đơn định và đầy đủ trên bảng chữ cái Σ, cần xây dựng nguồn K đơn định, đầy đủ tương đương với nó.

Thuật toán đơn định hoá nguồn như sau:

1) Đối với ký hiệu a tùy ý thuộc Σ và đỉnh s tùy ý của nguồn I xác định tập:

$T_1(s,a) = \{u \in D(I) \ | a \in N(s,u)\}$.

Đối với tập con M tùy ý của tập D(I) và mọi ký hiệu $a \in \Sigma$ xác định tập:

$$H_1(M,a) = \bigcup_{s \in M} T_1(s,a).$$

2) Xây dựng nguồn đơn định đầy đủ K

a. Đỉnh và cung:

Đỉnh vào của K: v(K)= {v(I)}.

Đối với mọi đỉnh M ∈ A(K) đã được xác định, ta xác định đỉnh $H_1(M,a)$ và kẻ một cung trên đó ghi ký hiệu a đi từ đỉnh M sang đỉnh $H_1(M,a)$.

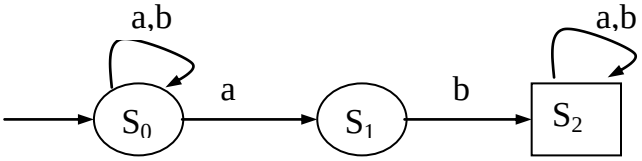
b.Tập đỉnh kết của K :

$$F(K) = \{M \in A(K) \mid M \cap F(I) \neq \varnothing \}$$

c.Với cách xây dựng như trên có nguồn K tương đương với nguồn I và K đơn định, đầy đủ.

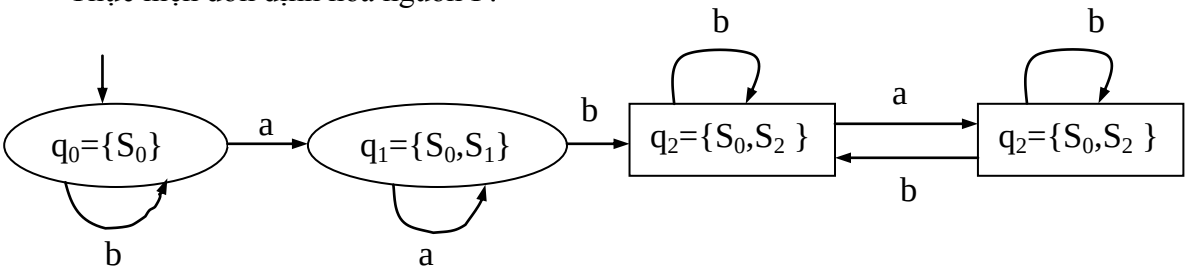
Chú ý: Nếu ta loại tập rỗng ra khỏi A(K) thì nguồn K nhận được đơn định, nhưng không đầy đủ.

Ví dụ : Cho nguồn chưa đơn định và đầy đủ I :

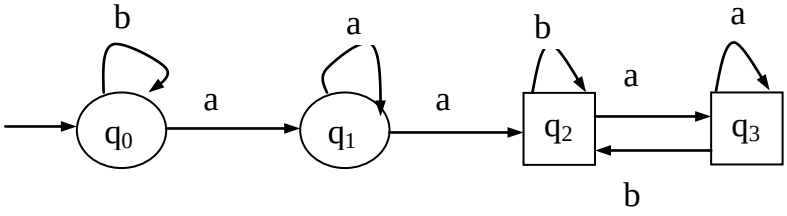


Hình 1.3. Nguồn I chưa đơn định đầy đủ

Thực hiện đơn định hóa nguồn I :



Nguồn đã đơn định đầy đủ, tương đương với I là :



Hình 1.4. Nguồn I' đơn định đầy đủ

2. 2. 2. Phép lấy phần bù

Giả sử I là nguồn tùy ý trên bảng chữ cái Σ. Để xây dựng nguồn sinh ngôn ngữ phần bù của ngôn ngữ do I sinh ra ta thực hiện một thuật toán gồm các bước sau:

1. Nếu I chưa đơn định và đầy đủ, ta xây dựng nguồn K đơn định, đầy đủ tương đương với I, tức là N(I)= N(K).

2. Trên nguồn K ta đổi các đỉnh kết thành không kết và không kết thành kết. Khi đó nguồn nhận được sẽ sinh ra ngôn ngữ phần bù của ngôn ngữ do nguồn K sinh ra.

Vậy $N(M) = C_\Sigma N(K) = C_\Sigma N(I)$.

2.2.3. Phép hợp nguồn

Giả sử I_1 và I_2 là các nguồn trên bảng chữ cái Σ. Cần xây dựng nguồn I sinh ngôn ngữ là hợp của hai ngôn ngữ do I_1 và I_2 sinh ra, tức là $N(I) = N(I_1) \cup N(I_2)$. Nguồn I được gọi là nguồn hợp của các nguồn I_1 và I_2.

Thuật toán:

Để xây dựng nguồn I hợp của hai nguồn I_1 và I_2, ta giữ nguyên cấu trúc của I_1 và I_2, thêm vào một đỉnh mới (không phụ thuộc I_1 cũng như I_2) và thừa nhận đỉnh này là đỉnh vào của nguồn I, đồng thời từ đỉnh mới thêm vào kẻ các cung rỗng đi tới các đỉnh $v(I_1)$ và $v(I_2)$.

Tập đỉnh kết của nguồn I là $F(I) = F(I_1) \cup F(I_2)$.

Với cách xây dựng trên dễ dàng nhận thấy rằng: $N(I) = N(I_1) \cup N(I_2)$.

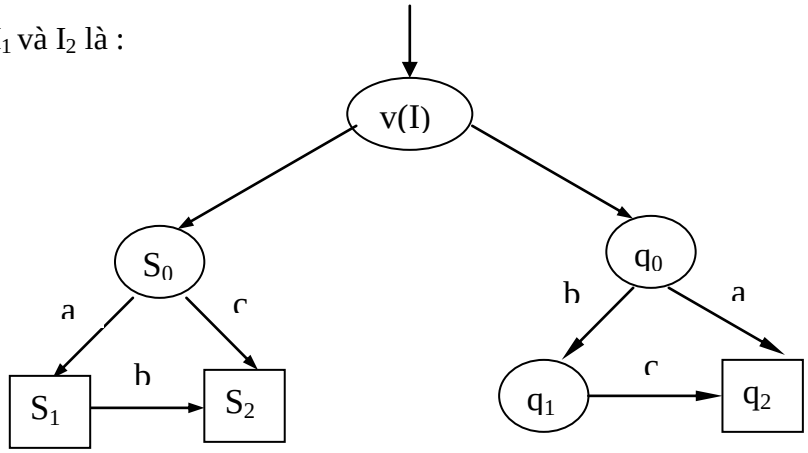
Chú ý: Ta có thể mở rộng phép hợp trên cho s (s>2) nguồn tùy ý.

Định lý: Đối với s (s > 1) nguồn tùy ý $I_1, I_2, \ldots, I_n$ luôn luôn xây dựng được nguồn I, để nó sinh ra ngôn ngữ là hợp của các ngôn ngữ do $I_1, I_2, \ldots, I_n$ sinh ra, tức là $N(I) = \bigcup_{i=1}^{s} N(I_i)$.

Cho 2 nguồn I_1 và I_2 :

Hình 1.5. Nguồn I_1 và I_2

Hợp của I_1 và I_2 là :



Hình 1.6. Hợp của I_1 và I_2

2.2.4. Phép giao nguồn

Giả sử có các nguồn I_1, I_2. Cần xây dựng nguồn I, sinh ngôn ngữ là giao của các ngôn ngữ do I_1, I_2 sinh ra, tức là $N(I)= N(I_1) \cap N(I_2)$. Nguồn I được gọi là nguồn giao của các nguồn I_1, I_2.

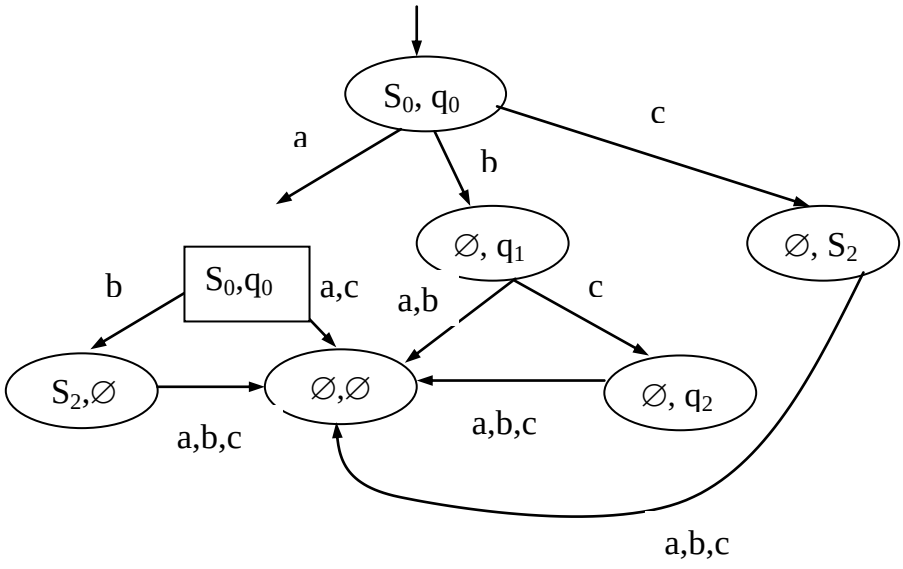
Thuật toán xây dựng nguồn giao I:

Để có nguồn giao I của các nguồn I_1, I_2 ta xác định đỉnh và cung của nó bằng phép quy nạp như sau:

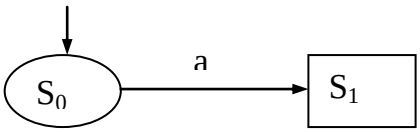
1. Đỉnh vào của nguồn I: $v(I) = \{v(I_1), v(I_2)\}$;
2. Giả sử B = (P,Q), trong đó $P \subseteq D(I_1)$, $Q \subseteq D(I_2)$ là các đỉnh tùy ý đã được xác định và a là ký hiệu bất kỳ thuộc Σ. Khi đó xác định đỉnh C = H_{I_1}(P,a), H_{I_2} (Q,a)), đồng thời kẻ một cung trên đó ghi ký hiệu a, đi từ đỉnh B sang C.
3. Đỉnh D = (S,R) được thừa nhận là đỉnh kết của nguồn I ($D \in F(I)$) khi và chỉ khi $S \cap F(I_1) \neq \varnothing$ và $R \cap F(I_2) \neq \varnothing$.

Với cách xây dựng trên nguồn I sinh ngôn ngữ là giao của các ngôn ngữ do I_1, I_2 sinh ra**.**

Lấy ví dụ với 2 nguồn I_1 và I_2 (hình 1.5)



Nguồn giao của chúng là :



Hình 1.7. Nguồn giao cuả I_1 và I_2

2.2.5. Phép tích ghép hai nguồn

Đối với hai nguồn I_1, I_2 tùy ý, cần xây dựng nguồn I, sinh ngôn ngữ là tích ghép của các ngôn ngữ do I_1, I_2 sinh ra, tức là $N(I)= N(I_1).N(I_2)$.

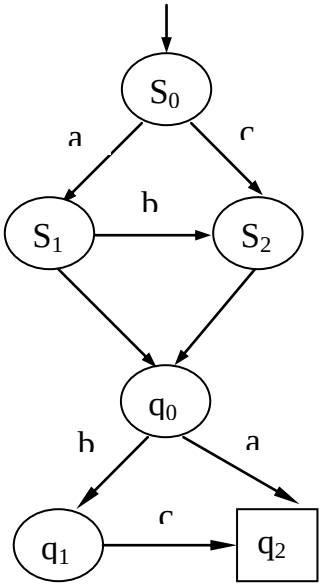
Nguồn I được gọi là nguồn tích ghép của nguồn I_1, I_2.

Thuật toán xây dựng nguồn tích ghép:

Để xây dựng nguồn tích ghép (I) của các nguồn I_1,I_2, ta giữ nguyên cấu trúc của I_1,I_2 thừa nhận đỉnh vào I_1 là đỉnh vào của nguồn I($v(I)=v(I_1)$), các đỉnh kết của nguồn I_2 là đỉnh kết của nguồn I($F(I)=F(I_2)$), đồng thời từ mỗi đỉnh kết của nguồn I_1 kẻ một cung rỗng đi tới đỉnh vào của nguồn I_2. Đồng thời chuyển các đỉnh kết của I_1 thành đỉnh thường.

Với cách xây dựng như trên nguồn I, sinh ngôn ngữ là tích ghép của các ngôn ngữ do I_1, I_2 sinh ra.

Tích ghép của hai nguồn I_1 và I_2 (hình 1.5) :



Hình 1.8. Tích ghép 2 nguồn I_1 và I_2

2.2.6. Phép lặp

Đối với nguồn I_1 tùy ý, cần xây dựng nguồn I_1 sinh ngôn ngữ là lặp của ngôn ngữ do I sinh ra. Nguồn I_1 được gọi là nguồn lặp của nguồn I.

Thuật toán xây dựng nguồn lặp I:

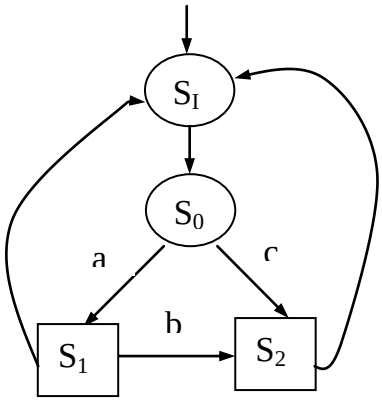
Để xây dựng nguồn I, sinh ngôn ngữ là lặp của ngôn ngữ do nguồn I_1 sinh ra, ta giữ nguyên

cấu trúc của I_1, thêm vào một đỉnh mới và thừa nhận đỉnh này là đỉnh vào đồng thời là đỉnh kết duy nhất của nguồn I. Từ đỉnh mới kẻ thêm một cung rỗng đi tới đỉnh $v(I_1)$, đồng thời từ mỗi đỉnh kết của nguồn I_1 kẻ một cung rỗng đi tới đỉnh mới thêm.

Nguồn sinh ngôn ngữ là lặp cắt của ngôn ngữ được sinh bởi I_1 chỉ khác với nguồn trên ở chỗ: đỉnh mới thêm chỉ được thừa nhận là đỉnh vào, còn tập đỉnh kết của I_1 được thừa nhận là đỉnh kết của nguồn mới.

Với cách xây dựng trên nguồn lặp (nguồn lặp cắt I) sinh ngôn ngữ là lặp (lặp cắt) của ngôn ngữ do I_1 sinh ra, tức là $N(I) = N(I_1)^*$ ($N(I) = N(I_1)^*$) .

Nguồn lặp của nguồn I_1 :



Hình 1.9. Nguồn lặp của I_1

2.2.7. Phép soi gương nguồn

Đối với nguồn I_1 tùy ý, cần xây dựng nguồn I sinh ngôn ngữ soi gương của ngôn ngữ do I_1 sinh ra. Nguồn I được gọi là nguồn soi gương của nguồn I_1.

Thuật toán xây dựng nguồn soi gương:

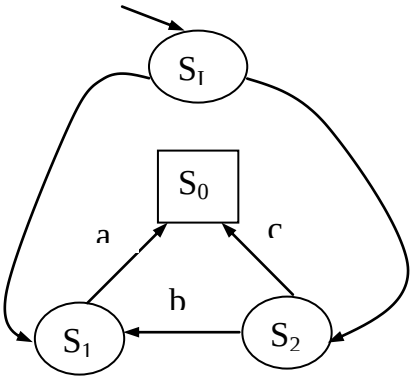
Để được nguồn I sinh ngôn ngữ soi gương của ngôn ngữ do I_1 sinh ra ta thực hiện các bước sau:

1) Thêm vào một đỉnh mới, thừa nhận nó là đỉnh vào của nguồn I, đồng thời từ đỉnh mới thêm (v(I)) kẻ các cung rỗng đi tới đỉnh kết của nguồn I_1.
2) Thừa nhận đỉnh vào của I_1 ($v(I_1)$) là đỉnh kết duy nhất của nguồn I.

   $F(I) = \{v(I_1)\}$;
3) Đối với mỗi cung của nguồn I_1, giữ nguyên ký hiệu ghi trên nó, nhưng đổi chiều ngược lại.

Với cách xây dựng trên ta có : $N(I) = N(I_1)$.

Nguồn soi gương của I_1 :



Hình 1.10. Nguồn soi gương của I_1

2.2.8. Phép chia trái

Giả sử I_1, I_2 là các nguồn trên bảng chữ cái Σ. Cần xây dựng nguồn K sinh ngôn ngữ là thương bên trái của các ngôn ngữ do I_1, I_2 sinh ra.Nguồn K được gọi là nguồn thương bên trái của các nguồn I_1, I_2.

