



*Giáo trình*  
*Công nghệ phần mềm*

# Mục lục

---

<b>CHƯƠNG 1 ĐẠI CƯƠNG VỀ CÔNG NGHỆ PHẦN MỀM.....</b>	<b>5</b>
I. KHÁI QUÁT VỀ LỊCH SỬ LẬP TRÌNH .....	5
I.1. Lập trình tuyến tính.....	5
I.2. Lập trình có cấu trúc .....	6
I.3. Lập trình định hướng đối tượng (ĐHĐT).....	6
I.4. Lập trình trực quan.....	7
I.5. Những tư tưởng cách mạng trong lập trình .....	7
II. CÁC PHƯƠNG DIỆN CỦA CÔNG NGHỆ PHẦN MỀM .....	8
II.1. Công nghệ phần mềm là gì?.....	8
II.2. Những yếu tố chất lượng bên ngoài và bên trong .....	8
II.3. Sản phẩm phần mềm là gì ? .....	9
III. NHỮNG NỘI DUNG CƠ BẢN CỦA CNPM.....	11
III.1. Tổng quan về công nghệ phần mềm .....	11
III.2. Chu kỳ sống của phần mềm .....	12
<b>CHƯƠNG 2 THIẾT KẾ PHẦN MỀM .....</b>	<b>18</b>
I. NỀN TẢNG CỦA THIẾT KẾ PHẦN MỀM.....	18
II. PHƯƠNG PHÁP LẬP TRÌNH CẤU TRÚC .....	20
II.1. Khái niệm về lập trình cấu trúc.....	22
II.2. Những ý tưởng cơ bản lập trình cấu trúc.....	22
II.3. Các cấu trúc điều khiển chuẩn .....	25
II.4. Một số ví dụ viết chương trình theo sơ đồ khối .....	28
III. CẤU TRÚC TỐI THIỂU .....	29
III.1. Các cấu trúc lồng nhau.....	31
IV. LẬP TRÌNH ĐƠN THỂ .....	32
IV.1. Khái niệm về đơn thể .....	32
IV.2. Mối liên hệ giữa các đơn thể .....	33
IV.2.1. Phân loại đơn thể.....	33
IV.2.2. Tổ chức một chương trình có cấu trúc đơn thể .....	33
V. PHÁT TRIỂN CHƯƠNG TRÌNH BẰNG TINH CHẾ TỪNG BƯỚC .....	35
V.1. Nội dung phương pháp.....	35
V.2. Ví dụ minh họa.....	36
V.2.1. Ví dụ 1.....	36
V.2.2. Bài toán 8 quân hậu.....	38

V.3.	Sửa đổi chương trình .....	42
VI.	PHỤ LỤC - ĐƠN VỊ TRONG TURBO PASCAL.....	50
VI.1.	Giới thiệu Unit .....	50
VI.2.	Cấu trúc của Unit .....	50
VI.3.	Cách sử dụng Unit.....	52
VI.4.	Ví dụ về Unit.....	53
VI.5.	Bài tập .....	55
<b>CHƯƠNG 3</b>	<b>HỢP THỨC HÓA PHẦN MỀM.....</b>	<b>57</b>
I.	XÁC MINH VÀ HỢP THỨC HÓA PHẦN MỀM.....	57
II.	CHỨNG MINH SỰ ĐÚNG ĐẮN CỦA CHƯƠNG TRÌNH.....	58
II.1.	Suy luận Toán học .....	59
II.1.1.	Các quy tắc suy luận Toán học .....	59
II.1.2.	Khái niệm về chứng minh tính đúng đắn của chương trình .....	60
II.1.3.	Tiên đề và quy tắc suy diễn.....	61
II.1.4.	Quy tắc điều kiện if B then P .....	62
II.1.5.	Quy tắc điều kiện if B then P else Q .....	63
II.1.6.	Quy tắc vòng lặp while .....	63
II.1.7.	Các quy tắc khác.....	64
II.2.	Phương pháp của C.A.R. Hoare .....	66
II.2.1.	Phát biểu .....	66
II.2.2.	Chứng minh tính đúng đắn từng phần của Div.....	66
II.3.	Chứng minh dừng.....	69
II.3.1.	Chứng minh dừng của một chương trình.....	69
II.3.2.	Chứng minh dừng của Div .....	70
II.3.3.	Đánh giá một chương trình lặp.....	71
III.	XÂY DỰNG CHƯƠNG TRÌNH .....	72
III.1.	Mở đầu .....	72
III.2.	Bài toán cờ tam tài .....	73
III.2.1.	Lời giải thứ nhất.....	74
III.2.2.	Lời giải thứ hai.....	75
III.2.3.	Chứng minh tính đúng đắn của chương trình (I) .....	76
III.3.	In ra một danh sách theo thứ tự ngược .....	80
III.3.1.	TILDA1 .....	81
IV.	CÁC TIÊN ĐỀ VÀ QUY TẮC SUY DIỄN.....	82
IV.1.	Điều kiện trước yếu nhất và điều kiện sau mạnh nhất của một dãy lệnh.....	82
IV.1.1.	Hàm fppre .....	83
IV.1.2.	Hàm fppost.....	83
IV.1.3.	Sử dụng điều kiện trước yếu nhất và điều kiện sau mạnh nhất để chứng minh tính đúng đắn của chương trình.....	84

IV.2.	Các tiên đề gán.....	86
IV.2.1.	Điều kiện trước yếu nhất và điều kiện sau mạnh nhất của lệnh gán .....	86
IV.2.2.	Quy tắc tính toán điều kiện sau mạnh nhất của một phép gán.....	87
V.	BÀI TẬP.....	89
<b>CHƯƠNG 4 THỬ NGHIỆM CHƯƠNG TRÌNH .....</b>		<b>90</b>
I.	KHẢO SÁT PHẦN MỀM .....	90
II.	CÁC PHƯƠNG PHÁP THỬ NGHIỆM.....	92
II.1.	Định nghĩa và mục đích thử nghiệm.....	92
II.2.	Thử nghiệm trong chu kỳ sống của phần mềm.....	94
II.2.1.	Thử nghiệm đơn thể.....	94
II.2.2.	Thử nghiệm tích hợp.....	95
II.2.3.	Thử nghiệm hệ thống.....	96
II.2.4.	Thử nghiệm hồi quy.....	97
II.3.	Dẫn dắt các thử nghiệm.....	97
II.4.	Thiết kế các phép thử phá hủy (Defect Testing).....	98
II.4.1.	Các phương pháp dựa trên chương trình.....	98
II.4.2.	Các phương pháp dựa trên đặc tả.....	100
II.4.3.	Kết luận.....	101
II.4.4.	Các tiêu chuẩn kết thúc thử nghiệm.....	101
II.5.	Các phép thử nghiệm thống kê.....	102
II.5.1.	Mở đầu .....	102
II.5.2.	Ước lượng độ ổn định của một phần mềm .....	104
<b>CHƯƠNG 5 ĐẶC TẢ PHẦN MỀM.....</b>		<b>105</b>
I.	MỞ ĐẦU ĐẶC TẢ PHẦN MỀM.....	105
I.1.	Khái niệm về đặc tả .....	105
I.1.1.	Đặc tả là gì ?.....	105
I.1.2.	Các phương pháp đặc tả.....	105
I.1.3.	Các thí dụ minh họa.....	106
I.2.	Đặc tả và lập trình.....	107
II.	ĐẶC TẢ CẤU TRÚC DỮ LIỆU .....	109
II.1.	Khái niệm về Cấu trúc dữ liệu cơ sở vectơ .....	109
II.1.1.	Dẫn nhập.....	109
II.1.2.	Đặc tả hình thức.....	110
II.2.	Truy nhập một phần tử của vectơ.....	110
II.3.	Các thuật toán xử lý vectơ.....	111
II.3.1.	Truy tìm tuần tự một phần tử của vectơ (sequential search).....	111
II.3.2.	Tìm kiếm nhị phân (Binary search) .....	113
III.	ĐẶC TẢ ĐẠI SỐ : MÔ HÌNH HÓA PHÁT TRIỂN PHẦN MỀM.....	117
III.1.	Mở đầu .....	117
III.2.	Phân loại các phép toán.....	119
III.3.	Hạng và biến .....	120
III.4.	Phép thế các hạng.....	120

III.5.	Các thuộc tính của đặc tả .....	122
III.5.1.	Mô hình lập trình (triển khai) .....	122
III.5.2.	Mô hình đặc biệt .....	123
III.5.3.	Mô hình đồng dư .....	123
III.6.	Phép chứng minh trong đặc tả đại số .....	123
III.6.1.	Lý thuyết tương đương .....	124
III.6.2.	Khái niệm về lý thuyết quy nạp .....	125
III.6.3.	Chứng minh tự động bởi viết lại .....	126
III.6.4.	Phân cấp trong đặc tả đại số .....	128
IV.	ĐẶC TẢ HAY CÁCH CỤ THỂ HÓA SỰ TRỪU TƯỢNG .....	129
IV.1.	Đặc tả phép thay đổi bộ nhớ .....	129
IV.2.	Hàm .....	131
IV.3.	Hợp thức hóa và phục hồi .....	134
IV.4.	Bắt đầu triển khai thực tiễn .....	137
IV.5.	Phép hợp thành (cấu tạo) .....	140
IV.6.	Triển khai thứ hai .....	141
IV.7.	Triển khai thực hiện lần thứ ba .....	146
IV.8.	Đặc tả làm gì ? .....	149

## CHƯƠNG 1

---

# Đại cương về công nghệ phần mềm

## I. Khái quát về lịch sử lập trình

Lập trình (programming), hay *lập chương trình cho máy tính điện tử* (MTĐT) là một ngành còn rất mới mẻ. MTĐT đầu tiên lập trình được mới chỉ xuất hiện cách đây hơn bốn mươi năm <sup>1</sup>. Suốt hơn bốn thập kỷ qua, lập trình không ngừng được cải tiến và phát triển, càng ngày càng hướng về nhu cầu của người lập trình.

Lập trình là một công việc nặng nhọc, năng suất thấp so với các hoạt động trí tuệ khác. Ví dụ nếu một sản phẩm phần mềm khoảng 2000 – 3000 dòng lệnh đòi hỏi 3 người lập trình chính trong vòng 6 tháng thì năng suất mỗi người chỉ dao động trong khoảng từ 5 đến 6 lệnh mỗi ngày (!).

Chính vì các sản phẩm phần mềm khi tung ra thị trường chưa thực sự hoàn hảo ngay nên người ta thường dùng mẹo thương mại bằng cách gán cho sản phẩm một cái đuôi "phiên bản" (version) để nói rằng phiên bản ra sau đã khắc phục được những khiếm khuyết của phiên bản trước đó.

### Ví dụ 1 :

Hệ điều hành MS-DOS đã có các phiên bản 1.0, 3.3, 5.0, 6.0, 7.0 v.v...

Microsoft Windows đã có các phiên bản 1.0, 2.0, 3.0, 3.1, 3.11.

Nay là Windows 95, 97, 98 v.v...

Turbo Psacal của hãng Borland Inc. đã có các phiên bản 5.0, 6.0, 7.0, 8.0 v.v...

### I.1. Lập trình tuyến tính

Với những MTĐT đầu tiên, người ta sử dụng ngôn ngữ máy (machine language) hay ngôn ngữ bậc thấp (low level) để lập trình và dùng các khoá cơ khí để nạp chương trình vào máy. Theo đà phát triển của các thiết bị phần cứng, các ngôn ngữ bậc cao (high level) với các dòng lệnh tựa tiếng Anh bắt đầu được sử dụng. Máy sẽ dịch chương trình đó sang ngôn ngữ máy trước khi thực hiện.

Với những ngôn ngữ lập trình ban đầu, chương trình viết ra gồm những dòng lệnh có khuynh hướng nối nhau theo dây dài, khó hiểu về mặt logic. Người ta sử

---

<sup>1</sup> ENIAC (Electronic Numerical Integrator and Computer) là chiếc MTĐT đầu tiên ra đời năm 1945 tại trường Đại học Tổng hợp Pennsylvania, nước Mỹ.

dụng các lệnh nhảy (goto) để điều khiển chương trình một cách tùy tiện. Chương trình là một mớ rối rắm không khác gì món mì sợi (spaghetti) của nước Ý.

Các ngôn ngữ lập trình tuyến tính không kiểm soát được những sự thay đổi của dữ liệu. Mọi dữ liệu sử dụng trong chương trình đều có tính toàn cục và có thể bị thay đổi vào bất cứ lúc nào. Vào giai đoạn này, người ta xem việc lập trình như một hoạt động nghệ thuật nhuộm màu sắc tài nghệ cá nhân hơn là khoa học, với thuật ngữ "the art of programming".

## 1.2. Lập trình có cấu trúc

Vào cuối những năm 1960 và đầu 1970, khuynh hướng lập trình cấu trúc (structured programming) ra đời. Theo phương pháp này, một chương trình có cấu trúc được tổ chức theo các phép toán mà nó phải thực hiện. Chương trình bao gồm nhiều thủ tục, hay hàm, riêng rẽ. Các thủ tục hay hàm này độc lập với nhau, có dữ liệu riêng, giải quyết những vấn đề riêng, nhưng có thể trao đổi qua lại với nhau bằng các tham biến.

Lập trình cấu trúc làm cho việc kiểm soát chương trình dễ dàng hơn, và do vậy, giải quyết bài toán dễ dàng hơn. Tính hiệu quả của lập trình cấu trúc thể hiện ở khả năng trừu tượng hoá. Trong một chương trình có cấu trúc, người ta chỉ quan tâm về mặt chức năng : một thủ tục hay hàm nào đó có thực hiện được công việc đã cho hay không ? Còn việc thực hiện như thế nào là không quan trọng, chúng ta nào còn đủ tin cậy.

Mặc dù kỹ thuật thiết kế và lập trình cấu trúc được sử dụng rộng rãi nhưng vẫn bộc lộ những khiếm khuyết. Khi độ phức tạp tăng lên thì sự phụ thuộc của chương trình vào kiểu dữ liệu mà nó xử lý cũng tăng theo. Cấu trúc dữ liệu trong một chương trình có vai trò quan trọng cũng như các phép toán thực hiện trên chúng. Một khi có sự thay đổi trên một kiểu dữ liệu thì một thủ tục nào đó tác động lên kiểu dữ liệu này cũng phải thay đổi theo.

Khiếm khuyết trên cũng ảnh hưởng đến tính hợp tác giữa các thành viên lập trình. Một chương trình có cấu trúc được giao cho nhiều người thì khi có sự thay đổi về cấu trúc dữ liệu của một người sẽ ảnh hưởng đến công việc của những người khác.

## 1.3. Lập trình định hướng đối tượng (ĐHĐT)

Lập trình ĐHĐT (oriented-object programming) được xây dựng trên nền tảng của lập trình cấu trúc và trừu tượng hoá dữ liệu (data abstraction).

Chương trình ĐHĐT được thiết kế xung quanh dữ liệu mà nó thao tác chứ không bản thân các thao tác. Tính ĐHĐT làm rõ mối quan hệ giữa dữ liệu và thao tác trên dữ liệu.

Trừu tượng hoá dữ liệu là làm cho việc sử dụng các cấu trúc dữ liệu trở nên độc lập đối với việc cài đặt cụ thể. Ví dụ số dấu chấm động (floating point number) đã được trừu tượng hoá trong mọi ngôn ngữ lập trình. NSD thao tác trên các số dấu chấm động mà không quan tâm đến cách biểu diễn nhị phân trong máy của chúng như thế nào.

Lập trình ĐHĐT liên kết các cấu trúc dữ liệu với các phép toán. Một cấu trúc nào đó thì tương ứng, ta có những phép toán nào đó. Ví dụ : một bản ghi về nhân sự có thể được đọc, cập nhật sự thay đổi và được cất giữ, còn một số phức thì được dùng trong tính toán. Không thể viết số phức lên tệp như một bản ghi nhân sự, cũng không thể cộng trừ nhân chia hai bản ghi nhân sự với nhau như cách của số phức.

Lập trình ĐHĐT đưa vào nhiều thuật ngữ và khái niệm mới, chẳng hạn khái niệm lớp (class), khái niệm kế thừa (inheritance).

Ưu điểm của lập trình ĐHĐT là làm cho việc phát triển phần mềm nhanh chóng hơn với khả năng dùng lại các chương trình cũ. Một lớp mới được xem như lớp suy diễn, có thể được kế thừa cấu trúc dữ liệu và các phương pháp của lớp gốc hoặc lớp cơ sở.

Một trong những ngôn ngữ lập trình ĐHĐT được nói đến là SMALLTALK, được phát triển năm 1980 tại Xerox Palo Alto Research Center (PARC). Hiện nay, nhiều ngôn ngữ lập trình thông dụng cũng được trang bị thêm khả năng ĐHĐT, như là C++, Delphi, v.v...

#### **1.4. Lập trình trực quan**

Lập trình trực quan (visual programming) được phát triển trên nền tảng của lập trình ĐHĐT. Khi thiết kế chương trình, người lập trình nhìn thấy ngay kết quả qua từng thao tác và giao diện người dùng (user interface) khi chương trình được thực hiện. Người lập trình có thể dễ dàng chỉnh sửa về màu sắc, kích thước, hình dáng và các xử lý thích hợp lên các đối tượng có mặt trong giao diện.

Các ngôn ngữ lập trình trực quan thông dụng hiện nay thường được phát triển trong môi trường Microsoft Windows, như Visual Basic, Visual C++, Visual Foxpro, Java. v.v...

#### **1.5. Những tư tưởng cách mạng trong lập trình**

Lập trình là một trong những lĩnh vực khó nhất của toán học ứng dụng. Người ta coi lập trình là một khoa học nhằm đề xuất những nguyên lý và phương pháp để nâng cao năng suất lao động của lập trình viên. Năng suất ở đây được hiểu là tính đúng đắn của chương trình, tính dễ đọc, dễ sửa, tận dụng hết khả năng của thiết bị mà không phụ thuộc vào thiết bị.



Thực chất của quá trình lập trình là người ta không lập trình trên một ngôn ngữ cụ thể mà lập trình hướng tới nó. Chương trình phải được viết dưới dạng các thao tác có cấu trúc trên các đối tượng có cấu trúc và các mệnh đề nhằm khẳng định tính đúng đắn của kết quả.

Những tư tưởng cách mạng trong lập trình thể hiện ở hai điểm sau :

- Chương trình và lập trình viên trở thành đối tượng nghiên cứu của lý thuyết lập trình.
- Làm thế nào để làm chủ được sự phức tạp của hoạt động lập trình ?

## II. Các phương diện của công nghệ phần mềm

### II.1. Công nghệ phần mềm là gì?

Theo từ điển *Computer Dictionary* của Microsoft Press® (1994), Software Engineering : The design and development of software (computer program), from concept through execution and documentation.

Từ điển Larousse (1996) định nghĩa chi tiết hơn : *Công nghệ phần mềm là tập hợp các phương pháp, mô hình, kỹ thuật, công cụ và thủ tục liên quan đến các giai đoạn xây dựng một sản phẩm phần mềm. Các giai đoạn đó là : đặc tả (specification), thiết kế (design), lập trình (programming), thử nghiệm (testing), sửa sai (debugging), cài đặt (setup) để đem vào ứng dụng (application), bảo trì (maintenance) và lập hồ sơ (documentation).*

Mục đích chính của công nghệ phần mềm là để sản xuất ra những phần mềm có chất lượng. Chất lượng phần mềm không là một khái niệm đơn giản, bao gồm nhiều yếu tố. Chẳng hạn chương trình chạy nhanh, dễ sử dụng, có tính cấu trúc, dễ đọc dễ hiểu, v.v...

Người ta thường đánh giá theo hai kiểu chất lượng : những yếu tố chất lượng bên ngoài và những yếu tố chất lượng bên trong.

### II.2. Những yếu tố chất lượng bên ngoài và bên trong

Những yếu tố chất lượng bên ngoài người dùng có thể nhận biết được, như tốc độ nhanh, chạy ổn định, tính dễ sử dụng, dễ thích nghi với những thay đổi (tính mở rộng), tính công thái học (ergonomy, human factor), v.v...

Những yếu tố chất lượng bên ngoài của một sản phẩm phần mềm là :

<i>Tính đúng đắn</i>	Khả năng thực hiện chính xác công việc đặt ra.
<i>Tính bền vững</i>	Có thể hoạt động trong những điều kiện bất thường.
<i>Tính có thể mở rộng</i>	Khả năng dễ sửa đổi để thích nghi với những thay đổi mới

<i>Tính sử dụng lại</i>	Khả năng sử dụng lại toàn bộ hay một phần của hệ thống cho những ứng dụng mới.
<i>Tính tương thích</i>	Có thể dễ dàng kết hợp với các sản phẩm phần mềm khác.
<i>Các chất lượng khác</i>	Hiệu quả đối với nguồn tài nguyên của MTĐT như bộ xử lý, bộ nhớ..., dễ chuyển đổi (không phụ thuộc vào cấu hình phần cứng), dễ kiểm chứng và an toàn (được bảo vệ quyền truy nhập), dễ sử dụng, v.v...

Những yếu tố chất lượng bên trong là tính đơn thể, tính dễ đọc, dễ hiểu mà chỉ những người làm Tin học chuyên nghiệp mới biết được. Yếu tố chất lượng bên ngoài là mục đích cuối cùng nhưng yếu tố chất lượng bên trong lại là mấu chốt để đạt được những yếu tố chất lượng bên ngoài.

### II.3. Sản phẩm phần mềm là gì ?

Mặc dù người ta không định nghĩa nhưng khái niệm sản phẩm phần mềm được hiểu như là một hệ thống chương trình thực hiện một nhiệm vụ tương đối độc lập nhằm phục vụ cho một ứng dụng cụ thể trong cuộc sống của con người (và có thể được thương mại hoá). Ví dụ các sản phẩm phần mềm :

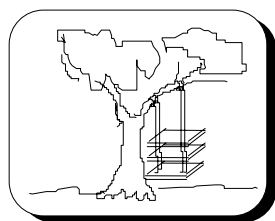
- Hệ điều hành : MS – DOS, OS/2, Unix, MAC OS...
- Hệ điều hành mạng máy tính : Unix, Novell Netware, Windows NT... và các ứng dụng trên mạng LAN, WAN, Internet/Intranet (các Browsers, các dịch vụ khai thác Internet...).
- Các ngôn ngữ lập trình (chương trình dịch) : Turbo Pascal, Turbo C, C++...
- Hệ quản trị cơ sở dữ liệu : Microsoft Foxpro, Microsoft Access, Oracle, Paradox...
- Microsoft Windows và các ứng dụng trên Windows.
- Các trò chơi (games).
- Các phần mềm trợ giúp thiết kế (CAD, Designers...), trợ giúp giảng dạy...
- Các hệ chuyên gia, trí tuệ nhân tạo, người máy, v.v...
- Các chương trình phòng chống virus, v.v...

Dưới đây là bảng tóm tắt quá trình tiến hóa của sản phẩm phần mềm :

Thời kỳ đầu tiên 1950 – 1960	Xử lý theo lô (Batch processing) Phần mềm được viết theo đơn đặt hàng
Thời kỳ thứ hai 1960 – 1970	Đa người dùng (Multiusers) Thời gian thực (Real time) Cơ sở dữ liệu (Database) Phần mềm sản phẩm

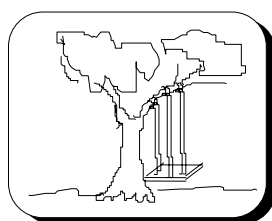
Thời kỳ thứ ba 1970 – 1990	Hệ thống xử lý phân bố (Distributed processing system) Thông minh (Intelligence) Phần cứng giá thành hạ Hiệu quả tiêu thụ
Thời kỳ thứ tư 1990 trở đi	Hệ thống để bàn (Desktop – Personal – Notebook computers) Lập trình hướng đối tượng (Object oriented programming) Lập trình trực quan (Visual programming) Hệ chuyên gia (Expert system) Mạng thông tin toàn cầu (Worldwide communication network) Xử lý song song (Paralell processing) ...

Sau đây là một tranh vui về quá trình tạo ra một sản phẩm phần mềm đã khá quen thuộc đối với những người làm Tin học từ hơn 20 năm nay (theo J. CLAVIER, "Diriger un projet informatique", Édition J. C. I. Inc, Canada 1993) :

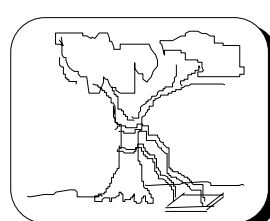


1. Người đặt hàng

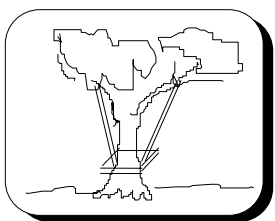
Ví dụ : Công ty Công viên



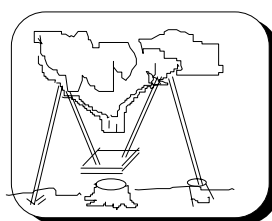
2. Thiết kế của chủ trì đề tài



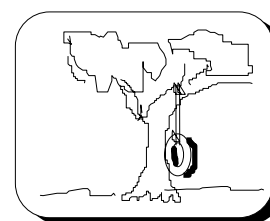
3. Sản phẩm của người lập trình



4. Sau khi sửa sai với  
nhiều sáng kiến cải tiến



5. Triển khai cho khách hàng



6. Ước mơ của người sử dụng !

*Hình 1.1. Quá trình tạo ra một sản phẩm phần mềm*

### III. Những nội dung cơ bản của CNPM

#### III.1. Tổng quan về công nghệ phần mềm

Công nghệ phần mềm đặc trưng bởi tập hợp các phương pháp để phát triển một chương trình (phần mềm nói chung). Sự phát triển một chương trình, hay *tiến trình phần mềm* (software process), không chỉ nằm ở chỗ lập trình theo nghĩa hẹp mà còn là việc triển khai các giai đoạn dẫn đến lập trình. Tập hợp các giai đoạn này được gọi là *chu kỳ sống* (hay vòng đời) của phần mềm (life cycle).

Với một dự án Tin học lớn, nhiều người lập trình tham gia được chia thành nhóm, mỗi nhóm phụ trách giải quyết một phần của dự án. Người phụ trách dự án có nhiệm vụ phân bổ công việc cho từng nhóm, đảm bảo mối liên lạc giữa các nhóm, kiểm tra tiến trình phát triển của dự án, chất lượng của sản phẩm phần mềm khi hoàn tất.

Tiến trình phát triển phần mềm gồm 3 giai đoạn chính là *xác định*, *phát triển* và *bảo trì*, không phụ thuộc vào miền áp dụng, độ lớn và độ phức tạp của dự án phát triển, cũng như mô hình được lựa chọn.

##### **Giai đoạn xác định :**

Giai đoạn này trả lời câu hỏi *là cái gì ?* (What?) và khi nào (When?) về dữ liệu (thông tin) cần xử lý, mục đích chức năng và môi trường phát triển. Gồm 3 bước :

- Phân tích hệ thống.
- Lập kế hoạch dự án phần mềm.
- Phân tích yêu cầu thực tiễn.

##### **Giai đoạn phát triển :**

Giai đoạn này trả lời câu hỏi *làm như thế nào ?* (How?). Gồm 3 bước :

- Thiết kế phần mềm : Sử dụng các công cụ đặc tả và lập trình cấu trúc.
- Chọn công cụ hoặc các ngôn ngữ lập trình để tiến hành viết chương trình.
- Kiểm thử (phát hiện sai sót, nhầm lẫn...).

##### **Giai đoạn bảo trì :**

Giai đoạn này tập trung vào các thay đổi (Modify). Có 3 kiểu thay đổi :

- *Sửa đổi* : Dù phần mềm có chất lượng tốt, vẫn tồn tại những khiếm khuyết từ việc sử dụng của khách hàng (người sử dụng). Bảo trì sửa đổi làm thay đổi phần mềm, khắc phục khiếm khuyết.

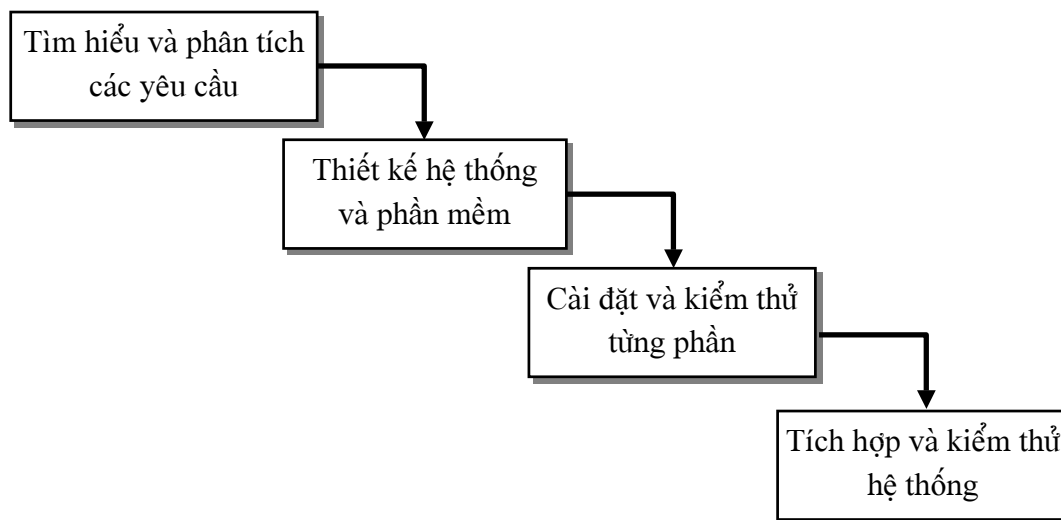
- *Thích nghi* : Nhằm làm phần mềm thích nghi với môi trường phần cứng, như CPU, OS, các thiết bị ngoại vi.

- *Nâng cao* : Khách hàng tìm ra những chức năng phụ của phần mềm. Bảo trì hoàn thiện để mở rộng phần mềm ra ngoài những chức năng vốn có.