Thuật toán xây dựng nguồn thương bên trái:

1) Xây dựng nguồn giao I của các nguồn I_1, I_2.
2) Thêm vào một đỉnh mới và thừa nhận đỉnh này là đỉnh vào của nguồn K. ký hiệu bằng v(K);
3) Tập đỉnh kết: $F(K) = F(I_1)$;
4) Giả sử Q = (S,T) là một đỉnh tùy ý của nguồn giao I. Khi đó đối với mỗi đỉnh $x \in S$ ($S \subseteq A(I_1)$) từ đỉnh vào của nguồn K (v(K)) có cung đi tới x khi và chỉ khi $T \cap F(I_2) \neq \varnothing$.

Với cách xây dựng này ta có: $N(K) = \begin{matrix} N(I_1) \\ \diagdown \\ N(I_2) \end{matrix}$

2. 2. 9. Phép chia phải

Giả sử I_1, I_2 là các nguồn trên bảng chữ cái Σ. Cần xây dựng nguồn K sinh ngôn ngữ là thương bên phải của các ngôn ngữ do I_1, I_2 sinh ra. Nguồn K được gọi là nguồn thương bên phải của các nguồn I_1, I_2.

Thuật toán xây dựng nguồn thương bên phải:

1) Dựa vào cấu trúc của I_1 đối với mỗi $s_1 \in D(I_1)$ xây dựng nguồn I_{s_i} có đỉnh vào là s_i ($v(I_{s_i}) = s_i$) và $F(I_{s_i}) = F(I_1)$

2) Xây dựng nguồn giao K_1 của I_1 và I_2.

Đối với mỗi I_{s_i} xây dựng nguồn giao K_{s_i} của I_{s_i} và I_2.

3) Xây dựng nguồn thương bên phải (K) của nguồn I_1 và I_2.

Nguồn K nhận được trên cơ sở biến đổi ngẫu nhiên bằng cách sau:

- Thừa nhận đỉnh vào của I_1 là đỉnh vào của nguồn K, tức là $v(K) \equiv v(I_1)$.

- Đỉnh $s_i \in D(I_1)$ được thừa nhận là đỉnh kết của nguồn K và chỉ khi $N(K_{s_i}) \neq \varnothing$

Với cách xây dựng như trên nguồn K sinh ngôn ngữ là thương bên phải của các ngôn ngữ I_1,I_2 sinh ta, tức là N(K $N(I_1)$ / $N(I_2)$

2. 3. Otomat hữu hạn không lối ra và ngôn ngữ được đoán nhận bởi otomat hữu hạn không lối ra:

Nghiên cứu các Otomat là một phần rất quan trọng trong lý thuyết tin học. Chúng được dùng để đoán nhận các ngôn ngữ. Có bốn loại Otomat thường được sử dụng, đó là Otomat hữu hạn không lối ra đoán nhận các lớp ngôn ngữ chính quy. Otomat hữu hạn có lối ra với chức năng biến đổi từ vào thành một từ có cùng độ dài. Otomat xác suất đoán nhận lớp ngôn ngữ ngẫu nhiên. Và cuối cùng là Otomat đẩy xuống dùng để đoán nhận lớp ngôn ngữ phi ngữ cảnh. Phần này ta sẽ nghiên cứu về Otomat hữu hạn.

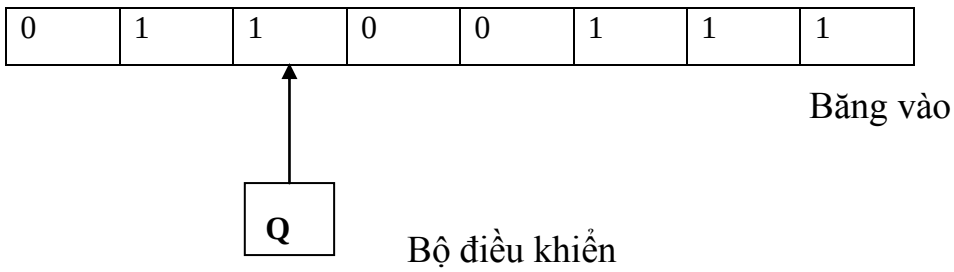
2. 3.1. Otomat đơn định (ÔHĐ)

Một cách trực quan ta có thể quan niệm Otomat hữu hạn như một "máy" đoán nhận xâu, mà các bộ phận và cung cách làm việc của nó như sau:

- Có một băng vào, dùng để ghi xâu vào (xâu cần được đoán nhận); mỗi ký hiệu của xâu vào (thuộc một bộ chữ cái Σ) được ghi trên một ô của băng vào.

- Có một đầu đọc, ở mỗi thời điểm quan sát một ô trên băng vào.

- Có một bộ điều khiển Q gồm một số hữu hạn trạng thái; ở mỗi thời điểm nó có một trạng thái (Hình 1.3.1).



Hình 1.3.1. Các bộ phận của Otomat hữu hạn

- Otomat hữu hạn làm việc theo từng bước rời rạc. Mỗi bước làm việc được mô tả như sau: Tùy theo trạng thái hiện thời của bộ điều khiển và ký hiệu đầu đọc quan sát được, mà otomat chuyển sang một trạng thái mới, đồng thời đầu đọc dịch chuyển sang phải một ô. Quy luật để chuyển sang trạng thái mới đó được cho bởi một hàm, gọi là hàm chuyển, $\delta : Q \times \Sigma \to Q$.

- Trong Q có phân biệt một trạng thái q_0, gọi là *trạng thái đầu* và một tập hợp F các trạng thái gọi là các *trạng thái cuối.*

- Ta nói Otomat đoán nhận (hay thừa nhận) một xâu vào $v \in \Sigma^*$, nếu Otomat xuất phát từ

trạng thái đầu q_0, với đầu đọc quan sát ký hiệu bên trái nhất của xâu v, sau một số hữu hạn bước làm việc, nó đọc xong xâu v (tức là đầu đọc vượt khỏi mút phải của v) và rơi vào một trong các trạng thái cuối.

- Tập hợp mọi xâu được đoán nhận bởi Otomat hợp thành ngôn ngữ được đoán nhận bởi Otomat đó.

Ta chú ý rằng tập Q thể hiện các trạng thái ghi nhớ của Otomat trong quá trình đoán nhận, và như vậy khả năng ghi nhớ của Otomat là hữu hạn. Mặt khác, hàm chuyển δ là hàm toàn phần và đơn trị, cho nên bước chuyển của Otomat luôn luôn được xác định một cách duy nhất. Chính vì hai đặc điểm này mà Otomat mô tả như trên được gọi là Otomat hữu hạn đơn định.

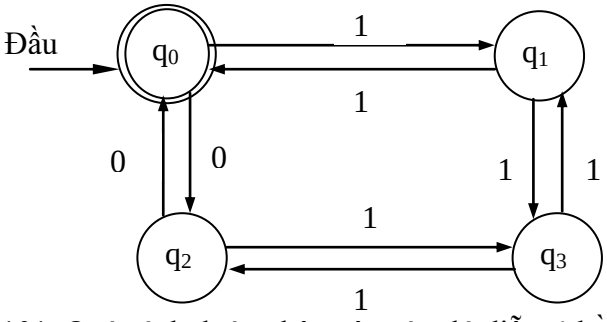
Ví dụ 1.3.1 : Xét ÔHĐ A, trong đó

$\Sigma = \{0,1\}$, $Q = \{q_0, q_1, q_2, q_3\}$, $F= \{q_0\}$

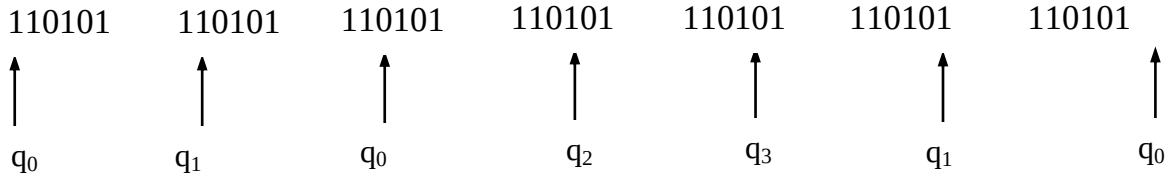
Hàm δ cho như sau :

δ	q_0	q_1	q_2	q_3
0	q_2	q_3	q_0	q_1
1	q_1	q_0	q_3	q_2

Để cho dễ hình dung hơn, ta thường biểu diễn hàm chuyển dưới dạng một đồ thị định hướng, gọi là biểu đồ chuyển như sau: Mỗi nút tương ứng với một trạng thái, nút đầu trỏ bởi một mũi tên có chữ "đầu" nút cuối được khoanh bởi hai vòng. Nếu có $\delta(q,a) = p$ thì có một cung đi từ nút q đến p, và cung đó mang nhãn a. Biểu đồ chuyển cho Otomat hữu hạn nói trên sẽ như hình sau :



Cho xâu vào 110101. Quá trình đoán nhận xâu vào đó diễn tả bằng các bước chuyển như sau, trong đó để cho gọn hình vẽ ta lược bớt các khung của băng vào và của bộ điều khiển :



Vì $q_0 \in F$, vậy xâu vào 110101 được thừa nhận.

2. 3. 1. 1. Định nghĩa

Otomat hữu hạn đơn định, không lối ra là một bộ năm :

$A = \{S, \Sigma, S_0, \delta, F\}$

Trong đó:

S : Tập hữu hạn các trạng thái của Otomat

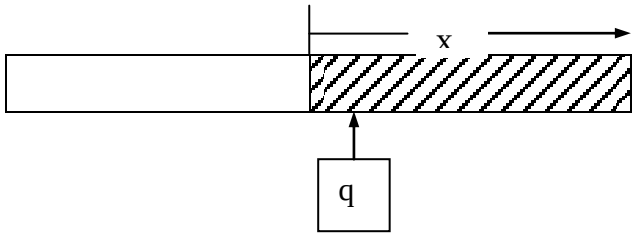
Σ : Bảng chữ cái vào của Otomat

$S_0 \in \Sigma$: Trạng thái khởi đầu

$\delta : S \times \Sigma \rightarrow S$: Hàm chuyển trạng thái

$F \in S$: Tập các trạng thái cuối

Ta gọi một hình trạng của ÔHĐ là một xâu có dạng qx với $q \in Q$ và $x \in \Sigma^*$. Có thể hiểu rằng xâu qx đó biểu diễn cho tình huống tức thời của ÔHĐ ở một lúc nào đó : ÔHĐ đang ở trạng thái q, còn x là phần xâu vào chưa được vượt qua ở trên bảng, và ký hiệu được nhòm lúc đó là ký hiệu bên trái nhất của x (hình dưới)



Hình 1.3.3 . Một tình huống tức thời của ÔHĐ

Quá trình đoán nhận một xâu vào của ÔHĐ là quá trình biến đổi các hình trạng qx p dựa vào hàm chuyển δ. Nếu có xâu vào $w = x_1x_2 \ldots x_n$ thì quá trình đoán nhận như sau:

$S_0x_1x_2 \ldots x_n \Rightarrow S_1x_2 \ldots x_n \Rightarrow \ldots \Rightarrow S_nx_n \Rightarrow S_{n+1}$. Trong đó S_0w (với S_0 là trạng thái đầu) được gọi là hình trạng đầu. Nếu $S_{n+1} \in F$ thì ta nói xâu w được A thừa nhận (đoán nhận)

Ví dụ 1.3.2 Cho ÔHĐ A :

$A=(\{S_0,S_1,S_2\}, \{a,b,c\}, S_0, \delta, \{S_2\})$ với hàm chuyển δ như sau :

δ	S_0	S_1	S_2
a	S_1	S_0	S_1
b	S_0	S_1	S_2
c	S_1	S_2	S_2

Quá trình đoán nhận xâu w = aabccb là :

$S_0aabccb \Rightarrow S_1abccb \Rightarrow S_0bccb \Rightarrow S_0ccb \Rightarrow S_1cb \Rightarrow S_2b \Rightarrow S_2$

*) Các tính chất của hàm chuyển trạng thái

- $\delta(s, \wedge)=S, \forall s \in S$
- $\delta(s, ax) = \delta(\delta(s, a), x), \forall a \in \Sigma, x \in \Sigma^*, s \in S$

2. 3. 1. 2. Ngôn ngữ đoán nhận bởi Otomat hữu hạn đơn định

Là tất cả các từ (xâu) $\in \Sigma^*$ được đoán nhận bởi ÔHĐ, ký hiệu là T(A)

$$T(A) = \{x \in \Sigma^* \mid \delta(S_0, x) \in F\}$$

2. 3. 2. Otomat không đơn định (ÔHK)

2. 3. 2. 1. Định nghĩa

Otomat hữu hạn không đơn định là một bộ năm

$$A=\{S, \Sigma, S_0, \delta, F\}$$

Trong đó:

S: Tập hữu hạn các trạng thái

Σ: Bảng chữ cái

$S_0 \in S$: Trạng thái khởi đầu

$\delta : S \times \Sigma \rightarrow P$ (P là một tập trạng thái $\in S$)

$F \in S$: Tập các trạng thái kết

Ví dụ 1.3.3. $A=(\{t_0, t_1, t_2\}, \{0, 1\}, \{t_0\}, \delta, \{t_1,t_2\}$

δ	t_0	t_1	t_2
0	t_1	$\{t_0, t_1\}$	t_1
1	$\{t_1,t_2\}$		t_0

*) Một số tính chất của ÔHK

- $\delta(Q, \wedge)= Q, \quad (Q= \bigcup_{Si \in S} Si)$
- $\delta(Q, ax)=\delta(\delta(Q,a),x), \quad a \in \Sigma, x \in \Sigma^*$

- $\delta(Q,a)= P, \quad (P= \bigcup_{Si \in Q} \delta(S,a))$

Quá trình đoán nhận xâu vào của ÔHK tương tự như của ÔHĐ.

Ví dụ đối với ÔHK A đoán nhận xâu 00101 như sau:

$t_0 00101 \Rightarrow t_1 0101 \Rightarrow \{t_0, t_1\}101 \Rightarrow \{t_0, t_1\}01 \Rightarrow \{t_0, t_1\}1 \Rightarrow \{t_1, t_2) \in F$

2. 3.2.2. Ngôn ngữ được đoán nhận bởi Otomat hữu hạn không đơn định

Ngôn ngữ được đoán nhận bởi Otomat không đơn định là các xâu $\in \Sigma^*$ đẩy Otomat từ trạng thái đầu tới một trong các trạng thái kết, ký hiệu là T(A)

$$T(A) =\{x \in \Sigma^* | \delta(s_0,x) \cap F \neq \varnothing\}$$

Lưu ý: Otomat A và otomat B được gọi là hai otomat tương đương nhau nếu chúng đoán nhận cùng một ngôn ngữ, tức là T(A) = T(B)

2. 3.3. Tính đầy đủ và đơn định của Otomat

Định nghĩa: Otomat hữu hạn (đơn định hay không đơn định)

$$A = (S, \Sigma, S_0, \delta, F)$$

được gọi là một otomat đầy đủ nếu hàm chuyển trạng thái xác định khắp nơi trên tập $S \times \Sigma$, nghĩa là $\forall s \in S, \forall a \in \Sigma$ đều có $\delta(s, a) \in S$ trong trường hợp A đơn định và $\delta(s, a) \cap S \neq \varnothing$ trong trường hợp A không đơn định.

Định nghĩa: Otomat A được gọi là otomat đơn định và đầy đủ nếu nó vừa là otomat đơn định vừa là otomat đầy đủ.

Ví dụ 1.3.4 Cho Otomat

$$A= (\{s_0, s_1, s_2, s_3, s_4\}, \{a,b,c\}, s_0, \delta, \{s_1, s_2\})$$

Trong đó hàm chuyển trạng thái được xác định bằng bảng sau :

δ	s_0	s_1	s_2	s_3	s_4
a	s_1	s_2	s_1	s_4	s_4
b	s_2	s_3	s_4	s_3	s_4
c	s_3	s_4	s_3	s_2	s_4

Nhận xét : Căn cứ vào bảng chuyển của otomat

$$A = (S, \Sigma , S_0, \delta, F)$$

ta có nhận xét sau đây :

1. Otomat A đơn định khi và chỉ khi trong bảng chuyển của nó không có vị trí nào chứa quá một trạng thái thuộc S.

2. Otomat A đầy đủ khi và chỉ khi mọi vị trí của bảng chuyển đều chứa ít nhất một trạng thái thuộc S.

2. 3.4. Phép đơn định hoá một Otomat

Giả sử A = (S, Σ, S_0, δ, F) là một Otomat chưa đơn định hoặc chưa đầy đủ. Cần xây dựng otomat M đơn định và đầy đủ tương đương với A (đoán nhận cùng ngôn ngữ do A đoán nhận).

Thuật toán đơn định hoá:

Giả sử A=(S, Σ, S_0, δ, F) là một Otomat hữu hạn không lối ra tùy ý. Để xây dựng otomat M hữu hạn đơn định, dầy đủ và tương đương với A ta thực hiện một số bước sau:

1) Xác định hàm hai biến :

$$T: 2^s \times \Sigma \to 2^s$$

bằng các quan hệ sau:

- $\forall s \in S, \forall a \in \Sigma , T(s, a)=\{s' \in S \mid s \in \delta(s,a) \}$
- $\forall B \subseteq S, \forall a \in \Sigma, T(B,a)= \bigcup_{s\in B} T(s,a)$

2) Khi đó otomat M là một bộ năm:

$$M = (Q, \Sigma, Q_0, f, P)$$

trong đó: $Q \subseteq 2^s$, $Q_0 = \{S_0\}$, $P = \{q \in Q \mid q \cap F \neq \varnothing \}$

Và hàm chuyển trạng thái f được xác định bằng quan hệ sau:

$$\forall q \in Q, \forall a \in \Sigma \ (f(q, a) = T(q, a))$$

Với cách xác định như trên ta có T(M) = T(A).

Ví dụ 1.3.5. Cho Otomat đơn định

$$A_1 = (\{s_0, s_1, s_2, s_3\}, \{a, b, c\}, s_0, \delta_1, \{s_2, s_3\})$$

với hàm chuyển δ được xác định bằng bảng chuyển sau :

δ_1	s_0	s_1	s_2	s_3
a	s_1	s_2	s_1	
b	s_2	s_2		s_3
c	s_3		s_3	s_2

Xây dựng Otomat M_1 đơn định, đầy đủ tương đương với otomat A .

Vì A đã đơn định nhưng chưa đầy đủ nên ta chỉ cần thêm vào một trạng thái mới (s_4) để cho hàm chuyển xác định khắp nơi, ta được otomat M_1 tương đương với A_1

$$M_1 = (\{s_0, s_1, s_2, s_3, s_4\}, \{a, b, c\}, s_0, \delta_1, \{s_2, s_3\})$$

Trong đó hàm chuyển được xác định bằng bảng sau:

δ_1	s_0	s_1	s_2	s_3	s_4
a	s_1	s_2	s_1	s_4	s_4
b	s_2	s_2	s_4	s_3	s_4
c	s_3	s_4	s_3	s_2	s_4

Ví dụ 1.3.6 : Cho Otomat A

$$A= (\{ s_0, s_1, s_2\}, \{0,1\}, \{s_0\}, \delta, \{s_2\})$$

trong đó δ xác định bằng bảng sau :

δ	s_0	s_1	s_2
0	$\{s_0, s_1\}$		s_1
1		s_2	s_0

A chưa đơn định và đầy đủ, tiến hành đơn định hoá Otomat ta được Otomat đơn định, đầy đủ như sau :

$T(s_0, 0) = \{s_0, s_1\} = q_1$

$T(s_0, 1) = \varnothing = q_2$

$T(s_1, 0) = \varnothing = q_2$

$T(s_1, 1) = \{s_2\} = q_3$

$T(s_2, 0) = \{s_1\} = q_4$

$T(s_2, 1) = \{s_0\} = q_0$

$T(q_1, 0)= T(\{s_0, s_1\}, 0)= T(s_0, 0) \cup T(s_1, 0) = \{s_0, s_1\} \cup \varnothing = \{s_0, s_1\} = q_1$

$T(q_1, 1) = T(\{s_0, s_1\}, 1) = T(s_0, 1) \cup T(s_1, 1) = \{s2\} = q_3$

$T(q_3, 0) = T(s_2, 0) =\{s_1\}= q_4$

$T(q_3, 1) = T(s_2, 1) =\{s_0\} = q_0$

$T(q_4, 0) = T(s_1, 0) = \varnothing = q_2$

$T(q_4, 1) = T(s_1, 1) = \{s_2\} = q_3$

Ta xác định được Otomat đơn định đầy đủ A_1 với Otomat A như sau :

$$A_1 = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{q_0\}, \delta_1, \{q_3\})$$

và hàm chuyển δ_1được cho bằng bảng sau :

δ_1	q_0	q_1	q_2	q_3	q_4
0	q_1	q_1	q_2	q_4	q_2
1	$q_{2`}$	q_3	q_2	q_0	q_3

2. 4. Sự tương đương của nguồn và Otomat hữu hạn không lối ra

Định nghĩa: Nguồn I và Otomat hữu hạn không lối ra A được gọi là tương đương nếu chúng sinh ra cùng một ngôn ngữ, nghĩa là N(I)=N(A).

Vấn đề đặt ra là: Khi có nguồn I cần xây dựng Otomat A tương đương với nó, và ngược lại, khi cho một otomat hữu hạn không lối ra cần xây dựng nguồn I tương đương với otomat này.

2.4.1. Xây dựng otomat tương đương với nguồn

2.4.1.1. Xây dựng Otomat có thể không đơn định tương đương với nguồn

Thuật toán:

Giả sử có nguồn I trên bảng chữ cái Σ .Để xây dựng otomat A có thể không đơn định đầy đủ tương đương với nguồn I ta thực hiện một số bước sau:

1) Xây dựng hàm hai biến T:

$T: 2^{D(I)} \times \Sigma \rightarrow 2^{D(I)}$

được xác định bằng quan hệ sau:

- $\forall s \in (D(I) \cup \{v(I)\}), \forall a \in \Sigma, T(s,a)=\{q \subseteq D(I)\}$
- $\forall a \in \Sigma, T(\varnothing,a) = \varnothing$

2) Xác định tập trạng thái kết và trạng thái khởi đầu

- Tập $\{v(I)\}$ được thừa nhận là trạng thái khởi đầu của otomat A và được ký hiệu là Q_0.
- Tập trạng thái kết (F) của otomat được xác định như sau:

$$F = \begin{cases} \{q \subseteq D(I) \mid q \cap F(I) \neq \phi\} \cup \{q_0\} & \text{nếu } \wedge \in N(I) \\ \{q \subseteq D(I) \mid q \cap F(I) \neq \phi\} & \text{nếu } \wedge \notin N(I) \end{cases}$$

3) Otomat không đơn định A có dạng :

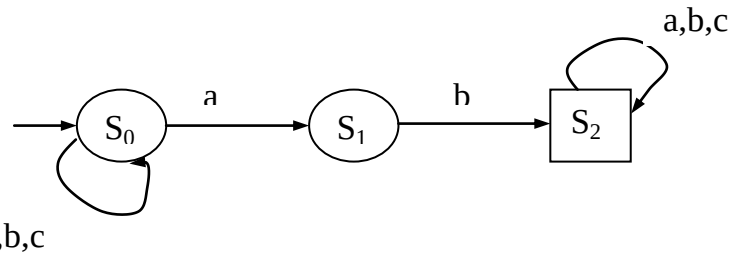
$A=(Q, \Sigma, Q_0, \delta, F)$

Trong đó Q là tập trạng thái của Otomat và hàm chuyển trạng thái δ được xác định bằng quan hệ:

$\forall q \in Q, \forall a \in \Sigma(\delta(q,a) = T(q,a))$.

Bằng thuật toán này ta có xây dựng được otomat A tương đương với nguồn I, tức là N(A)=N(I).

Ví dụ Cho nguồn :



- Xác định tập trạng thái :

$T(s_0,a)= \{s_0, s_1\}$

$T(s_0,b)= \{s_0\}$

$T(s_0,c)= \{s_0\}$

$T(s_1,a) = \varnothing$

$T(s_1,b)= \{s_2\}$

$T(s_1,c)= \varnothing$

$T(s_2,a)= \{s_2\}$

$T(s_2,b)= \{s_2\}$

$T(s_2,c)= \{s_2\}$

- Otomat tương đương với nguồn có dạng sau :

$$A= (\{s_0, s_1, s_2\}, \{a, b, c\}, s_0, \delta, s_2)$$

và hàm chuyển δ được xác định bằng bảng sau :

δ	s_0	s_1	s_2
A	$\{s_0, s_1\}$		s_2
B	s_0	s_2	s_2
C	s_0		s_2

2.4.1.2. Xây dựng Otomat tương đương đơn định, đầy đủ tương đương với nguồn.

Thuật toán:

Giả sử có nguồn I trên bảng chữ cái Σ. Để xây dựng otomat A đơn định đầy đủ tương đương với nguồn I ta thực hiện một số bước sau:

1) Xây dựng hàm hai biến T:

$T: 2^{D(I)} \times \Sigma \rightarrow 2^{D(I)}$

được xác định bằng quan hệ sau:

- $\forall s \in (D(I) \cup \{v(I)\}), \forall a \in \Sigma, T(s,a)=\{s' \in D(I) \mid a \in N_I(s, s')\}$

- $\forall q \subseteq D(I), \forall a \in \Sigma, T(q,a) = \bigcup_{s \in q} T(s,a)$
- $\forall a \in \Sigma, T(\varnothing,a) = \varnothing$

2) Xác định tập trạng thái kết và trạng thái khởi đầu

- Tập $\{v(I)\}$ được thừa nhận là trạng thái khởi đầu của otomat A và được ký hiệu là Q_0.
- Tập trạng thái kết (F) của otomat được xác định như sau:

$$F = \begin{cases} \{q \subseteq D(I) \mid q \cap F(I) \neq \phi\} \cup \{q_0\} & \text{nếu } \wedge \in N(I) \\ \{q \subseteq D(I) \mid q \cap F(I) \neq \phi\} & \text{nếu } \wedge \notin N(I) \end{cases}$$

3) Otomat đơn định đầy đủ A có dạng :

$$A=(Q, \Sigma, Q_0, \delta, F)$$

Trong đó Q là tập trạng thái của Otomat và hàm chuyển trạng thái δ được xác định bằng quan hệ:

$$\forall q \in Q, \forall a \in \Sigma(\delta(q,a) = T(q,a)).$$

Bằng thuật toán này ta có xây dựng được otomat A tương đương với nguồn I, tức là N(A)=N(I).

Xét ví dụ trên, ta đi xây dựng Otomat đơn định đầy đủ tương đương với nguồn.

$T(s_0,a) = \{s_0, s_1\} = q_1$

$T(s_0,b) = \{s_0\} = q_0$

$T(s_0,c) = \{s_0\} = q_0$

$T(s_1,a) = \varnothing = q_2$

$T(s_1,b) = \{s_2\} = q_3$

$T(s_1,c) = \varnothing = q_2$

$T(s_2,a) = \{s_2\} = q_3$

$T(s_2,b) = \{s_2\} = q_3$

$T(s_2,c) = \{s_2\} = q_3$

$T(q_1,a) = T(\{s_0, s_1\},a) = T(s_0,a) \cup T(s_1,a) = \{s_0, s_1\} \cup \varnothing = \{s_0, s_1\} = q_1$

$T(q_1,b) = T(\{s_0, s_1\},b) = T(s_0,b) \cup T(s_1,b) = \{s_0, s_2\} = q_4$

$T(q_1,c) = T((\{s_0, s_1\},c) = T(s_0,c) \cup T(s_1,c) = \{s_0\} \cup \varnothing = s_0 = q_0$

$T(q_3,a) = T(s_2,a) = q_3$

$T(q_3,b) = T(s_2,b)= q_3$

$T(q_3,c) = T(s_2,c)= q_3$

$T(q_4,a) = T(\{s_0, s_2\},a) = T(s_0,a) \cup T(s_2,a) = \{s_0, s_1\} \cup \{s_2\} = \{s_0, s_1, s_2\} = q_5$

$T(q_4,b) = T(\{s_0, s_2\},b) = T(s_0,b) \cup T(s_2,b) =\{s_0, s_2\} = q_4$

$T(q_4,c) = T(\{s_0, s_2\},c) = T(s_0,c) \cup T(s_2,c) =\{s_0, s_2\} = q_4$

$T(q_5,a) = T(\{s_0, s_1, s_2\},a) =T(s_0,a) \cup T(s_1,a) \cup T(s_2,a) = (\{s_0, s_1, s_2\} = q_5$

$T(q_5,b) = T(\{s_0, s_1, s_2\},b) =T(s_0,b) \cup T(s_1,b) \cup T(s_2,b) = (\{s_0, s_2\} = q_4$

$T(q_5,c) = T(\{s_0, s_1, s_2\},c) =T(s_0,c) \cup T(s_1,c) \cup T(s_2,c) = (\{s_0, s_2\} = q_4$

Vậy Otomat A đơn định và đầy đủ tương đương với nguồn là :

$A=(\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{a, b, c\}, q_0, \delta, \{q_3, q_4, q_5\})$

và hàm chuyển trạng thái δ được xác định bằng bảng sau :

δ	q_0	q_1	q_2	q_3	q_4	q_5
a	q_1	q_1	q_2	q_3	q_5	q_5
b	q_0	q_4	q_2	q_3	q_4	q_4
c	q_0	q_0	q_2	q_3	q_4	q_4

2.4.2. Xây dựng nguồn tương đương với Otomat

Thuật toán:

Giả sử có otomat $A =(S, \Sigma, S_0, \delta, F)$.Ta xây dựng nguồn I tương đương với otomat bằng các bước thực hiện sau:

1) Xác định đỉnh: Lấy các điểm trên mặt phẳng hoặc trong khôn gian tương ứng các trạng thái của otomat và dùng ngay các ký hiệu trạng thái để đặt tên cho các điểm tương ứng.

Đỉnh tương ứng với trạng thái khởi đầu được thừa nhận là đỉnh vào của nguồn, đồng thời cũng đươc ký hiệu bằng v(I) ($v(I) \equiv S_0$).

Đỉnh s được thừa nhận là đỉnh kết khi và chỉ khi $s \in F$.

2) Xác định cạnh :$\forall$ s, s' $\in$ S, $\forall$ a $\in \Sigma$ .Từ đỉnh s sang tập các đỉnh $\bigcup$ s'_i có cung ghi ký hiệu a khi và chỉ khi $s'_i \in \delta(s,a)$.

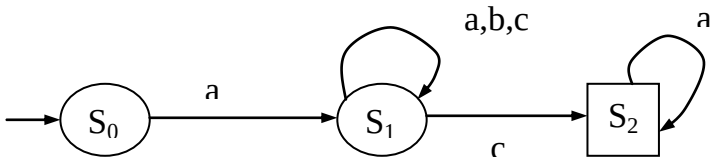
Bằng thuật toán này ta xây dựng được nguồn tương đương với otomat A.

Ví dụ : Cho Otomat A= ($\{s_0, s_1, s_2\}$, {a, b, c}, s_0, δ, s_2)

δ	s_0	s_1	s_2

a	s_1	s_1	s_2
b		s_1	
c		$\{s_1,s_2\}$	

Xây dựng nguồn tương đương với Otomat :



2. 5. Sự tương đương của nguồn và văn phạm chính quy

2.5.1. Xây dựng nguồn tương đương với văn phạm chính quy.

Thuật toán

Ví dụ

2.5.2. Xây dựng văn phạm chính quy tương đương với nguồn.

Thuật toán

Ví dụ

2. 6. Sự tương đương của nguồn và biểu thức chính quy

2. 6.1. Xây dựng nguồn tương đương với biểu thức chính quy.

Thuật toán

Ví dụ

2. 6.2. Xây dựng biểu thức chính quy tương đương với nguồn.

Thuật toán

Ví dụ

2. 7. Bài tập tổng hợp

Cho một ngôn ngữ chính quy L. Xây dựng nguồn, văn phạm chính quy, Otomat hữu hạn và biểu thức chính quy sinh ngôn ngữ L

Cho bảng chữ cái $\sum = \{a_1, a_2, ..., a_n\}$. Xây dựng văn phạm sinh các ngôn ngữ sau:

VD 1. L = Σ*

VD 2. L = Σ^+

VD 3. L = Các xâu có độ dài chẵn dương trên Σ

VD 4. L = Các xâu có độ dài chẵn trên Σ

VD 5. L = Các xâu có độ dài lẻ trên Σ

VD 6. Cho bảng chữ cái $\Sigma \{0,1,2\}$. ây dựng văn phạm G sinh ngôn ngữ L như sau:

L = gồm các từ bắt đầu bằng 011, chứa từ con 11211 và kết thúc bằng 1210

2. 8. Tính đóng của lớp ngôn ngữ chính quy

2.8.1. Lớp ngôn ngữ chính quy

2.8.2. Tính đóng của lớp ngôn ngữ chính quy

2. 9. Điều kiện cần của lớp ngôn ngữ chính quy

2.9.1. Otomat tối tiểu

2.9.2. Điều kiện cần của lớp ngôn ngữ chính quy.