### III.2. Chu kỳ sống của phần mềm

Có nhiều mô hình khác nhau để thể hiện một chu kỳ sống (life cycle). Sau đây là một chu kỳ sống kiểu cổ điển theo mô hình thác nước ("waterfall" model) gồm các giai đoạn như sau :

- Tìm hiểu và phân tích các yêu cầu (RAD – Requirements analysis and definition)
- Thiết kế hệ thống và phần mềm (SSD – System and software design)
- Cài đặt và kiểm thử từng phần (IUT – Implementtation and Unit testing)
- Tích hợp và kiểm thử hệ thống (IST – Integrgion and system testing)



Hình 1.2. Mô hình thác nước

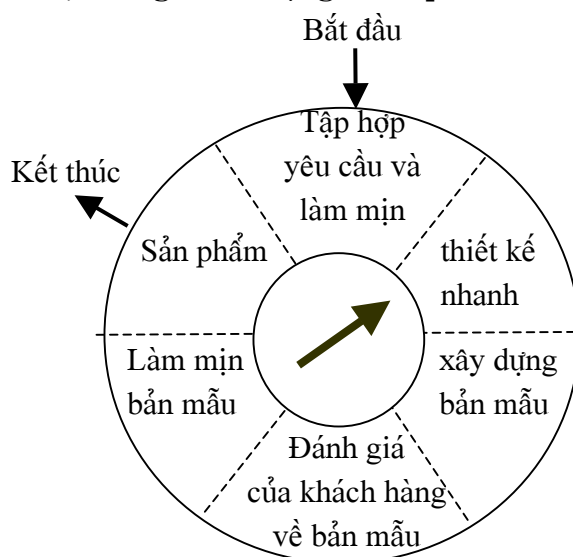
Dẫu rằng mô hình thác nước trên đây có ích lợi trong việc quản lý (management), lập kế hoạch và lập báo cáo tiến độ phát triển phần mềm nhưng chỉ thích hợp với một lớp hệ thống phần mềm nào đó mà thôi, không phù hợp với các hoạt động đã chỉ ra trong mô hình.

Tiến trình phần mềm gồm các hoạt động phức tạp và biến động mà không thể biểu diễn trên một mô hình đơn giản. Những mô hình tốt về tiến trình phần mềm vẫn còn là chứng chủ đề nghiên cứu. Hiện nay, các mô hình tổng quát khác nhau hay tính thực dụng của sự phát triển phần mềm, gắn bó chặt chẽ với nhau.

Mô hình thác nước nguyên thủy (original) là một trong những mô hình tổng quát mang tính thực dụng sâu sắc.

Sau đây là một số tiếp cận :

1. *Tiếp cận thác nước* (the waterfall approach) : Bao gồm các giai đoạn đặc tả yêu cầu, thiết kế phần mềm, cài đặt, kiểm thử, v.v..., sau mỗi giai đoạn là sự kết thúc (signed-off) và tiếp tục giai đoạn tiếp theo.
2. *Lập trình thăm dò* (exploratory programming) : Cho phép tăng nhanh quá trình để dẫn đến tính thỏa đáng của hệ thống. Lập trình thăm dò thường được áp dụng trong lĩnh vực trí tuệ nhân tạo, khi NSD không thể định hình được các đặc tả yêu cầu. NSD quan tâm đến tính thỏa đáng của kết quả hơn là tính chính xác.
3. *Bản mẫu* (prototyping) : Tương tự tiếp cận lập trình thăm dò. Pha đầu tiên bao gồm phát triển một chương trình cho phép thử nghiệm. Tuy nhiên, mục đích của phát triển là thiết lập các yêu cầu hệ thống. Sau đó là sự cài đặt lại phần mềm để đưa đến hệ thống chất lượng - sản phẩm.



Hình 1.3. Tiếp cận kiểu bản mẫu

4. *Biến đổi hình thức* (formal transformation) : Là sự biến đổi các đặc tả hình thức (formal specification) của hệ thống phần mềm đang xét để thành một chương trình khả thi nhưng bảo toàn được tính chính xác (correctness - preserving transformations).
5. *Lắp ráp hệ thống từ các thành phần dùng lại được* (system assembly from reusable components). Kỹ thuật này cho phép xây dựng hệ thống từ các thành phần đã có. Tiến trình phát triển hệ thống là sự lắp ráp hơn là sự sáng tạo.

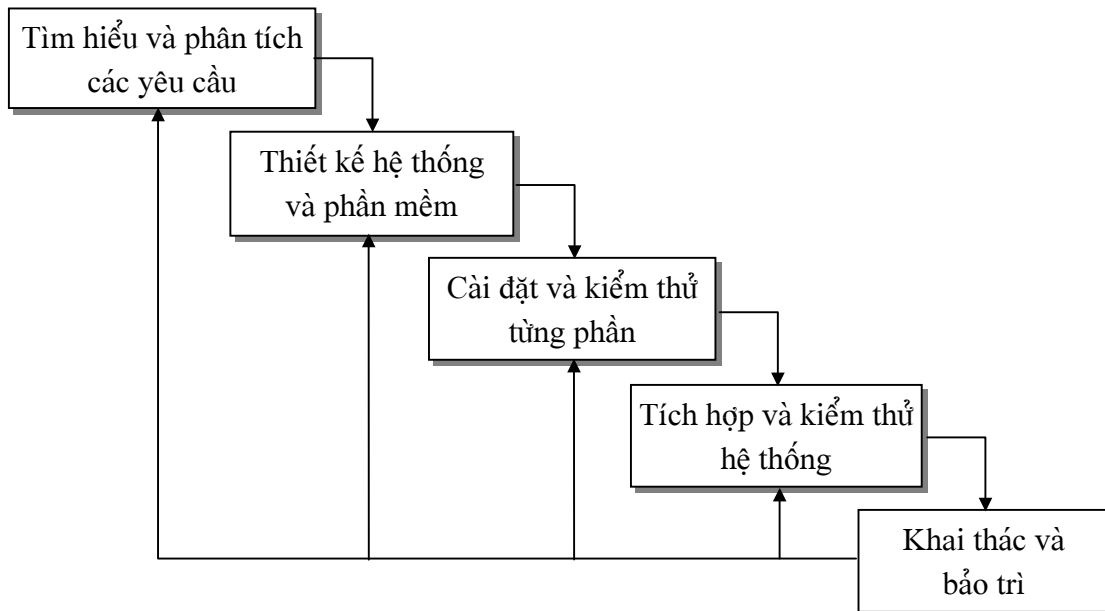
Hiện nay, các tiếp cận 1, 2, 3 được ứng dụng nhiều trong thực tiễn.

Trên thực tế, các giai đoạn phát triển phần mềm không phải rời riêng mà là gối lên nhau (overlap) và thông tin được cung cấp lẫn nhau.

Trong khi thiết kế, những vấn đề và các yêu cầu gắn bó với nhau, trong khi lập trình, những vấn đề thiết kế được tìm thấy, v.v... Lúc này, tiến trình phần mềm không đơn giản là một mô hình tuyến tính mà bao gồm một dãy các tương tác của các hoạt động phát triển.

Tuy nhiên, một mô hình chứa các vòng lặp sẽ làm khó khăn cho việc quản lý và báo cáo. Có nhiều dạng mô hình trong tiến trình phần mềm. Sau đây là một số mô hình :

### 1. Mô hình thác nước cải tiến

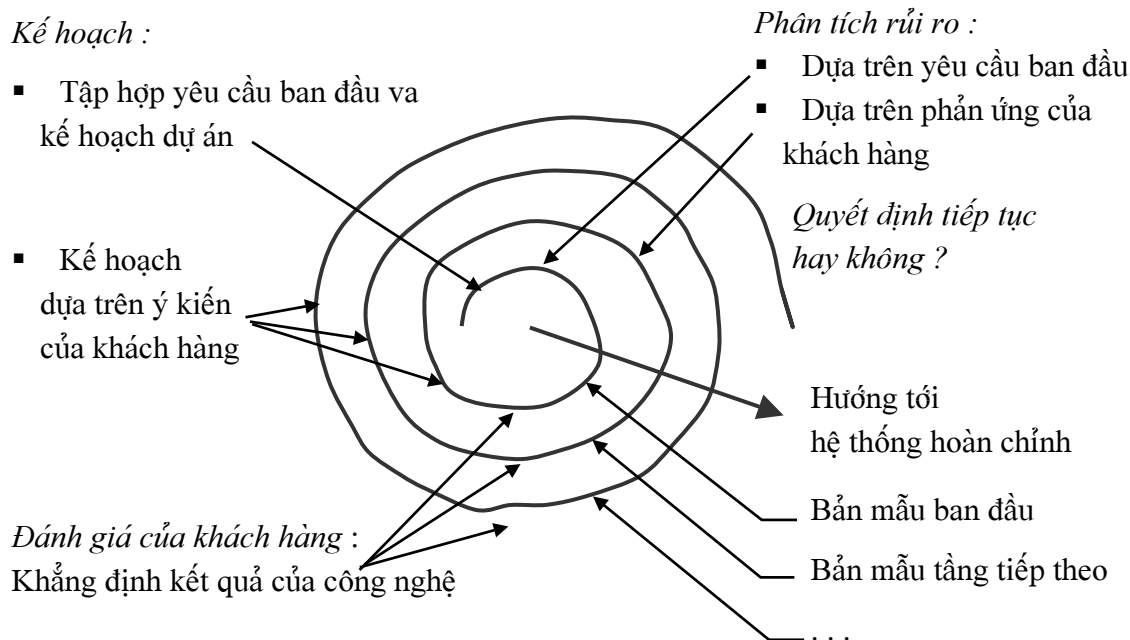


Hình 1.4. Mô hình thác nước cải tiến

1. *Tìm hiểu và phân tích các yêu cầu*: NSD hệ thống và người phát triển hệ thống bàn bạc, trao đổi (consultation) với nhau để thiết lập mục đích, ràng buộc và các dịch vụ của hệ thống phần mềm, lĩnh hội được những đòi hỏi của bài toán.
2. *Thiết kế hệ thống và phần mềm* : Tiến trình thiết kế hệ thống phân chia các yêu cầu thành các hệ thống phần cứng, phần mềm và thiết lập một kiến trúc hệ thống toàn bộ (overall system architecture). Việc thiết kế phần mềm bao gồm việc thể hiện các chức năng hệ thống phần mềm (software system functions) để biến đổi thành các chương trình khả thi.
3. *Cài đặt và kiểm thử từng phần* : Trong giai đoạn này, các đơn vị chương trình hay tập hợp các chương trình được kiểm thử lần lượt sao cho thỏa mãn các đặc tả tương ứng.
4. *Tích hợp và kiểm thử hệ thống* : Các đơn vị chương trình được tích hợp và kiểm thử như là một hệ thống đầy đủ để đảm bảo các yêu cầu đặt ra ban đầu. Sau giai đoạn này, hệ thống phần mềm được giao cho khách hàng.
5. *Khai thác và bảo trì* (operation and maintenance) : Đây là một pha dài nhất của chu kỳ sống. Hệ thống được cài đặt và đưa vào sử dụng thực tế. Việc bảo trì bao gồm việc khắc phục những sai sót xảy ra đã không xuất hiện trong các giao đoạn trước đó của chu kỳ sống. Việc tối ưu hóa các dịch vụ của hệ thống được xem như là những yêu cầu mới được phát hiện.

## 2. Mô hình xoắn ốc

Phát triển trên tính ưu việt của vòng đời cổ điển và bản mẫu, bổ sung những yếu tố còn thiếu và thêm các yếu tố mới, phân tích rủi ro.



Hình 1.5. Mô hình xoắn ốc

**Ưu điểm :**

Các phiên bản (hay sản phẩm) được hoàn thiện dần theo chiều xoáy ốc từ trong ra ngoài.

**Nhược điểm :**

- Khó đánh giá chính xác, nhất là khi gặp rủi ro, khó kiểm soát. Do đó khó thuyết phục được các khách hàng lớn
- Mô hình này còn mới, chưa được kiểm nghiệm nhiều trong thực tiễn.

## 3. Kỹ thuật thế hệ 4 (4th Generation Technology)

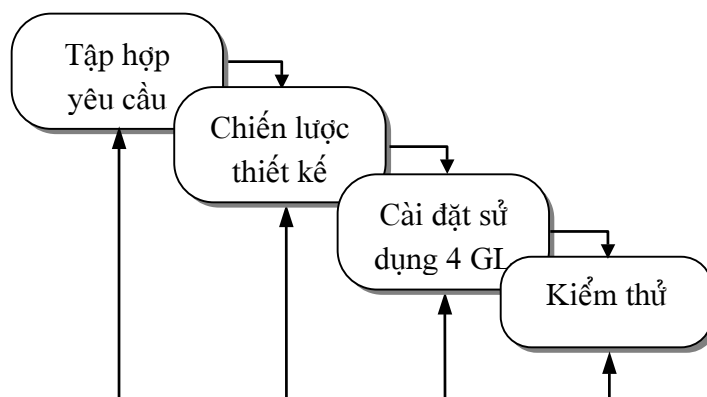
Bao gồm các công cụ phần mềm trên cơ sở tự động sản sinh mã chương trình gốc theo nhu cầu của người phát triển :

- Ngôn ngữ phi thủ tục<sup>2</sup> (non procedural language) để truy cập cơ sở dữ liệu.
- Bộ sinh báo cáo.
- Bộ thao tác dữ liệu.

<sup>2</sup> là ngôn ngữ lập trình không tuân theo cách gọi thủ tục hay gọi chương trình con thông thường, không sử dụng các cấu trúc điều khiển, tuần tự, mà dựa trên tập hợp các yếu tố và quan hệ để dẫn về kết quả yêu cầu. Ví dụ ngôn ngữ vấn tin SQL thuộc loại này.



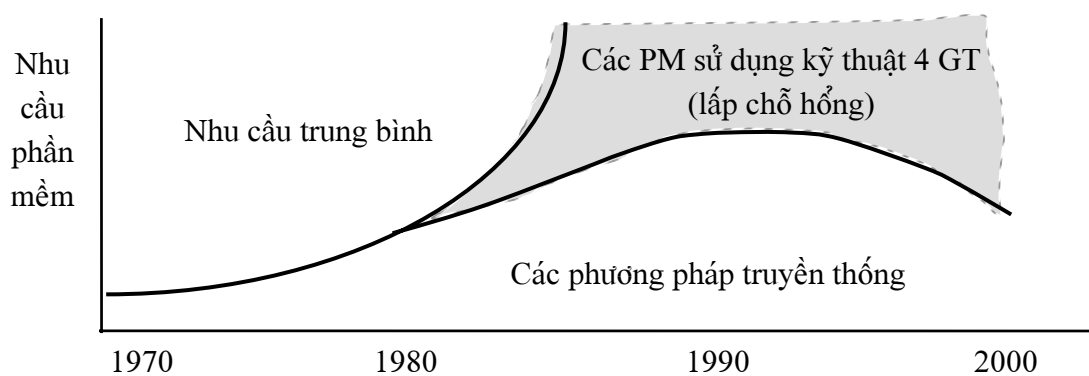
- Bộ tương tác và thiết kế màn hình.
- Bộ sinh chương trình.
- Bảng tính.
- Công cụ đồ họa.



Hình 1.6. Kỹ thuật thế hệ 4

Ưu điểm :

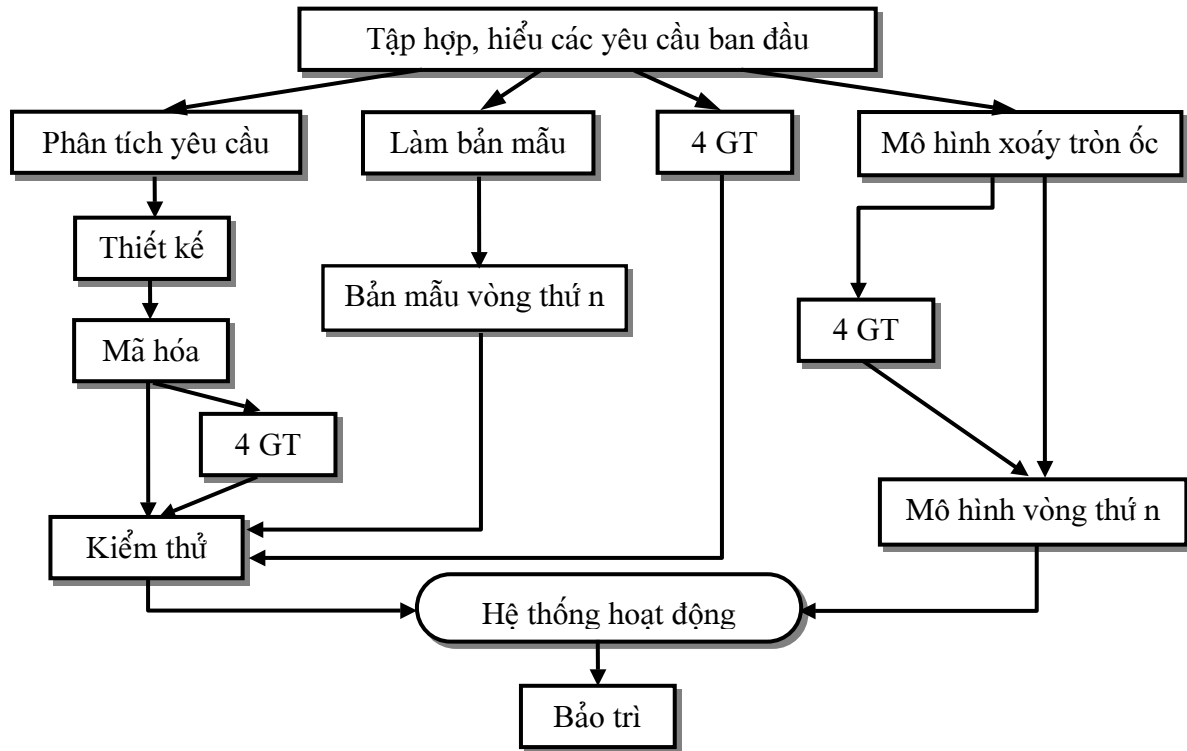
Thường được sử dụng để xây dựng các hệ thống tin và tương lai là các ứng dụng kỹ nghệ phát triển phần mềm thời gian thực.



Hình 1.7. Nhu cầu phần mềm

## 5. Tích hợp các kỹ thuật

Nhằm tăng cường tính tối ưu trong phát triển phần mềm, người ta có xu hướng tích hợp các kỹ thuật cổ điển, xoáy tròn ốc và 4GT đã nêu.



Hình 1.8. Tích hợp các kỹ thuật

## CHƯƠNG 2

---

# Thiết kế phần mềm

## I. Nền tảng của thiết kế phần mềm



## **II. Phương pháp lập trình cấu trúc**



## II.1. Khái niệm về lập trình cấu trúc

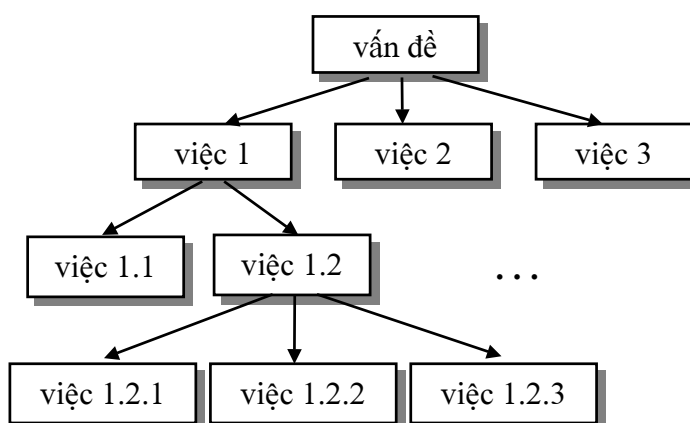
Lập trình cấu trúc (Structured Programming) là trường phái lập trình xuất hiện vào những năm 70 và được duy trì phát triển từ đó đến nay. Lập trình cấu trúc phản ánh quan niệm : lập trình là công việc sáng tạo nhưng có tính khoa học và có phương pháp, không phải là ngẫu hứng cá nhân.

Tính logic và trong sáng của chương trình đảm bảo độ tin cậy, dễ hiểu, dễ sửa và dễ thừa kế chương trình.

## II.2. Những ý tưởng cơ bản lập trình cấu trúc

### a) Chương trình là một hệ thống phân cấp từ trên xuống

Trong lập trình cấu trúc, chương trình là một hệ thống phân cấp từ trên xuống, trong đó các thành phần tương tác với nhau tối thiểu. Vấn đề cần giải quyết bao gồm các vấn đề nhỏ hơn, mỗi vấn đề đó lại bao gồm các vấn đề nhỏ hơn nữa, v.v... cho đến mức cuối cùng là những công việc đơn giản và dễ giải quyết hoặc đã giải quyết rồi.



Hình 2.1. Chương trình là một hệ thống phân cấp

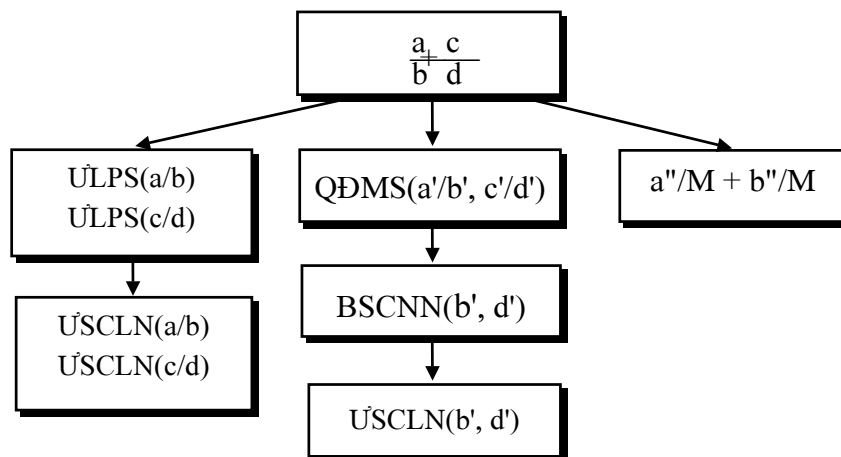
Ví dụ 2 :

Phân tích bài toán cộng hai phân số để đưa về bài toán tìm ước số chung lớn nhất.

Để cộng hai phân số, trước tiên cần ước lượng chúng, tiếp đó quy đồng mẫu số để lấy mẫu số chung. Cuối cùng tiến hành cộng hai tử số của hai phân số đã có chung mẫu số. Việc ước lượng phân số được đưa về tìm ước số chung lớn nhất của tử số và mẫu số (sử dụng thuật toán Euclide).

Để quy đồng mẫu số, cần tìm bội số chung nhỏ nhất. Việc tìm bội số chung nhỏ nhất của hai số lại được đưa về tìm ước số chung lớn nhất của chúng :

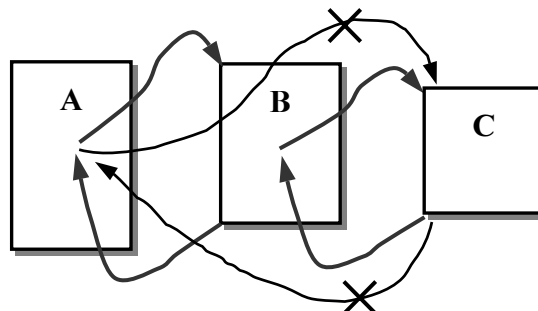
$$BSCNN(b', d') = b' * d' / ƯSCLN(b',d').$$



Hình 2.2. Phân tích bài toán cộng hai phân số

Như vậy, chương trình là một hệ thống gồm nhiều thành phần phân cấp, mỗi thành phần có nhiệm vụ giải quyết một vấn đề sơ cấp và có tính độc lập cao. Các thành phần nên tương tác với nhau tối thiểu. Giữa hai thành phần trong hệ thống chỉ nên có tối đa một đường tương tác là đường trao đổi thông tin để dễ quản lý và dễ kiểm soát.

- Giữa chương trình chính và chương trình con có đường giao tiếp là việc truyền tham biến. Không được gọi chương trình con theo kiểu vượt cấp.





*Hình 2.3. A gọi B, B gọi C, nhưng A không gọi được C*

- Hạn chế dùng biến toàn cục (global variables) trong chương trình con vì sẽ tạo thêm những đường giao tiếp khó quản lý. Chẳng hạn, một chương trình con nào đó làm thay đổi một biến toàn cục thì ở một nơi khác, trong một chương trình con khác hoặc ngay trong chương trình chính, cũng sẽ khó nhận biết sự thay đổi này.

**b) Không sử dụng lệnh nhảy goto**

Lệnh goto (jump statement) dùng để chuyển điều khiển đến một điểm khác trong chương trình. Lệnh goto làm khó quản lý và khó kiểm soát chương trình nên khó đọc, khó sửa sai (rối rắm như món mì sợi Spaghetti của Ý).

Các chương trình viết trên ngôn ngữ assembly hoặc trên các ngôn ngữ bậc cao như Fortran, Algol, Cobol... thường sử dụng lệnh goto.

*Ví dụ 3 :*

Chương trình Algol sau đây sử dụng lệnh goto để điều khiển vòng lặp tính tổng các phần tử của mảng a gồm N số thực :

```
S := 0; I := 0;
Start : S := S + a[ I ] ; I := I + 1;
      if I <= N then goto Start;
...

```

**c) Chương trình có tính cấu trúc**

Chương trình chỉ sử dụng các cấu trúc điều kiện chuẩn, dễ hiểu, dễ thể hiện thuật toán. Cấu trúc của chương trình phản ánh được cấu trúc của vấn đề và cách giải quyết vấn đề (làm như thế nào?). Phương pháp hay được sử dụng để thiết kế chương trình là *phân tích từ trên xuống* (Top-Down Analysis) và *tổng hợp từ dưới lên* (Bottom up Synthesis).

Nội dung phương pháp phân tích từ trên xuống là nhìn nhận xem xét tổng quát toàn bộ vấn đề, xuất phát từ mục tiêu (đỉnh) đi xuống các thành phần trong hệ thống, chia các thành phần thành các thành phần nhỏ hơn theo một cấu trúc phân cấp chặt chẽ.

Nội dung phương pháp tổng hợp từ dưới lên là xuất phát từ các vấn đề cụ thể và cách giải quyết cụ thể, sau đó tích hợp chúng lại thành vấn đề lớn hơn và cách giải tổng quát hơn, hướng từ dưới lên trên để nhận được vấn đề cần phải giải quyết ban đầu

Cách thiết kế này gây khó khăn vì khó kiểm soát và dễ lạc hướng, khó đáp ứng đầy đủ các yêu cầu của vấn đề.

### II.3. Các cấu trúc điều khiển chuẩn

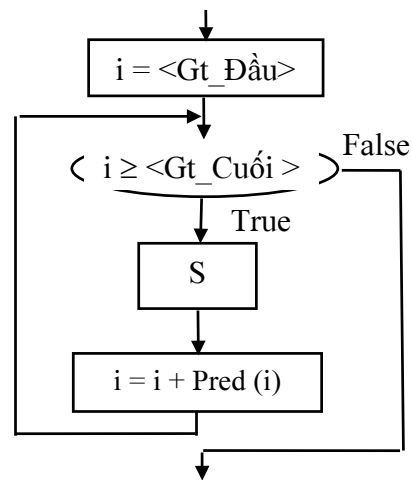
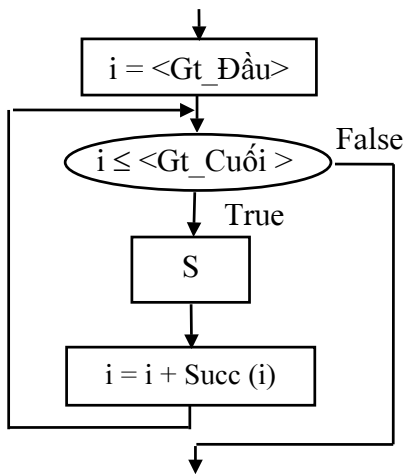
Trong chương trình, chỉ nên sử dụng 7 cấu trúc điều khiển sau đây với quy ước S là một lệnh (Statement) và C là một biểu thức điều kiện (Condition) nào đó :

Stt	Cấu trúc điều khiển	Lưu đồ tương đương	Mô tả
1	<b>Tuần tự</b> (Sequential) <b>begin</b> S1 ... Sn <b>end</b>		Được coi như là một lệnh ghép (khối), thực hiện tuần tự các lệnh S1, S2, ..., Sn.
2	<b>Rẽ nhánh</b> (Branching) <i>a) Rẽ nhánh thiếu</i> <b>if C then S</b>		Nếu C thỏa mãn (True) thì thực hiện S. Nếu C không thỏa mãn (False) thì không làm gì cả.
3	<i>b) Rẽ nhánh đủ</i> <b>if C then S1</b> <b>else S2</b>		Nếu C đúng thì thực hiện lệnh S1. Nếu C sai thì thực hiện S2.
4	<b>Lựa chọn</b> (Selection) <b>case</b> C1 : S1 C2 : S2 ... Cn : Sn <b>endcase</b>		Nếu C1 đúng thì thực hiện S1. Nếu không, nếu C2 đúng thì thực hiện S2, v.v... Cuối cùng, nếu Cn đúng thì thực hiện Sn. Nếu không thì thôi.
5	Cấu trúc lặp (Iteration) Kiểm tra điều kiện trước khi thực hiện vòng lặp :		Khi C còn đúng thì còn thực hiện S. Khi C sai thì dừng.

	<b>while C do S</b>		
--	---------------------	--	--

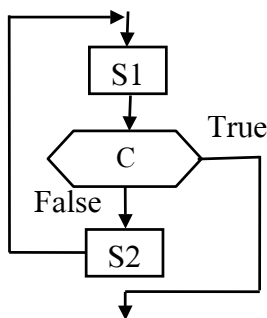
Stt	Cấu trúc điều khiển	Lưu đồ tương đương	Mô tả
6	Lặp với kiểm tra điều kiện sau khi thực hiện xong thân vòng lặp : <b>do S until C</b>		Còn thực hiện S khi C còn chưa thoả mãn (sai). Ít nhất lặp được một lần. Dừng khi C đúng.