2. 10. Điều kiện cần và đủ của lớp ngôn ngữ chính quy

2.10.1. Quan hệ tương đương bất biến phải chỉ số hữu hạn

2.10.2. Điều kiện cần và đủ

2. 11. Otomat hữu hạn có lối ra

2.10.1. Định nghĩa

2.10.2 Cách biểu diễn

2. 12. ω-Ngôn ngữ chính quy (Siêu ngôn ngữ chính quy)

2.12.1. Các định nghĩa

2.12.2. Các phép toán trên siêu ngôn ngữ

2. 12. 3. Tính đóng của siêu ngôn ngữ chính quy.

BÀI TẬP

Bài 1. Cho bảng chữ cái $\Sigma = \{a, b, c\}$.

Xây dựng văn phạm sinh ngôn ngữ $L = \{a^n b^{2n+1} c^k \in \Sigma^* \mid n > 0, k \geq 0\}$

Bài 2. Cho 2 nguồn I_1 và I_2. Tìm nguồn bù, nguồn hợp, nguồn tích ghép, nguồn soi gương của I_1 và I_2.

Bài 3.. Cho $\Sigma = \{a, b, c\}$. Xây dựng nguồn I sinh ngôn ngữ sau đây:

a. $N(I) = \{\alpha = ab^n cc$, với n nguyên dương$\}$

b. $N(I) = \{\alpha = ab^{2n} ca$, với n nguyên dương$\}$

c. $N(I) = \{\alpha = ab^{2n-1} cb$, với n nguyên dương$\}$

d. $N(I) = \{\alpha = ab\beta c$, với $\beta \in \Sigma^*\}$

Bài 4. Cho $\Sigma = \{0, 1\}$. Xây dựng nguồn I sinh ngôn ngữ sau đây:

a. $N(I) = \{\alpha = 01\beta 0$, với $\beta \in \Sigma^*\}$

b. N(I) = {$\alpha = 1\beta 1$, với β có độ dài lẻ}

c. N(I) = {$\alpha = 11\beta 01$, với β có độ dài chẵn dương}

d. N(I) = {$\alpha = 11\beta 01$, với β có độ dài chẵn}

Chương 3. NGÔN NGỮ PHI NGỮ CẢNH VÀ OTOMAT ĐẨY XUỐNG

3.1. Cây và phép thế cây

3.2. Dạng chuẩn Chomsky

3.3. Cây dẫn xuất và các tính chất.

3.3.1. Vấn đề vô hạn của ngôn ngữ phi ngữ cảnh

3.3.2. Tính đóng của lớp ngôn ngữ phi ngữ cảnh.

3.3.3 Tính nhập nhằng của ngôn ngữ phi ngữ cảnh.

3.4. Khái niệm về Otomat đẩy xuống

3.4.1. Otomat đẩy xuống không đơn định và ngôn ngữ phi ngữ cảnh

3.4.1.1. Định nghĩa

3.4.1.2. Hình trạng

3.4.1.3. Hàm chuyển

3.4.1.4. Ngôn ngữ được đoán nhận bởi Otomat đẩy xuống không đơn định.

3.4.2. Giao của ngôn ngữ phi ngữ cảnh và ngôn ngữ chính quy.

3.4.3. Otomat đẩy xuống đơn định.

3.4.3.1. Định nghĩa.

3.4.3.2. Ví dụ.

3.4.3.3. Tính chất.

BÀI TẬP

Bài 1. Trình bầy điều kiện cần của ngôn ngữ chính quy và ứng dụng.

Bài 2. Trình bầy điều kiện cần và đủ của ngôn ngữ chính quy. cho ví dụ về phân hoạch.

Bài 3. Trình bầy định nghĩa Otomat hữu hạn có lối ra, các cách biểu diễn Otomat hữu hạn có lối ra. Cho ví dụ minh họa.

CHƯƠNG 4. CƠ BẢN VỀ CHƯƠNG TRÌNH DỊCH

4.1. Giơi thiệu về chương trình dịch

Chương này sẽ giới thiệu một cách tổng quan về vấn đề biên dịch bằng cách mô tả các thành phần của một trình biên dịch và môi trường hoạt động của nó.

Mục đích của môn học:

Nắm vững các nguyên lý xây dựng một chương trình dịch từ đó hiểu về bản chất và sử dụng có hiệu quả các ngôn ngữ lập trình . Có khả năng tự thiết kế và xây dựng được một chương trình dịch cho một ngôn ngữ đơn giản (ngôn ngữ PL/0).

Nâng cao khả năng lập trình thông qua các bài thực hành để xây dựng các thành phần cho mộ t chương trình dịch. Các bài thực hành trong môn học là các bài phức tạp , kết hợp nhiều kiến thức về ngôn ngữ hình thức , phân tích văn phạm , cấu trúc và giải thuật , vì vậy sinh viên có điều kiện luyện tập thiết kế và viết các chương trình lớn, độ phức tạp cao.

Những kiến thức của môn học cũng có thể được sử dụng trong các lĩnh vực khác như xử lý ngôn ngữ tự nhiên.

4.2. Chương trình dịch

4.2.1. Định nghĩa

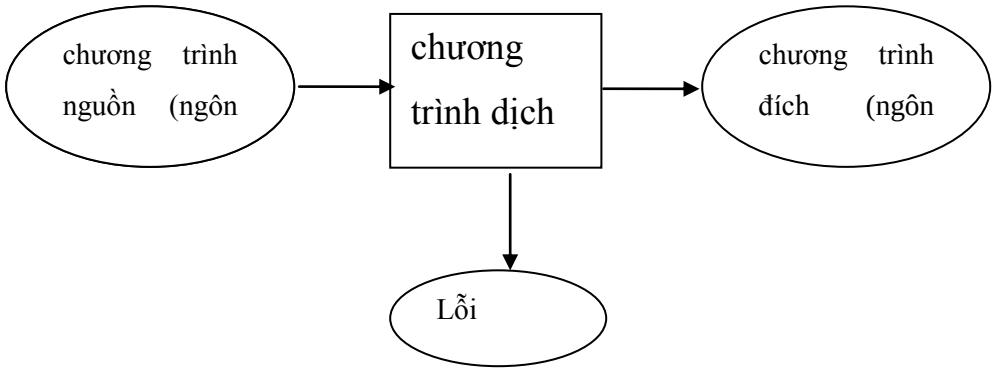
Chúng ta đều biết có rất nhiều loại máy tính khác nhau như máy PC, máy Macshintos với các chíp khác nhau và một ngôn ngữ máy khác nhau (tập các chỉ thị lệnh khác nhau).Việc xây dựng các ứng dụng trực tiếp trên ngôn ngữ máy là rất khó và phức tạp. Đối với các ứng dụng lớn thì điều này gần như là không khả thi.

Vì vậy nhu cầu có một ngôn ngữ trung gian, gần với ngôn ngữ tự nhiên là tất yếu, và khi đó cần có một hệ thống (chương trình) để dịch các chương trình trên ngôn ngữ này sang mã máy để có thể chạy được. Những chương trình làm nhiệm vụ như vậy gọi là các chương trình dịch. Ngoài ra nhiệm vụ của một chương trình dịch không chỉ là chuyển một chương trình từ một ngôn ngữ lập trình sang ngôn ngữ máy mà tổng quát là chuyển một chương trình viết ở một ngôn ngữ này sang ngôn ngữ khác . Thông thường ngôn ngữ nguồn là ngôn ngữ bậc cao và ngôn ngữ đích là ngôn ngữ bậc thấp, ví dụ như từ C hay Pascal sang Asembly.

Nói một cách đơn giản, trình biên dịch là một chương trình làm nhiệm vụ đọc một chương trình được viết bằng một ngôn ngữ - ngôn ngữ nguồn (source language) - rồi dịch nó thành một chương trình tương đương ở một ngôn ngữ khác - ngôn ngữ đích (target languague). Một phần quan trọng trong quá trình dịch là ghi nhận lại các lỗi có trong chương trình nguồn để thông báo lại cho người viết chương trình.

Định nghĩa: Chương trình dịch là một chương trình thực hiện việc chuyển đổi một chương trình hay một đoạn chương trình từ một ngôn ngữ này (gọi là ngôn ngữ nguồn) sang ngôn ngữ khác (gọi là ngôn ngữ đích) tương đương.

Sơ đồ một chương trình dịch như sau:



Hình 1.1 - *Một trình biên dịch*

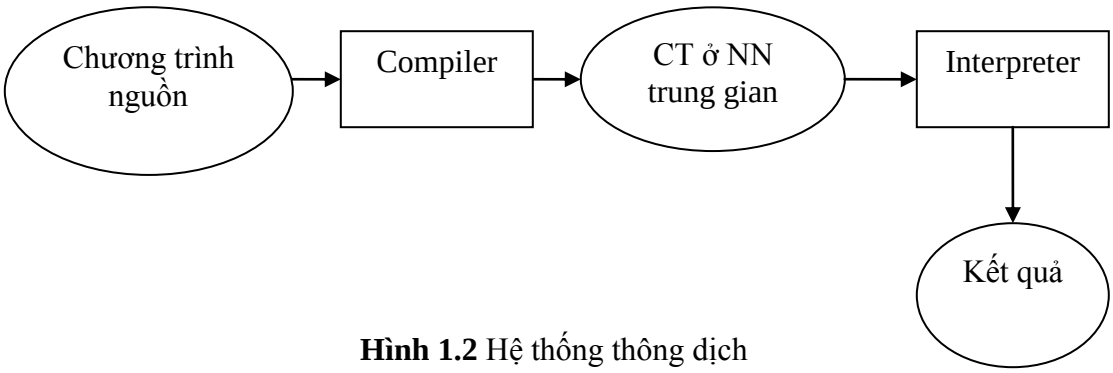
Hệ thống biên dịch

Trong hệ thống biên dịch như Borland C, hay Turbo Pascal, toàn bộ chương trình nguồn được trình biên dịch chuyển sang chương trình đích ở dạng mã máy. Chương trình đích này có thể chạy độc lập trên máy mà không cần hệ thống biên dịch nữa.

Hệ thống Thông dịch (Interpreter)

Trong một hệ thống thông dịch, chương trình thông dịch đọc chương trình nguồn theo từng lệnh và phân tích rồi thực hiện nó, ví dụ như hệ điều hành thực hiện các câu lệnh DOS, hay hệ quản trị cơ sở dữ liệu Foxpro.

Một kiểu khác trong một hệ thống thông dịch là ngôn ngữ nguồn không được chuyển sang ngôn ngữ máy mà chuyển sang một ngôn ngữ trung gian. Một chương trình sẽ có nhiệm vụ đọc chương trình ở ngôn ngữ trung gian này và thực hiện từng câu lệnh. Ngôn ngữ trung gian được gọi là ngôn ngữ của một máy ảo, và chương trình thông dịch thực hiện ngôn ngữ này còn được gọi là máy ảo. Một hệ thống như vậy có thể được mô tả như sơ đồ dưới đây:



Hình 1.2 Hệ thống thông dịch

Một ví dụ rất đặc trưng cho hệ thống thông dịch kiểu này là hệ thông dịch Java. Mã nguồn Java được dịch ra dạng Bytecode. File đích này được một trình thông dịch gọi là máy ảo Java thực hiện. Chính vì vậy mà người ta nói Java có thể chạy trên mọi hệ điều hành có cài máy ảo Java.

Một chương trình thông dịch có kích thước nhỏ trình biên dịch nhưng chạy chậm hơn. Một chương trình biên dịch kết hợp với một chương trình thông dịch tạo thành một chương trình dịch.

Các đặc trưng của một ngôn ngữ lập trình bậc cao

Mục đích chính của môn học này là xây dựng một chương trình dịch cho một ngôn ngữ bậc cao. Vì vậy trong phần này chúng ta nhắc lại những đặc trưng của một ngôn ngữ bậc cao, mà những đặc trưng này sẽ được chúng ta phân tích trong các giai đoạn của một chương trình dịch. Chúng ta sẽ không phân tích mọi đặc điểm trong ngôn ngữ bậc cao như lớp, cấu trúc, đối tượng, … mà chỉ xét những đặc trưng cơ bản nhất để có những tìm hiểu bước đầu cho việc thiết kế và xây dựng một chương trình dịch.

Từ vựng

Cũng như ngôn ngữ tự nhiên, ngôn ngữ lập trình cũng được xây dựng dựa trên bộ từ vựng. Từ vựng trong ngôn ngữ lập trình thường được xây dựng dựa trên bộ chữ gồm có:

+ chữ cái: A .. Z, a . . z

+ chữ số: 0..9

+ các ký hiệu toán học: +, - , *, /, (,), =, <, >, !, %, /

+ các ký hiệu khác: [,], . . .

Các từ vựng được ngôn ngữ hiểu bao gồm các từ khóa, các tên hàm, tên hằng, tên biến, các phép toán, . . .

Các từ vựng này có những qui định nhất định ví dụ như tên thì viết bởi chữ cái đầu tiên và sau đó là không hoặc nhiều chữ cái hoặc chữ số, phép gán trong ngôn ngữ C là = còn trong Pascal là :=, v. v. . .

Để xây dựng một chương trình dịch, hệ thống phải tìm hiểu tập từ vựng của ngôn ngữ nguồn và phân tích để biết được từng loại từ vựng và các thuộc tính của nó, nhiệm vụ này thuộc modun phân tích từ vựng.

Cú pháp

Cú pháp là thành phần quan trọng nhất trong một ngôn ngữ. Như chúng ta đã biết trong ngôn ngữ hình thức thì ngôn ngữ là tập các câu thỏa mãn văn phạm của ngôn ngữ đó. Ví dụ như

câu = chủ ngữ + vị ngữ

vị ngữ = động từ + bổ ngữ

v.v. . .

Trong ngôn ngữ lập trình, cú pháp của nó được thể hiện bởi một bộ luật cú pháp. Bộ luật này dùng để mô tả cấu trúc của chương trình, các câu lệnh. Chúng ta quan tâm đến các cấu trúc này bao gồm:

1) các khai báo
2) biểu thức số học, biểu thức logic
3) các lệnh: lệnh gán, lệnh gọi hàm, lệnh vào ra, . . .
4) câu lệnh điều kiện if
5) câu lệnh lặp: for, while
6) chương trình con (hàm và thủ tục)

Nhiệm vụ trước tiên là phải biết được bộ luật cú pháp của ngôn ngữ mà mình định xây dựng chương trình cho nó.

Chương trình phải phân tích chương trình nguồn thành các cấu trúc cú pháp của ngôn ngữ, từ đó để kiểm tra tính đúng đắn về mặt ngữ pháp của chương trình nguồn. Vấn đề này thuộc công việc của modun phân tích cú pháp.

Một vấn đề nữa là từ các cấu trúc này, phải chuyển thành các cấu trúc tương đương ở ngôn ngữ đích thế nào. Nhiệm vụ này là của phần sinh mã.

Ngữ nghĩa

Kiểm tra ngữ nghĩa của chương trình và một phần của một chương trình dịch. Ngữ nghĩa của một ngôn ngữ lập trình liên quan đến:

+ Kiểu, phạm vi của hằng và biến

+ Phân biệt và sử dụng đúng tên hằng, tên biến, tên hàm

Chương trình dịch phải kiểm tra được tính đúng đắn trong sử dụng các đại lượng này. Ví dụ kiểm tra không cho gán giá trị cho hằng, kiểm tra tính đúng đắn trong gán kiểu, kiểm tra phạm vi, kiểm tra sử dụng tên như tên không được khai báo trùng, dùng cho gọi hàm phải là tên có thuộc tính hàm, . . .

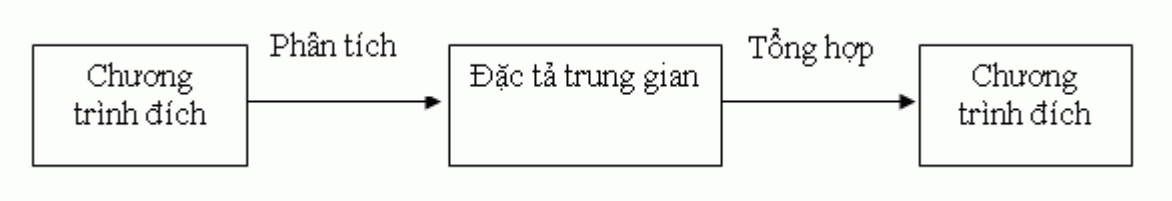
Nhiệm vụ này là của phần phân tích ngữ nghĩa.