7 Lặp hết trước số lần (for)



**for** I ← Gt\_Đầu **to** Gt\_Cuối **do** S      **for** I ← Gt\_Đầu **downto** Gt\_Cuối **do** S

Ngoài các cấu trúc lặp hay gặp thông thường trên đây, người ta còn sử dụng các cấu trúc lặp có thoát (loop exit) như sau :

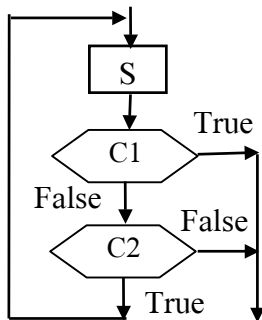


{ sử dụng khoá key để đánh dấu lối thoát,  
key không xuất hiện trong S và trong C }

key := **False**  
**While not key do begin**  
    S1  
    **if** C **then** key := **True** **else** S2  
**End**

## II.4. Một số ví dụ viết chương trình theo sơ đồ khối

Ví dụ 4 :



{ Vòng lặp này dùng **Repeat** }

**Repeat**

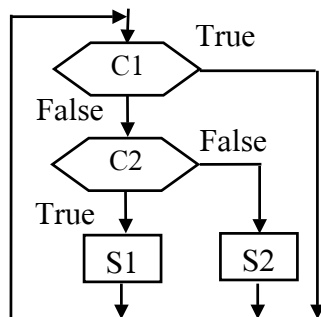
S

**Until C1 or not C2**

*Chú ý* : Có điều kiện cuối vòng lặp có thể dùng

**Repeat**

Ví dụ 5 :



{ Vòng lặp này dùng **While** }

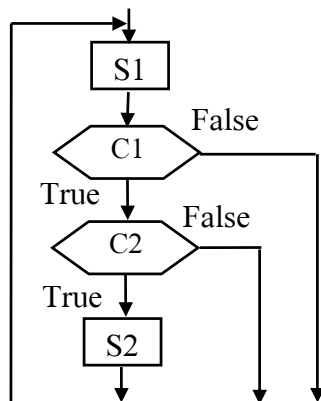
**While C1 do**

**if C2 then S1 else S2**

*Chú ý* : Có điều kiện trước vòng lặp có thể dùng

**While**

Ví dụ 6 :



{ Đây là cấu trúc loop-exit, có thể dùng **While** như sau :}

key := **False**

**While not key do begin**

S1

**if not C1 then key := True**

**else if C2 then S2**

**End**

Cấu trúc loop-exit trên đây có thể dùng **Repeat** như sau :

key := **False**

**Repeat**

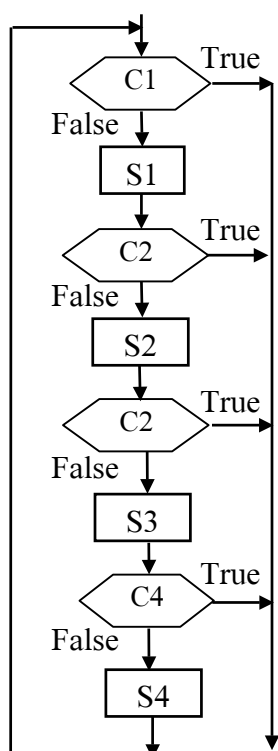
S1

**if C1 then key := True**

**else if C2 then S2**

**Until key**

Ví dụ 7 :



{ Trường hợp Loop-exit mở rộng.  
Dùng khóa key để đánh dấu lối thoát như sau :}

key := **False**

**Repeat**

**If C1 then key := True**

**Else begin**

S1

**If C2 then key := True**

**Else begin**

S2

**If C3 then key := True**

**Else begin**

S3

**If C4 then key := True**

**Else S4**

**End**

**End**

**End**

**Until key**

### III. Cấu trúc tối thiểu

Các cấu trúc điều khiển chuẩn là kết quả của những cố gắng lớn trong cuộc cách mạng về lập trình những năm 60. Những nhà tin học có tên tuổi đã đóng góp công sức là Bohm C. và Jacopini G., Dijkstra E.W. và Warier, v.v...

Để đảm bảo tính trong sáng, đơn giản và tự nhiên, người ta khuyên rằng chỉ nên xây dựng chương trình với 3 cấu trúc điều khiển cơ bản là *tuần tự*, *rẽ nhánh* và *lặp*.

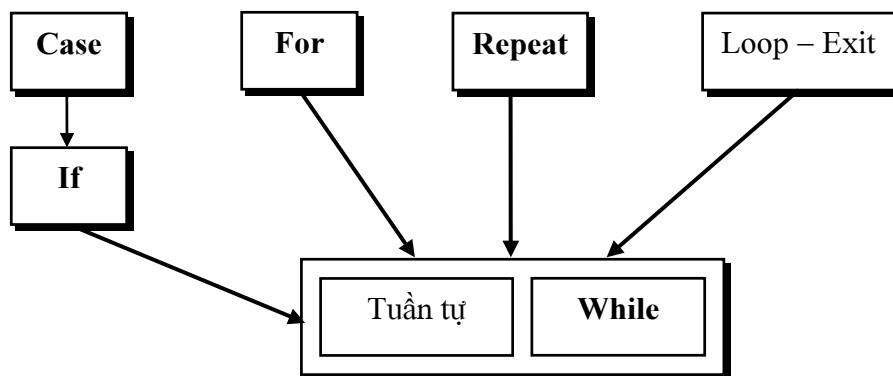
Tuy nhiên, Bohm và Jacopini đã chứng minh được rằng chỉ cần tối thiểu hai cấu trúc tuần tự và lặp là đủ.

#### Định lý Bohm và Jacopini 1986

Với mọi chương trình viết dưới dạng sơ đồ khối P (Flowchart), đều tồn tại một chương trình Q tương đương với P theo nghĩa sau :

- Với mọi dữ liệu vào  $X$  thuộc miền xác định  $X$ , ta có  $P(x) = Q(x) : P$  và  $Q$  biến đổi những cái vào giống nhau thành các ra giống nhau.
- Các thao tác trên các biến của  $Q$  là giống như của  $P$ .
- Các biến của  $Q$  cũng là các biến của  $P$ , có thể  $Q$  chứa thêm một số biến logic.
- $Q$  sử dụng hai cấu trúc điều khiển duy nhất là **tuần tự** và vòng lặp **while** (SW: Sequence & While).

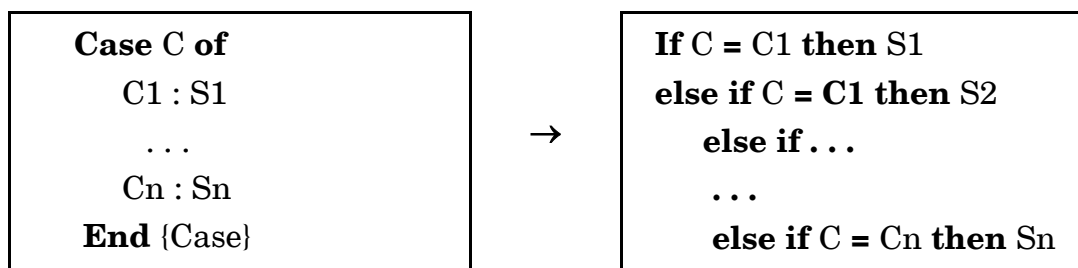
Sơ đồ chuyển cấu trúc như sau :



Hình 2.4. Chuyển về cấu trúc **tuần tự** và lặp **while** (SW)

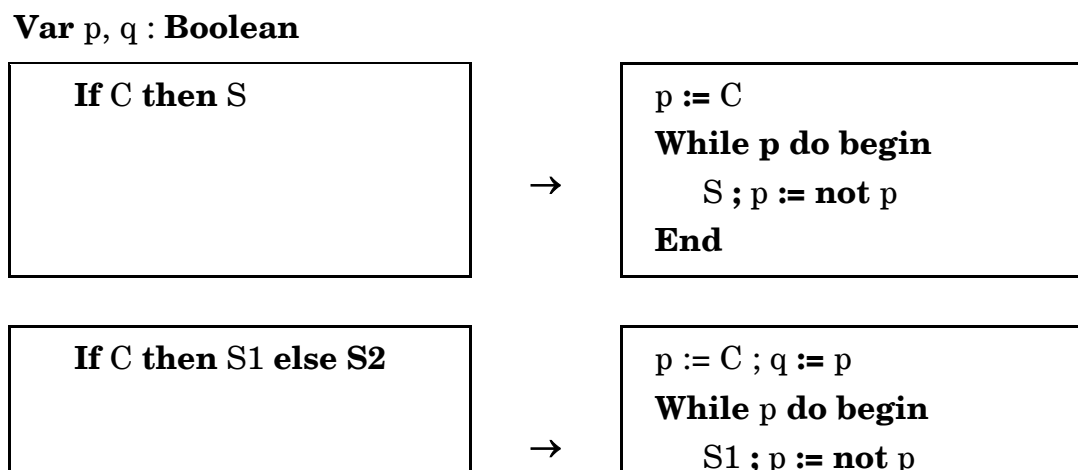
Sau đây là cách chuyển đổi của các cấu trúc **Case**, **If**, **For**, **Repeat** và **Loop - Exit** :

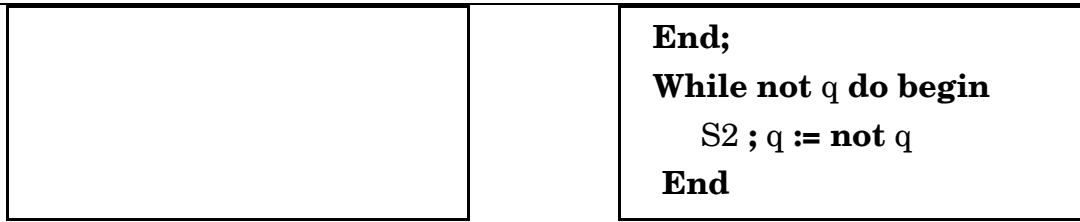
#### 1. Case → if



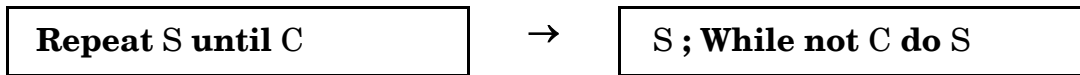
#### 2. if → SW

Dùng hai biến phụ kiểu logic để thực hiện vòng lặp **While** đúng một lần :

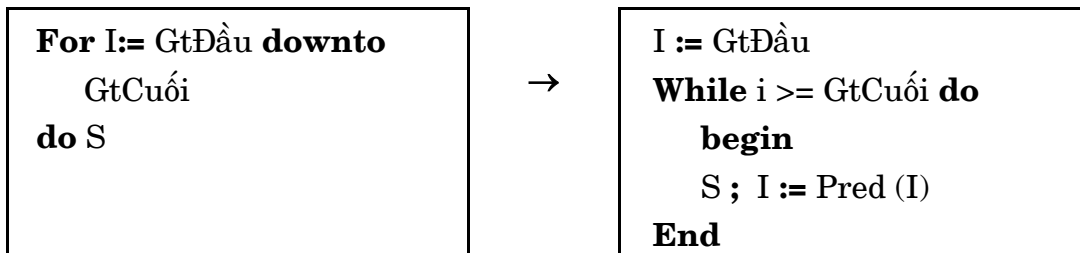
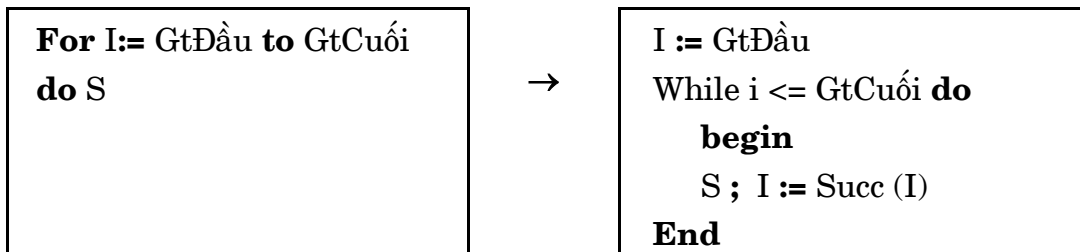




3. Repeat → SW



4. For → SW



### III.1. Các cấu trúc lồng nhau

Bản thân lệnh S trong mỗi cấu trúc điều khiển cơ bản lại có thể là một cấu trúc điều khiển khác.

*Ví dụ 8 : Với cấu trúc điều kiện*

**If C then S1 else S2**

Tại S1 và S2, ta có thể đặt các cấu trúc điều khiển khác, chẳng hạn thế S1 bởi :

**While C1 do S3**

và thế S2 bởi :

**Repeat S4 until C2**

Ta có :

**If C then**

**While C1 do S3**

**Else Repeat S4 until C2**



Đến lượt S3 và S4 lại có thể thay thế bởi các cấu trúc khác, v.v...

Với các phép thế như vậy, cấu trúc của chương trình ngày càng phức tạp và dẫn đến khó hiểu và dễ sai sót. Chính vì vậy mà người ta chú trọng triển khai chương trình từ trên xuống và viết các cấu trúc theo từng khối.

Các khối có thể thụt vào, thụt ra để phản ánh tính cấu trúc và mức độ lồng nhau của các cấu trúc.

**Nguyên tắc :** *Cấu trúc con được viết lọt vào trong (thụt vào) cấu trúc cha. Điểm vào và điểm ra của mỗi cấu trúc phải nằm trên cùng một hàng dọc.*

## IV. Lập trình đơn thể

### IV.1. Khái niệm về đơn thể

Ý tưởng cơ bản của lập trình cấu trúc là phân rã vấn đề lớn thành các vấn đề nhỏ hơn cho đến khi nhận được các vấn đề tương đối đơn giản, mỗi vấn đề này được giải quyết bởi một đơn thể chương trình (module). Mỗi đơn thể có các tính chất như sau :

#### a) Tính đơn thuần

- Chỉ giải quyết những đối tượng dữ liệu có liên hệ với nhau trong phạm vi của vấn đề.
- Có một lối vào và một lối ra, bên trong chỉ dùng những cấu trúc điều khiển chuẩn.
- Hoạt động chỉ phụ thuộc vào dữ liệu đưa vào chứ không phụ thuộc vào tình trạng trước đó của nó. Mỗi đơn thể là một hàm dữ liệu vào, kết quả tiên đoán được.

#### b) Tính chuyên biệt

- Chỉ thực hiện một chức năng, nhiệm vụ nhất định.
- Không quá dài hoặc quá ngắn (lý tưởng là mỗi đơn thể có từ 60 đến 70 dòng lệnh vừa nằm trọn trong một trang A4).
- Chỉ được khởi động bằng cách gọi.

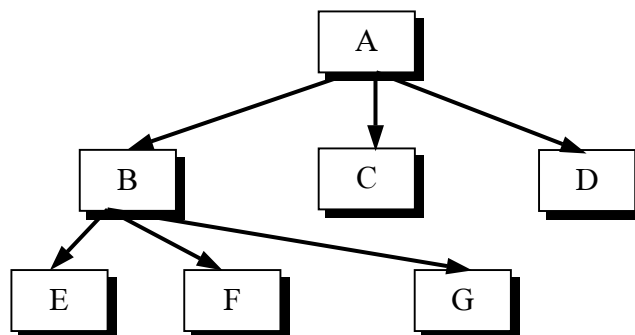
#### c) Tính độc lập

- Là một đơn vị biên dịch. Có thể viết và chạy thử độc lập.

Các ngôn ngữ lập trình bậc cao như Pascal (kỹ thuật dùng Unit), C, C++ (include các tệp chương trình ) và hầu hết các công cụ lập trình thường gặp hiện nay đều cho phép lập trình theo đơn thể.

## IV.2. Mối liên hệ giữa các đơn thể

Các đơn thể nối kết với nhau thành chương trình, tổ chức phân cấp dạng cây (tree).



Hình 2.5. Mối liên hệ giữa các đơn thể

### IV.2.1. Phân loại đơn thể

Có 4 loại đơn thể :

#### a) Đơn thể điều khiển

Đơn thể điều khiển (Director Module) có chức năng gọi các đơn thể khác xử lý.

#### b) Đơn thể xử lý

Đơn thể xử lý (Processing Module) chuyên trách một nhiệm vụ nào đó trên vùng dữ liệu độc lập. Đơn thể xử lý được đơn thể điều khiển gọi tới và sau khi thực hiện xong chức năng, đơn thể xử lý trả quyền điều khiển trở lại cho đơn thể điều khiển.

#### c) Đơn thể vào/ra

Đơn thể vào/ra (IO Module) chuyên trách vào/ra dữ liệu, có sự kiểm tra và xử lý sai sót. Đơn thể cũng do đơn thể điều khiển gọi tới giống như hoạt động của đơn thể xử lý.

#### d) Đơn thể chương trình con

(Subroutine Module) nhằm giải quyết một nhiệm vụ trọn vẹn nhưng có quan hệ với các đơn thể khác. Đơn thể chương trình con được gọi thực hiện nhiều lần trong chương trình.

### IV.2.2. Tổ chức một chương trình có cấu trúc đơn thể

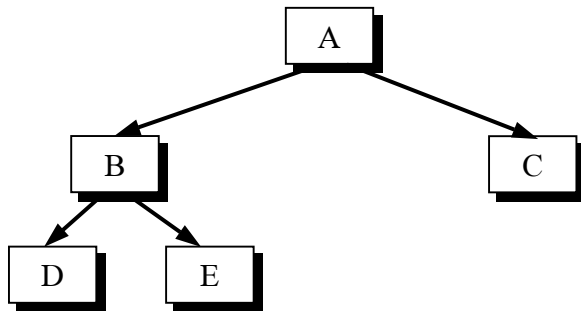
Cấu trúc chương trình gồm :

- Cấu trúc nội tại của các đơn thể.
- Mối liên hệ giữa các đơn thể.

Các đơn thể được tổ chức phân cấp dạng cây nhưng phải thỏa mãn tính *cục bộ tham chiếu* (locality of Reference) : chỉ có đơn thể mức cao hơn (cha) mới có quyền tham chiếu (gọi) đến đơn thể mức thấp hơn kề đó (con).

Những cấu trúc cây thỏa mãn tính cục bộ tham chiếu được gọi là *cấu trúc cây thuần túy* (Pure tree Structure).

Ví dụ 9 :



– D chỉ phục vụ B, chỉ có B mới có quyền điều khiển D, C không thể gọi D.

– Giữa B và D chỉ có một đường tương tác duy nhất là trao đổi tham biến.

Hình 2.6. Cấu trúc cây thuần túy

#### a) Đặc điểm của cấu trúc cây thuần túy

– Mỗi đơn thể chỉ được quyền điều khiển đơn thể con trực tiếp, giảm được tính phức tạp của chương trình.

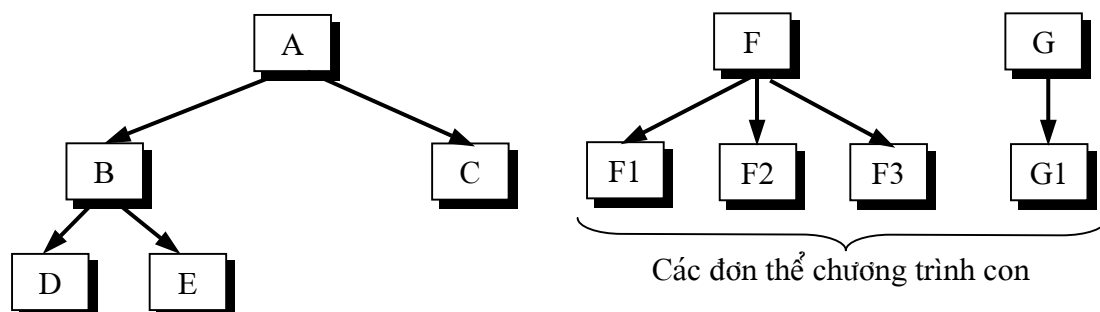
– Các nhánh hoàn toàn tách biệt nhau nên có tính tương tác tối thiểu.

*Ngoại lệ* : Nếu một công việc nào đó cần thực hiện nhiều lần, nhiều chỗ trong chương trình thì nên tổ chức thành một đơn thể chương trình con và vẽ riêng, không vẽ vào cấu trúc cây.

Như vậy, các đơn thể chương trình con chỉ đóng vai trò thư viện, một sự mở rộng của ngôn ngữ lập trình. Trường hợp đơn thể chương trình con phức tạp thì có thể tổ chức theo cấu trúc cây thuần túy.

Như vậy toàn bộ chương trình là một tập hợp các cấu trúc cây thuần túy.

Ví dụ 10 :



Hình 2.7. Cấu trúc cây thuần túy của chương trình và chương trình con

**b) Thử nghiệm chương trình trên cấu trúc cây thuần túy**

Quá trình thử nghiệm một chương trình :

- Thử nghiệm các đơn thể chương trình con trước.
- Thử nghiệm các đơn thể trong chương trình chính, từ dưới lên và riêng từng nhánh.

Cần phân biệt :

- Các đơn thể xử lý phụ thuộc vào ngữ cảnh (context) nào, là con của đơn thể nào ?
- Các đơn thể chương trình con độc lập với ngữ cảnh.

Để thử nghiệm chương trình cho trong hình vẽ trong ví dụ ở trên :

- Thử F và G trước (sau khi đã thử F1, F2, F3 và G1).
- Thử D và E rồi thử B.
- Thử C.
- Thử cả chương trình.

## V. Phát triển chương trình bằng tinh chế từng bước

### V.1. Nội dung phương pháp

Nguyên lý phát triển CHTR bằng tinh chế từng bước (hay thiết kế từ trên xuống) do Niclus Wirth (tác giả của ngôn ngữ lập trình Pascal) đề xuất vào năm 1971, trong bài báo của mình "Program Development by Stepwise Refinement".

Ban đầu, CHTR là những câu được viết bằng ngôn ngữ tự nhiên (chẳng hạn tiếng Việt) thể hiện sự phân tích tổng thể của người lập trình.

Sau đó, tại mỗi bước, mỗi câu được phân tích chi tiết hơn thành những câu khác. Có nghĩa đã phân tích một công việc thành những công việc bé hơn.

- Mỗi câu được gọi là một đặc tả (Specification).
- Mỗi bước phân tích được gọi là đã tinh chế (refine) câu (công việc) đó.

Sự tinh chế được hướng về phía ngôn ngữ lập trình sẽ dùng. Nghĩa là càng ở bước sau, những câu chữ trên ngôn ngữ tự nhiên càng đơn giản dễ hiểu hơn và được thay thế bằng các câu lệnh của ngôn ngữ lập trình. Nếu câu còn tỏ ra phức tạp, có thể coi đó là một CHTR con và tiếp tục tinh chế nó.

Trong quá trình tinh chế, cần đưa ra các cấu trúc dữ liệu tương ứng với từng bước. Như vậy sự tinh chế các đặc tả CHTR và dữ liệu là song song.

Phương pháp tinh chế từng bước thể hiện tư duy giải quyết vấn đề từ trên xuống, trong đó sự phát triển của các bước là hướng về ngôn ngữ lập trình sẽ sử dụng. Đáy của sự đi xuống trong hoạt động phân tích là các câu lệnh và các mô tả dữ liệu viết bằng ngôn ngữ lập trình.

*Ý nghĩa* : Việc lập trình có sự định hướng và có sự ngăn nắp trên giấy nháp, tránh mò mẫm thử nghiệm mang tính trực giác.

## V.2. Ví dụ minh họa

### V.2.1. Ví dụ 1

Nhập vào dãy các ký hiệu liên tiếp từ bàn phím cho đến khi kí tự dấu chấm (.) được gõ. In ra số lượng từng chữ số từ 0..9 đã đọc.

Chẳng hạn, nếu nhập vào dãy :

*Kiki1t2047655kp412.*

thì in ra :

*số chữ số 0 đã đọc = 1,*

*số chữ số 1 đã đọc = 2,*

*số chữ số 2 đã đọc = 2*

...

#### 1. Phác thảo lời giải

Cần in ra 10 giá trị ứng với các chữ số từ 0..9. Có thể dùng 10 biến đơn ZERO, MOT, HAI, BA... nhưng tốt nhất nên dùng một mảng có 10 phần tử :

Số ['0'] chứa kí tự '0' đã đọc;

Số ['1'] chứa kí tự '1' đã đọc;

v.v...

Ta mô tả như sau :

```
Type dãy = array [ '0'..'9' ] of integer;
```

```
var số = dãy;
```

```
    c: Char; { ký tự được đọc }
```

Từ đó lời giải có thể được viết như sau :

```
Repeat
```

```
    đọc_một_kí_tự; { là ký tự c }
```

```
    if kí_tự_là_chữ_số then đếm_chữ_số_đó;
```

```
    { ví dụ, nếu đọc '2' thì tăng số ['2'] lên 1 }
```

```
Until c = dấu_chấm;
```

```
for c := '0' to '9' do
  writeln('số các chữ số', c, ' đã đọc =', số [ c ] : 2);
```

Ta tinh chế bước *kí tự là chữ số* bằng cách chuyển ra dạng biểu thức Pascal như sau :

```
('0' < c) and (c <= '9')
```

Việc *đọc một kí tự* được viết như sau : Read (c);

Dấu chấm có thể dùng hằng :

```
Const dấu chấm '=';
```

Ta thấy trước lúc đếm, các phần tử của mảng số phải bằng 0. Ta có :

```
for c := '0' to '9' do số [ c ] := 0;
```

Bây giờ ta có chương trình hoàn chỉnh như sau :

```
Program Đếm chữ số;
Const dấu chấm = '.';
Type dãy = array ['0'..'9'] of integer;
Var số: dãy;
    c: char;
begin
  for c := '0' to '9' do số [ c ] := 0;
  writeln ('Hãy gõ vào các kí tự');
  writeln ('và kết thúc bằng dấu chấm (.) :');
  Repeat
    Read (c);
    if ('0' <= c) and (c <= '9') then
      số [ c ] := số [ c ] + 1;
  Until c = dấu chấm;
  for c := '0' to '9' do
    writeln ('Số các chữ số', c, ' đã đọc =', số [ c ] : 2)
  Readln
end.
```

Cho chạy chương trình ta được kết quả như sau :

```
Hãy gõ vào các kí tự
và kết thúc bằng dấu chấm (.) :
ytr7657g858450020820.
Số các chữ số 0_ đã đọc = 4
Số các chữ số 1_ đã đọc = 0
Số các chữ số 2_ đã đọc = 2
Số các chữ số 3_ đã đọc = 0
Số các chữ số 4_ đã đọc = 1
Số các chữ số 5_ đã đọc = 3
```

Số các chữ số 6\_đã đọc = 1  
 Số các chữ số 7\_đã đọc = 2  
 Số các chữ số 8\_đã đọc = 3  
 Số các chữ số 9\_đã đọc = 0

### V.2.2. Bài toán 8 quân hậu

Hãy đặt 8 quân hậu lên bàn cờ vua (có 8 x 8 ô) sao cho không có quân nào ăn được quân nào? Một quân hậu có thể ăn được bất cứ quân nào nằm trên cùng cột, cùng hàng hay cùng đường chéo thuận nghịch với nó.

Bài toán này do Carl Friedrich Gauss đưa ra vào năm 1850 nhưng không có lời giải hoàn toàn theo phương pháp giải tích. Lý do là loại bài toán này không phù hợp với các phương pháp giải tích mà phải tìm cách khác để giải trên MTĐT, có thể thử đi thử lại nhiều lần.

Niclaus Wirth trình bày phương pháp *thử-sai* (trial-and-error) như sau :

Đặt một quân hậu vào cột 1 (trên một hàng tùy ý);

Đặt tiếp một quân hậu thứ hai sao cho 2 quân không ăn nhau;

Tiếp tục đặt quân thứ 3, v.v...

Lời giải có dạng một vòng lặp như sau :

*Xét-cột-đầu;*

Repeat

*Thử\_cột;*

if *an\_toàn* then begin

*Đặt\_quân\_hậu\_vào;*

*Xét\_cột\_kế\_tiếp;*

end

else *Quay\_lại;*

until *Đã\_xong\_với\_cột\_cuối* or *Đã\_quay\_lại\_quá\_cột\_đầu;*

Các công việc được tinh chế dần dần bằng cách chọn các việc đơn giản, có cách giải ngay để tiến hành trước như sau :

Gọi bàn cờ vua 8 x 8 gồm các ô (i, j) ở cột j, hàng i với j=1..8 và i=1..8, ta có :

*Xét\_cột\_đầu* : Bắt đầu với cột j=1.

*Xét\_cột\_kế\_tiếp* : Tức là chuyển qua xét cột kế tiếp và chuẩn bị xét hàng đầu tiên :

$j := j + 1; i := 0;$

*Đã\_xong\_với\_cột\_cuối* : Lúc này đã xong cả 8 cột, quân hậu cuối cùng đã được đặt vào bàn cờ : thành công, ta có biểu thức :

$j > 8$

*Đã quay lại quá cột đầu*: Tức là đã lùi quá cột đầu tiên : tình trạng bế tắc xảy ra : không tìm ra lời giải !

$$j < 1$$

*Thử cột* : Tìm xem có thể đặt quân hậu tại hàng nào ở cột đang xét. Bước *Thử cột* sẽ có dạng :

```
repeat
```

```
  Xét_một_hàng ; { là hàng thứ i }
```

```
  Kiểm_tra_an_toàn ; { khi đặt quân hậu vào hàng này }
```

```
Until An_toàn or Đã_xét_đến_hàng_cuối;
```

Lúc đầu  $I=0$ , việc *Xét\_một\_hàng* tức :  $i := i+1$

Từ đó ta có ngay *Đã\_xét\_đến\_hàng\_cuối* tức là :  $i = 8$

Lúc này người ta tìm cách biểu diễn dữ liệu tương ứng vì các công việc đã có vẻ "mịn" rồi. Theo lời khuyên của Niclaus Wirth, sự biểu diễn dữ liệu càng trì hoãn lâu càng tốt (đến khi không thể trì hoãn được nữa) !

Vì bàn cờ có  $8 \times 8$  ô nên có thể nghĩ ngay đến việc sử dụng một ma trận Boolean hai chiều để biểu diễn :

```
Var B : array [ 1..8, 1..8] of Boolean;
```

$B[i, j]$  có giá trị true nếu có quân hậu ở hàng  $i$ , cột  $j$ .

Tuy nhiên, cách biểu diễn này gây khó khăn cho việc kiểm tra hai đường chéo có 2 quân hậu nào ăn nhau không theo luật cờ vua ?

Bây giờ ta dùng 3 dãy Boolean  $a, b, c$  với :

$a[i] = \text{true}$  nếu không tồn tại quân hậu nào nằm trên hàng  $i$ .

$b[k] = \text{true}$  nếu không tồn tại quân hậu nào nằm trên đường chéo thuận thứ  $k$ .

$c[l] = \text{true}$  nếu không tồn tại quân hậu nào nằm trên đường chéo nghịch thứ  $l$ .

Với mỗi ô  $(i, j)$  hàng  $i$  cột  $j$ , ta có quan hệ như sau :

–đường chéo thuận thứ  $k$  thoả mãn  $i + j = k$ ;

–đường chéo nghịch thứ  $l$  thoả mãn  $i - j = l$ ;

Vì vậy, nếu :  $1 \leq i, j \leq 8$  thì :  $2 \leq k \leq 16$  và :  $-7 \leq l \leq 7$ .

Ta có các mảng  $a, b, c$  như sau :

```
var a : array [ 1..8] of boolean;
    b : array [ 2..16] of boolean;
    c : array [ -7..7] of boolean;
```



Để biểu diễn sự kiện đặt quân hậu tại cột  $j$  vào hàng  $i$ , ta dùng dãy nguyên  $x$  sao cho  $x[j] = i$  nếu như có một quân hậu ở ô  $(i, j)$  :

```
var x : array [1..8] of integer;
```

Việc đặt quân hậu vào ô  $(i, j)$  sẽ làm cho :

```
a[i] = b[i+j] = c[i-j] = false
```

*Kiểm\_tra\_an\_toàn* : Cho đến lúc này, chưa có hai quân hậu nào trong số những quân đã đặt lên bàn cờ có thể ăn lẫn nhau. Điều kiện *An\_toàn* để đặt quân hậu vào ô  $(i, j)$  là :

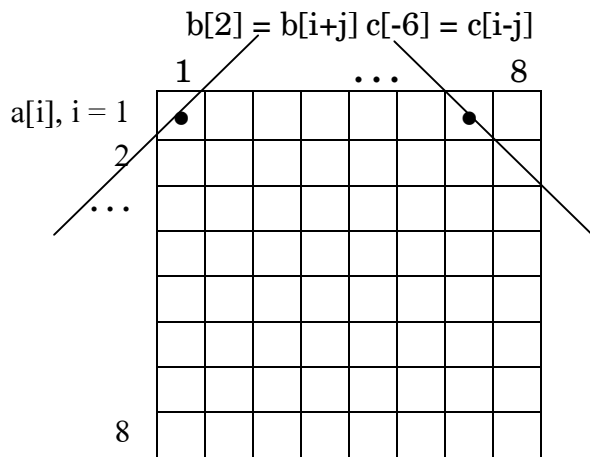
```
a[i] = b[i+j] = c[i-j] = true;
```

Bằng cách sử dụng một biến logic :

```
Var Antoan: Boolean;
```

Việc *Kiểm\_tra\_an\_toàn* được dịch ra Pascal như sau :

```
An_toàn := a[i] and b[i + j] and c[i - j] ;
```



Hình 2.8. Bàn cờ vua cho bài toán tám quân hậu

Đặt quân hậu vào ô  $(i, j)$  *Đặt\_quân\_hậu\_vào* sẽ là :

```
x[j] := i;
a[i] := false;
b[i+j] := false;
c[i-j] := false;
```

Tiếp tục tình chế bước phức tạp nhất là *Quay\_lại* :

*Quay\_lại* : là quay lại một cột ở trước cột đang xét để đặt lại quân hậu cho cột đó khi tình thế hiện trạng là bế tắc.

Bước *Quay\_lại* có dạng :

```
Xét_lại_cột_trước;
```

```
if not Đã_quay_lại_quá_cột_đầu then begin
```

```
  Bỏ_quân_hậu_ở_cột_đó; { tức cột trước cột đang xét, ô  $(i, j)$  }
```

```
  if Đang_ở_hàng_cuối_cùng then begin
```

```

    Xét_lại_cột_trước ;
    if not Đã_quay_lại_quá_cột_đầu then
        Bỏ_quân_hậu_ở_cột_đó
    end
end;

```

Để dàng ta thấy *Xét\_lại\_cột\_trước* tức là :

```
j = j - 1;
```

Còn *Đã\_quay\_lại\_quá\_cột\_đầu* thì đã xét trước đây, tức là :

```
j < 1;
```

Thao tác *Bỏ\_quân\_hậu\_ở\_cột\_đó* sẽ có dạng:

```
i := x [ j ] ; a [ i ] := true ; b [ i+j ] := true ; c [ i-j ] := true ;
```

Chương trình hoàn chỉnh như sau :

```

Program TamQuânHau;
Uses Crt;
Const Hau='Q ';ov='#';
Var x: array[1..8] of Integer;
    a: array[1..8] of Boolean;
    b: array[2..16] of Boolean;
    c: array[-7..7] of Boolean;
    i,j: Integer;
    antoan: Boolean;
Begin
for i:=1 to 8 do a[ i ]:=true;
for i:=2 to 16 do b[ i ]:=true;
for i:=-7 to 7 do c[ i ]:=true;
j:=1; i:=0;
repeat
repeat
i:=i+1;
antoan:=a[ i ] and b[ i+j ] and c[ i-j ] ;
until antoan or (i=8);
if antoan then begin
x[ j ]:=i;
a[ i ]:=false; b[ i+j ]:=false; c[ i-j ]:=false;
j:= j+1; i:= 0
end else begin
j:=j-1;
if j>=1 then begin
i:=x[ j ] ;
a[ i ]:=true; b[ i+j ]:=true; c[ i-j ]:=true;
if i=8 then begin

```

```

        j:=j-1;
        if j>=1 then begin
            i:=x[ j] ;
            a[ i ]:=true; b[ i+j] :=true; c[ i-j] :=true
        end
    end
end
end
until (j>8) or (j<1);
if j<1 then writeln('Khong co loi giai!')
else
for i:=1 to 8 do begin
    for j:=1 to 8 do
        if x[ j]=i then write(Hau) else write(ov);
        writeln
    end;
end;
readln
end.

```

**Kết quả chạy chương trình như sau :**

```

Turbo Pascal Version 7.0 Copyright (c) 1983,92 Borland International
Q # # # # # # #
# # # # # # Q #
# # # # Q # # #
# # # # # # # Q
# Q # # # # # #
# # # Q # # # #
# # # # # Q # #
# # Q # # # # #

```

### V.3. Sửa đổi chương trình

Chương trình viết xong chạy tốt chưa có nghĩa quá trình lập trình đã xong. Do nhu cầu, có thể cần sửa đổi lại theo một cách nào đó cho phù hợp. Nhờ phương pháp tinh chế từng bước mà người lập trình có thể dễ dàng nhìn thấy những chỗ cần chỉnh sửa trong chương trình. Đây là khả năng duy trì (Maintainability) của phương pháp.

Một đặc tính khác của phương pháp tinh chế từng bước là tính phổ cập (portability) của chương trình : ta dễ dàng chuyển đổi sang một môi trường khác, tức là chuyển sang một ngôn ngữ lập trình khác, hoặc một hệ thống máy tính khác. Để minh họa, ta xét bài toán 8 quân hậu tổng quát như sau :

*Tìm tất cả các phương án có thể đặt 8 quân hậu lên bàn cờ sao cho không có hai quân nào ăn lẫn nhau.*

Từ tinh chế lần 1 trong mục trước, ta cần có hai sửa đổi như sau :

- Khi đã đến cột cuối cùng và đặt quân hậu cuối cùng vào bàn cờ, ta in lời giải ra nhưng chưa kết thúc chương trình ngay mà tiếp tục quay trở lại để tìm lời giải khác.
- Chương trình ngừng khi sự quay lại đã quá cột đầu.

Lời giải có dạng phác thảo như sau :

```

Xét_cột_đầu ;
repeat
  Thử_cột ;
  if An_toàn then begin
    Đặt_quân_hậu_vào ;
    Xét_cột_kế ;
    if Cột_kế_vượt_quá_cột_cuối_cùng then begin
      In_ra_lời_giải ;
      Quay_lại
    end
  end else Quay_lại
Until Đã_quay_lại_quá_cột_đầu ;

```

Từ đây, với các bước làm mịn đã giải quyết ở mục trước, ta có thể viết lại thành chương trình hoàn chỉnh.

### Bài toán mã đi tuàn

Cho bàn cờ  $n \times n$  ô và một quân mã đang ở tọa độ  $x_0, y_0$ . Hãy tìm cách cho quân mã đi theo luật cờ vua để qua hết tất cả các ô của bàn cờ, mỗi ô đi qua đúng một lần ?

Cách giải quyết để quân mã đi qua hết  $n^2 - 1$  ô của bàn cờ là tại mỗi ô mà quân mã đang đứng, hãy xác định xem có thể thực hiện một nước đi kế tiếp nữa hay không ? Như vậy thuật toán để tìm nước đi kế tiếp có thể viết thành thủ tục đệ quy dạng phác thảo như sau :

```

Procedure Thử_nước_đi_kế ;
Begin
  Khởi_động_nước_đi_có_thể
  Repeat
    Chọn_một_nước_đi
    if OK then begin
      Thực_hiện_nước_đi
      if Chưa_hết_nước then begin
        Thử_nước_đi_kế ;
        if NotOK then Xoá_nước_trước
      end else Thành_công
    end
  end
Until Đi_được or (Hết_nước_đi) ;
Kết_thúc
End ;

```

Bây giờ ta cần tìm cấu trúc dữ liệu để biểu diễn bàn cờ  $n \times n$  ô, mỗi ô có tọa độ  $(i, j)$ , với  $1 \leq i, j \leq n$ . Dễ dàng ta tìm được mô tả như sau :

```
Type Idx = 1..n;
Var H: Array[Idx, Idx] of Integer;
```

Trong mô tả trên, thay vì sử dụng giá trị kiểu Boolean để đánh dấu ô đó đã được đi qua chưa, ta đưa vào giá trị kiểu Integer để dò theo quá trình di chuyển của quân mã theo quy ước như sau :

$H[x, y] = 0$       ô  $\langle x, y \rangle$  chưa được quân mã đi qua

$H[x, y] = i$       ô  $\langle x, y \rangle$  đã được quân mã đi qua ở nước thứ  $i$ ,  $1 \leq i \leq n^2$

Để chỉ một nước đi có thành công hay không, ta sử dụng biến Boolean  $q$  với quy ước như sau :

$q = \text{true}$       nước đi thành công

$q = \text{false}$       không có nước đi

Ta thấy điều kiện *Chưa\_hết\_nước* được biểu diễn bởi biểu thức :  $i \leq n^2$

Giả sử gọi  $u, v$  là tọa độ nước đi kế tiếp của quân mã theo luật cờ vua thì điều kiện OK phải thoả mãn :

- Ô mới  $\langle u, v \rangle$  phải thuộc vào bàn cờ, nghĩa là  $1 \leq u \leq n$  và  $1 \leq v \leq n$ .
- Quân mã chưa đi qua ô  $\langle u, v \rangle$ , nghĩa là  $H[u, v] = 0$ .

Bằng cách xây dựng tập hợp :

```
Var s: set of Idx;
```

biểu thức điều kiện OK bây giờ có thể viết :

$$(u \text{ in } s) \text{ and } (v \text{ in } s) \text{ and } H[u, v] = 0$$

Để ghi nhận nước đi hợp lệ *Thực\_hiện\_nước đi*, ta sử dụng phép gán :

```
H[ u, v] := i;
```

Từ đó, việc *Xoá\_nước\_trước* có thể sử dụng phép gán :

```
H[ u, v] := 0
```

Để ghi nhận kết quả lời gọi đệ quy, ta sử dụng biến Boolean  $q_1$  cho biểu thức điều kiện *Đi\_được*. Như vậy, *Thành\_công* sẽ là :

```
q1 := true
```

và *Kết\_thúc* sẽ là :

```
q := q1
```

Bây giờ ta có lời giải mịn hơn như sau :

```
Procedure Try(i:Integer; x, y : Idx; Var q: Boolean);
```

```

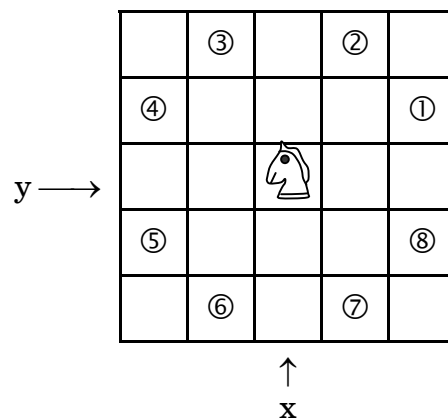
Var  u, v: Integer;
      q1: Boolean;
Begin
  Khởi_động_nước_đi_có_thể
  Repeat
    Chọn_một_nước_đi
    if (u in s) and (v in s) and H[u, v]=0 then begin
      H[u, v] := i;
      if n < sqr(n) then begin
        Try(i+1, u, v, q1);
        if not q1 then H[u, v] := 0
      end else q1 := true
    end
  end
  Until q1 or (Hết_nước_đi);
  q := q1
End;

```

Cho đến lúc này, ta chưa xét đến luật đi của quân mã, nghĩa là chương trình xây dựng ở trên độc lập với luật cờ vua với chủ ý giảm nhẹ những chi tiết chưa cần thiết khi phát triển chương trình.

Như vậy ta vẫn còn hai việc chưa giải quyết là : *Khởi\_động\_nước\_đi\_có\_thể* và *Chọn\_một\_nước\_đi*.

Cho trước một tọa độ bất kỳ  $\langle x, y \rangle$  của quân mã trên bàn cờ, ta có thể có tám ô  $\langle u, v \rangle$  được đánh số từ 1..8 (theo chiều ngược kim đồng hồ) mà quân mã có thể nhảy đến như hình dưới đây :



Hình 2.9. Các vị trí khác nhau của quân mã

Để có được  $\langle u, v \rangle$ , từ  $\langle x, y \rangle$ , ta cần xác định giá trị chênh lệch theo tọa độ. Ta sẽ dùng hai mảng một chiều a và b, mỗi mảng sẽ có kích thước 8 phần tử, để lưu giữ 8 giá trị chênh lệch theo tọa độ  $\langle x, y \rangle$ , với quy ước chiều đi  $\uparrow$  và  $\rightarrow$  mang dấu +, chiều đi  $\leftarrow$  và  $\downarrow$  mang dấu -. Ta có khai báo như sau :

```
Var a, b: Array[1..8] of integer;
```

Chẳng hạn nếu cho  $\langle x, y \rangle = \langle x_0, y_0 \rangle$  với điều kiện  $n-2 \geq x_0, y_0 \geq 3$  thì ta có thể có 8 cặp giá trị như sau :

- ①  $a[1] := 2; \quad b[1] := 1;$
- ②  $a[2] := 1; \quad b[2] := 2;$
- ③  $a[3] := -1; \quad b[3] := 2;$
- ④  $a[4] := -2; \quad b[4] := 1;$
- ⑤  $a[5] := -2; \quad b[5] := -1;$
- ⑥  $a[6] := -1; \quad b[6] := -2;$
- ⑦  $a[7] := 1; \quad b[7] := -2;$
- ⑧  $a[8] := 2; \quad b[8] := -1;$

Bây giờ, để đánh số các nước đi có thể, ta sử dụng một biến  $k$  nguyên,  $k$  sẽ nhận giá trị trong phạm vi 1..8. Như vậy, đầu thủ tục, việc *Khởi động nước đi có thể* tương ứng với lệnh gán :

```
k := 0;
```

Việc *Chọn một nước đi* tương ứng với các lệnh gán :

```
k := k + 1;
q1 := false;
u := x + a[k]; v := y + b[k];
```

Còn biểu thức điều kiện *Hết nước đi* sẽ tương ứng với :  $k = 8$

Thủ tục đệ quy được bắt đầu bởi toạ độ  $\langle x_0, y_0 \rangle = \langle 1, 1 \rangle$ , kể từ nước đi  $k=2$ , các ô của bàn cờ đều có thể là đích của quân mã với khởi động :

```
for i:=1 to n do for j:=1 to n do H[i, j] := 0;
```

Lời gọi thủ tục như sau :

```
H[1, 1] := 1; Try(2, 1, 1, q);
```

Cuối cùng là một thay đổi nhỏ bằng cách thêm biến nguyên  $nsq$  để tính  $sqr(n)$  ngoài thủ tục. Chú ý rằng  $n \geq 5$ . Sau đây là chương trình hoàn chỉnh :

*Chương trình mã đi tuần :*

```
PROGRAM KnightsTour;
Uses CRT, Dos;
Const NMax=50;
Type Idx = 1..Nmax;
Var i, j: Idx;
    N, Nsq: integer;
    q: Boolean;
    s: set of Idx;
    H: Array[Idx, Idx] of Integer;
    a, b: Array[1..8] of integer;

    gio, phut, giay, hund : Word; { Để tính thời gian }
Procedure Try(i:Integer; x, y : Idx; Var q: Boolean);
```



```

Var k, u, v:Integer; q1: Boolean;
Begin
k:= 0;
  Repeat
    k:= k + 1; q1:= false; u:= x + a[k]; v:= y + b[k];
    if (u in s) and (v in s) and (H[u, v]=0) then begin
      H[u, v] := i;
      if i < Nsq then begin
        Try(i+1, u, v, q1);
        if not q1 then H[u, v] := 0
      end else q1:= true
    end
  Until q1 or (k=8);
  q := q1
End { Try };
Begin { KnightsTour main }
  ClrScr;
  a[1]:= 2;   b[1]:= 1;
  a[2]:= 1;   b[2]:= 2;
  a[3]:= -1;  b[3]:= 2;
  a[4]:= -2;  b[4]:= 1;
  a[5]:= -2;  b[5]:= -1;
  a[6]:= -1;  b[6]:= -2;
  a[7]:= 1;   b[7]:= -2;
  a[8]:= 2;   b[8]:= -1;
  Write('Cho N = '); Readln(N);
  While (N>1) and (N<Nmax) do begin
    { Giờ bắt đầu tính }
    GetTime(gio, phut, giay, hund);
    Writeln('Bắt đầu =', gio:2,':', phut:2,':', giay:2,':',
    hund);
    nsq:= sqr(N);
    s:=[1..n];
    for i:=1 to n do for j:=1 to n do H[i, j]:= 0;
    H[1, 1]:= 1; Try(2, 1, 1, q);
    if q then
      for i:=1 to N do begin
        for j:=1 to N do
          write(h[i, j]:5);
        Writeln
      end
    else Writeln('Không có lời giải !');
    { Giờ bắt đầu tính }
    GetTime(gio, phut, giay, hund);
    Writeln('Kết thúc=', gio:2,':', phut:2,':', giay:2,':',
    hund);
    Write('Cho N = '); Readln(N);
  End {While}
End.

```

Sau đây là kết quả với  $N = 5$  :

Cho  $N = 5$

Bắt đầu = 5:59:57:30

1	6	15	10	21
14	9	20	5	16
19	2	7	22	11
8	13	24	17	4
25	18	3	12	23

Kết thúc = 5:59:57:36

**Kết quả với  $N = 6$  :**

Cho  $N = 6$

Bắt đầu = 6: 0:40:80

1	16	7	26	11	14
34	25	12	15	6	27
17	2	33	8	13	10
32	35	24	21	28	5
23	18	3	30	9	20
36	31	22	19	4	29

Kết thúc = 6: 0:41:79

## VI. Phụ lục - Đơn vị trong Turbo Pascal

Đơn vị (Unit) trong Turbo Pascal thể hiện tính cấu trúc trong lập trình : cho phép chia chương trình lớn thành một hệ thống phân cấp gồm một chương trình chính và nhiều đơn vị chương trình con.

### VI.1. Giới thiệu Unit

Unit là tập hợp khai báo các hằng, các kiểu dữ liệu, các biến, các thủ tục và hàm có quan hệ với nhau để đưa vào sử dụng trong chương trình chính.

Mỗi đơn vị, được cất giữ trên thiết bị nhớ phụ (đĩa từ) dưới dạng một tệp chương trình Pascal \*.PAS và được dịch (compile) riêng rẽ. Kết quả dịch là một tệp mới có phần mở rộng là \* .TPU.

Để gọi các Unit, trong phần đầu chương trình sử dụng lệnh :

```
USES <tên Unit>
```

trong phần đầu chương trình.

Thư viện các chương trình mẫu Turbo Pascal có 8 Unit chuẩn như sau :

<b>System</b>	chứa các hàm và thủ tục thư viện thông dụng mức hệ thống : xử lý tệp, xử lý chuỗi, tính các hàm toán học. System được gọi mặc nhiên mà không cần khai báo : <b>USES System</b>
<b>Dos</b>	cung cấp các chức năng của hệ điều hành MS-DOS
<b>Crt</b>	để điều khiển các thiết bị vào (bàn phím) và ra (màn hình) : Goto XY, Clrscr...
<b>Graph</b>	cung cấp các khả năng đồ họa (graphics) cho các loại màn hình khác nhau : Hercule, CGA, EGA, VGA...
<b>Turbo3</b>	để tương thích với các chương trình Turbo Pascal V3.0 đã có.
<b>Graph3</b>	thực hiện các chương trình con đồ họa theo kiểu con rùa (tortoise) của Turbo Pascal V3.0

### VI.2. Cấu trúc của Unit

Unit do NSD tạo ra. Mỗi Unit được đặt trong một tệp chương trình, gồm những thành phần như sau :

#### a) Phần tên của Unit (Unit Heading)

```
Unit <tên Unit>
```

**b) Phần giao tiếp (Interface Section)****Interface**

{ Các khai báo sau đây không bắt buộc phải có }

**Uses** <Ds các Units dùng cho Unit này>

**Const** <Ds các hằng>

**Type** <Ds các mô tả kiểu>

**Var** <Ds các khai báo biến>

<Ds các phần đầu của các thủ tục và hàm>

Các khai báo Const, Type và Var sau Interface cho Unit cũng dùng được tại nơi sử dụng đơn vị này. Ta nói chúng là "thấy được" (Visible).

Danh sách các tên thủ tục và hàm được các Unit khác dùng đến sẽ được khai báo đầy đủ trong phần tiếp theo :

**c) Phần hiện thực (Implementation Section)****Implementation**

{ Các khai báo sau đây không bắt buộc phải có }

**Uses** <Ds các Uses sử dụng đến>

**Label** <Ds các nhãn>

**Const** <Ds các hằng>

**Type** <Ds các mô tả kiểu>

**Var** <Ds các khai báo biến>

<Các mô tả hàm và/hoặc thủ tục>

Các mô tả hàm và/hoặc thủ tục trong phần này gồm có :

- Các hàm và /hoặc thủ tục đã mô tả ở phần giao tiếp.
- Các hàm và/hoặc thủ tục nội bộ dùng riêng trong phần hiện thực (không khai báo trong phần giao tiếp).

Các mô tả nhãn, hằng, kiểu dữ liệu, biến và các hàm, thủ tục nội bộ trong phần hiện thực của một Unit là không dùng được tại nơi sử dụng Unit này. Ta gọi chúng là "bị dấu" (Hidden).

**d) Phần khởi động (Initialization section)****Begin**

<Các lệnh Pascal>

Phần này có thể vắng mặt nhưng **end.** phải có mặt !

```
end.
```

Khi nơi sử dụng một Unit có phần khởi động Initialization thì phần khởi động của Unit này sẽ được gọi chạy trước khi thân của nơi sử dụng chạy.

Nếu có một nơi sử dụng nhiều Unit thì phần khởi động của các Unit đó sẽ được chạy theo thứ tự xuất hiện của tên của các Unit trong khai báo Uses của nơi gọi.

Thường thì trong phần khởi động của một đơn vị, người ta làm các động tác chuẩn bị như khởi gán cho các biến, mở các tệp, thông báo chế độ chạy chương trình...

### VI.3. Cách sử dụng Unit

#### a) Một chương trình hay một Unit có thể sử dụng nhiều Unit khác

Sử dụng một Unit có nghĩa là được quyền sử dụng các hằng, kiểu dữ liệu, biến, các hàm và/hoặc thủ tục đã được khai báo trong phần giao tiếp Interface của Unit đó.

Ví dụ : Để sử dụng các Units có tên lần lượt là U1, U2, ..., Un, dùng mệnh đề Uses đặt sau khai báo Program trong chương trình :

```
Uses U1, U2, ...,Un ;
```

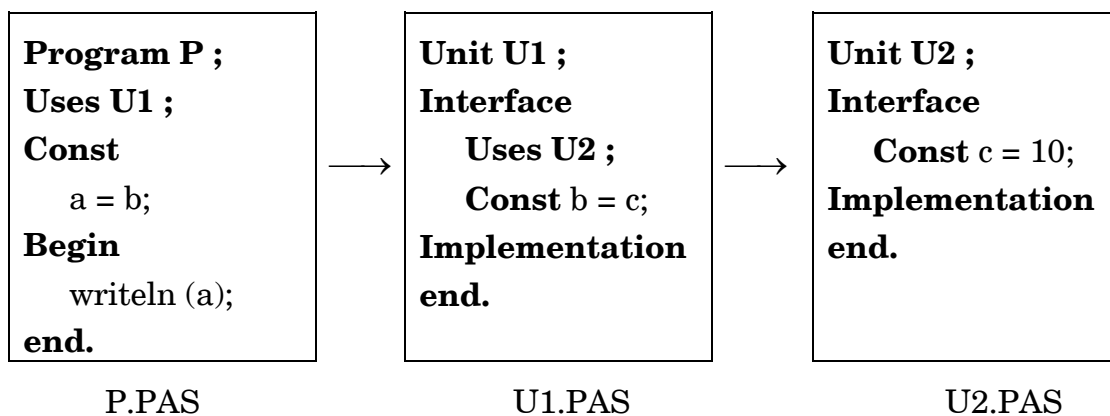
#### b) Khi có một sửa đổi nào đó trong phần Interface của một Unit

Khi đó, những (tệp) chương trình hay (tệp) Unit đó cần được dịch (Compile) lại.

Khi trong một Unit chỉ có sự sửa đổi ở phần Implementation hay ở phần Initialization thì không cần dịch lại những (tệp) chương trình hay (tệp) Unit sử dụng Unit đó.

Ví dụ 11 :

Giả sử chương trình P gọi U1 và U1 sử dụng U2 :



Nếu trong U2, đổi c=5 chẳng hạn thì cần phải dịch lại U2.PAS và U1.PAS (vì U1 sử dụng U2) và dịch lại P.PAS.

**c) Trùng tên hằng, biến, kiểu dữ liệu và tên CT con (hàm, thủ tục)**

Để phân biệt, người ta đặt trước tên trùng đó tên Unit có chứa tên này và cách một dấu chấm (.). Riêng chương trình chính (khai báo Program) thì không cần.

Ví dụ 12 :

Giả sử chương trình P sử dụng các Unit U1 và U2.

Nếu trong P, trong Interface của U1 và U2 đều có khai báo biến i thì để phân biệt i nào là của P, i nào là của U1 và U2 người ta viết :

```

i      {i của P}
U1.i   {i của U1}
U2.i   {i của U2}

```

**VI.4. Ví dụ về Unit**

Viết chương trình tính nhiều lần diện tích hình tròn với bán kính nhập vào từ bàn phím, cho đến khi bán kính nhập vào là 0 thì dừng.