4.2.2. Cấu trúc của chương trình dịch

4.2.2.1. Mô hình phân tích - tổng hợp của một trình biên dịch

Chương trình dịch thường bao gồm hai quá trình : phân tích và tổng hợp

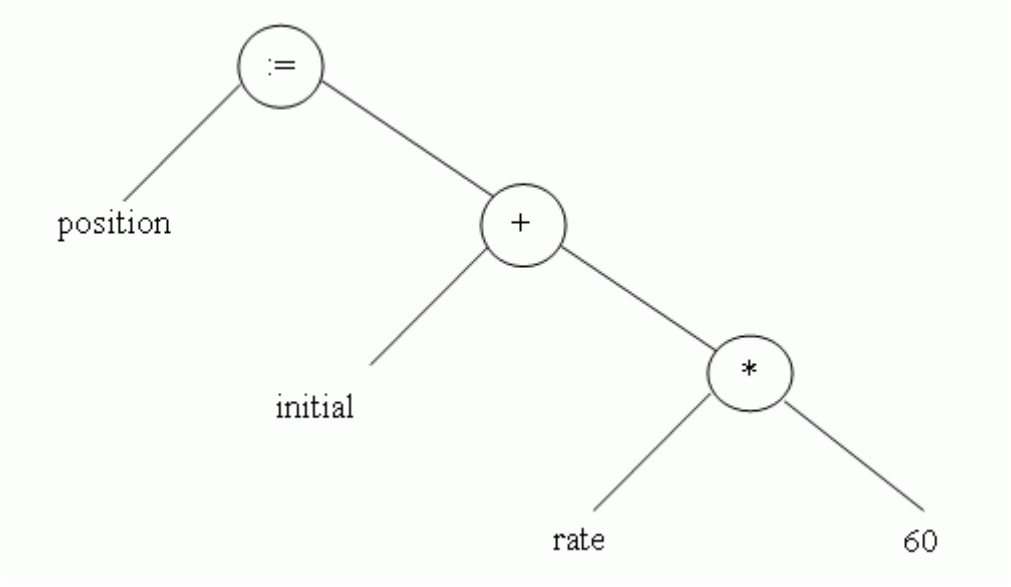
- Phân tích đặc tả trung gian
- Tổng hợp chương trình đích



Hình 1.2 - *Mô hình phân tích - tổng hợp*

Trong quá trình phân tích chương trình nguồn sẽ được phân rã thành một cấu trúc phân cấp, thường là dạng cây - cây cú pháp (syntax tree) mà trong đó có mỗi nút là một toán tử và các nhánh con là các toán hạng.

Ví dụ 1.1: Cây cú pháp cho câu lệnh gán **position := initial + rate * 60**


:=
position
+
initial
*
rate
60

4.2.2.2. Môi trường của trình biên dịch

Ngoài trình biên dịch, chúng ta có thể dùng nhiều chương trình khác nữa để có thể tạo ra một chương trình đích có thể thực thi được (executable). Một chương trình nguồn có thể được phân thành các module và được lưu trong các tập tin riêng rẽ. Công việc tập hợp lại các tập tin này thường được giao cho một chương trình riêng biệt gọi là bộ tiền xử lý (preprocessor). Bộ tiền xử lý có thể "bung" các ký hiệu tắt được gọi là các macro thành các câu lệnh của ngôn ngữ nguồn.

Ngoài ra, chương trình đích được tạo ra bởi trình biên dịch có thể cần phải được xử lý thêm trước khi chúng có thể chạy được. Thông thường, trình biên dịch chỉ tạo ra mã lệnh hợp ngữ (assembly code) để trình dịch hợp ngữ (assembler) dịch thành dạng mã máy rồi được liên kết với một số thủ tục trong thư viện hệ thống thành các mã thực thi được trên máy.

Thông thường một chương trình dịch là một chương trình trong hệ thống liên hoàn giúp cho người lập trình có được một môi trường hoàn chỉnh để phát triển các ứng dụng của họ. Ví dụ như một hệ thống soạn thảo, một hệ thống cho phép tìm lỗi, phần chính là một chương trình dịch sang ngôn ngữ đích cho phép tải và chạy chương trình.

+ bộ soạn thảo chương trình nguồn.

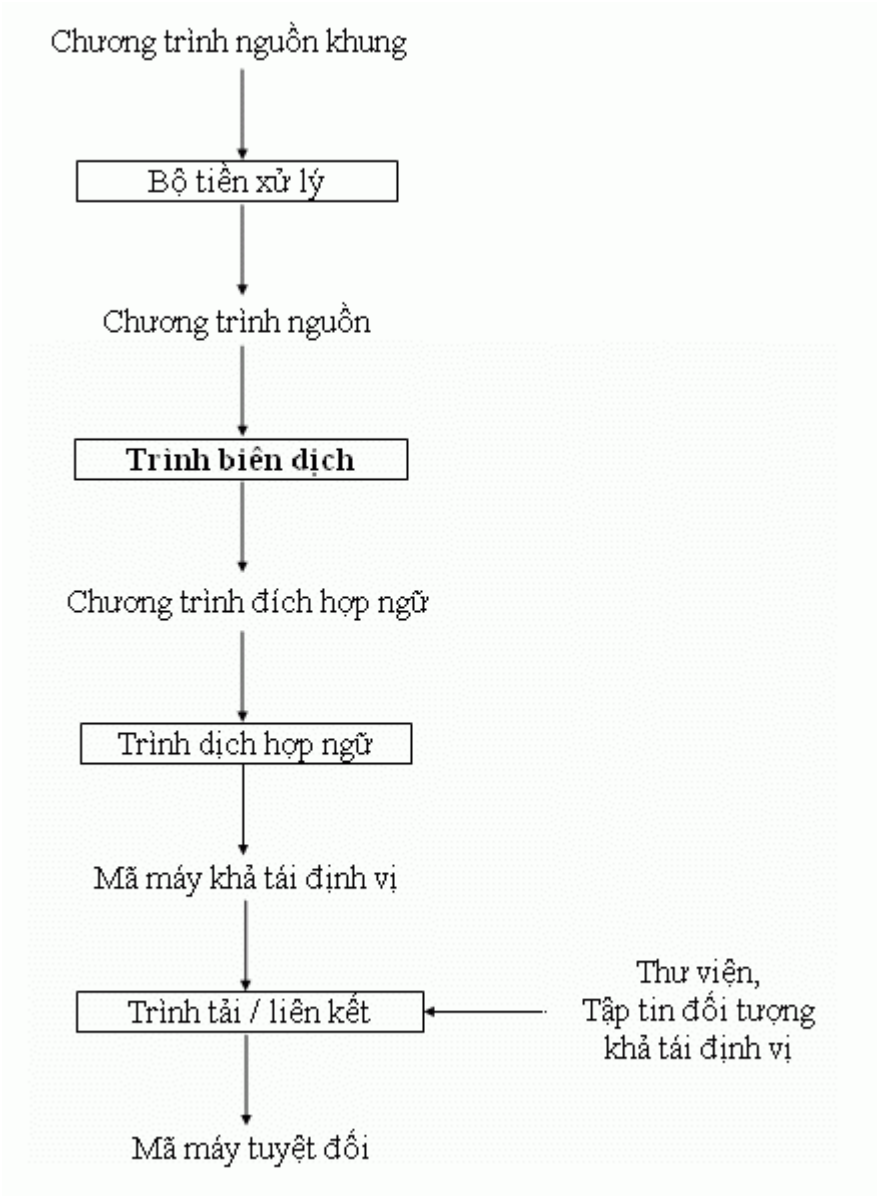
+ tiền xử lý: xử lý một số chức năng ban đầu để tạo một chương trình nguồn hoàn chỉnh, ví dụ như bỏ qua các chú thích; xử lý các macro, kết hợp các tập tin, . . .

+ kiểm lỗi: bao gồm bộ kiểm lỗi chương trình

+ dịch ra ngôn ngữ đích: dịch ra ngôn ngữ đích nhưng đang ở dạng định vị lại được, hay có thể ở dạng ngôn ngữ assembly.

+ tải/liên kết: tải vào bộ nhớ máy để có thể tạo thành một chương trình chạy được trên một cấu trúc máy cụ thể.

Hình sau trình bày một quá trình biên dịch điển hình :



Hình 1.3 - *Một trình xử lý ngôn ngữ điển hình*

4.2.2.3. Sự phân tích chương trình nguồn

Phần này giới thiệu về các quá trình phân tích và cách dùng nó thông qua một số ngôn ngữ định dạng văn bản.

1. Phân tích từ vựng (Lexical Analysis)

Trong một trình biên dịch, giai đoạn phân tích từ vựng sẽ đọc chương trình nguồn từ trái sang phải (quét nguyên liệu - scanning) để tách ra thành các thẻ từ (token).

Ví dụ 1.2:

Quá trình phân tích từ vựng cho câu lệnh gán **position := initial + rate * 60** sẽ tách thành các token như sau**:**

1. Danh biểu position
2. Ký hiệu phép gán :=
3. Danh biểu initial
4. Ký hiệu phép cộng (+)
5. Danh biểu rate
6. Ký hiệu phép nhân (*)
7. Số 60

Trong quá trình phân tích từ vựng các khoảng trắng (blank) sẽ bị bỏ qua.

2. Phân tích cú pháp (Syntax Analysis)

Giai đoạn phân tích cú pháp thực hiện công việc nhóm các thẻ từ của chương trình nguồn thành các ngữ đoạn văn phạm (grammatical phrase), mà sau đó sẽ được trình biên dịch tổng hợp ra thành phẩm. Thông thường, các ngữ đoạn văn phạm này được biểu diễn bằng dạng cây phân tích cú pháp (parse tree) với :

- Ngôn ngữ được đặc tả bởi các luật sinh.
- Phân tích cú pháp dựa vào luật sinh để xây dựng cây phân tích cú pháp.

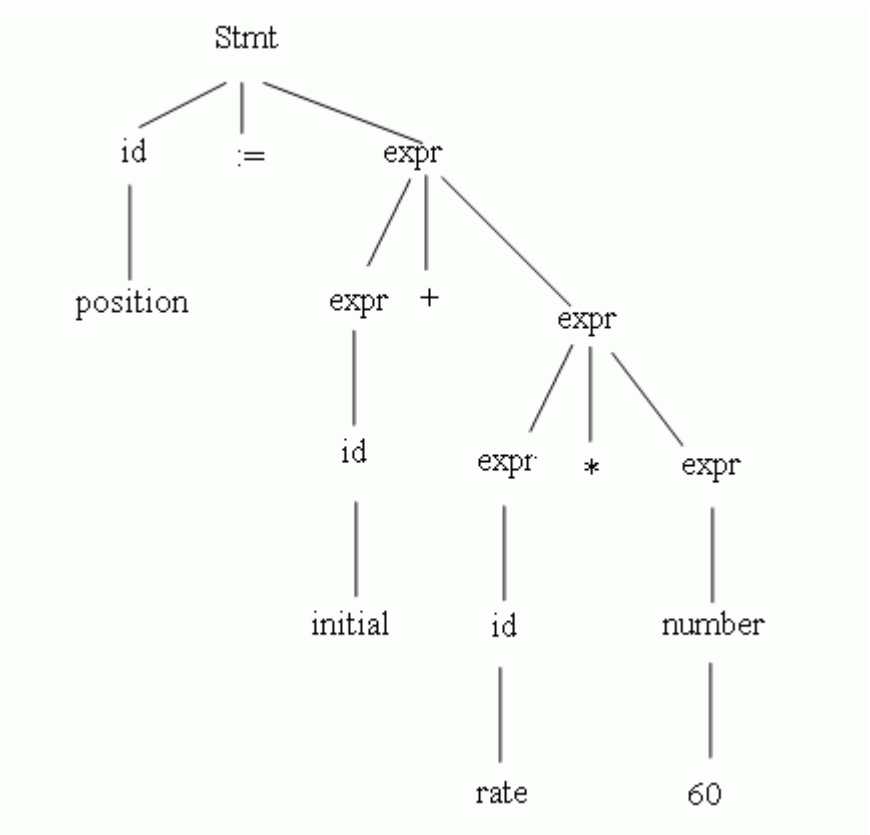
Ví dụ 1.3:

Giả sử ngôn ngữ đặc tả bởi các luật sinh sau :

Stmt id := expr

expr expr + expr | expr ***** expr | id | number

Với câu nhập: position := initial + rate * 60, cây phân tích cú pháp được xây dựng như sau :



Hình 1.4 - *Một cây phân tích cú pháp*

Cấu trúc phân cấp của một chương trình thường được diễn tả bởi quy luật đệ qui.

Ví dụ 1.4:

1) Danh biểu (identifier) là một biểu thức (expr).

2) Số (number) là một biểu thức.

3) Nếu expr1 và expr2 là các biểu thức thì:

expr1 + expr2

expr1 ***** expr2

(expr)

cũng là những biểu thức.

Câu lệnh (statement) cũng có thể định nghĩa đệ qui :

1) Nếu id1 là một danh biểu và expr2 là một biểu thức thì id1 := expr2 là một lệnh (stmt).

2) Nếu expr1 là một biểu thức và stmt2 là một lệnh thì

while (expr1) do stmt2

If (expr1) then stmt2

đều là các lệnh.

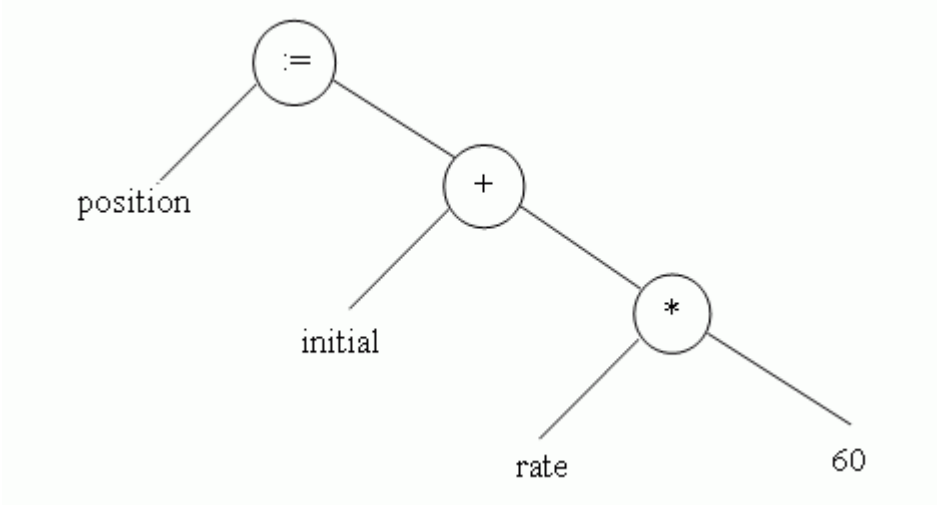
Người ta dùng các qui tắc đệ qui như trên để đặc tả luật sinh (production) cho ngôn ngữ. Sự phân chia giữa quá trình phân tích từ vựng và phân tích cú pháp cũng tuỳ theo công việc thực hiện.

3. Phân tích ngữ nghĩa (Semantic Analysis)

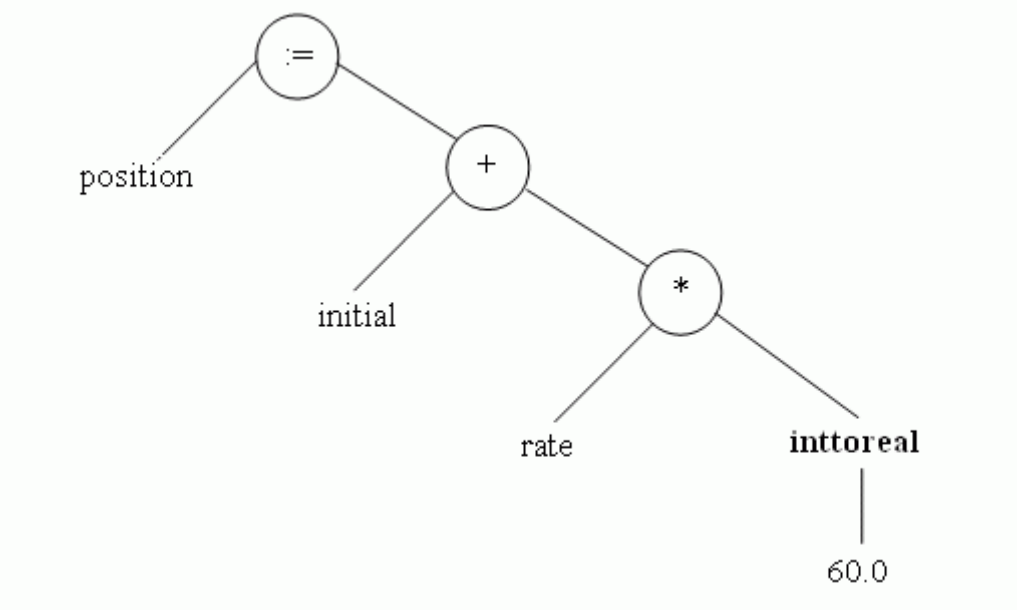
Giai đoạn phân tích ngữ nghĩa sẽ thực hiện việc kiểm tra xem chương trình nguồn có chứa lỗi về ngữ nghĩa hay không và tập hợp thông tin về kiểu cho giai đoạn sinh mã về sau. Một phần quan trọng trong giai đoạn phân tích ngữ nghĩa là kiểm tra kiểu (type checking) và ép chuyển đổi kiểu.