Gọi chương trình chính là HINHTRON và Unit sử dụng để tính diện tích hình tròn là DTHTRON, ta có :

```

Program HINHTRON ; {Tập HINHTRON.PAS}
Uses Crt, DTHTRON ;
Var bk : Real ;
Begin
  Write ('Số Pi = ', Pi) ; {hằng Pi khai báo trong Unit DTHTRON}
  Repeat
    Write (bán kính =) ; Readln (bk) ;
    if bk > 0 then
      Writeln ('Diện tích = ', Dientich (bk) ) ;
  Until bk = 0
End. {HINHTRON}

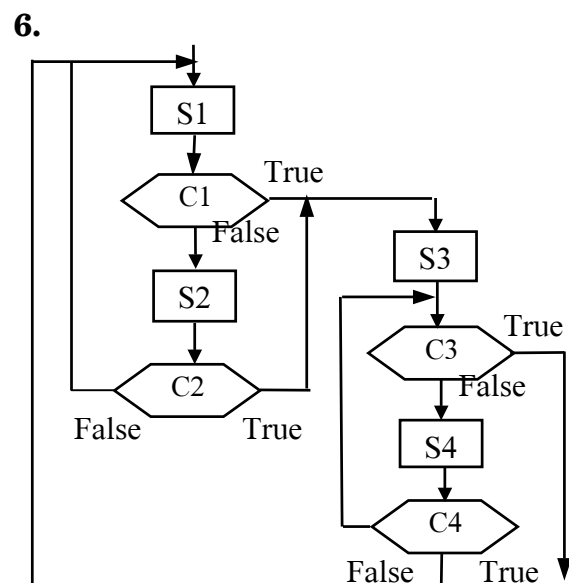
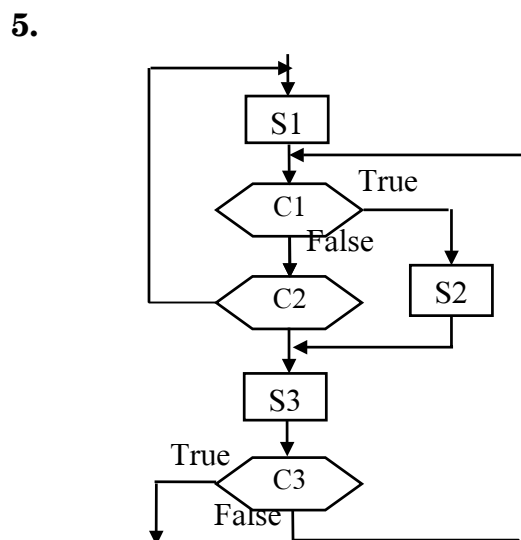
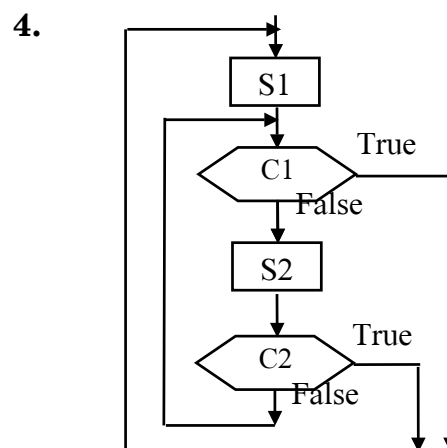
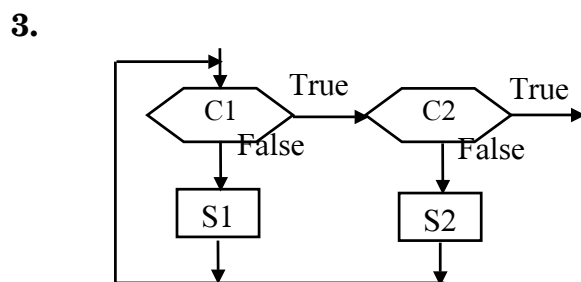
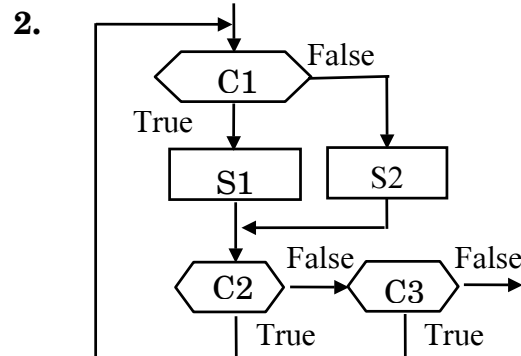
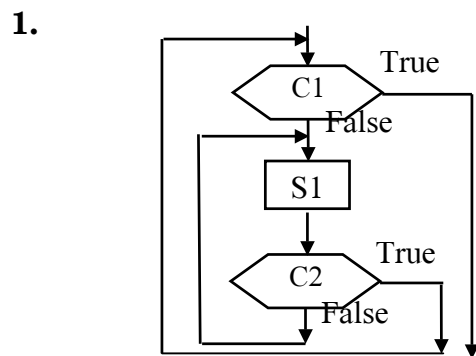
Unit DTHTRON ; {Tập DTHTRON.PAS}
Interface
  Const Pi = 3.1415926535 ; {1/π = 0.318309886}
  Function Dientich (R : Real) : Real;
Implementation
  Function Dientich;
  Begin
    Dientich := Pi * Sqr (R)
  End;
Begin {phần khởi động Inialization}

```

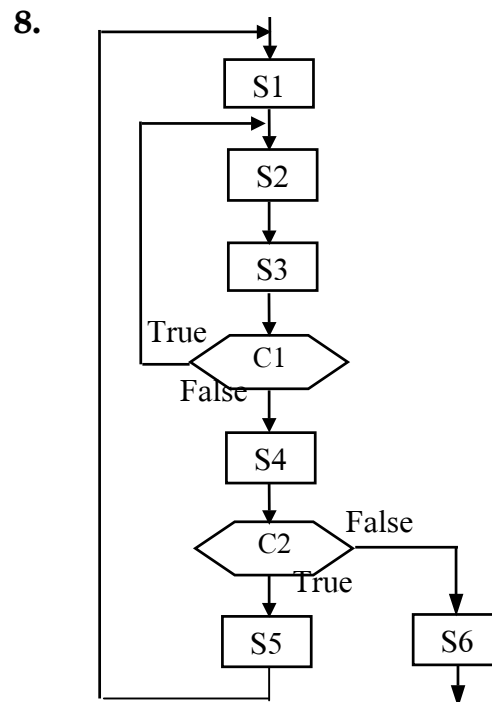
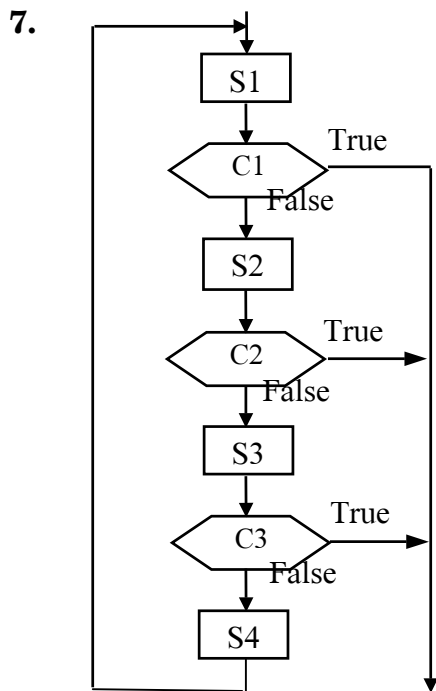
```
Writeln ( 'trong chương trình chính sẽ sử dụng Unit DTHTRON !' )  
End.
```

### VI.5. Bài tập

1. Sử dụng các câu lệnh Pascal để viết chương trình theo sơ đồ khối dưới đây bằng cách chỉ sử dụng các cấu trúc điều khiển cơ bản. Sau đó hãy đổi về dạng chương trình chỉ sử dụng ba cấu trúc điều khiển là tuần tự, điều kiện **if then else** và lặp **While**.







2. **Xử lý chuỗi** (string) : Viết chương trình đọc một câu (kết thúc bởi Enter ↵) sau đó tiến hành các công việc :

- Thống kê số từ, số ký tự trong câu.
- Tách câu ra thành các từ cách nhau bởi dấu cách (space). In ra các từ này.
- Nén câu (bỏ các dấu cách giữa các từ). In kết quả.
- Thay thế những xuất hiện của câu con S1 thành câu con S2 và in kết quả. (S1 và S2 là các câu con nhập vào)

Yêu cầu sử dụng Unit cho mỗi việc. Chương trình chính dùng menu để gọi.

3. Viết chương trình xử lý ma trận vuông A cấp  $n \times n$  dưới dạng menu, gồm các việc sau :

- Đọc vào ma trận vuông cấp  $n \times n$ .
- Tính định thức của ma trận.
- Kiểm tra tính đối xứng qua đường chéo chính của ma trận.
- Xác định xem ma trận có dạng tam giác trên không ? (các phần tử phía dưới đường chéo chính đều = 0)
- Xác định xem ma trận có dạng tam giác dưới không ? (các phần tử phía trên đường chéo chính đều = 0)

Yêu cầu dùng Unit. Bốn việc sau cùng chỉ có hiệu lực khi ma trận đã được đọc.

## CHƯƠNG 3

# Hợp thức hóa phần mềm

## I. Xác minh và hợp thức hóa phần mềm

Người ta thường phân biệt 2 yếu tố trong hoạt động sản xuất phần mềm :

1. Xây dựng đặc tả, lập trình và lập hồ sơ (viết các hướng dẫn sử dụng v.v ...) Yếu tố này tạo nên giai đoạn phát triển chương trình.
2. Xác minh (hay kiểm tra) và hợp thức hóa các sản phẩm phần mềm là việc so sánh các sản phẩm phần mềm đó với các đặc tả của chúng sao cho có quan hệ thỏa mãn.

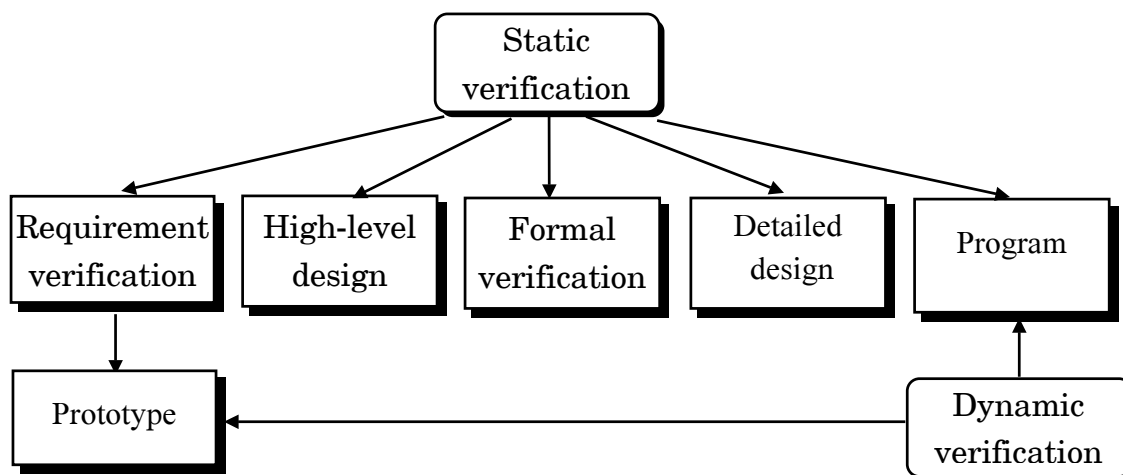
Người ta dùng thuật ngữ *xác minh* (verification) khi so sánh một sản phẩm với một đặc tả chặt chẽ (rigorous). Khi đặc tả là không hình thức (informal), người ta dùng thuật ngữ *hợp thức hóa* (validation).

Giai đoạn xác minh và hợp thức hóa, gọi tắt là giai đoạn V & V (Verification and Validation) bao giờ cũng có mặt trong mọi dự án Tin học.

Sự khác nhau cơ bản giữa V & V được Boehm B.W. (1979) tóm tắt như sau :

Validation : *Are we building the right product?*

Verification : *Are we building the product right?*



Hình 3.1. Kỹ thuật tĩnh và kỹ thuật động của quá trình V&V

Để thực hiện quá trình V&V, người ta sử dụng các kỹ thuật tĩnh (static techniques) và động (dynamic techniques) để kiểm tra hệ thống.

Kỹ thuật tĩnh nhằm phân tích biểu diễn hệ thống qua việc phân tích yêu cầu, phân tích thiết kế và hiển thị (listing) chương trình. Kỹ thuật động là việc thử nghiệm (testing) chương trình

Kỹ thuật tĩnh bao gồm việc thanh tra (inspection) chương trình, phân tích và xác minh hình thức (formal verification) hay chứng minh sự đúng đắn (proving) của chương trình.

## II. Chứng minh sự đúng đắn của chương trình

Phương pháp chứng minh là sử dụng các định lý để minh họa tính đúng đắn của sản phẩm cần xác minh. Phương pháp này không có khả năng hợp thức hóa các đặc tả phi hình thức, bởi vì không thể chứng minh một cách toán học các tính chất không được định nghĩa chặt chẽ. Người ta phân biệt các phép chứng minh hình thức, được diễn tả trong lý thuyết logic, và các chứng minh phi hình thức nhưng chặt chẽ, như trong các cuốn sách về Toán học.

Mọi phép chứng minh hình thức tính đúng đắn của chương trình được xây dựng một cách tường minh từ các tiên đề và các quy tắc suy diễn logic. Thực tiễn cho thấy không thể xây dựng một phép chứng minh như vậy mà không sử dụng đến những công cụ như là các công cụ chứng minh định lý.

Những phép chứng minh hình thức cho các chương trình cỡ hàng ngàn dòng lệnh đã được thực hiện. Chúng cho phép khẳng định tính được phê phán của chương trình.

Trong phép chứng minh không hình thức, người ta định nghĩa kiến trúc tổng quan của phép chứng minh, xử lý những điểm khó khăn, để lại cho người đọc sự chăm sóc chi tiết đến các điểm khác.

Ví dụ để chứng minh không hình thức sự đúng đắn của một chương trình, người ta có thể diễn tả các tất biến của vòng lặp và của thủ tục đệ quy. Một phép chứng minh phi hình thức không cần thiết phải sử dụng các công cụ, vấn đề là người đọc sẽ tự kiểm chứng thông qua các cuộc trao đổi, thảo luận (chẳng hạn tổ chức thanh tra căn cứ trên việc xác minh).

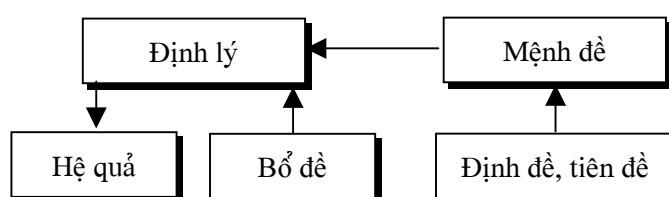
Người ta còn có thể thử chứng minh phi hình thức các tính chất đã phát biểu ít nhiều có tính chặt chẽ (chẳng hạn đề cập đến vấn đề hợp thức hóa), khái niệm chứng minh tính đúng đắn theo nghĩa Toán học được thay thế bởi khái niệm biện luận (reasoning - argumentation) nhằm thuyết phục các chuyên gia Tin học.

## II.1. Suy luận Toán học

Trong lĩnh vực suy luận Toán học, người ta thường đặt ra hai vấn đề :

1. Khi nào thì một suy luận là đúng ?
2. Có thể sử dụng những phương pháp nào để xây dựng các suy luận Toán học ?

Suy luận Toán học là một hình thức tư duy mà từ một hay nhiều mệnh đề logic đã có (phán đoán) rút ra được một mệnh đề logic mới. Kết quả của một suy luận nào đó phải là đúng hoặc là sai. Trong Toán học, định lý là một phát biểu có thể chứng minh được là đúng. Người ta hay gặp mô hình chứng minh một định lý Toán học (là đúng) như sau :



Hình 3.2. Chứng minh một định lý Toán học

### II.1.1. Các quy tắc suy luận Toán học

Để trình bày các quy tắc suy luận Toán học, chúng ta nhắc lại các phép toán logic sau :

- $\neg$  **not** (không)
- $\wedge$  **and** (và)
- $\vee$  **or** (hoặc)
- $\rightarrow$  **implicate** (kéo theo)
- $\sim$  **equivalence** (tương đương)

Chú ý :  $a \rightarrow b$  tương đương với  $\neg a \vee b$ , hay **if a then b else true**

$a \sim b$  có nghĩa  $(a \rightarrow b) \wedge (b \rightarrow a)$ .

Thứ tự ưu tiên của các phép toán logic là  $\neg, \wedge, \vee, \rightarrow, \sim$ . Bảng logic như sau :

a	b	$\neg a$	$a \wedge b$	$a \vee b$	$a \rightarrow b$	$a \sim b$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

Để chứng minh một định lý, người ta sử dụng một số tiên đề và quy tắc suy luận, hay là các hằng đúng. Bảng dưới đây trình bày các quy tắc suy luận được sử dụng trong chứng minh tính đúng đắn của chương trình.

Stt	Quy tắc suy luận	Tên gọi	Ví dụ
1	$\frac{p}{\therefore p \vee q}$	Luật khẳng định	Nhạc của Trịnh Công Sơn hay. Vậy nhạc của Trịnh Công Sơn hay hoặc ca sĩ Khánh Ly hát hay.
2	$\frac{p \wedge q}{\therefore p}$	Luật rút gọn	Tháng này trời nắng hạn và sông Đà thì cạn nước. Vậy trời nắng hạn.
3	$\frac{p \rightarrow q}{p} \therefore q$	Luật tách rời (Modus Ponens)	Nếu cơm chín thì cần tắt lửa. Cơm đã chín. Vậy cần tắt lửa.
4	$\frac{p \rightarrow q}{\neg q} \therefore \neg p$	Luật phủ định (Modus Tollens)	Nếu mặt trời ở đỉnh đầu thì bóng ngắn nhất. Bóng không ngắn nhất. Vậy mặt trời không ở đỉnh đầu
5	$\frac{p \rightarrow q}{q \rightarrow r} \therefore p \rightarrow r$	Tam đoạn luận giả định	Nếu trời mưa thì đường HP bị ngập. Nếu đường HP bị ngập thì phải xuống xe dắt bộ. Vậy trời mưa thì phải xuống xe dắt bộ.
6	$\frac{p \vee q}{\neg p} \therefore q$	Tam đoạn luận chuyển	Cu Tý thuộc bài hoặc là cu Tý ham chơi. Mà Cu Tý không thuộc bài. Vậy cu Tý ham chơi.

Trong bảng trên, dấu  $\therefore$  được đọc là *vậy thì*. Mỗi luật (cơ sở của phép suy luận), chẳng hạn luật tách rời (Modus Ponens), có thể viết dưới dạng hằng đúng :

$$(p \wedge (p \rightarrow q)) \rightarrow q$$

### II.1.2. Khái niệm về chứng minh tính đúng đắn của chương trình

Một chương trình P xác định một thuật toán cho phép nhận vào một tập hợp dữ liệu L để đưa ra một tập hợp kết quả R. Nói cách khác, với mọi  $d \in L$ , chương trình P xác định hoặc một dãy hữu hạn các phép tính để cho ra một kết quả  $P(d) \in R$ , hoặc một dãy vô hạn các phép tính : chương trình bị "quẩn" với dữ liệu d.

Mặt khác, P được viết để tính một hàm f nào đó từ  $D \subseteq L$  vào R. P đúng nếu và chỉ nếu P tính đúng hàm f, nghĩa là nếu  $\forall d \in D, P(d)$  xác định (P không quẩn với dữ liệu vào d) và bằng f(d).

Thông thường, để kiểm tra tính đúng đắn của chương trình, người ta dùng phương pháp thử (test) : người ta chọn một dãy các dữ liệu mẫu  $d_1, d_2, \dots, d_n$ , rồi cho P chạy lần lượt với mỗi dữ liệu để kiểm tra rằng  $P(d_1) = f(d_1), P(d_2) = f(d_2), \dots, P(d_n) = f(d_n)$ .

Tính không đầy đủ của phương pháp này thể hiện ở chỗ không thể thử hết mọi dữ liệu của  $D$ , dù  $D$  hữu hạn : có thể xảy ra  $P$  cho kết quả đúng với mọi dữ liệu mẫu  $d_i$  đã chọn nhưng với một dữ liệu  $d \neq d_i \forall i$ ,  $P$  cho một kết quả sai.

Hơn nữa, phương pháp thử không bao giờ chứng minh được một chương trình là đúng đắn, chỉ có thể chứng minh được là không đúng, nếu với một giá trị  $d_i$  nào đó đã chọn, thì  $P(d_i) \neq f(d_i)$ .

Về mặt lý thuyết, phương pháp chứng minh tính đúng đắn của chương trình mang tính Toán học bằng cách chứng minh một định lý tương đương :  $\forall d \in D, P(d) = f(d)$ .

### II.1.3. Tiên đề và quy tắc suy diễn

Một cách tổng quát, các phát biểu cần chứng minh có dạng  $E\{P\}S$ , trong đó  $E$  và  $S$  là các điều kiện,  $P$  là dãy các lệnh (hay là một chương trình) theo nghĩa rằng : nếu  $E$  đúng trước khi thực hiện  $P$  thì nếu  $P$  dừng,  $S$  đúng sau khi thực hiện  $P$ . Người ta gọi  $E$  là điều kiện trước (precondition) và  $S$  là điều kiện sau (postcondition) của chương trình  $P$ .

Các điều kiện trước  $E$  và điều kiện sau  $S$  được xây dựng từ các biểu thức logic có thể nhận giá trị đúng (1) hoặc sai (0), chúng là mối liên hệ giữa các biến của chương trình (ví dụ các biến  $a, b, c, p, q, r$  trong các biểu thức  $a = bq + r, q \geq 0, b^2 - 4ac > 0, v.v...$ ) cùng các phép toán logic (nếu có).

#### Tính chất :

Với mọi chương trình  $P$  và mọi điều kiện  $C$ , ta đều có :  
**false {P} C và C {P} true**

Việc chứng minh nếu một chương trình  $P$  dừng thì  $P$  sẽ cho kết quả đúng được gọi là chứng minh tính đúng đắn từng phần.

Trong các chứng minh tính đúng đắn từng phần, các tiên đề và định lý sẽ là các phát biểu có dạng  $E\{P\}S$ . Sau đây là danh sách các tiên đề và các quy tắc suy diễn cho phép chứng minh các định lý dạng  $E\{P\}S$ .

Để chứng minh các quan hệ giữa các điều kiện (ví dụ  $E_1 \sim E_2, E_1 \rightarrow E_2, v.v...$ ), người ta sử dụng các tính chất của đại số Boole và miền xác định các biến chương trình (số nguyên trong trường hợp Div).

### a) Tiên đề về phép gán

Cho phép gán  $\mathbf{x} := \langle \mathbf{bt} \rangle$  và một điều kiện sau S, ta có tiên đề :

$$E \{ \mathbf{x} := \langle \mathbf{bt} \rangle \} S$$

trong đó E nhận được từ S bằng phép thế các biến x bởi biểu thức  $\langle \mathbf{bt} \rangle$ . E là *điều kiện yếu nhất* phải làm thoả mãn các biến trước khi thực hiện phép gán sao cho S là đúng sau đó.

#### Ví dụ 1

$$(xy \geq 0) \{ \mathbf{z} := \mathbf{x} * \mathbf{y} \} (z \geq 0)$$

$$(q + 1 \geq 0) \{ \mathbf{q} := \mathbf{q} + 1 \} (q \geq 0)$$

$$((x+y)^2 = y) \{ \mathbf{x} := \mathbf{x} + \mathbf{y} \} (x^2 = y)$$

#### Chú ý quan trọng :

Nhờ quy tắc trên đây, người ta có thể chứng minh điều kiện trước của một lệnh gán, bằng cách sử dụng một điều kiện sau, nhưng ngược lại là không thể. Bởi vậy, để chứng minh tính đúng đắn của chương trình, người ta xuất phát từ điểm kết thúc (điều kiện sau) để tiến hành ngược lên điểm bắt đầu (điều kiện trước).

### b) Quy tắc ";" hay tổ hợp các lệnh và lệnh ghép

Gọi E, F, S là các điều kiện, P và Q là các dãy lệnh, ta có :

$$\frac{E \{ P \} F \quad F \{ Q \} S}{\therefore E \{ P ; Q \} S} \qquad \frac{E \{ P \} S}{\therefore E \{ \mathbf{begin} \ P \ \mathbf{end} \} S}$$

#### Ví dụ 13 :

Chúng minh  $(\mathbf{x}=1) \{ \mathbf{y}:= 2; \mathbf{z}:= \mathbf{x} + \mathbf{y} \} (\mathbf{z}=3)$

Là đúng đắn với khẳng định đầu  $E \equiv (\mathbf{x} = 1)$

Và khẳng định cuối  $S \equiv (\mathbf{z} = 3)$

Giả sử S đúng, tức  $z = 3$ , khi đó nhận được  $x + y = 3$ , ta có :

$$(\mathbf{x+y=3}) \{ \mathbf{z}:= \mathbf{x} + \mathbf{y} \} (\mathbf{z=3}) \quad (1)$$

Do y được gán giá trị 2, nên nhận được giá trị của x là 1, tức là :

$$(\mathbf{x=1}) \{ \mathbf{y}:= 2 \} (\mathbf{x+y=3}) \quad (2)$$

Vậy theo quy tắc ";", từ (1) và (2) ta có P kết thúc thì S đúng (đpcm).

### II.1.4. Quy tắc điều kiện if B then P

$$\frac{E \wedge B \{ P \} S \quad E \wedge \neg B \rightarrow S}{\therefore E \{ \mathbf{if} \ B \ \mathbf{then} \ P \} S}$$

**Chú ý :**

B phải được xem như một điều kiện sao cho có thể ứng dụng một trong những quy tắc được trình bày ở đây. Điều này có nghĩa rằng việc tính B không làm thay đổi các giá trị của các biến của chương trình.

**Ví dụ 2 :**

Chúng minh :  $E\{\text{if } x > y \text{ then } y := x\} (y \geq x)$ , với **E** là điều kiện đầu nào đó.

Khi E đúng và  $x > y$  đúng (điều kiện B) thì y có giá trị x, vậy  $y \geq x$  (S), ta có :

$$E \wedge (x > y) \{ y := x \} (y \geq x) \quad (1)$$

Khi E đúng và  $x > y$  sai ( $\neg B$ ) có nghĩa  $x \leq y$ , vậy  $y \geq x$  (S), tức là :

$$E \wedge (x \leq y) \rightarrow (y \geq x) \quad (2)$$

Vậy từ (1) và (2) ta có nhận được đpcm.

**II.1.5. Quy tắc điều kiện if B then P else Q**

$$E \wedge B \{P\} S$$

$$E \wedge \neg B \{Q\} S$$

---


$$\therefore E \{ \text{if } B \text{ then } P \text{ else } Q \} S$$

**Ví dụ 3 :**

Chúng minh :  $E \{ \text{if } x < 0 \text{ then } \text{abs} := -x \text{ else } \text{abs} := x \} (\text{abs} = |x|)$   
với **E** là điều kiện đầu nào đó.

Vậy chương trình là đúng với điều kiện đầu **E** và điều kiện sau  $\text{abs} = |x|$ .

Khi E đúng và  $x < 0$  đúng (B) thì abs có giá trị  $-x$ , tức  $\text{abs} = -x = |x|$  và điều kiện sau S đúng, ta có :

$$E \wedge x < 0 \{ \text{abs} := -x \} (\text{abs} = |x|) \quad (1)$$

Khi E đúng và có  $x < 0$  sai ( $\neg B$ ) thì  $x \geq 0$ , khi đó abs có giá trị x tức  $\text{abs} = x = |x|$  và điều kiện sau S đúng, tức là :

$$E \wedge x < 0 \{ \text{abs} := x \} (\text{abs} = |x|) \quad (2)$$

Vậy từ (1) và (2) ta có nhận được đpcm.

**II.1.6. Quy tắc vòng lặp while**

$$E \wedge B \{ P \} E$$

---


$$\therefore E \{ \text{while } B \text{ do } P \} E \wedge \neg B$$

Ở đây, E và B là những điều kiện. Riêng điều kiện E được gọi là *bất biến* của vòng lặp.



Một trong những khó khăn của việc chứng minh tính đúng đắn của chương trình là tìm được bất biến cho mỗi vòng lặp (nghĩa là một bất biến cho phép chứng minh đúng cái yêu cầu). Thực tế *không tồn tại một phương pháp có tính hệ thống và tổng quan để tìm ra những bất biến như vậy.*

Ví dụ 4 :

Sử dụng bất biến của vòng lặp, chứng minh đoạn chương trình tính  $fac = n!$ , với  $n \in \mathbb{N}$  sau đây :

```

i := 1; fac := 1;
while i < n do begin
  i := i + 1;
  fac := fac * i
end;

```

Gọi  $P \equiv \{\text{begin } i := i + 1; \text{ fac} := \text{fac} * i \text{ end}\}$

Giả sử điều kiện  $E \equiv (fac = i!) \wedge (i \leq n)$ , ta cần chứng minh  $E$  là bất biến của vòng lặp.

Ta sẽ chứng minh bằng quy nạp :

$E$  đúng trước khi vào vòng lặp, vì  $i = 1, fac = 1 = 1!$  Và  $1 \leq n$ .

Giả sử  $E$  đúng với  $i < n$  sau khi thực hiện vòng lặp và sau đó, **while** còn được thực thi một lần nữa. Trước hết  $i$  được tăng thêm 1 (với lệnh gán  $i := i + 1$ ) và do vậy vẫn còn  $i \leq n$ . Do giả thiết quy nạp  $fac = (i - 1)!$  trước khi vào vòng lặp nên  $fac$  sẽ có giá trị là :

$$fac = (i - 1)! * i = i!$$

Từ đó  $E$  quả thật là bất biến của vòng lặp và mệnh đề :

$$(E \wedge (i < n)) \{P\} E \text{ đúng.}$$

Từ đó suy ra khẳng định :

$$E \{ \text{while } i < n \text{ do } P \} E \wedge (i \geq n) \text{ cũng đúng.}$$

Vì vòng lặp kết thúc sau khi lặp  $n - 1$  lần, khi đó  $i = n$  và  $fac = n!$ .

### II.1.7. Các quy tắc khác

**Quy tắc điều kiện trước :**

$$\frac{E \{P\} S \quad E' \rightarrow E}{\therefore E' \{P\} S}$$

**Quy tắc "và" :**

$$\frac{E \{P\} S \quad E \{P\} S'}{\therefore E \{P\} S \wedge S'}$$

Quy tắc điều kiện sau :

$$\frac{E \{P\} S \quad S \rightarrow S'}{\therefore E \{P\} S'}$$

Quy tắc "hoặc" :

$$\frac{E \{P\} S \quad E' \{P\} S}{\therefore E \vee E' \{P\} S \wedge S'}$$

Ví dụ 5 :

Chúng minh chương trình con P tính tích hai số nguyên  $m$  và  $n$  là đúng :

```
P ≡ function product(m, n: Integer): Integer;
begin
  {P1 ≡} if n < 0 then a := -n else a := n;
  {P2 ≡} k := 0; x := 0;
  {P3 ≡} while k < a do begin
    x := x + m;
    k := k + 1
  end;
  {P4 ≡} if n < 0 then product := -x
  else product := x
end;
```

Ta sẽ chứng minh rằng sau khi thực hiện P thì hàm trả về giá trị là  $mn$ .

Ta chia P gồm bốn đoạn CT là {P1; P2; P3; P4} như trên.

Gọi E là điều kiện đầu  $E \equiv \langle m, n \text{ nguyên} \rangle$  và  $S1 \equiv E \wedge (a = |n|)$ .

Khi đó có thể chỉ ra  $E \{P1\} S1$  là đúng.

Gọi  $S2 \equiv S1 \wedge (k = 0) \wedge (x = 0)$ . Dễ dàng kiểm tra rằng  $S1 \{P2\} S2$  là đúng.

Ta cũng thấy điều kiện  $(x = mk) \wedge (k \leq a)$  là một bất biến trong vòng lặp P3 tương tự với lý luận quy nạp trong vòng lặp tính  $n!$ . Vòng lặp này kết thúc sau  $a$  bước lặp khi  $k = a$ , tức  $x = ma$  tại điểm này.

Gọi  $S3 \equiv (x = ma) \wedge (a = |n|)$ . Từ đó suy ra  $S2 \{P3\} S3$  là đúng.

Cuối cùng có thể chỉ ra P4 là đúng với điều kiện đầu S3 và điều kiện cuối S, với  $S \equiv \text{product} = mn$ . Vậy  $S3 \{P4\} S$  đúng và hàm trả về giá trị là  $mn$ .

Từ các mệnh đề  $E \{P1\} S1$ ,  $S1 \{P2\} S2$ ,  $S2 \{P3\} S3$  và  $S3 \{P4\} S$  là đúng, theo quy tắc hợp thành, ta có P cũng đúng, tức  $E \{P\} S$  đúng.

Ngoài ra do cả P1, P2, P3 và P4 đều dừng nên P cũng dừng <sub>(qed)</sub>.

## II.2. Phương pháp của C.A.R. Hoare

### II.2.1. Phát biểu

Sau đây là một ví dụ sử dụng phương pháp của C.A.R. Hoare :

```
Div : r:=a; q:= 0;
      while r >= b do begin
          r:= r - b;
          q:= q + 1;
      end;
```

$a, b, q, r$  là các biến nguyên,  $a$  và  $b$  được khởi gán giá trị đầu lần lượt là  $A$  và  $B$ . Với  $A \in \mathbb{N}$  và  $B \in \mathbb{N}^+$ , hàm Div tính thương  $q$  và số dư  $r$  của phép chia  $A$  cho  $B$ . Như vậy với hàm Div ta có :

$D = \mathbb{N} \times \mathbb{N}^+, R = \mathbb{N} \times \mathbb{N}$  và  $f$  là hàm xác định trên  $\mathbb{N} \times \mathbb{N}^+$  vào trong  $\mathbb{N} \times \mathbb{N}$ , nghĩa là từ cặp  $(A, B) \in \mathbb{N} \times \mathbb{N}^+$ , xác định cặp  $(q, r) \in \mathbb{N} \times \mathbb{N}$  sao cho  $A = Bq + r$  và  $r < B$ .

Bây giờ cần chứng minh Div tính đúng hàm  $f$  theo hai bước như sau :

1. *Chứng minh tính đúng đắn từng phần* : nếu Div dừng thì Div sẽ cho kết quả đúng.
2. *Chứng minh tính dừng* : Div dừng với mọi dữ liệu thuộc  $\mathbb{N} \times \mathbb{N}^+$ .

Trong trường hợp Div, ta cần chứng minh phát biểu sau :

Nếu trước khi thực hiện lệnh đầu tiên của Div,  $a$  và  $b$  được gán giá trị đầu  $A \in \mathbb{N}$  và  $B \in \mathbb{N}^+$ . Sau khi thực hiện,  $q$  và  $r$  thoả mãn các điều kiện  $A = Bq + r, r < B, r \geq 0, q \geq 0$ . Ta có :

$$(a=A) \wedge (b=B) \wedge (A \geq 0) \wedge (B > 0) \{Div\} (A=Bq+r) \wedge (q \geq 0) \wedge (r \geq 0) \wedge (r < B).$$

### II.2.2. Chứng minh tính đúng đắn từng phần của Div

Ta cần chứng minh :

$$(a=A) \wedge (b=B) \wedge (A \geq 0) \wedge (B > 0) \{Div\} (A=Bq+r) \wedge (q \geq 0) \wedge (r \geq 0) \wedge (r < B)$$

Để dễ theo dõi, ta đặt tên các điều kiện như sau :

$$P1 = (a=A) \wedge (b=B)$$

$$P2 = (A \geq 0) \wedge (B > 0)$$

$$Q1 = (A=Bq+r) \wedge (q \geq 0) \wedge (r \geq 0) \wedge (r < B)$$

Chú ý rằng :

$$P1 \wedge P2 \quad \rightarrow \quad P1 \wedge (a \geq 0) \wedge (b > 0) \text{ và}$$

$$P1 \wedge (a=bq+r) \wedge (q \geq 0) \wedge (r \geq 0) \wedge (r < b) \quad \rightarrow \quad Q1$$

Theo các quy tắc điều kiện trước và điều kiện sau, chỉ cần chứng minh :

$$(I) \quad \boxed{P1 \wedge (a \geq 0) \wedge (b > 0) \{Div\} P1 \wedge (a = bq + r) \wedge (q \geq 0) \wedge (r \geq 0) \wedge (r < b)}$$

Ta sẽ chứng minh hai giai đoạn :

$$1. P1 \{Div\} P1$$

$$2. (a \geq 0) \wedge (b > 0) \{Div\} (a = bq + r) \wedge (q \geq 0) \wedge (r \geq 0) \wedge (r < b)$$

### ❶ $P1 \{Div\} P1$

Trước hết cần chỉ ra P1 là bất biến của vòng lặp. Theo quy tắc gán, ta có :

$$P1 \{ q := q + 1 \} P1$$

$$P1 \{ r := r - b \} P1$$

như vậy, theo quy tắc tổ hợp :

$$P1 \{ r := r - b ; q := q + 1 \} P1, \text{ nhưng :}$$

$$P1 \wedge (r \geq b) \rightarrow P1$$

Theo quy tắc điều kiện trước :

$$P1 \wedge (r \geq b) \{ r := r - b ; q := q + 1 \} P1$$

Theo quy tắc vòng lặp :

$$(1) P1 \{ \text{while } r \geq b \text{ do begin } r := r - b ; q := q + 1 \text{ end} \} P1 \wedge (r < b)$$

Áp dụng lần nữa quy tắc gán và quy tắc tổ hợp, ta có :

$$(2) P1 \{ r := a ; q := 0 \} P1$$

Từ (1) và (2), theo quy tắc tổ hợp, ta có :

$$P1 \{Div\} P1 \wedge (r < b)$$

Cuối cùng, theo quy tắc điều kiện sau :

$$P1 \{Div\} P1 \text{ (đpcm)}$$

### ❷ $(a \geq 0) \wedge (b > 0) \{Div\} (a = bq + r) \wedge (q \geq 0) \wedge (r \geq 0) \wedge (r < b)$

Đặt :

$$P3 = (a = bq + r) \wedge (q \geq 0) \wedge (r \geq 0)$$

$$P4 = (a = b(q+1) + r) \wedge (q+1 \geq 0) \wedge (r \geq 0) \quad \text{bằng cách thay } q \text{ trong } P3 \text{ bởi } q+1$$

$$P5 = (a = b(q+1) + r - b) \wedge (q+1 \geq 0) \wedge (r - b \geq 0) \quad \text{bằng cách thay } r \text{ trong } P4 \text{ bởi } r-b$$

(2.1.) Ta xem rằng P3 là một bất biến của vòng lặp. Theo quy tắc gán :

$$P4 \{ q := q + 1 \} P3$$

$$\text{và } P5 \{ r := r - b \} P4$$

Từ đó theo quy tắc tổ hợp :

$$(3) P5 \{ r := r - b ; q := q + 1 \} P3$$

Xét điều kiện trước, ta có :

$$(i) P5 \sim (a=bq+r) \wedge (q \geq -1) \wedge (r \geq b)$$

(ii) Vì rằng :

$$(q \geq 0) \rightarrow (q \geq -1) \quad \text{và}$$

$$(r \geq 0) \wedge (r \geq b) \rightarrow (r \geq b), \quad \text{ta có :}$$

$$P3 \wedge (r \geq b) \rightarrow (a=bq+r) \wedge (q \geq -1) \wedge (r \geq b)$$

Theo quy tắc điều kiện trước, phát biểu (3) sẽ là :

$$P3 \wedge (r \geq b) \{ r := r - b ; q := q + 1 \} P3$$

Theo quy tắc vòng lặp dẫn đến :

$$(4) P3 \{ \text{while } r \geq b \text{ do begin } r := r - b ; q := q + 1 \text{ end} \} P3 \wedge (r < b)$$

(2.2.) Bây giờ sử dụng bất biến vòng lặp như là điều kiện sau cho những lệnh đầu tiên của dãy. Ta có quy tắc gán :

$$(a=b.0+r) \wedge (0 \geq 0) \wedge (r \geq 0) \{ q := 0 \} P3$$

$$\text{và } (a=b.0+r) \wedge (0 \geq 0) \wedge (r \geq 0) \sim (a=r) \wedge (r \geq 0)$$

Áp dụng cho lần nữa quy tắc gán :

$$(a=a) \wedge (a \geq 0) \{ r := a \} (a=r) \wedge (r \geq 0) \quad \text{và :}$$

$$(a=a) \wedge (a \geq 0) \sim (a \geq 0)$$

Theo quy tắc tổ hợp, ta có :

$$(5) (a \geq 0) \{ r := a ; q := 0 \} P3$$

Quy tắc tổ hợp áp dụng cho (4) và (5) cho ta :

$$(a \geq 0) \{ \text{Div} \} P3 \wedge (r < b)$$

Quy tắc điều kiện sau dẫn đến kết luận :

$$(a \geq 0) \wedge (b > 0) \{ \text{Div} \} P3 \wedge (r < b)$$

Từ ❶ và ❷ đã chứng minh, suy ra (I) theo quy tắc "và", đồng thời kết thúc việc chứng minh tính đúng đắn từng phần của Div.

## II.3. Chứng minh dừng

### II.3.1. Chứng minh dừng của một chương trình

Một chương trình không là đệ quy, không chứa lệnh goto sẽ chỉ có thể một dãy vô hạn tính toán nếu nó thực hiện vô hạn lần thân một vòng lặp. Việc chứng minh một chương trình như vậy dừng với mọi dữ liệu  $d \in D$  dẫn đến việc chứng minh rằng mỗi vòng lặp của chương trình chỉ có thể thực hiện một số hữu hạn lần  $\forall d \in D$ .

Cho vòng lặp :

**while B do P**

với  $x_1, x_2, \dots, x_n$  là các biến của chương trình. Gọi  $W_E$  là tập hợp các giá trị của vectơ :

$$w = \langle x_1, x_2, \dots, x_n \rangle$$

sao cho các biến thoả mãn điều kiện E.

Giả thiết rằng điều kiện E là bất biến với vòng lặp đang xét và thoả mãn trước khi thực hiện vòng lặp này.

Nếu  $w = w_1 \in W_{E \wedge B}$  trước khi thực hiện vòng lặp, khi đó  $w = w_2 \in W_E$  sau khi thực hiện thân vòng lặp P.

Nếu  $w_2 \in W_{E \wedge \neg B}$ , vòng lặp dừng ; nếu không  $w = w_3 \in W_E$ , sau khi thực hiện P, v.v...

Như vậy để chứng minh vòng lặp dừng, chỉ cần chỉ ra rằng mọi dãy  $w_1, w_2, w_3, \dots$  đã xây dựng là hữu hạn.

Phương pháp để chứng minh là định nghĩa một hàm  $m$  từ  $W_{E \wedge B}$  vào  $\mathbb{N}$  sao cho có thể chỉ ra rằng :

$$E \wedge B \wedge (m(w) = m_0) \{ P \} \neg B \wedge (m(w) < m_0) \quad \forall m_0 \in \mathbb{N}$$

#### Chú ý :

Phát biểu trên có nghĩa rằng nếu  $w = w_i$  trước khi thực hiện P, thì sau khi thực hiện P, hoặc vòng lặp dừng, hoặc  $m(w_{i+1}) < m(w_i)$ .

Trong trường hợp này (đã chứng minh được P dừng), ta có thể khẳng định rằng với mọi dãy  $w_1, w_2, w_3, \dots$ , dãy  $m(w_1), m(w_2), m(w_3) \dots$  giảm dần thực sự trong  $\mathbb{N}$ , là hữu hạn.

Điều đó chứng minh rằng dãy  $w_1, w_2, w_3, \dots$  là hữu hạn, và vòng lặp dừng.

### II.3.2. Chứng minh dừng của Div

Div chỉ chứa một vòng lặp :

while  $r \geq b$  do begin  $r := r - b$  ;  $q := q + 1$  end

Vòng lặp này luôn luôn dừng.

Điều kiện  $\{ b=B \}$  là một bất biến của vòng lặp này, được thoả mãn trước khi thực thể vòng lặp (xem chứng minh đúng từng phần mục ❶).

Tương tự như vậy đối với điều kiện  $(r \geq 0)$  (xem ❷).

Ta xét tập hợp  $W_1 = W_{(b=B) \wedge (r \geq 0) \wedge (r \geq b)}$  và hàm  $m : W_1 \rightarrow \mathbf{N}$  sao cho  $m(w) = r$ .

#### Chú ý :

Việc chọn hàm này dẫn việc chứng minh dừng đến việc chứng minh rằng biến  $r$  nhận các giá trị giảm ngặt và liên tục.

Đặt :

$$P6 = (b=B) \wedge (r \geq 0) \wedge (r \geq b)$$

Tiếp theo, ta còn phải chứng minh rằng :

$$(1) P6 \wedge (r=m_0) \{ r := r - b ; q := q + 1 \} (r < b) \wedge (r < m_0) \quad \forall m_0 \in \mathbf{N}$$

Theo quy tắc gán :

$$(r-b < m_0) \{ r := r - b ; q := q + 1 \} (r < m_0) \quad \forall m_0 \in \mathbf{N}$$

áp dụng các quy tắc điều kiện trước và điều kiện sau, ta có :

$$(2) P6 \wedge (r=m_0) \wedge (r-b < m_0) \{ r := r-b ; q := q+1 \} (r < b) \wedge (r < m_0) \quad \forall m_0 \in \mathbf{N}$$

Mặt khác :

$$(b=B) \wedge (r=m_0) \wedge (r-b < m_0) \sim (b=B) \wedge (r=m_0) \wedge (m_0-B < m_0)$$

hay :

$$(b=B) \wedge (r=m_0) \wedge (r-b < m_0) \sim (b=B) \wedge (r=m_0) \wedge (B > 0)$$

Điều kiện  $B > 0$  luôn luôn đúng vì rằng  $(A, B) \in D = \mathbf{N} \times \mathbf{N}^+$ , như vậy :

$$(b=B) \wedge (r=m_0) \wedge (r-b < m_0) \sim (b=B) \wedge (r=m_0)$$

Từ đó dẫn đến (1) xuất phát từ (2). Việc chứng minh Div dừng đã xong.

#### Chú ý :

Để chứng minh Div dừng với  $\forall d \in D$ , nói chung không áp dụng được cho  $\forall d \in L$ .

Ví dụ nếu  $L = \mathbf{Z} \times \mathbf{Z}$ , Div có thể quẩn với  $B = 0$ .

### II.3.3. Đánh giá một chương trình lặp

Đánh giá một chương trình P là xác định thời gian và kích thước bộ nhớ cần thiết để thực hiện chương trình P :

- một hàm  $T_P : L \rightarrow \mathbb{R} \cup \{+\infty\}$  sao cho,  $\forall d \in L$ ,  $T_P(d)$  là thời gian thực hiện P đối với dữ liệu d ( $T_P(d) = +\infty$  nếu chương trình quẩn).
- một hàm  $N_P : L \rightarrow \mathbb{N} \cup \{+\infty\}$  sao cho,  $\forall d \in L$ ,  $N_P(d)$  là số đơn vị bộ nhớ cần thiết để thực hiện P đối với dữ liệu d ( $N_P(d)$  có thể vô hạn nếu chương trình quẩn).

Ở đây, ta chỉ quan tâm đến việc xác định thời gian thực hiện chương trình gồm các lệnh cơ sở như lệnh gán (quy tắc ";"), lệnh điều kiện và vòng lặp while.

Ta sẽ gọi  $f_P$  là hàm riêng phần (partial function) của L trong tập hợp các kết quả R, được tính bởi chương trình P, và W là tập hợp  $W_{\text{true}}$  các giá trị có thể của các biến trong P.

Hàm  $f_P$  có thể mở rộng thành một hàm  $W \rightarrow W$ . Tương tự, nếu P' là một dãy các lệnh của P, hàm  $f_{P'}$  tính bởi P có thể được định nghĩa như là một hàm của

$W \rightarrow W$ .

#### Ví dụ 14 :

Trong trường hợp Div, nếu  $L = D$ ,  $W = \mathbb{N} \times \mathbb{N}^+ \times \mathbb{N} \times \mathbb{N} =$  tập hợp các giá trị có thể của a, b, r, q.

$$f_{\text{Div}}(a, b, r, q) = (a, b, \text{thương của } a \text{ và } b, \text{ phần dư của } a \text{ chia cho } b).$$

Nếu xét lệnh  $q := q + 1$  của Div :

$$f_{q := q+1}(a, b, r, q) = (a, b, q+1, r).$$

Thời gian thực hiện một chương trình có thể được định nghĩa như sau :

1. Mỗi lệnh cơ sở và mỗi điều kiện có một thời gian thực hiện độc lập với giá trị của các biến. Như vậy,  $\forall w \in W$  :

$$\text{Với mọi phép gán } x := y, \quad T_{x := y}(w) = \text{constant. Ta viết } T_{x := y}.$$

$$\text{Với mọi điều kiện } B, \quad T_B(w) = \text{constant. Ta viết } T_B.$$

$$2. T_{P; Q}(w) = T_P(w) + T_Q(f_P(w))$$

$$3. T_{\text{if } B \text{ then } P}(w) = \text{if } B \text{ then } (T_B + T_P(w)) \text{ else } T_B$$

$$T_{\text{if } B \text{ then } P \text{ else } Q}(w) = \text{if } B \text{ then } (T_B + T_P(w)) \text{ else } (T_B + T_Q(w))$$

$$4. T_{\text{while } B \text{ do } P}(w) = (n(w) + 1) T_B + \sum_{i=0}^{n(w)-1} T_P(f_P^i(w))$$



trong đó,  $n$  là hàm  $W \rightarrow \mathbb{N} \cup \{+\infty\}$ , sao cho  $n(w)$  là số lần thực hiện thân vòng lặp  $P$  với giá trị đầu  $w$  của các biến.

*Đánh giá thời gian thực hiện của Div :*

Số lần lặp của Div là giá trị cuối của biến  $q$ , giả sử là  $\left\lfloor \frac{A}{B} \right\rfloor$ , phần nguyên của  $A$  chia cho  $B$ , trong đó,  $A$  và  $B$  là các dữ liệu của Div.

Mặt khác, thời gian thực hiện thân vòng lặp độc lập với các dữ liệu và bằng :

$$T_{r:=r-b} + T_{q:=q+1}$$

Ta có :

$$T_{\text{Div}}(A, B) = T_{r:=a} + T_{q:=0} + \left( \left\lfloor \frac{A}{B} \right\rfloor + 1 \right) T_{r \geq b} + \left\lfloor \frac{A}{B} \right\rfloor (T_{r:=r-b} + T_{q:=q+1})$$

Như vậy,  $T_{\text{Div}}(A, B)$  có dạng  $K_1 \left\lfloor \frac{A}{B} \right\rfloor + K_2$ , trong đó  $K_1$  và  $K_2$  độc lập với các dữ liệu  $A$  và  $B$ .

$T_{\text{Div}}(A, B)$  là bậc (order) của  $\left\lfloor \frac{A}{B} \right\rfloor$ .

**Chú ý :** Người ta thường quan tâm đến bậc của hàm TP thay vì bản thân hàm TP, vì TP phụ thuộc vào phần cứng và phần mềm sử dụng để thực thi chương trình  $P$ . Như vậy chỉ cần xác định số lần thực hiện dãy các lệnh hay thực hiện nhất của chương trình. Điều này có thể nhận được bằng cách đếm số lần lặp trong mỗi vòng lặp.

### III. Xây dựng chương trình

#### III.1. Mở đầu

Trong mục trước, ta thấy việc chứng minh tính đúng đắn của một chương trình, dù chỉ là một chương trình rất ngắn và đơn giản, rất khó khăn và mệt mỏi. Để khắc phục, người ta đưa ra một phương pháp hiệu quả hơn là *vừa thiết kế vừa chứng minh tính đúng đắn của chương trình*.

Ví dụ, đặc tả một bất biến cho một vòng lặp trước khi viết, kiểm tra tính đúng của vòng lặp này với điều kiện sau, rồi viết thân của vòng lặp sao cho giữ được bất biến và chỉ thực hiện một số hữu hạn lần. Để tính USCLN của hai số nguyên dương, ta sử dụng các tính chất số học như sau :

$$\text{Nếu } a = b \quad \text{USCLN}(a, b) = a = b$$

$$\text{Nếu } a < b \quad \text{USCLN}(a, b) = \text{USCLN}(a, b - a)$$

$$\text{Nếu } a > b \quad \text{USCLN}(a, b) = \text{USCLN}(a - b, b)$$

Giả sử  $a$  và  $b$  là hai biến của chương trình nhận giá trị hai dữ liệu lần lượt  $A$  và  $B$ . Sau khi gán, điều kiện  $USCLN(a, b) = USCLN(A, B)$  thoả mãn.

Nếu thân chương trình chứa một vòng lặp của bất biến  $USCLN(a, b) = USCLN(A, B)$  với điều kiện kiểm tra dừng là  $a = b$ , thì ta có :

$$USCLN(a, b) = a = b = USCLN(A, B)$$

sau khi thực hiện vòng lặp này. Kết quả tìm được là một trong hai biến  $a$  và  $b$ .

Cần xác định thân của vòng lặp này sao cho, một mặt, bất biến được thoả mãn, mặt khác, vòng lặp chỉ thực hiện một số hữu hạn lần. Theo tính chất của  $USCLN$ , nếu  $a \neq b$ , lệnh :

```
if a > b then a := a - b else b := b - a
```

làm cho điều kiện  $USCLN(a, b) = USCLN(A, B)$  trở nên bất biến. Chỉ còn phải chứng minh rằng vòng lặp đã viết chỉ thực hiện một số hữu hạn lần.

Ta thấy quan hệ :  $(a > 0) \wedge (b > 0)$

là bất biến trong vòng lặp (điều kiện của vòng lặp là  $a \neq b$ , lệnh  $b := b - a$  chỉ được thực hiện nếu  $b > a$ ).

Vậy  $\max(a, b)$  cũng là bất biến. Hơn nữa  $\max(a, b)$  giảm ngặt mỗi lần thực hiện thân vòng lặp, và vòng lặp luôn luôn dừng.

Như vậy ta đã viết và chứng minh không hình thức chương trình sau đây :

```
P : while a <> b do
      if a > b then a := a - b else b := b - a ;
      ketqua := a ;
```

### III.2. Bài toán cờ tam tài

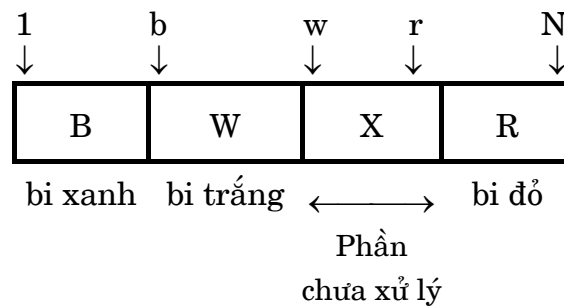
Cho trước :

1. Một mảng các hòn bi đánh số từ 1 đến  $N$ , mỗi hòn bi mang một màu hoặc xanh, hoặc trắng, hoặc đỏ.
2. Các vị từ  $B(i)$ ,  $W(i)$  và  $R(i)$  là đúng nếu và chỉ nếu hòn bi thứ  $i$  ( $1 \leq i \leq N$ ) là xanh, trắng và đỏ tương ứng.
3. Cặp hoán vị  $(i, j)$  để đặt hòn bi thứ  $i$  thành  $j$ , hòn bi thứ  $j$  thành  $i$ ,  $\forall i, j \in 1..N$ , không loại trừ trường hợp  $i = j$ .

Ta cần sắp xếp các hòn bi theo thứ tự "xanh, trắng, đỏ", mỗi vị từ  $B$ ,  $W$  và  $R$  chỉ được tính đến một lần cho mỗi hòn bi đã cho. Hơn nữa, các hoán vị phải càng ít càng tốt.

### III.2.1. Lời giải thứ nhất

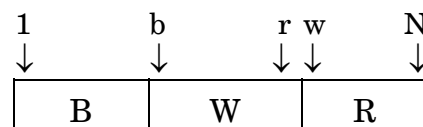
Ta sẽ sử dụng một vòng lặp while với bất biến như hình dưới đây và điều kiện kiểm tra dừng là "vùng X rỗng".



Cần có 3 chỉ số b, w và r để phân cách 4 vùng của mảng. Ta thực hiện phép chọn như sau : các chỉ số b và w đứng ngay sau các vùng B và W tương ứng của các viên bi. Chỉ số r là của viên bi đứng ngay trước vùng R.

Ta có :  $P_{b,w,r} = (1 \leq \alpha < b \rightarrow B(\alpha)) \wedge (b \leq \alpha < w \rightarrow W(\alpha)) \wedge (r < \alpha \leq N \rightarrow R(\alpha))$

Điều kiện sau của chương trình như sau :



hay có thể viết :  $P_{b,w,r} \wedge (w = r + 1)$ . Chương trình sẽ có dạng như sau :

{ Khởi gán :  $P_{b,w,r}$  true }

**while** (vùng\_X\_không\_rỗng) **do**

{ Thân vòng lặp của bất biến  $P_{b,w,r}$  }

Sau khi thực hiện chương trình này, ta kiểm tra được rằng :  $P_{b,w,r} \wedge$   
(vùng\_X\_rỗng)

và mảng các viên bi đã được sắp xếp.

Chương trình sau đây chứa biến n nhận giá trị N :

```

w := 1; b := 1; r := n;
while w <= r do
(I)   if W(w) then w := w+1
      else if B(w) then
          begin HoánVị(b, w); b := b+1; w := w+1 end
      else
          begin HoánVị(r, w); r := r-1 end

```

Ta nhận thấy rằng chương trình này thoả mãn các vị từ B, W và R.

**Phân tích :**

Số lần lặp = N

Số lần hoán vị  $= ne_1 = \#B + \#R$  ( $\#B$  và  $\#R$  lần lượt là số viên bi xanh và đỏ)

Kết quả này chưa là tối ưu, ta có thể làm giảm số lần hoán vị các viên bi đỏ như sau :

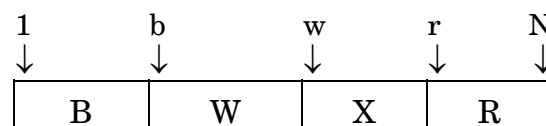
```

(Ibis)
w:= 1; b := 1; r := n;
while w <= r do
  if W(w) then w:= w+1
  else if B(w) then
    begin HoánVị(b, w); b:= b+1; w:= w+1 end
  else
    begin while R(r) and w < r then r:= r-1;
           HoánVị(r, w); r:= r-1
    end
end

```

### III.2.2. Lời giải thứ hai

Ta sử dụng bất biến biểu diễn như sau :



Từ đó đưa đến chương trình (tạm thời chấp nhận việc chứng minh không hình thức) :

```

(II)
w:= 1; b := 1; r := n;
while r <= n do
  if R(r) then r:= r-1
  else if W(r) then
    begin HoánVị(w, r); w:= w+1; r:= r-1 end
  else
    begin HoánVị(w, r); r:= r-1;
           HoánVị(b, w); b:= b+1; w:= w+1
    end
end

```

**Phân tích :**

Số lần lặp  $= N$

Số lần hoán vị  $= ne_2 = \#W + 2\#B$

So sánh với chương trình (I), ta có :  $ne_2 - ne_1 = \#B + \#W - \#R$

Chương trình (I) sẽ hiệu quả hơn (II), nếu như ít ra nửa số viên bi là đỏ.

Bài tập :

Từ bất biến trên đây, ta có thể nhận được chương trình như sau :

```

w:= 1; b := 1; r := n;
while r <= n do
  if R(r) then r:= r-1
  (IIbis) else if W(r) then
    begin HoánVị(w, r); w:= w+1; r:= r+1 end
  else
    begin HoánVị(b, r); b:= b+1;
      HoánVị(w, r); w:= w+1; r:= r+1
    end
end

```

Chương trình trên là sai. Hãy tìm một cách phân bố ban đầu của mảng hai viên bi để thấy sai.

### III.2.3. Chứng minh tính đúng đắn của chương trình (I)

#### a) Các định nghĩa và ký hiệu

Ta đã xây dựng chương trình (I) bằng cách sử dụng bất biến :

$$P_{b,w,r} = (1 \leq \alpha < b \rightarrow B(\alpha)) \wedge (b \leq \alpha < w \rightarrow W(\alpha)) \wedge (r < \alpha \leq N \rightarrow R(\alpha))$$

Để chứng minh tính đúng đắn của chương trình này, ta sử dụng ký hiệu đơn giản hoá :

$x \leq \alpha < y \rightarrow A(\alpha)$  là ký hiệu viết tắt của điều kiện :

$$\begin{cases} A(x) \wedge A(x+1) \wedge \dots \wedge A(y) & \text{nếu } x \leq y \\ \text{true} & \text{nếu không} \end{cases}$$

hay :  $(x \leq y) \rightarrow A(x) \wedge \dots \wedge A(y)$

Bây giờ ta xem xét các điều kiện đặc trưng của bài toán .

a. Các điều kiện (1) và (2) được biểu diễn bởi :

$$E_1 = (1 \leq \alpha \leq N) \rightarrow (B(\alpha) \vee W(\alpha) \vee R(\alpha))$$

b. Định nghĩa (3) của hoán vị (i, j) đưa đến tiên đề sau đây :

"Cho điều kiện sau S, ta có :

$$E \wedge (1 \leq i \leq N) \wedge (1 \leq j \leq N) \{ \text{hoán vị } (i, j) \} S$$

nếu E nhận được từ S bằng cách thay thế :

– mọi vị từ B(bt), W(bt), R(bt), trong đó bt có giá trị i, bởi B(j), W(j), R(j)

– và mọi vị từ B(bt), W(bt), R(bt), trong đó bt có giá trị j, bởi B(i), W(i), R(i)".

c. Điều kiện sau của chương trình (sắp xếp mảng) được biểu diễn bởi :

$$S_1 = (1 \leq b \leq N+1) \wedge (b-1 \leq r \leq N) \wedge (1 \leq \alpha < b \rightarrow B(\alpha)) \wedge \\ (b \leq \alpha < r \rightarrow W(\alpha)) \wedge (r < \alpha \leq N \rightarrow R(\alpha))$$

Bất biến của vòng lặp ta vừa sử dụng không hẳn là  $P_{b,w,r}$  nhưng :

$$Q_{b, w, r} = E_1 \wedge (1 \leq b \leq N) \wedge (w-1 \leq r \leq N) \wedge P_{b, w, r}$$

Giả sử  $A_1$  là vòng lặp while của chương trình (I) và  $A_2$  là thân của vòng lặp này.