Ví dụ 1.5: Trong biểu thức position := initial + rate * 60

Các danh biểu (tên biến) được khai báo là real, 60 là số integer vì vậy trình biên dịch đổi số nguyên 60 thành số thực 60.0



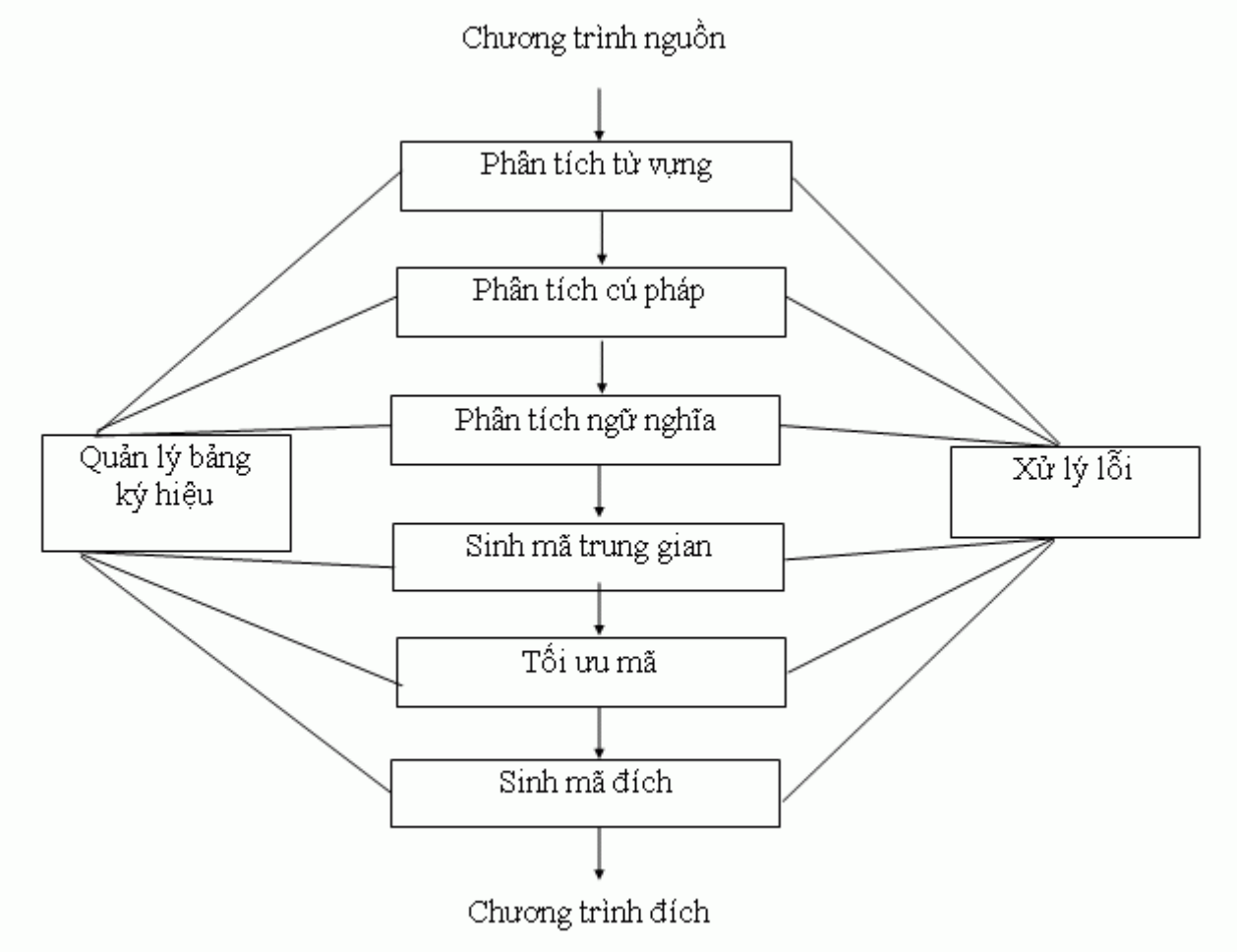
thành



Hình 1.5 - *Chuyển đổi kiểu trên cây phân tích cú pháp*

4.2.3. Cấu trúc tĩnh (cấu trúc logic) của chương trình dịch

Để dễ hình dung, một trình biên dịch được chia thành các giai đoạn, mỗi giai đoạn chuyển chương trình nguồn từ một dạng biểu diễn này sang một dạng biểu diễn khác. Một cách phân rã điển hình trình biên dịch được trình bày trong hình sau.



Hình 1.6 - *Các giai đoạn của một trình biên dịch*

Việc quản lý bảng ký hiệu và xử lý lỗi được thực hiện xuyên suốt qua tất cả các giai đoạn.

1. Quản lý bảng ký hiệu

Một nhiệm vụ quan trọng của trình biên dịch là ghi lại các định danh được sử dụng trong chương trình nguồn và thu thập các thông tin về các thuộc tính khác nhau của mỗi định danh. Những thuộc tính này có thể cung cấp thông tin về vị trí lưu trữ được cấp phát cho một định

danh, kiểu và tầm vực của định danh, và nếu định danh là tên của một thủ tục thì thuộc tính là các thông tin về số lượng và kiểu của các đối số, phương pháp truyền đối số và kiểu trả về của thủ tục nếu có.

Bảng ký hiệu (symbol table) là một cấu trúc dữ liệu mà mỗi phần tử là một mẩu tin dùng để lưu trữ một định danh, bao gồm các trường lưu giữ ký hiệu và các thuộc tính của nó. Cấu trúc này cho phép tìm kiếm, truy xuất danh biểu một cách nhanh chóng.

Trong quá trình phân tích từ vựng, danh biểu được tìm thấy và nó được đưa vào bảng ký hiệu nhưng nói chung các thuộc tính của nó có thể chưa xác định được trong giai đoạn này.

Ví dụ 1.6: Chẳng hạn, một khai báo trong Pascal có dạng

var position, initial, rate : real

thì thuộc tính kiểu real chưa thể xác định khi các danh biểu được xác định và đưa vào bảng ký hiệu. Các giai đoạn sau đó như phân tích ngữ nghĩa và sinh mã trung gian mới đưa thêm các thông tin này vào và sử dụng chúng. Nói chung giai đoạn sinh mã thường đưa các thông tin chi tiết về vị trí lưu trữ dành cho định danh và sẽ sử dụng chúng khi cần thiết.

Bảng ký hiệu

1	position	...
2	initial	...
3	rate	...
4		

2. Xử lý lỗi

Mỗi giai đoạn có thể gặp nhiều lỗi, tuy nhiên sau khi phát hiện ra lỗi, tùy thuộc vào trình biên dịch mà có các cách xử lý lỗi khác nhau, chẳng hạn :

- Dừng và thông báo lỗi khi gặp lỗi đầu tiên (Pascal).
- Ghi nhận lỗi và tiếp tục quá trình dịch (C).

Giai đoạn phân tích từ vựng thường gặp lỗi khi các ký tự không thể ghép thành một token.

Giai đoạn phân tích cú pháp gặp lỗi khi các token không thể kết hợp với nhau theo đúng cấu trúc ngôn ngữ.

Giai đoạn phân tích ngữ nghĩa báo lỗi khi các toán hạng có kiểu không đúng yêu cầu của phép toán hay các kết cấu không có nghĩa đối với thao tác thực hiện mặc dù chúng hoàn toàn đúng về mặt cú pháp.

3. Các giai đoạn phân tích

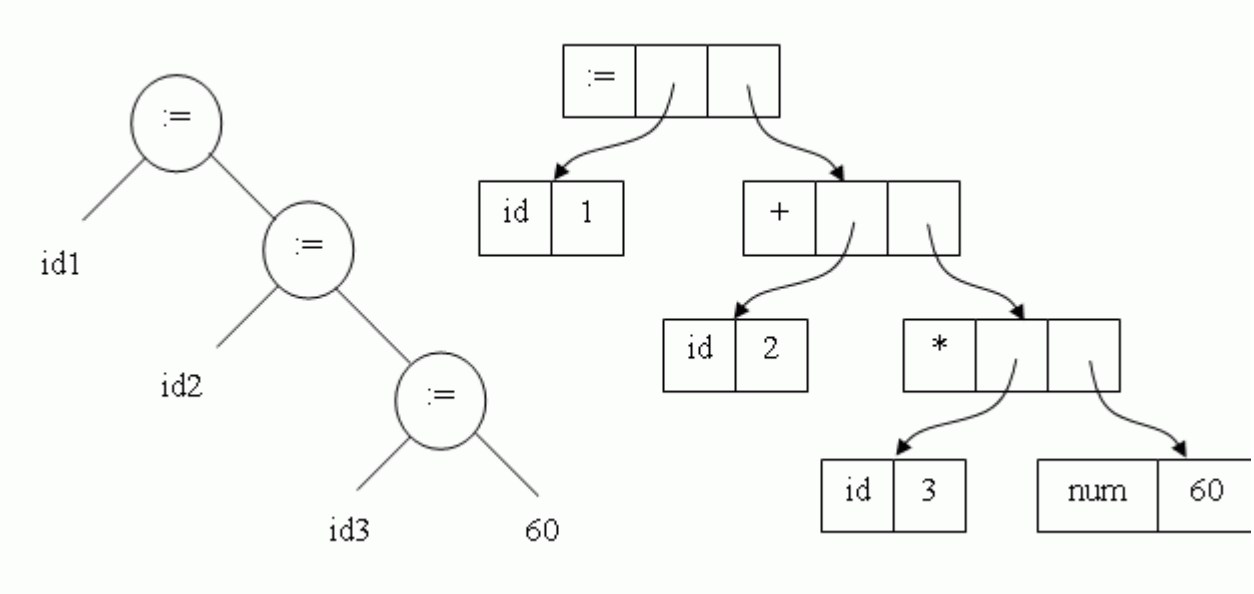
Giai đoạn phân tích từ vựng: Đọc từng ký tự gộp lại thành token, token có thể là một danh biểu, từ khóa, một ký hiệu,...Chuỗi ký tự tạo thành một token gọi là lexeme - trị từ vựng của token đó.

Ví dụ 1.7: Danh biểu rate có token **id**, trị từ vựng là **rate** và danh biểu này sẽ được đưa vào bảng ký hiệu nếu nó chưa có trong đó.

Giai đoạn phân tích cú pháp và phân tích ngữ nghĩa: Xây dựng cấu trúc phân cấp cho chuỗi các token, biểu diễn bởi cây cú pháp và kiểm tra ngôn ngữ theo cú pháp.

Ví dụ 1.8: Cây cú pháp và cấu trúc lưu trữ cho biểu thức

position := initial + rate ***** 60



Hình 1.7 - *Cây cú pháp và cấu trúc lưu trữ*

4. Sinh mã trung gian

Sau khi phân tích ngữ nghĩa, một số trình biên dịch sẽ tạo ra một dạng biểu diễn trung gian của chương trình nguồn. Chúng ta có thể xem dạng biểu diễn này như một chương trình dành cho một máy trừu tượng. Chúng có 2 đặc tính quan trọng : dễ sinh và dễ dịch thành chương trình đích.

Dạng biểu diễn trung gian có rất nhiều loại. Thông thường, người ta sử dụng dạng "mã máy 3 địa chỉ" (three-address code), tương tự như dạng hợp ngữ cho một máy mà trong đó mỗi vị trí bộ nhớ có thể đóng vai trò như một thanh ghi.

Mã máy 3 địa chỉ là một dãy các lệnh liên tiếp, mỗi lệnh có thể có tối đa 3 đối số.

Ví dụ 1.9:

```
t1 := inttoreal (60)
t2 := id3 * t1
t3 := id2 + t2
id1 := t3
```

Dạng trung gian này có một số tính chất:

- Mỗi lệnh chỉ chứa nhiều nhất một toán tử. Do đó khi tạo ra lệnh này, trình biên dịch phải xác định thứ tự các phép toán, ví dụ * thực hiện trước +.
- Trình biên dịch phải tạo ra một biến tạm để lưu trữ giá trị tính toán cho mỗi lệnh.
- Một số lệnh có ít hơn 3 toán hạng.

5. Tối ưu mã

Giai đoạn tối ưu mã cố gắng cải thiện mã trung gian để có thể có mã máy thực hiện nhanh hơn. Một số phương pháp tối ưu hóa hoàn toàn bình thường.

Ví dụ 1.10:

Mã trung gian nêu trên có thể tối ưu thành:

```
t1 := id3 * 60.0
```

id1 := id2 + t1

Để tối ưu mã, ta thấy việc đổi số nguyên 60 thành số thực 60.0 có thể thực hiện một lần vào lúc biên dịch, vì vậy có thể loại bỏ phép toán inttoreal. Ngoài ra, t3 chỉ được dùng một lần để chuyển giá trị cho id1 nên có thể giảm bớt.

Có một khác biệt rất lớn giữa khối lượng tối ưu hoá mã được các trình biên dịch khác nhau thực hiện. Trong những trình biên dịch gọi là "trình biên dịch chuyên tối ưu", một phần thời gian đáng kể được dành cho giai đoạn này. Tuy nhiên, cũng có những phương pháp tối ưu giúp giảm đáng kể thời gian chạy của chương trình nguồn mà không làm chậm đi thời gian dịch quá nhiều.

6. Sinh mã

Giai đoạn cuối cùng của biên dịch là sinh mã đích, thường là mã máy hoặc mã hợp ngữ. Các vị trí vùng nhớ được chọn lựa cho mỗi biến được chương trình sử dụng. Sau đó, các chỉ thị trung gian được dịch lần lượt thành chuỗi các chỉ thị mã máy. Vấn đề quyết định là việc gán các biến cho các thanh ghi.

Ví dụ 1.11:

Sử dụng các thanh ghi (chẳng hạn R1, R2) cho việc sinh mã đích như sau:

```
MOVF    id3, R2
MULF    #60.0, R2
MOVF    id2, R1
ADDF    R2, R1
MOVF    R1, id1
```

Toán hạng thứ nhất và thứ hai của mỗi chỉ thị tương ứng mô tả đốïi tượng nguồn và đích. Chữ F trong mỗi chỉ thị cho biết chỉ thị đang xử lý các số chấm động (floating_point). Dấu # để xác định số 60.0 xem như một hằng số.

Tóm lại quá trình thực hiện của một chương trình biên dịch như sau:

(1) Phân tích từ vựng *(Lexical Analysis)*

(2) Phân tích cú pháp *(Syntatic Analysis)*

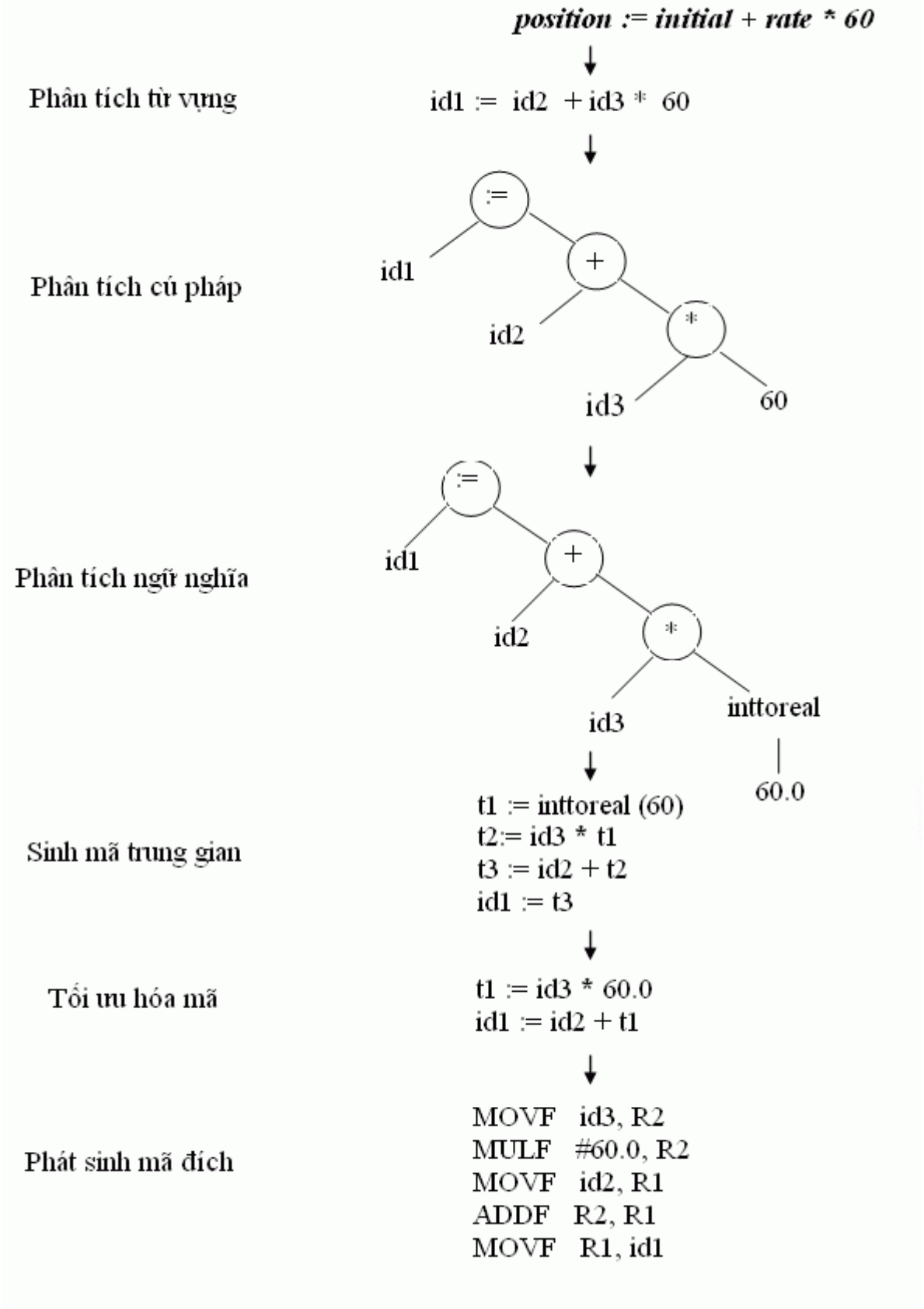
(3) Phân tích ngữ nghĩa *(Semantic Analysis)*

(4) Sinh mã trung gian *(Intermediate code generation)*

(5) Tối ưu mã *(Code Optimization)*

(6) Sinh mã đích *(Code generation)*

(7) Quản lý bảng ký hiệu

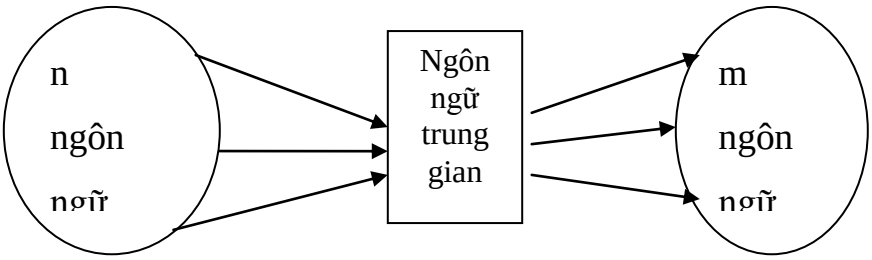


Hình 1.8 - *Minh họa các giai đoạn biên dịch một biểu thức*

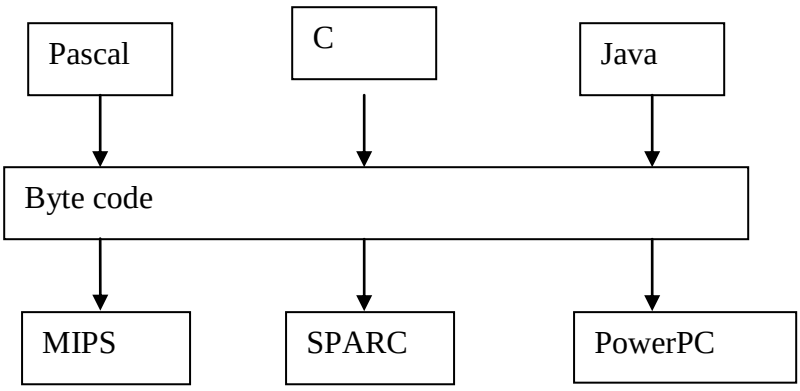
4.2.4. Cấu trúc động (cấu trúc theo thời gian) của chương trình dịch

Các giai đoạn mà chúng ta đề cập ở trên là thực hiện theo trình tự logic của một trình biên dịch. Nhưng trong thực tế, cài đặt các hoạt động của nhiều hơn một giai đoạn có thể được nhóm lại với nhau. Thông thường chúng được nhóm thành hai nhóm cơ bản, gọi là: kỳ đầu (Front end) và kỳ sau (Back end).

Kỳ đầu gồm các giai đoạn: phân tích từ vựng, phân tích cú pháp, phân tích ngữ nghĩa và sinh mã trung gian. Kỳ sau gồm các giai đoạn tối ưu mã trung gian và sinh mã đích. Bằng thiết kế này, đối với các ngôn ngữ nguồn, chúng ta chỉ cần quan tâm đến việc sinh ra mã trung gian mà không cần biết mã máy đích của nó là gì. Điều này làm cho công việc đơn giản, không phụ thuộc vào máy đích. Còn giai đoạn sau thì cũng trở nên đơn giản hơn vì ngôn ngữ trung gian thường thì gần với mã máy. Và nó còn thể hiện ưu điểm khi chúng ta xây dựng nhiều cặp ngôn ngữ. Ví dụ có n ngôn ngữ nguồn, muốn xây dựng chương trình dịch cho n ngôn ngữ này sang m ngôn ngữ đích thì chúng ta cần *n*m* chương trình dịch; còn nếu chúng ta xây dựng theo kiến trúc *front end* và *back end* thì chúng ta chỉ cần *n+m* chương trình dịch.



Ví dụ:



1. Kỳ đầu (Front end)

Kỳ đầu bao gồm các giai đoạn hoặc các phần giai đoạn phụ thuộc nhiều vào ngôn ngữ nguồn và hầu như độc lập với máy đích. Thông thường, nó chứa các giai đoạn sau: Phân tích từ vựng, Phân tích cú pháp, Phân tích ngữ nghĩa và Sinh mã trung gian. Một phần của công việc tối ưu hóa mã cũng được thực hiện ở kỳ đầu.

Front end cũng bao gồm cả việc xử lý lỗi xuất hiện trong từng giai đoạn.

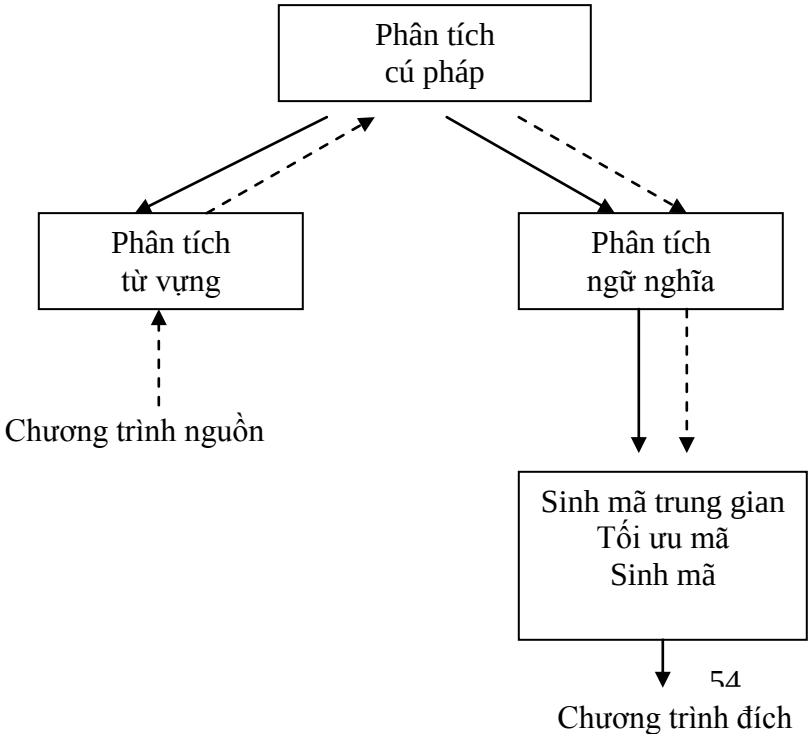
2. Kỳ sau (Back end)

Kỳ sau bao gồm một số phần nào đó của trình biên dịch phụ thuộc vào máy đích và nói chung các phần này không phụ thuộc vào ngôn ngữ nguồn mà là ngôn ngữ trung gian. Trong kỳ sau, chúng ta gặp một số vấn đề tối ưu hoá mã, phát sinh mã đích cùng với việc xử lý lỗi và các thao tác trên bảng ký hiệu.

3. Thiết kế duyệt một lượt và nhiều lượt

Cấu trúc động của chương trình dịch dịch (hay cấu trúc theo thời gian) cho biết quan hệ giữa các phần khi nó hoạt động. Các giai đoạn của chương trình dịch (phân tích từ vựng, phân tích cú pháp, phân tích ngữ nghĩa, tối ưu, sinh mã) có thể hoạt động theo hai cách: lần lượt hay đồng thời.

Một số giai đoạn biên dịch thường được cài đặt bằng một lượt (pass) duy nhất bao gồm việc đọc một file dữ liệu vào, rồi phân tích và cho kết quả ra một file đích. Người ta hay nhóm nhiều giai đoạn vào một lượt và hoạt động của các giai đoạn này đan xen lẫn nhau. Ví dụ như các giai đoạn phân tích từ vựng, phân tích cú pháp, phân tích ngữ nghĩa và sinh mã trung gian có thể được nhóm lại thành một lượt. Khi đó dòng từ tố sau giai đoạn phân tích có thể được dịch trực tiếp thành mã trung gian.



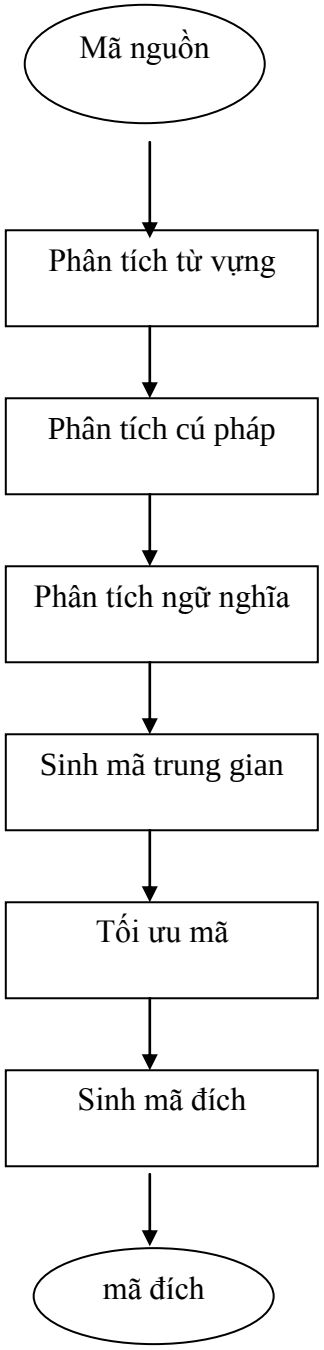
Ở đây chúng ta xem xét hai thiết kế của một chương trình.

Thứ nhất là **thiết kế duyệt một lượt.** Trong thiết kế này một số thành phần của chương trình được thực hiện đồng thời. Bộ phân tích cú pháp đóng vai trò trung tâm, nó sẽ gọi bộ phân tích từ vựng khi nó

cần một từ tố tiếp theo và nó gọi bộ phân tích ngữ nghĩa khi nó muốn chuyển cho một cấu trúc cú pháp đã được phân tích. Bộ phân tích ngữ nghĩa lại đưa cấu trúc sang phần sinh mã trung gian để sinh ra các mã trong một ngôn ngữ trung gian rồi đưa vào bộ tối ưu và sinh mã.

Trong cấu trúc duyệt nhiều lượt, các thành phần trong chương trình được thực hiện lần lượt và độc lập với nhau. Qua mỗi một phần, kết quả sẽ được lưu lại và làm đầu vào cho bước tiếp theo.

Sau đây là Sơ đồ thiết kế duyệt nhiều lượt



Người ta chỉ muốn có một số ít lượt bởi vì mỗi lượt đều mất thời gian đọc và ghi ra tập tin trung gian. Ngược lại nếu chúng ta gom quá nhiều giai đoạn vào trong một lượt thì có thể sẽ phải duy trì toàn bộ chương trình trong bộ nhớ, bởi vì một giai đoạn có thể cần thông tin theo một thứ tự khác với thứ tự nó được tạo ra. Dạng biểu diễn trung gian của chương trình có thể lớn hơn nhiều so với chương trình nguồn hoặc chương trình đích, vì thế sẽ gặp vấn đề về bộ nhớ lưu trữ.

Đối với một số giai đoạn, nhóm chúng vào một lượt làm nảy sinh một số vấn đề. Chẳng hạn các ngôn ngữ như PL/1, Algol 68 hay Foxpro cho phép các biến được dùng trước khi khai báo. Chúng ta không thể tạo ra mã đích cho một kết cấu nếu không biết được kiểu các biến có mặt trong kết cấu đó. Tương tự phần lớn các ngôn ngữ lập trình đều cho phép dùng lệnh goto với một nhãn khai báo sau. Chúng ta không thể xác định được địa chỉ đích của một lệnh nhảy như thế cho đến khi chúng ta thấy được mã nguồn ở trong đoạn đó và mã đích được sinh ra. Trong những trường hợp như thế này, chúng ta có thể dùng kỹ thuật điền sau (backpatching): dành một chỗ trống cho các thông tin đang thiếu, và điền vào khoảng này khi có được thông tin đó.

Nếu so sánh giữa hai thiết kế này thì thiết kế duyệt nhiều lượt đơn giản hơn về mặt logic thực hiện vì cứ thực hiện hết giai đoạn này lại đến giai đoạn khác. Tuy nhiên chương trình sẽ chạy chậm hơn nhiều lần vì phải truy xuất lại kết quả của các giai đoạn trước từ thiết bị lưu trữ ngoài.

Trong giáo trình này chúng ta nghiên cứu các giai đoạn của một

chương trình dịch một cách riêng rẽ nhưng theo thiết kế duyệt một lượt.

4. Giới thiệu ngôn ngữ PL/0

Ngôn ngữ PL/0 là một ngôn ngữ lập trình nhỏ, các câu lệnh của nó tựa như ngôn ngữ lập trình Pascal. Nó thường được sử dụng để minh họa cách xây dựng một chương trình dịch. Mặc dù rất đơn giản, nhưng ngôn ngữ PL/0 chứa những nét điển hình của ngôn ngữ bậc cao, đơn giản và thích hợp việc tìm hiểu một ngôn ngữ lập trình bậc cao.

Về kiểu dữ liệu thì PL/0 chỉ có kiểu dữ liệu nguyên

Về các câu lệnh thì PL/0 chứa hầu hết các câu lệnh cơ bản: câu lệnh gán, câu lệnh điều kiện IF, câu lệnh lặp WHILE, các toán tử số học. Nó không có câu lệnh vào/ra.

Ngôn ngữ PL/0 Có cấu trúc khối, thể hiện khá đầy đủ các khái niệm về định nghĩa chương trình con. Nó cũng cho phép xây dựng một chương trình con đệ qui.

Ví dụ về một chương trình viết trong ngôn ngữ PL/0:

```
Const m:=7, n:=82;
Var x, y, z, q, r;
Procedure Multiply;
Var a, b;
Begin
    a := b; b := y; z := 0;
    while b>0 do
    begin
    if b=a then z := z+a;
    a := 2*a; b := b/2;
    end;
End;
```

```
Procedure Divide;
Var w;
Begin
    r := x; q := 0; w := y;
    while w<=r do w := 2*w;
    while w>y do
    begin
            q := 2*q; w := w/2;
            if w<=z then
            begin
                    r:=r-w; q:=q+1;
            end;
    end;
End;
Begin
    x:=m; y:=n; call multiply;
    x:=25; y:=0; call Divide;
End.
```

Văn phạm PL/0 được cho dưới dạng các luật sản xuất như sau:

program -> block .

block -> { CONST idetifier := number **(** , identifier := number **)*** ; }

{ VAR identifier **(** , identifier **)** * ; }

{ **(** PROCEDURE identifier ; block ; **)** * }

statement

statement -> identifier := expression

| CALL identifier

| BEGIN statement **(** ; statement **)** * END

| IF condition THEN statement

| WHILE condition DO statement

| ε

expression -> fragment **((** + | - | * | / **)** fragment **)***

fragment -> identifier

| number

| **(** + | - **)** fragment

| (expression)

condition -> ODD expression

| expression **(** = | <> | < | <= | > | >= **)** expression

Ngôn ngữ PL/0 sẽ được sử dụng để thực hành minh họa các bước xây dựng một chương trình dịch hoàn chỉnh trong tài liệu này . Việc này sẽ giúp chúng có được sự tìm hiểu cụ thể và sâu sắc hơn về việc xây dựng chương trình dịch cho một ngôn ngữ hoàn chính.

Nhiệm vụ học chương trình dịch

+ xây dựng bộ phân tích từ vựng
+ xây dựng bộ phân tích cú pháp
+ xây dựng bộ phân tích ngữ nghĩa
+ xây dựng bộ sinh mã trung gian
+ xây dựng bộ sinh mã máy ảo
+ chương trình thông dịch chạy máy ảo
+ chương trình quản lý bảng ký hiệu

Chúng ta sẽ xây dựng các phần việc này trên một ngôn ngữ cụ thể là ngôn ngữ PL/0.