### b) Chứng minh tính đúng đắn từng phần

**Bổ đề :**  $x \leq \alpha \leq y \rightarrow A(\alpha)$  tương đương với  $(x \leq \alpha \leq y-1 \rightarrow A(\alpha)) \wedge (x \leq y \rightarrow A(y))$

Chứng minh :

$$\begin{aligned} x \leq \alpha \leq y &\rightarrow A(\alpha) \\ &\sim (x \leq y) \rightarrow A(x) \wedge \dots \wedge A(y) \text{ theo định nghĩa} \\ &\sim ((x < y) \rightarrow A(x) \wedge \dots \wedge A(y)) \wedge ((x = y) \rightarrow A(y)) \\ &\sim ((x < y) \rightarrow A(x) \wedge \dots \wedge A(y-1)) \wedge ((x < y) \rightarrow A(y)) \wedge ((x = y) \rightarrow A(y)) \\ &\sim x \leq \alpha \leq y-1 \rightarrow A(\alpha) \wedge (x \leq y \rightarrow A(y)) \text{ đpcm} \end{aligned}$$

#### ( $\alpha$ ) Chứng minh của bất biến $Q_{b, w, r}$ trong $A_2$

Ta kiểm tra rằng  $E_1 \wedge (1 \leq i \leq N) \wedge (1 \leq j \leq N) \{ \text{hoán vị } (i, j) \} E_1$

##### (i) Trường hợp $W(w)$

Rõ ràng ta có :  $Q_{b, w+1, r} \{ w := w+1 \} Q_{b, w, r}$

Vả lại :  $Q_{b, w, r} \wedge (w \leq r) \wedge W(w) \sim E_1 \wedge (1 \leq b \leq w) \wedge (w \leq r \leq N) \wedge P_{b, w, r} \wedge W(w)$

và :  $P_{b, w, r} \wedge W(w) \rightarrow P_{b, w+1, r}$

như vậy :  $Q_{b, w, r} \wedge (w \leq r) \wedge W(w) \rightarrow Q_{b, w+1, r}$

Điều này chứng minh :

$$Q_{b, w, r} \wedge (w \leq r) \wedge W(w) \{ w := w+1 \} Q_{b, w, r}$$

##### (ii) Trường hợp $B(w)$

Rõ ràng ta có :  $Q_{b+1, w+1, r} \{ b := b+1 ; w := w+1 \} Q_{b, w, r}$

Vả lại :

$$\begin{aligned} P_{b+1, w+1, r} &= (1 \leq \alpha < b+1 \rightarrow B(\alpha)) \wedge (b+1 \leq \alpha < w+1 \rightarrow W(\alpha)) \wedge \\ &\quad (r \leq \alpha < N \rightarrow R(\alpha)) \end{aligned}$$

Từ đó theo bổ đề :

$$\begin{aligned} P_{b+1, w+1, r} &\sim (1 \leq \alpha < b+1 \rightarrow B(\alpha)) \wedge (b+1 \leq \alpha < w \rightarrow W(\alpha)) \wedge \\ &\quad (r \leq \alpha < N \rightarrow R(\alpha)) \wedge ((b \geq 1) \rightarrow B(b)) \wedge ((w \geq b+1) \rightarrow W(\alpha)) \end{aligned}$$

Áp dụng tiên đề định nghĩa bởi phép hoán vị cho  $Q_{b+1, w+1, r}$  :

$$F \{ \text{hoán vị}(b, w) \} Q_{b+1, w+1, r}$$

với :

$$\begin{aligned}
F &= E_1 \wedge (1 \leq b+1 \leq w+1) \wedge (w \leq r \leq N) \\
&\wedge (1 \leq \alpha < b \rightarrow B(\alpha)) \wedge (b+1 \leq \alpha < w \rightarrow W(\alpha)) \\
&\wedge (r < \alpha \leq N \rightarrow R(\alpha)) \wedge ((b \geq 1) \rightarrow B(w)) \\
&\wedge ((w \geq b+1) \rightarrow W(b)) \wedge (1 \leq b \leq N) \wedge (1 \leq w \leq N)
\end{aligned}$$

Nhưng :

$$(b+1 \leq \alpha < w \rightarrow W(\alpha)) \wedge ((w \geq b+1) \rightarrow W(b)) \sim (b \leq \alpha < w \rightarrow W(\alpha))$$

Như vậy :

$$F \sim E_1 \wedge (1 \leq b \leq w) \wedge (w \leq r \leq N) \wedge B(w) \wedge P_{b, w, r}$$

Từ đó suy ra :

$$Q_{b, w, r} (w \leq r) \wedge \neg W(w) \wedge B(w) \rightarrow F$$

Điều này chứng minh :

$$Q_{b, w, r} \wedge (w \leq r) \wedge \neg W(w) \wedge B(w) \{ \text{hoán vị}(b, w) ; b := b+1 ; w := w+1 \} Q_{b, w, r}$$

(iii) Trường hợp  $R(w)$

Ta có :

$$Q_{b, w, r-1} \{ r := r-1 \} Q_{b, w, r}$$

Mặt khác theo bổ đề :

$$P_{b, w, r-1} \sim P_{b, w, r} \wedge ((r \leq N) \rightarrow R(r))$$

Áp dụng tiên đề định nghĩa bởi phép hoán vị cho  $Q_{b, w, r-1}$  :

$$G \{ \text{hoán vị}(r, w) \} Q_{b, w, r-1}$$

với :

$$\begin{aligned}
G &= E_1 \wedge (1 \leq b \leq w) \wedge (w-1 \leq r-1 \leq N) \wedge P_{b, w, r} \wedge ((r \leq N) \rightarrow R(w)) \\
&\wedge (1 \leq r \leq N) \wedge (1 \leq w \leq N)
\end{aligned}$$

Thực tế ta có :

$$P_{b, w, r} \wedge (b \leq w) \wedge (w \leq r) \wedge (1 \leq w \leq N) \wedge (1 \leq r \leq N)$$

$$\{ \text{hoán vị}(r, w) \} P_{b, w, r} \wedge (b \leq w) \wedge (w \leq r)$$

Ta có :

$$G \sim E_1 \wedge (1 \leq b \leq w) \wedge (w \leq r \leq N) \wedge R(w) \wedge P_{b, w, r}$$

Vả lại :

$$E_1 \wedge \neg W(w) \wedge \neg B(w) \wedge (1 \leq w \leq N) \rightarrow R(w)$$

Như vậy :

$$Q_{b,w,r} \wedge (w \leq r) \wedge \neg W(w) \wedge \neg B(w) \rightarrow G$$

Điều này chứng minh :

$$Q_{b,w,r} \wedge (w \leq r) \wedge \neg W(w) \wedge \neg B(w) \{ \text{hoán vị}(r, w) ; r := r - 1 \} Q_{b,w,r}$$

Từ 3 kết quả trên ta suy ra :

$$Q_{b,w,r} \wedge (w \leq r) \{ A_2 \} Q_{b,w,r}$$

### (β) Chứng minh tính đúng đắn của chương trình (I)

Ta đã có :

$$Q_{b,w,r} \{ A_2 \} Q_{b,w,r} \wedge (w > r)$$

#### (i) Điều kiện sau

Ta có :

$$Q_{b,w,r} \wedge (w > r) \rightarrow (1 \leq b \leq w) \wedge (w=r+1) \wedge (r \leq N) \wedge P_{b,w,r}$$

và :

$$P_{b,w,r} \wedge (w=r+1) \rightarrow (1 \leq \alpha < b \rightarrow B(\alpha)) \wedge (b \leq \alpha < r \rightarrow W(\alpha)) \wedge (r < \alpha \leq N \rightarrow R(\alpha))$$

Như vậy :

$$Q_{b,w,r} \wedge (w > r) \rightarrow S_1$$

trong đó :

$$Q_{b,w,r} \{ A_1 \} S_1$$

#### (ii) Khởi gán

Ta có :

$$E_1 \wedge (0 \leq n \leq N) \wedge (n < \alpha \leq N \rightarrow R(\alpha)) \{ w := 1; b := 1; r := n \} Q_{b,w,r}$$

và lại :

$$(n=N) \rightarrow (0 \leq n \leq N) \wedge (n < \alpha \leq N \rightarrow R(\alpha)),$$

do đó :

$$E_1 \wedge (n=N) \{ \text{chương trình (I)} \} S_1$$

Như vậy ta đã chứng minh xong tính đúng đắn từng phần.

### c) Chứng minh dừng

Chương trình (I) chỉ có một vòng lặp  $A_1$  và thân của vòng lặp là  $A_2$ .

Giả sử :  $W_{Q_{bwr}} \wedge (w \leq r) = X_1$  và hàm  $m : X_1 \rightarrow N$  sao cho  $m(x) = r - w$ .



*Chú ý* : Việc chứng minh dừng là chỉ ra rằng kích thước của vùng chưa xử lý  $r - w$  giảm dần mỗi lần thực hiện  $A_2$ .

Để chứng minh việc dừng của  $A_1$ , chỉ cần chỉ ra rằng :

$$Q_{b, w, r} \wedge (w \leq r) \wedge (r - w = m_0) \{ A_2 \} (w > r) \vee (r - w < m_0) \quad \forall m_0 \in \mathbb{N}$$

Chú ý rằng :

$(r - w = m_0) \{ w := w + 1 \} (r - w = m_0 - 1)$  trường hợp của  $W(w)$  và  $B(w)$

và  $(r - w = m_0) \{ r := r - 1 \} (r - w = m_0 - 1)$  trường hợp của  $R(w)$

Ta chứng minh được rằng :

$$Q_{b, w, r} \wedge (w \leq r) \wedge (r - w = m_0) \{ A_2 \} (r - w = m_0 - 1) \quad \forall m_0 \in \mathbb{N}$$

Từ đó chương trình (I) dừng.

#### ***d) Chứng minh chương trình (I) thoả mãn điều kiện của bài toán***

Bây giờ chỉ còn phải chứng minh rằng điều kiện "mỗi vị từ  $B$ ,  $W$  và  $R$  chỉ được xét tính nhiều nhất một lần cho mỗi viên bi" chưa xuất hiện trong  $S_1$  cũng thoả mãn.

Thật vậy, ở trên, ta đã suy ra được rằng  $A_2$  được thực hiện đúng  $N$  lần (giá trị của  $r - w$  giảm 1 mỗi lần, giảm từ  $N - 1$  đến 0). Mặt khác, việc thực hiện  $A_2$  xét tính (nhiều nhất một lần) các vị từ liên quan đến chỉ một viên bi. Trong quá trình sắp xếp, màu của  $N$  viên bi là hoàn toàn xác định và vì có  $N$  lần thực hiện, mỗi thực hiện của  $A_2$  chỉ liên quan đến 1 viên bi phân biệt.

### **III.3. In ra một danh sách theo thứ tự ngược**

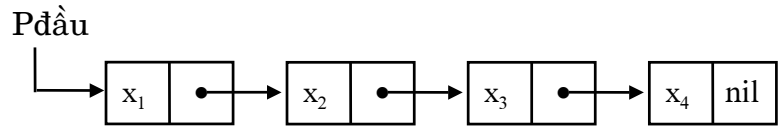
Giả sử ta có một danh sách tuyến tính gồm các phần tử có cấu trúc và một con trỏ chỉ đến phần tử đầu tiên của danh sách như sau :

```

const n = ...           { độ dài lớn nhất của danh sách }
type KiểuPhầnTử = ...   { kiểu của các phần tử trong danh sách }
KiểuConTrỏ = 0..n;      { kiểu của con trỏ, 0 là giá trị nil }
PhầnTử = record
    NộiDung: KiểuPhầnTử;
    TiếpTheo: KiểuConTrỏ; { trỏ đến phần tử tiếp theo }
end;
Danhsách = array[1..n] of PhầnTử;
var Ds : Danhsách;      { danh sách các phần tử }
    Đầu : KiểuConTrỏ;  { con trỏ chỉ đến phần tử đầu tiên của danh
sách }

```

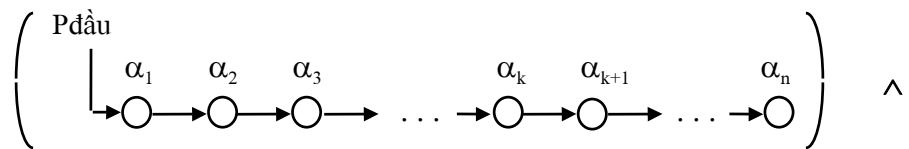
Giả thiết rằng danh sách chứa ít nhất một phần tử (đầu danh sách **PĐầu** ≠ **nil**). Bài toán đặt ra là in nội dung danh sách theo thứ tự ngược lại. Ví dụ, nếu ta có :



thì chương trình sẽ in ra như sau :  $x_4, x_3, x_2, x_1$ .

Ta sẽ đưa ra ba lời giải TILDA1, TILDA2 và TILDA3 cho danh sách có độ dài  $n$  phần tử. Đối với mỗi lời giải, ta sẽ phân tích thời gian và bộ nhớ cần thiết để thực hiện chương trình.

Cả ba lời giải đều sử dụng vòng lặp với bất biến có dạng sau :



(các trường nội dung các bản ghi  $\alpha_{k+1}, \dots, \alpha_n$  được in theo chiều ngược lại)

Mỗi bước của vòng lặp là tìm phần tử  $\alpha_k$  và viết nội dung ( $\alpha_k$ ). Ba chương trình phân biệt nhau cơ bản ở cách tiếp cận đến  $\alpha_k$ .

### III.3.1.TILDA1

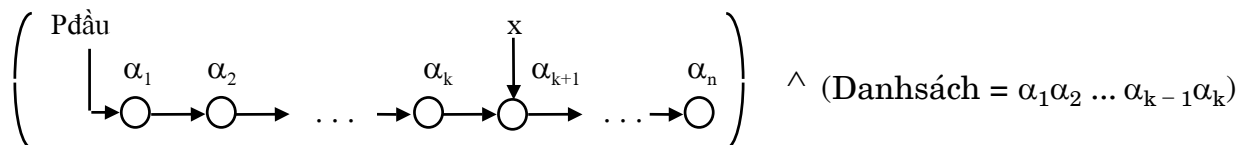
Ta sử dụng một danh sách con trỏ, đầu tiên danh sách ở trạng thái rỗng, sau đó hoạt động với hai thao tác như sau :

**put(expr)** đặt vào đỉnh danh sách giá trị của biểu thức **expr**.

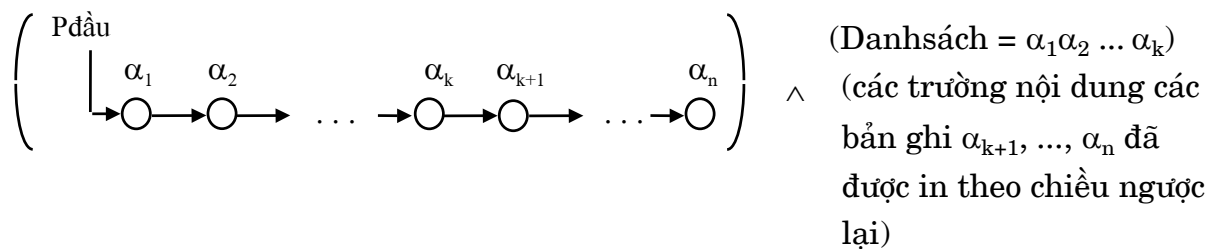
**get(x)** nếu danh sách khác rỗng, lấy giá trị ở đỉnh danh sách để gán cho biến **x**, nếu không, thao tác không xác định.

Chương trình TILDA1 gồm hai vòng lặp :

(a) Vòng lặp thứ nhất :



(b) Vòng lặp thứ hai :



Mỗi bước của vòng lặp là tìm phần tử  $\alpha_k$  và viết nội dung ( $\alpha_k$ ). Ba chương trình phân biệt nhau cơ bản ở cách tiếp cận đến  $\alpha_k$ .

## IV. Các tiên đề và quy tắc suy diễn

Mục này sẽ nghiên cứu các vấn đề sau :

- – Khái niệm về điều kiện trước yếu nhất và điều kiện sau mạnh nhất của một dãy lệnh.
- – Các kiểu tiên đề gán khác nhau.
- – Các tiên đề và quy tắc suy diễn cho một số cấu trúc ngôn ngữ lập trình (khối, thủ tục).
- – Phân tích chương trình.

### IV.1. Điều kiện trước yếu nhất và điều kiện sau mạnh nhất của một dãy lệnh

Trong mục trước, ta đã ký hiệu :

- $W_E$  tập hợp các giá trị các biến của một chương trình thoả mãn điều kiện E,
- $f_P$  là hàm tính được bởi dãy các lệnh P của một chương trình.

Nếu  $f_P$  được định nghĩa cho mọi giá trị của  $W_E$ , P không quản và không thực hiện các phép tính vô định (ví dụ chia cho 0) nếu điều kiện trước E thoả mãn. Tính chất này được ký hiệu là  $\text{term}_E P$ .

Bây giờ ta xét phát biểu E {P} S. Ta có các tính chất sau đây :

- (i) E {P} S là true nếu và chỉ nếu  $f_P(W_E) \subseteq W_S$
- (ii) Nếu  $\text{term}_E P$  thì :  
E {P} S là true nếu và chỉ nếu  $W_E \subseteq f_P^{-1}(W_S)$

#### Ví dụ 15 :

- (1) Cho phát biểu ( $q > 0$ ) {  $q := q+1$  } ( $q > 0$ ), trong đó q là biến duy nhất của chương trình. Ta có :

$$W_E = W_S = \mathbb{N}^+, \quad f_{q:=q+1}(\mathbb{N}^+) = \mathbb{N}^+ - \{1\} \subseteq \mathbb{N}^+,$$

và  $N^+ \subseteq f_{q:=q+1}^{-1}(N^+) = N$

(2) Cho phát biểu  $(q \geq 0) \wedge (y \geq 0) \{ q := q \text{ div } y \} (q \geq 0) \wedge (y \geq 0)$ , trong đó  $q$  và  $y$  là các biến duy nhất của chương trình. Ta có :

$$W_E = W_S = N \times N, \quad f_{q:=q \text{ div } y}(N \times N) = N \times N^+ \subset W_S,$$

nhưng  $f_{q:=q \text{ div } y}^{-1}(N \times N) = N \times N^+ \not\subseteq W_E$

Trong trường hợp  $W_E = f_P^{-1}(W_S)$ ,  $E$  là *điều kiện trước yếu nhất* (la plus faible précondition) phải được thoả mãn trước khi thực hiện  $P$  để cho  $S$  được thoả mãn sau đó.

Thực tế, nếu  $E' \{P\} S$  và  $\text{term}_E \cdot P$  thì  $W_{E'} \subseteq f_P^{-1}(W_S) = W_E$  và như vậy  $E' \rightarrow E$ .

Mặt khác, nếu  $f_P(W_E) = W_S$ ,  $S$  là *điều kiện sau mạnh nhất* (la plus forte postcondition) phải được thoả mãn sau khi thực hiện  $P$  nếu  $E$  là đúng trước.

Thực tế, nếu  $E \{P\} S'$  thì  $f_P(W_E) = W_S \subseteq W_{S'}$ , như vậy  $S \rightarrow S'$ .

Ta ký hiệu hai hàm *fppre* và *fp post* như sau :

- với một dãy lệnh và với một điều kiện sau, *fppre* trả về điều kiện trước yếu nhất tương ứng
- với một điều kiện trước và một dãy lệnh, *fp post* trả về điều kiện sau sau mạnh nhất tương ứng.

Chú ý rằng các hàm *fppre(P, S)* và *fp post(E, P)* được định nghĩa gần như tương đương.

Bây giờ ta sẽ trình bày các tính chất của các hàm *fppre* và *fp post*.

#### IV.1.1. Hàm *fppre*

Với mọi điều kiện  $S$  và dãy lệnh  $P$ ,  $\text{fp post}(\text{pfpre}(P, S), P) \rightarrow S$

*Chứng minh :*

Đặt  $E \sim \text{pfpre}(P, S)$ .

Ta có  $W_E = f_P^{-1}(W_S)$ , từ đó suy ra  $f_P(W_E) = f_P(f_P^{-1}(W_S)) \subseteq W_S$

Hay có thể nói  $E \sim \text{pfpre}(P, S)$ , từ đó suy ra  $\text{fp post}(E, P) \rightarrow S_{\text{đpcm}}$

#### IV.1.2. Hàm *fp post*

Với mọi điều kiện  $E$  và dãy lệnh  $P$ ,  $E \rightarrow \text{pfpre}(P, \text{fp post}(E, P))$ .

*Chứng minh :*

Đặt  $S \sim \text{fp post}(E, P)$ .

Ta có  $W_S = f_P(W_E)$ , từ đó suy ra  $f_P^{-1}(W_S) = f_P^{-1}(f_P(W_E)) \supseteq W_E$

Hay có thể nói  $S \sim \text{pfpst}(E, P)$ , từ đó suy ra  $E \rightarrow \text{pfpre}(P, S)_{\text{dpcm}}$

Bài tập :

1. Cho điều kiện E và một dãy lệnh P, những điều kiện nào làm thoả mãn  $f_P$  sao cho :

$$E \sim \text{pfpre}(P, \text{pfpst}(E, P)) ?$$

#### **IV.1.3. Sử dụng điều kiện trước yếu nhất và điều kiện sau mạnh nhất để chứng minh tính đúng đắn của chương trình**

##### **a) Trường hợp điều kiện trước yếu nhất**

Sau khi định nghĩa hàm  $\text{pfpre}$ , với mọi điều kiện E và S, và mọi dãy lệnh P, ta có :

$$(E \{P\} S \wedge \text{term}_E P) \sim (E \rightarrow \text{pfpre}(P, S))$$

đặc biệt :

$$\text{term}_E P \sim (E \rightarrow \text{pfpre}(P, \text{true}))$$

Điều này có nghĩa rằng tập hợp các giá trị của  $\text{pfpre}(P, \text{true})$  là miền xác định của hàm  $f_P$  (nghĩa là tập hợp các giá trị sao cho chương trình P đưa ra một kết quả).

Khả năng biểu diễn  $\text{pfpre}(P, S)$ ,  $\forall P$  và  $\forall S$ , cho phép chứng minh tính đúng đắn (CMTĐĐ) toàn cục của chương trình.

Một hệ thống CMTĐĐ toàn cục sử dụng điều kiện trước yếu nhất bao gồm :

- (a) Với mỗi lệnh sơ cấp, các tiên đề cho điều kiện trước yếu nhất ứng với một điều kiện sau đã cho. Đối với phép gán, các tiên đề đã cho ở mục I nói chung sẽ thay thế vai trò này.
- (b) Các quy tắc suy diễn cho phép xây dựng điều kiện trước yếu nhất của một lệnh không sơ cấp P, xuất phát từ các điều kiện trước yếu nhất của các lệnh trong P.

##### **Ví dụ 16 :**

Giả sử cần xác định :  $\text{pfpre}(\text{if } B \text{ then } P \text{ else } Q, S)$

Nếu B cho một kết quả, ta có :

$$\begin{aligned} & \text{pfpre}(\text{if } B \text{ then } P \text{ else } Q, S) \\ & \sim (B \wedge \text{pfpre}(P, S)) \vee (\neg B \wedge \text{pfpre}(Q, S)) \end{aligned}$$

Nhưng việc tính B để cho một kết quả nếu và chỉ nếu  $\text{pfpre}(\text{if } B \text{ then } , \text{true})$  là đúng.

Như vậy trong trường hợp tổng quát :

**pfpre (if B then P else Q, S)**  
 $\sim$ if pfpre (if B then , true)  
 then (B  $\wedge$  pfpre (P, S)  $\vee$  ( $\neg$ B  $\wedge$  pfpre (Q, S)) else false

Không có quy tắc đơn giản cho vòng lặp while.

Độ phức tạp của các quy tắc đưa ra để xác định dkt của một vòng lặp while có thể được minh họa như sau :

– Vấn đề dừng của vòng lặp while trừu tượng không là quyết định được.

Vì vậy, điều kiện

**pfpre (while B do P, true)**

không tính được cho mọi B, mọi P.

– Việc giải quyết một số bài toán nổi tiếng đưa về việc chứng minh một chương trình là dừng. Ví dụ, người ta không biết nếu :

**pfpre (while n > 1 do**  
 $n :=$ if not odd(n) then n div 2 else 3\*n+1, true)  $\sim$  (n >= 1)

Đây là sự giả định (conjecture) của Collatz.

**Chú ý :** Các tiên đề và các quy tắc suy diễn xét ở mục I cho phép chứng minh tính đúng đắn của các phát biểu E {P} S, trong đó E  $\neq$  pfpre (P, S). Ví dụ, ta đã chứng minh tính đúng đắn từng phần rằng :

(a  $\geq$  0) {Div} (a = bq + r)  $\wedge$  (q  $\geq$  0)  $\wedge$  (0  $\leq$  r < b)

### **b) Trường hợp điều kiện sau mạnh nhất**

Theo định nghĩa của pfpost, ta có : E {P} S  $\sim$  (pfpost (E, P)  $\rightarrow$  S)

Như vậy, một hệ thống chứng minh tính đúng đắn dựa trên việc tính toán các điều kiện sau mạnh nhất cho phép chứng minh tính đúng đắn *từng phần*.

Đối với lệnh điều kiện, quy tắc suy diễn được cho bởi tính chất :

**pfpost (E, if B then P else Q)**  
 $\sim$  pfpost (E  $\wedge$  B, P)  $\wedge$  pfpost (E  $\wedge$   $\neg$ B, Q)

với giá trị rằng B luôn luôn cho một kết quả.

Trong trường hợp tổng quát :

**pfpost (E, if B then P else Q)**  
 $\sim$  pfpost (E  $\wedge$  (if pfpre (if B then, true) then B else false) , P)  
 $\vee$  pfpost (E  $\wedge$  (if pfpre (if B then, true) then  $\neg$ B else false) , Q)

## IV.2. Các tiên đề gán

### IV.2.1. Điều kiện trước yếu nhất và điều kiện sau mạnh nhất của lệnh gán

Điều kiện trước E được tính toán theo quy tắc đã trình bày ở mục I để tạo ra các tiên đề gán là điều kiện trước yếu nhất của một lệnh gán  $x := \langle bt \rangle$  và của một điều kiện sau S, khi đại lượng  $\text{term}_E(x := \langle bt \rangle)$  có giá trị true. Trong trường hợp này, tiên đề được ký hiệu bởi :

$$\text{pfpref}(x := \langle \text{biểu thức} \rangle, S) \{ x := \langle bt \rangle \} S$$

Trong trường hợp tổng quát, ta có :

$$\text{pfpref}(x := \langle bt \rangle, S) \sim \text{if } \text{pfpref}(x := \langle bt \rangle, \text{true}) \text{ then } S(x / \langle bt \rangle) \text{ else false.}$$

Điều kiện  $\text{pfpref}(x := \langle bt \rangle, \text{true})$  thể hiện :

- x và các toán hạng của  $\langle bt \rangle$  được định nghĩa :  
chỉ số của mảng nằm giữa cận dưới và cận trên, các con trỏ chỉ đến các biến khác nil, v.v...,
- mọi phép toán thực thi đều cho kết quả :  
có sự tương thích về kiểu của các toán hạng, không có phép chia cho 0, v.v...

Ở đây ta không thấy sự vi phạm về tính hợp thức của các tiên đề đã nêu trong mục I về tính đúng đắn từng phần : nếu  $\text{term}_E(x := \langle bt \rangle)$  có giá trị false, phát biểu  $E \{ x := \langle bt \rangle \} S$  là true. Người ta có thể củng cố điều kiện trước của các tiên đề này bởi các điều kiện do  $\text{pfpref}(x := \langle bt \rangle, \text{true})$  đưa đến.

Ví dụ, nếu m và n lần lượt là cận dưới và cận trên của mảng a, thay vì :

$$(y > 0) \{ x := a[j] \} (y > 0) \text{ và} \\ (a[j] = y) \{ x := a[j] \} (x = y)$$

Ta có thể viết :  $(m \leq j \leq n) \wedge (y > 0) \{ x := a[j] \} (y > 0)$

$$\text{if } (m \leq j \leq n) \text{ then } (a[j] = y) \text{ else false} \quad \{ x := a[j] \} \quad (x = y)$$

Theo mệnh đề III.1.1, nếu  $E(x := \langle bt \rangle) S$  là một tiên đề, S không nhất thiết là điều kiện sau mạnh nhất ứng với E. Như vậy các tiên đề gán nói chung không thể ký hiệu :

$$E(x := \langle bt \rangle) \text{pfpref}(E, x := \langle bt \rangle)$$

Chẳng hạn ta có tiên đề :  $(\frac{1}{x} \geq 0) \{ x := 1/x \} (x \geq 0)$

Ta có :  $(\frac{1}{x} \geq 0) \sim (x > 0)$ , nhưng  $\text{pfpref}(x > 0, x := 1/x) \sim (x > 0)$

Bây giờ ta sẽ xét các quy tắc xây dựng các tiên đề từ điều kiện trước về điều kiện sau. Các tiên đề này sẽ luôn luôn được ký hiệu bởi :

$$E(x := \langle bt \rangle) \text{pfpref}(E, x := \langle bt \rangle)$$

nhưng không thể ký hiệu bởi :  $\text{pfpref}(x := \langle bt \rangle, S) \{ x := \langle bt \rangle \} S$

### IV.2.2. Quy tắc tính toán điều kiện sau mạnh nhất của một phép gán

Với lệnh gán  $x := \langle bt \rangle$ , chỉ có biến  $x$  bị thay đổi. Như vậy,  $f_{x := \langle bt \rangle}$  là một hàm từ tập hợp  $W$  các giá trị của các biến của chương trình vào chính nó, đặt đồng nhất các thành phần, trừ biến  $x$ .

Ta ký hiệu  $\pi_x f_{x := \langle bt \rangle}$  là phép chiếu của  $f_{x := \langle bt \rangle}$  vào thành phần này.

Ví dụ, với lệnh  $q := q+1$  của chương trình Div ở mục I, ta có :

$$f_{q := q+1}(a, b, q, r) = (a, b, q+1, r) \text{ và } \pi_q f_{q := q+1}(a, b, q, r) = q+1$$

Tương tự, quy tắc điều kiện trước về tiên đề gán đã cho ở mục I có thể viết :

$$E \sim S(x / \pi_x f_{x := \langle bt \rangle})$$

Bây giờ ta sẽ định nghĩa hai quy tắc tính toán điều kiện sau mạnh nhất của một lệnh gán.

#### a) Trường hợp đặc biệt

Nếu hàm thu hẹp  $f_{x := \langle bt \rangle}$  vào  $W_E$  là toàn cục và đơn ánh, sẽ tồn tại hàm ngược  $f_{x := \langle bt \rangle}^{-1}$

Trong trường hợp này, có thể sử dụng quy tắc sau đây để tính pfpst :

$$\text{pfpst}(E, x := \langle bt \rangle) \sim E(x / \pi_x f_{x := \langle bt \rangle}^{-1})$$

nghĩa là ta nhận được điều kiện sau mạnh nhất của một điều kiện trước  $E$  và một phép gán  $x := \langle bt \rangle$  bởi việc thay thế những nơi  $x$  xuất hiện trong điều kiện  $E$  bởi hàm ngược của  $f_{x := \langle bt \rangle}$ .