Đọc thêm Các thế hệ ngôn ngữ lập trình

Các chương trình dịch đầu tiên xuất hiện vào những năm đầu thập kỷ 50. Khó có thể chỉ ra thời điểm chính xác của sự kiện đó vì đã có vài nhóm độc lập với nhau cùng nghiên cứu và thực hiện công việc này.

Các thế hệ đầu tiên

Trước khi máy tính có thể thực hiện một nhiệm vụ, nó cần phải được lập trình để hoạt động bằng cách đặt các thuật toán, biểu thức trong ngôn ngữ máy vào bộ nhớ chính. Nguồn gốc của việc có quá trình lập trình này là do có các nhu cầu đa dạng người lập trình mong muốn diễn đạt được tất cả các thuật toán bằng ngôn ngữ máy. Phương pháp này cần phải được cải tiến để ngoài nhiệm vụ sẵn sàng cho thiết kế thuật toán còn giúp người lập trình tránh hay phát hiện và định vị được các lỗi và giúp chữa lại trước khi công việc được hoàn thành.

Bước đầu tiên nhằm loại bỏ các khó khăn này từ quá trình lập trình là vứt bỏ việc dùng các con số buồn tẻ và dễ gây lỗi dùng để biểu diễn các mã phép toán và các toán tử có trong ngôn ngữ máy. Chỉ đơn giản bằng cách chọn các tên mô tả cho các ô bộ nhớ và các thanh ghi để biểu diễn các mã phép toán, người lập trình có thể tăng được rất nhiều tính đọc được của một chuỗi các lệnh. Ví dụ, chúng ta hãy xem một thủ tục viết bằng mã máy có nhiệm vụ cộng nội dung của các ô nhớ 6C và 6D lại với nhau và đặt kết quả vào ô 6E. Các lệnh thực hiện công việc này viết trong mã 16 như sau:

```
156C
166D
5056
306E
C000
```

Nếu bây giờ chúng ta gán một cái tên PRICE (giá tiền) cho vị trí 6C, TAX (thuế) cho 6D và TOTAL (tổng số) cho 6E, và chúng ta có thể biểu diễn cùng thủ tục này như dưới đây sử dụng kỹ thuật gọi là *đặt ký hiệu gợi nhớ;*

```
LD      R5, PRICE
LD      R6, TAX
ADDI        R0, R5 R6
ST      R0, TOTAL
HLT
```

Đa số chúng ta sẽ công nhận là dạng thứ hai dù vẫn còn khó, thì việc thực hiện công việc biểu diễn mục đích và ý nghĩa của thủ tục tốt hơn nhiều dạng đầu.

Khi kỹ thuật này lần đầu tiên được công bố, người lập trình dùng bộ ký hiệu này để thiết kế chương trình gốc trên giấy và sau đó sẽ dịch nó ra dạng mã máy. Việc này cũng không mất nhiều thời gian lắm do việc chuyển đổi thực hiện tương đối máy móc. Hệ quả là việc dùng các ký hiệu này dẫn đến việc hình thức hóa nó thành một ngôn ngữ lập trình gọi là ngôn ngữ Assembly, và một chương trình gọi là Assembler dùng để dịch tự động các chương trình khác viết trong ngôn ngữ Assembly thành ngôn ngữ máy tương ứng. Chương trình này được gọi là Assembler (trình dịch hợp ngữ) do nhiệm vụ của nó là tổng hợp thành các lệnh máy bằng cách dịch các ký hiệu gợi nhớ và các tên.

Ngày nay Assembler trở thành một chương trình tiện ích thông thường trong hầu hết các máy tính. Với những hệ thống như vậy, người lập trình có thể gõ một chương trình trong dạng ký hiệu gợi nhớ nhờ các bộ soạn thảo của hệ thống, rồi yêu cầu hệ thống dùng Assembler để dịch chương trình đó và lưu thành tệp mà sau đó có thể dùng để chạy được.

Như vậy các ngôn ngữ Assembly đã được phát triển đầu tiên, chúng xuất hiện như là một bước tiến khổng lồ trên con đường tìm kiếm các môi trường lập trình tốt hơn. Trong thực tế, có nhiều nghiên cứu về chúng để biểu diễn một ngôn ngữ lập trình mới và toàn diện. Do đó, các ngôn ngữ Assembly được coi là các ngôn ngữ thế hệ thứ hai, còn ngôn ngữ thế hệ đầu tiên chính là bản thân các ngôn ngữ máy.

Mặc dù ngôn ngữ thế hệ thứ hai này có rất nhiều ưu điểm so với ngôn ngữ máy, chúng vẫn còn quá vắn tắt. Ngôn ngữ Assembly về nguyên tắc cũng giống như ngôn ngữ máy tương ứng. Sự khác nhau đơn giản chí là cú pháp dùng để biểu diễn chúng. Sự giống nhau giữa ngôn ngữ assembly và ngôn ngữ máy còn dẫn đến việc ngôn ngữ Assembly phụ thuộc vào từng loại máy cụ thể. Các lệnh dùng trong chương trình chỉ là biểu diễn các thuộc tính của máy. Mặt khác, một chương trình viết trong ngôn ngữ assembly không dễ chuyển sang một loại máy khác và thường phải viết lại cho thích ứng với các thanh ghi và tập lệnh của máy mới.

Một nhược điểm nữa của ngôn ngữ assembly là một người lập trình, mặc dù không buộc phải mã các lệnh ở dạng từng bit, thì thường bị bắt phải nghĩ đến các chi tiết, các thành phần nhỏ này, không được tập trung vào việc tìm giải pháp tốt hơn. Tình cảnh này cũng giống như thiết kể một ngôi nhà mà phải chú ý đến xi măng, vôi vữa, gạch ngói, đinh, đá… Tuy quá trình thiết kế một ngôi nhà từ những thành phần nhỏ như thế cũng thực hiện được, nhưng việc thiết kế sẽ đơn giản đi rất nhiều nếu chúng ta suy nghĩ bắt đầu từ các phòng, nền, cửa sổ, mái….

Như vậy nguyên lý thiết kế dựa trên các phần tử nhỏ đi lên không phải là nguyên lý thích hợp trong việc thiết kế. Quá trình thiết kể tốt hơn là dùng các nguyên lý ở mức cao hơn, mỗi nguyên lý này biểu diễn một khái niệm liên quan với các thuộc tính chính của sản phẩm. Mỗi khi một thiết kế được hoàn tất, các thiết kế gốc có thể được dịch sang các khái niệm ở mức thấp hơn, liên quan đến việc thực hiện, giống một nhà xây dựng cuối cùng sẽ chuyển thiết kế trên giấy sang chi tiết các vật liệu xây dựng.

Theo triết lý này, các nhà khoa học máy tính bắt đầu phát triển các ngôn ngữ lập trình tốt hơn cho việc viết phần mềm so với các ngôn ngữ lập trình bậc thấp assembly. Kết quả là *các ngôn ngữ thế hệ thứ ba đã ra đời, khác với các thế hệ trước ở chỗ chúng vừa là ngôn ngữ ở mức cao, lại vừa độc lập với máy.*

Nói chung, phương pháp của các ngôn ngữ lập trình thế hệ thứ ba này là nhận biết bộ các nguyên lý bậc cao cho việc phát triển phần mềm. Mỗi một nguyên lý này được thiết kế sao cho nó có thể thực hiện như là một chuỗi các nguyên lý thấp có trong ngôn ngữ máy. Ví dụ, câu lệnh:

assign Total the value Price plus Tax

hoặc Total: =Price + Tax

cho thấy một hành động ở mức cao không cần quan tâm đến việc một máy tính cụ thể phải thực hiện nó như thế nào.

Các phát triển gần đây

Nói chung, thuật ngữ *ngôn ngữ thế hệ thứ tư* được dùng trong một số sản phẩm phần mềm mà có *cho phép người dùng tự biến đổi (tùy biến) phần mềm của họ mà không cần có chuyên môn*. Người lập trình trong các ngôn ngữ như vậy thường được yêu cầu chọn từ những gì hiện trên màn hình ở dạng câu hoặc biểu tượng. Những sản phẩm phần mềm bao gồm cả bảng tính giúp duy trì các bảng dữ liệu ở dạng các bản ghi kế toán, hệ CSDI giúp duy trì và lấy lại thông tin, các phần mềm đồ họa giúp phát triển các đồ họa, đồ thị, ảnh… các bộ xử lý văn bản mạnh cho phép các tài liệu có thể kết hợp, sắp xếp lại, và đổi dạng. Thêm nữa, các phần mềm này nhiều khi lại được bó lại tạo nên các hệ thống tổng thể. Với những hệ thống như vậy, một nhà kinh tế có thể kiến trúc nên và biến đổi các mô hình kinh tế, phân tích những thay đổi ảnh hưởng khác nhau có thể có trong nền kinh tế nói chung hoặc trong một lĩnh vực kinh doanh cụ thể nào đó, và đưa ra kết quả ở dạng một tài liệu viết với các hình đồ họa, lược đồ làm các phương tiện trợ giúp trực quan. Một người quản lý doanh nghiệp nhỏ có thể tùy biến cùng sản phẩm này để phát triển một hệ thống cho việc duy trì kho và tìm ra các ảnh hưởng của các mục lưu chuyển thấp nào đó… Rõ ràng là

nếu ta phải viết các chương trình làm tất cả các tùy biến này thì đó đều là những chương trình lớn.

Những hệ thống này *được coi là thay thế cho các ngôn ngữ lập trình* do môi trường lập trình của chúng gần gũi với ứng dụng hơn các môi trường mà các ngôn ngữ thế hệ thứ ba cung cấp. Ví dụ, thay cho việc mô tả các thông tin được biểu diễn trong máy như thế nào, một bảng dữ liệu có thể hiển thị trên màn hình máy tính ra làm sao, hoặc cả hệ thống được cập nhật thông tin như thế nào, thì người lập trình dùng các phần mềm thế hệ thứ tư chỉ cần mô tả các mục dữ liệu sẽ xuất hiện trên bảng tính và chúng quan hệ với nhau như thế nào. Do vậy, người dùng có thể tùy biến và dùng bảng tính mà không cần quan tâm (hoặc hiểu) về các chi tiết liên quan đến các kỹ thuật đang được sử dụng.

Thuật ngữ *ngôn ngữ thế hệ thứ năm được dùng cho khái niệm lập trình mô tả,* với việc nhấn mạnh phương pháp đặc biệt được biết như lập trình logic. Tư tưởng này thông minh hơn các tư tưởng trước đây. Đến giờ chúng ta sẽ chú ý rằng tư tưởng lập trình khai báo cho phép người dùng máy tính giải các bài tóan mà chỉ cần quan tâm đến đó là bài toán gì chứ không phải là nó được giải như thế nào. Quan điểm này có thể là thái quá đối với bạn. Tại sao chúng ta lại hy vọng giải được một bài toán mà không cần phải quan tâm đến cách giải nó như thế nào. Câu trả lời là chúng ta không giải bài toán này mà để máy tính giải hộ. Với phương pháp này, nhiệm vụ của chúng ta đơn thuần chỉ là khai báo về bài toán, trong khi đó máy tính phải thực hiện nhiệm vụ tìm lời giải cho nó.

Từ tư tưởng của thế hệ ngôn ngữ lập trình thứ năm, ta thấy nếu vẫn cứ được phát triển lên như vây, thì cho đến một lúc nào đó máy tính sẽ hiểu được trực tiếp ngôn ngữ tự nhiên của con người.

Truyền thông giữa người và máy bằng ngôn ngữ máy, trong đó con người phải mô tả chi tiết mỗi bước lời giải	Truyền thông giữa người và máy bằng ngôn ngữ tự nhiên của con người, trong đó máy sẽ tự sinh ra toàn bộ lời giải

4.2.5. Vị trí của chương trình dịch trong hệ thống dịch

Trong thực tế, chương trình dịch thường được dùng trong một hệ thống liên hoàn nhiều chức năng, tạo ra một môi trường chương trình dịch hoàn chỉnh.

4.3. Sự cần thiết phải nghiên cứu chương trình dịch

Việc nghiên cứu chương trình dịch sẽ giúp chúng ta:

- Nắm vững các nguyên lý ngôn ngữ lập trình và công cụ quan trọng nhất của các nhà tin học, đó là chương trình dịch. Trên cơ sở đó hiểu sâu được từng ngôn ngữ lập trình, nắm được điểm mạnh, điểm yếu của từng ngôn ngữ, từ đó chọn được ngôn ngữ phù hợp với đề án của mình.
- Biết lựa chọn chương trình dịch thích hợp của cùng một ngôn ngữ lập trình.
- Hiểu rx các lựa chọn trong các chương trình dịch, từ đó tùy chọn tối ưu cho công việc cần thực hiện
- Nâng cao trình độ tay nghề, cải thiện kỹ năng và hiểu biết về lập trình.
- Vận dụng có hiệu quả vào các công việc cụ thể, có thể tụ mình xây dựng được một chương trình dịch theo yêu cầu.

Dùng các kiến thức của môn chương trình dịch vào các ứng dụng khác.

- Áp dụng các kiến thức đã học về chương trình dịch vào các ngành nghề khác như xử lý ngôn ngữ tự nhiên.

4.4. Bộ phân tích cú pháp.

Phân tích cú pháp tách chương trình nguồn thành các phần theo văn phạm và biểu diễn cấu trúc này bằng một cây (gọi là cây phân tích), hoặc theo một cấu trúc tương đương với cây. Đây là bước quan trọng nhất của toàn bộ quá trình dịch.

BÀI TẬP

1. Thế nào là một chương trình dịch
2. So sánh hệ thống biên dịch và thông dịch
3. So sánh thiết kế duyệt một lượt và nhiều lượt
4. Ưu điểm của kiến trúc kỳ trước và kỳ sau
5. Tìm hiểu ngôn ngữ HTML về từ tố và cú pháp

MỘT SỐ ĐỀ THI MẪU

Đề số 1.

Bài 1: Cho bảng chữ cái ∑={a,b,c}

L1={x∈∑*| x chứa từ con bc và kết thúc bởi lẻ ký hiệu a}

a) Xây dựng nguồn sinh ngôn ngữ L1.

b) Xây dựng văn phạm tương đương với nguồn

Bài 2: Cho bảng chữ cái ∑={k,m,n}

L2={x∈∑*| x chứa từ con kmn}

a) Xây dựng văn phạm sinh ngôn ngữ L2.

b) Xây dựng Otomat hữu hạn đơn định tương đương với nguồn.

c) Xây dựng văn phạm sinh ngôn ngữ soi gương với ngôn ngữ L2.

Đề số 2.

Bài 1: Cho bảng chữ cái ∑={a,b,c}

L1={x∈∑*| x bắt đầu bởi lẻ b và chứa từ con ca}

a) Xây dựng nguồn sinh ngôn ngữ L1.

b) Xây dựng Otomat đơn định, đầy đủ tương đương với nguồn

Bài 2: Cho bảng chữ cái ∑={m,n,p}

L2={x∈∑*| x chứa từ con mn}

a) Xây dựng văn phạm G sinh ngôn ngữ L2.

b) Chuyển văn phạm G về dạng chuẩn Chomsky.

c) Xây dựng cây dẫn xuất sinh từ α=pmnnp từ văn phạm chuẩn Chomsky.

Đề số 3.

Bài 1: Cho bảng chữ cái ∑={m,n,p}

L1={x∈∑*| x chứa đúng 3 ký hiệu n, các ký hiệu m,p là bất kỳ}

a) Xây dựng nguồn sinh ngôn ngữ L1.

b) Xây dựng Otomat tương đương với nguồn

Bài 2: Cho bảng chữ cái ∑={a,b,c}

L2={x∈∑*| x bắt đầu bởi a và có độ dài lẻ}

a) Xây dựng nguồn I sinh ngôn ngữ L2.

b) Xây dựng Otomat hữu hạn tương đương với nguồn.

c) Xây dựng nguồn soi gương với nguồn I.

TÀI LIỆU THAM KHẢO

- [1] **Nguyễn Văn Ba**, *Ngôn ngữ hình thức*, Trường Đại học Bách khoa Hà nội, 1997
- [2] **Phan Đình Diệu**, *Lý thuyết otomat và thuật toán*, Nhà xuất bản Đại học và Trung học Chuyên nghiệp, 1971
- [3] **Đỗ Đức Giáo, Đặng Huy Ruận**, *Ngôn ngữ hình thức*, Nhà xuất bản KHKT, 1991
- [4] **Phạm Hồng Nguyên**, *Giáo trình chương trình dịch*, ĐH KHTN HN
- [5] **Nguyễn Văn Ba**, *Phân tích cú pháp*, ĐH Bách khoa HN