Từ quy tắc tính pfpst, ta có tiên đề gán từ điều kiện trước và điều kiện sau như sau :

$$E \{ x := \langle bt \rangle \} E(x / \pi_x f_{x := \langle bt \rangle}^{-1})$$

Sự tồn tại hàm ngược của  $f_{x := \langle bt \rangle}^{-1}$  cho phép tìm được giá trị của  $x$  trước khi thực hiện phép gán, từ các giá trị các biến sau khi gán.

**Chú ý :**  $\text{pfpst}(q \geq 0, q := q+1) \sim (q-1 \geq 0)$

$$\text{pfpst}(q \geq 0, q := q*2) \sim \left(\frac{q}{2} \geq 0\right)$$

$$\text{pfpst}(q = 0, q := q+a) \sim (q-a = 0)$$

Ta nhận được các tiên đề sau đây :

$$(q \geq 0) \{ q := q+1 \} (q-1 \geq 0)$$

$$(q \geq 0) \{ q := q*2 \} \left(\frac{q}{2} \geq 0\right)$$

$$(q = 0) \{ q := q+a \} (q-a = 0)$$

Quy tắc trên không dùng được cho các phép gán như  $q := 0$ , hoặc  $q := q \text{ div } 2$ .



Ta có thể dùng quy tắc để chứng minh chương trình theo chiều bắt đầu – kết thúc.

Ví dụ, để chứng minh  $(x > 0) \{ x := x*2 \} (x > 0)$  :

- Xét quy tắc từ điều kiện sau về điều kiện trước :

$$(2x > 0) \{ x := x*2 \} (x > 0)$$

Sử dụng quy tắc điều kiện trước, ta nhận được kết quả vì :

$$(2x > 0) \rightarrow (x > 0)$$

- Xét quy tắc từ điều kiện trước về điều kiện sau :

$$(x > 0) \{ x := x*2 \} \left(\frac{x}{2} > 0\right)$$

Sử dụng quy tắc điều kiện sau, ta nhận được kết quả vì :  $\left(\frac{x}{2} > 0\right) \rightarrow (x > 0)$

**Chú ý :** Các tiên đề gán đã đưa ra trong các ví dụ trên có thể nhận được từ điều kiện sau bởi tiên đề gán ở mục I. Trong các ví dụ này, hàm  $f_x := \langle bt \rangle : W \rightarrow W$  là đơn ánh.

Ví dụ sau đây chỉ ra rằng nếu hàm  $f_x := \langle bt \rangle : W \rightarrow W$  không là đơn ánh, người ta không còn có tính chất này.

### Ví dụ 17 :

Hàm  $f_x := x \text{ div } 2$  là toàn thể, nhưng không đơn ánh. Thu hẹp của nó vào  $W_{x=2y}$  là đơn ánh.

Áp dụng quy tắc tính pfpst, ta có tiên đề :  $(x = 2y) \{ x := x \text{ div } 2 \} (2x = 2y)$

tương đương với :  $(x = 2y) \{ x := x \text{ div } 2 \} (x = y)$

Tuy nhiên, xuất phát từ điều kiện sau  $(2x = 2y)$  và sử dụng tiên đề gán ở mục I, ta có tiên đề :

$$\left(2 \left\lfloor \frac{x}{2} \right\rfloor = 2y\right) \{ x := x \text{ div } 2 \} (2x = 2y)$$

tương đương với :  $\left(2 \left\lfloor \frac{x}{2} \right\rfloor = 2y\right) \{ x := x \text{ div } 2 \} (x = y)$

### b) Trường hợp tổng quát

Nếu không tồn tại hàm ngược của  $f_x := \langle bt \rangle$ , sẽ không thể tìm được giá trị ban đầu của  $x$ , xuất phát từ giá trị các biến sau khi thực hiện phép gán.

## V. Bài tập

**Bài 1 :** Sử dụng các quy tắc trên đây, chứng minh các tính chất sau :

1. Nếu  $E \{P\} S$  và  $E' \{P\} S'$  là các định lý, thì  $E \wedge E' \{P\} S \wedge S'$  và  $E \vee E' \{P\} S \vee S'$  cũng là các định lý.
2. Nếu  $E \{P\} F$  và  $G \{Q\} S$  là các định lý và nếu  $F \rightarrow G$ , thì  $E \{P ; Q\} S$  là một định lý.
3. Sử dụng quy tắc ";", chứng minh rằng :  $(y = 2) \{ x := y+1 ; z := x+y \} (z = 5)$
4. Sử dụng quy tắc điều kiện trước, chứng minh rằng :  $(q \geq 0) \{ q := q + 1 \} (q \geq 0)$
5. Phép thế sử dụng trong quy tắc trên đây có thể được viết  $E = S(x / \langle bt \rangle)$ . Trong điều kiện đặc biệt của một biểu thức điều kiện trước, phép thế này được định nghĩa bởi :

$$S(x / \text{if } a \text{ then } e_1 \text{ else } e_2) \stackrel{\text{dn}}{=} (a \wedge S(x / e_1)) \vee (\neg a \wedge S(x / e_2))$$

Sử dụng quy tắc trên đây để chứng minh :

$$(z \geq 0) \{ z := \text{if } b > 0 \text{ then } z + b \text{ else } z - b \} (z \geq 0)$$

**Bài 2 :** Chứng minh Div dừng :

Bằng cách sử dụng hàm  $m'(w) = \left\lfloor \frac{A}{B} \right\rfloor - q$ , trong đó  $\left\lfloor \frac{A}{B} \right\rfloor$  là phần nguyên của phép chia  $\frac{A}{B}$ .

Chứng minh sau khi ra khỏi vòng lặp,  $m'(w) = 0$ .

**Bài 3 :** Từ chương trình P trong mục II.2.1, hãy :

1. Chứng minh hình thức chương trình P.
2. Cho biết số lần tối đa thực hiện vòng lặp của P.
3. Sử dụng chương trình Div của mục trước để suy diễn từ P một chương trình tính USCLN của hai số nguyên dương A và B bởi việc tính số dư liên tiếp.

**Bài 4 :** Từ chương trình ( $I^{\text{bis}}$ ) trong mục II.2.2, hãy :

1. Chứng minh rằng chương trình ( $I^{\text{bis}}$ ) thoả mãn các ràng buộc của bài toán.
2. Chứng minh rằng chương trình ( $I^{\text{bis}}$ ) không còn thoả mãn nếu thay thế vòng lặp trong cùng bởi :

```
while R(r) and (w<=r) do r:=r-1;
if w<r then begin hoánvị(r, w); r:=r-1 end;
```

Chứng minh rằng chương trình này có thể dẫn đến tính R(0).

# Thử nghiệm chương trình

Như đã trình bày trong chương trước, người ta thường sử dụng các *kỹ thuật tĩnh* (static techniques) và *kỹ thuật động* (dynamic techniques) trong quá trình V&V để kiểm tra tính đúng đắn của một sản phẩm phần mềm.

Chương này sẽ trình bày một phương pháp tĩnh là *khảo sát* (inspection) chương trình, với vai trò như là một phép chứng minh phi hình thức và, một phương pháp động là thử nghiệm (testing) chương trình.

## I. Khảo sát

Khảo sát (hay thanh tra) là những cuộc họp nhằm mục đích xác minh một sản phẩm. Phần lớn các phương pháp sản xuất phần mềm đều ấn định trước những cuộc họp như vậy. Tùy theo bản chất của sản phẩm cần khảo sát, người ta nói về khảo sát *thiết kế toàn thể* (global design), khảo sát *thiết kế chi tiết* (detailed design), và khảo sát mã nguồn.

Một kịch bản mẫu (typical scenario) cho một khảo sát mã nguồn như sau :

1. Cần đến 4 người gồm một chủ tịch, một người lập trình, một người thiết kế và một khảo sát (đều là những chuyên gia về Tin học, riêng khảo sát phải có kiến thức chuyên môn về lĩnh vực ứng dụng của sản phẩm).
2. Các thành viên nhận chương trình nguồn và các đặc tả trước cuộc họp ít ngày để đọc và chuẩn bị.
3. Cuộc họp kéo dài khoảng 1 giờ 30 đến khoảng 2 giờ.
4. Trong quá trình họp khảo sát :
  - Người lập trình đọc và giải thích chương trình của mình, có thể đọc từng dòng lệnh một và trả lời các câu hỏi được đặt ra.
  - Chương trình được phân tích căn cứ trên một danh sách các lỗi sai (errors) thông dụng do khảo sát cung cấp.
5. Cuộc họp không sửa lỗi tìm thấy mà chỉ ghi nhận qua biên bản mà thôi. Chính người lập trình sẽ tự sửa lỗi sau khi họp xong.
6. Nếu khi khảo sát tìm thấy trong chương trình, nhiều khiếm khuyết (failures), hoặc nhiều lỗi trầm trọng thì phải tiếp tục khảo sát lần sau, sau khi sửa lỗi.

Một số kịch bản coi trọng việc tìm lỗi sai và khuyến khích việc chạy demo trực tiếp mã chương trình (hand made) nguồn : khảo sát mang đến cuộc họp cách tiến hành và các dữ liệu liên quan để mọi người tiến hành thử nghiệm. Người ta còn gọi cách thử nghiệm như vậy là *walk throughs* (chạy suốt).

Một số kịch bản lại coi trọng việc chứng minh không hình thức : khảo sát đề nghị xác minh các tính chất cho phép thử nghiệm tính đúng đắn của sản phẩm. Người ta nói đây là việc khảo sát căn cứ trên việc xác minh.

Việc kiểm lại (review) khác với khảo sát vì rằng việc kiểm lại không đòi hỏi phải họp : Sản phẩm được giao cho những người không tham gia vào việc lập trình, họ có những khuynh hướng đánh giá độc lập.

Có thể nói phương pháp khảo sát có hiệu quả đáng kể : những số liệu tìm thấy trong các văn bản ghi nhận khoảng 50% sai sót được phát hiện khi khảo sát. Những con số dưới đây (lấy từ tạp chí IEEE<sup>3</sup> năm 1992 của Dyer M. từ bài báo "Verification Based Inspection") cho thấy các sai sót tìm thấy khi phát triển dự án 5 phần mềm của hãng IBM :

Dự án	Khảo sát thiết kế toàn bộ	Khảo sát thiết kế chi tiết	Khảo sát mã	Thử nghiệm đơn vị	Thử nghiệm hệ thống
1			50	25	25
2	4	13	49	17	17
3	20	27	10	20	23
4	20	26	22	18	36
5	10	18	24	24	24

Một phương pháp khác, gọi là *phương pháp phòng sạch* (Clean-room Methodology), thay vì thử nghiệm (testing), khuyến khích việc khảo sát (inspection) bằng cách xác minh (verification) trong quá trình sản xuất phần mềm. Sự phát triển phần là liên tiếp làm mịn (refinement) sản phẩm. Mỗi giai đoạn, người ta tiến hành chứng minh tính đúng đắn (proving) một cách chặt chẽ, đồng thời với các cuộc khảo sát, sao cho sản phẩm phần mềm không chứa sai sót.

Việc thử nghiệm chỉ được tiến hành khi xác minh hậu nghiệm (a posteriori) nhờ các phương pháp thống kê, nhằm đạt được mục đích đặt ra lúc đầu. Phương pháp phòng sạch do H.Mills xây dựng tại IBM, đã được áp dụng để sản xuất các phần mềm cỡ lớn.

<sup>3</sup> IEEE, đọc là eye-triple-ee, tên viết tắt của Institute of Elechtrric and Engineers.

## II. Các phương pháp thử nghiệm

Phương pháp thử nghiệm là cho chạy chương trình từ một số dữ liệu thử được chọn trước. Phép thử nghiệm dùng cho cả hai quá trình xác minh và hợp thức hóa V&V, với điều kiện rằng chương trình là chạy được.

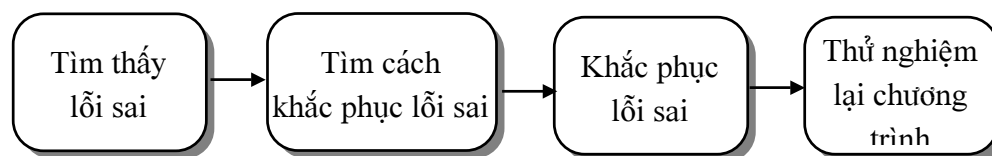
Việc thử nghiệm phân biệt :

1. Các phép chứng minh tính đúng đắn hay khảo sát mã nguồn mà không chạy chương trình, với quy trình "walkthroughs" bằng cách chạy demo (hand-made).
2. Chạy chương trình debugger để tìm sửa lỗi.

Các thử nghiệm và chạy debugger thường do các nhóm công tác khác nhau đảm nhiệm.

Để nâng cao hiệu quả, người ta thường phân công như sau : nhóm thử nghiệm là nhóm không lập trình, còn nhóm chạy debugger là nhóm tham gia lập trình ra sản phẩm. Quá trình debugger gồm nhiều giai đoạn :

1. Tìm thấy lỗi sai (Locate error)
2. Tìm cách khắc phục lỗi sai (Design error repair)
3. Khắc phục lỗi sai (Error repair)
4. Thử nghiệm lại chương trình (Re-test program)



Hình 4.1. Quá trình debugger

Với những phương pháp lập trình truyền thống, quá trình V & V là khảo sát và thử nghiệm chương trình. Thực tế, việc thử nghiệm chiếm một phần đáng kể trong quá trình sản xuất phần mềm, chiếm khoảng từ 30% đến 50%, tùy theo bản chất của dự án Tin học.

### II.1. Định nghĩa và mục đích thử nghiệm

Người ta đưa ra những định nghĩa sau đây :

- Một *thử nghiệm* là cho chạy (run) hay thực hiện (execution) một chương trình từ những dữ liệu được lựa chọn đặc biệt, nhằm để xác minh kết quả nhận được sau khi chạy là đúng đắn.
- Một *tập dữ liệu thử* là tập hợp hữu hạn các dữ liệu trong đó mỗi dữ liệu phục vụ cho một thử nghiệm.

- Mỗi *phép thử nghiệm* chỉ ra hoạt động từ việc thiết kế các tập dữ liệu thử, tiến hành thử nghiệm và đánh giá kết quả đến các giai đoạn khác nhau trong chu kỳ sống của phần mềm.
- *Người thử nghiệm* (hay *nhóm thử nghiệm*) có kiến thức chuyên môn Tin học có nhiệm vụ tiến hành phép thử nghiệm.
- Một *khiếm khuyết* (failure) xảy ra khi chương trình thực hiện cho ra kết quả không tương hợp với đặc tả của chương trình.
- Một lỗi sai (error) là một phần chương trình (lệnh) đã gây ra khiếm khuyết.

Người thử nghiệm có nhiệm vụ :

1. Tạo ra tập dữ liệu thử.
2. Triển khai các phép thử.
3. Lập báo cáo về kết quả thử nghiệm và lưu giữ.

Mục đích thử nghiệm là để :

### 1. Chứng minh rằng chương trình là đúng đắn

Để khẳng định tính đúng đắn của chương trình, cần tiến hành các thử nghiệm toàn bộ (exhaustive testing), đòi hỏi tập dữ liệu thử phải hữu hạn và có kích thước vừa phải sao cho đủ sức thuyết phục. Điều này trên thực tế rất khó thực hiện.

Sau đây là một tiêu chuẩn nổi tiếng của Dijkstra : "Các thử nghiệm cho phép chứng minh một chương trình là không đúng, bằng cách chỉ ra một phản ví dụ, tuy nhiên, không bao giờ có thể chứng minh được chương trình đó là đúng đắn".

### 2. Gây ra những khiếm khuyết của chương trình

Myers G. J. trong bài báo "The Art of Software Testing", Wiley 1979, đã định nghĩa thử nghiệm như sau :

"Phép thử nghiệm là cho chạy chương trình nhằm tìm ra những sai sót".

Từ đó, thường người ta nói về "thử nghiệm phá hủy" (destructive testings). Mục đích của những phép thử như vậy là tập trung tìm ra các lỗi sai từ những khiếm khuyết do người lập trình phạm phải. Người thử nghiệm tiến hành với mục đích nghịch (negative) : phép thử là thành công nếu tìm ra được khiếm khuyết, là thất bại trong trường hợp ngược lại. Việc thử nghiệm kiểu này thường được áp dụng trong quá trình viết chương trình.

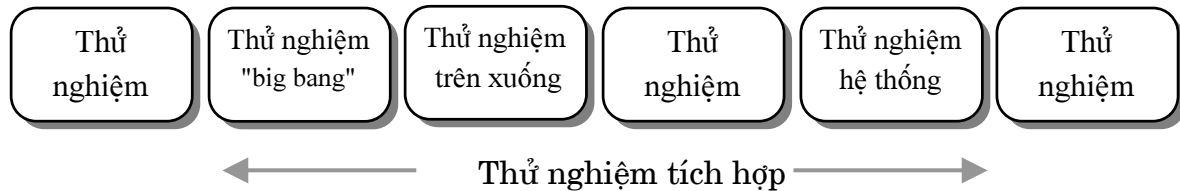
### 3. Đưa ra đánh giá tĩnh (static evaluation - static benchmark) về chất lượng của chương trình.

Người ta sử dụng phương pháp "thử nghiệm tĩnh" (static testing) cho mục đích này. Trong phương pháp phòng tránh, người ta chỉ tiến hành những phép thử tĩnh,

nhằm mục đích vừa đảm bảo công việc của người lập trình vừa đánh giá sự tin cậy của sản phẩm vận hành.

## II.2. Thử nghiệm trong chu kỳ sống của phần mềm

Người ta phân biệt nhiều phương pháp thử nghiệm, tương ứng với các giai đoạn sản xuất phần mềm khác nhau.



Hình 4.2. Nhiều phương pháp thử nghiệm

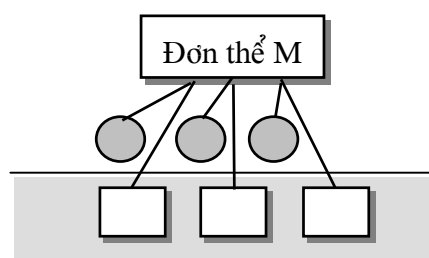
### II.2.1. Thử nghiệm đơn thể

Thử nghiệm đơn thể (Module testing), hay thử nghiệm đơn vị (Unit testing) do người lập trình tự tiến hành. Phương pháp này hay được sử dụng trong lập trình cấu trúc (top-down programming). Các phương pháp thử nghiệm khác do người thử nghiệm tiến hành.

Giả sử gọi M là một đơn thể cần thử nghiệm riêng biệt. Khi đó, xảy ra hai trường hợp như sau :

**Trường hợp 1** : những đơn thể do M gọi tới không có mặt lúc thử nghiệm.

Khi đó, những đơn thể do M gọi tới vắng mặt phải được thay thế bởi các chương trình cùng một giao diện với M. Các chương trình này thực hiện đúng chức năng mà chúng đại diện cho đơn thể vắng mặt và chúng được gọi là các trình stubs ("cuồng").



Hình 4.3. Các đơn thể vắng mặt được thay bởi các trình stubs

Ví dụ, nếu đơn thể đang được thử nghiệm gọi một thủ tục sắp xếp ở đầu :

```
Procedure Sort (T: Array ; n: Integer) ;
```

người ta có thể sử dụng trình Stub :

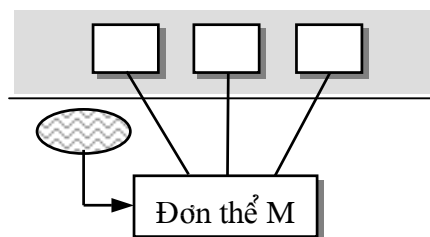
```
Procedure Sort (T: Array ; n: Integer) ;
  Writeln ('Dãy cần sắp xếp là : ') ;
  for i:= 1 to n do writeln (T[i]) ;
```

```
for i:= 1 to n do readln (T [i]) ;
```

Tiếp theo, người thử nghiệm sẽ tiến hành sắp xếp dãy đã nhập bằng tay để thay thế cho thủ tục sắp xếp vắng mặt.

**Trường hợp 2 :** những đơn thể gọi tới M không có mặt lúc thử nghiệm.

Khi đó, đơn thể gọi tới M nhưng vắng mặt phải được thay thế bởi một chương trình, được gọi là trình driver. Trình driver gọi M để M thực hiện trên các dữ liệu thuộc tập dữ liệu thử, sau đó ghi nhận các kết quả tính được bởi M để so sánh với các kết quả chờ đợi.



Hình 4.4. Dùng trình driver để gọi thực hiện M

Số lượng các trình stubs và các trình drivers cần thiết để tiến hành thử nghiệm các đơn thể phụ thuộc vào thứ tự các đơn thể được thử nghiệm.

### II.2.2. Thử nghiệm tích hợp

Thử nghiệm tích hợp vừa nhằm tạo mối liên kết giữa các đơn thể, vừa được tiến hành đối với những đơn thể lớn hình thành hệ thống chương trình hoàn chỉnh. Có nhiều phương pháp thử nghiệm tích hợp.

#### 1. Phương pháp "big bang"

Người ta xây dựng mối liên hệ giữa các đơn thể để tạo thành một phiên bản hệ thống hoàn chỉnh, sau đó thử nghiệm phiên bản này.

Như vậy người ta cần đến nhiều trình drivers, mỗi trình driver cho một đơn thể, một trình stubs cho tất cả các đơn thể của hệ thống, trừ đơn thể chính phải được thử nghiệm bằng phương pháp thử nghiệm đơn vị.

Phương pháp "big bang" nguy hiểm : tất cả các sai sót có thể đồng thời xuất hiện, việc xác định từng lỗi sai sẽ gặp khó khăn. Hơn nữa phương pháp này đòi hỏi một lượng tối đa các trình drivers và các trình stubs. Vì vậy thường người ta sử dụng các phương pháp tích hợp từ trên xuống, hay từ dưới lên.

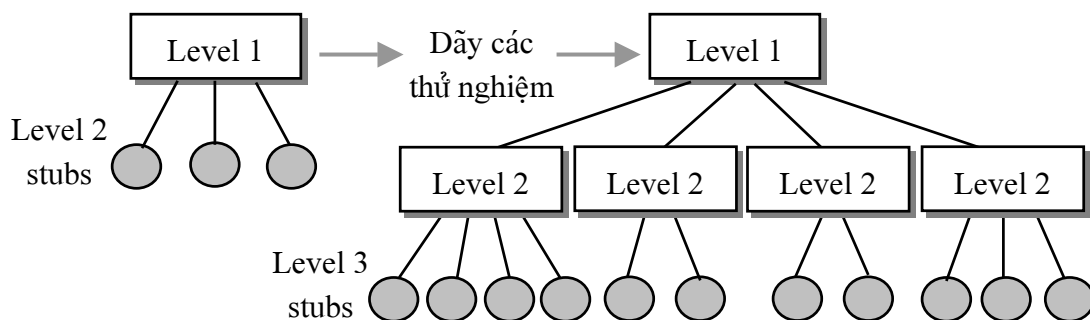
#### 2. Phương pháp thử nghiệm từ trên xuống

(Descendant hay Top-down Testing Method)

Bắt đầu thử nghiệm đơn thể chính, sau đó thử nghiệm chương trình nhận được từ sự liên kết giữa đơn thể chính và các đơn thể được gọi trực tiếp từ đơn thể chính, v.v...



Phương pháp này chỉ cần dùng một trình driver duy nhất cho đơn thể chính, nhưng cần một trình stub cho mỗi đơn thể còn lại.

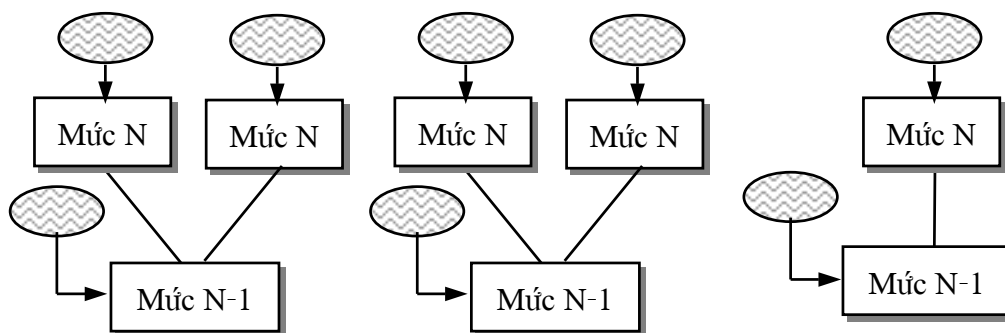


Hình 4.5. Phương pháp thử nghiệm từ trên xuống

### 3. Phương pháp thử nghiệm từ dưới lên

(Ascendant hay Bottom-Up Testing Method)

Bắt đầu thử nghiệm các đơn thể không gọi đến các đơn thể khác, sau đó các chương trình nhận được bởi sự liên kết giữa một đơn thể chỉ gọi đến các đơn thể đã được thử nghiệm với các đơn thể này, v.v . . . Phương pháp này đòi hỏi mỗi đơn thể một trình driver, nhưng không cần trình stub.



Hình 4.6. Thử nghiệm từ dưới lên

Phương pháp tiến tỏ ra ưu điểm hơn phương pháp lùi, do các trình driversử dụng dễ viết hơn các trình stubs và có các công cụ để tạo sinh tự động các trình drivers. Mặt khác, thứ tự tích hợp thường bị ràng buộc bởi thứ tự có mặt của các đơn thể.

#### II.2.3. Thử nghiệm hệ thống

Vấn đề là thử nghiệm phần mềm hoàn chỉnh và phân cứng để đánh giá hiệu năng, độ an toàn, tính tương hợp với các đặc tả, v.v . . .

Những thử nghiệm này đòi hỏi nhiều thời gian. Người ta nói đến thử nghiệm chấp nhận (Acceptance testing), là những thử nghiệm phù hợp với sản phẩm cuối cùng qua hợp đồng đã ký với khách hàng (nhiều khi việc thử nghiệm này do khách hàng tiến hành), còn được gọi là thử nghiệm alpha và cuối cùng là thử nghiệm cài đặt (Setup Testing), là thử nghiệm đối với sản phẩm cuối cùng, tiến hành tại vị trí

của khách hàng (với các máy tính và hệ điều hành họ đang sử dụng). Người ta gọi các thử nghiệm cho phiên bản đầu tiên của phần mềm do khách hàng được lựa chọn đặc biệt tiến hành là thử nghiệm beta.

#### **II.2.4. Thử nghiệm hồi quy**

Người ta còn gọi các thử nghiệm tiến hành sau khi sửa lỗi là thử nghiệm hồi quy, hay thoái lui (regression testing) nhằm để xác minh nếu các sai sót khác không được xử lý khi sửa chữa. Khi thử nghiệm này hay được dùng trong khi bảo trì. Để tiến hành hiệu quả các thử nghiệm này, cần lưu giữ lại những thử nghiệm đã làm trong quá trình sản xuất phần mềm, điều này giúp cho việc xác minh tự động các kết quả thử nghiệm thoái lui. Khó khăn gặp phải là trong số những thử nghiệm đã dặt dấn, cần phải chọn những thử nghiệm nào cho thử nghiệm thoái lui. Phương cách người ta hay làm là kết hợp mỗi lệnh của chương trình với tập hợp các thử nghiệm làm chạy chương trình.

### **II.3. Dẫn dắt các thử nghiệm**

Việc dẫn dắt các thử nghiệm bao gồm :

- Xác định kích thước của tập dữ liệu thử (vấn đề kết thúc các TN).
- Lựa chọn các dữ liệu cần thử nghiệm.
- Xác định tính đúng đắn hay không đúng đắn của các kết quả nhận được sau khi thực hiện chương trình đối với các dữ liệu của tập dữ liệu thử (Vấn đề lời tiên tri - oracle).

Việc dẫn dắt các thử nghiệm kèm theo việc viết các chương trình hỗ trợ như là các stubs và các drivers.

Vấn đề lời tiên tri

Một phép thử nghiệm (check program)

Một tập hợp hữu hạn các giá trị đưa vào.

Một tập hợp hữu hạn các cặp (Giá trị đưa vào, kết quả tương ứng).

Trong trường hợp 1, việc phát hiện ra các khiếm khuyết phải được làm bằng tay (by hand), từ đó dẫn đến một công việc xem xét kỹ lưỡng các kết quả mất thì giờ và mệt mỏi (làm hạn chế kích thước các tập dữ liệu thử). Có hai kiểu sai sót xảy ra khi xem xét :

- Một kết quả sai lại được xem như là đúng.
- Một kết quả đúng có thể được hiểu là sai.

Để tránh xác minh bằng tay, cần phải có những đặc tả khả thi, hay một phiên bản khác của chương trình (điều này có nguy cơ làm lan truyền sai sót từ phiên bản này sang phiên bản khác).

Trong trường hợp thứ hai, chính chương trình đang chạy tự phát hiện ra các khiếm khuyết, vấn đề là tìm ra được các giá trị đưa ra kết quả tương ứng với giá trị đưa vào. Điều này có thể làm "bằng tay" với một đặc tả khả thi, với một phiên bản của chương trình, với cùng những vấn đề đã gặp trong trường hợp đầu. Người ta cũng có thể vận dụng các phép thử cũ đã lưu giữ.

Chú ý rằng dùng chương trình xác minh tính đúng đắn của kết quả không luôn luôn đơn giản: nếu xảy ra có nhiều cái ra đúng tương ứng với một cái vào thì phải đặt kết quả do chương trình tính ra dưới dạng quy tắc trước khi xác minh tính nhất quán với kết quả dự kiến trong tập dữ liệu thử. Điều này không phải luôn luôn làm được. Chẳng hạn làm sao có thể xác minh được rằng mã sinh ra bởi một trình biên dịch là đúng đắn, nếu chỉ thử nghiệm mã đó mà thôi?

## II.4. Thiết kế các phép thử phá hủy (Defect Testing)

### II.4.1. Các phương pháp dựa trên chương trình

Các phương pháp này còn được gọi là phương pháp có cấu trúc (Structural Testing) hay thử nghiệm hộp trắng (white-box hay glass-box).

Mỗi chương trình tương ứng với một sơ đồ khối gồm các cấu trúc lựa chọn và các cấu trúc khối là một dãy tối đa các lệnh thực hiện (gồm các lệnh gán, lệnh gọi chương trình con, các lệnh vào-ra...) mà không có lệnh rẽ nhánh. Người ta gọi các *khối lệnh* là các đầu vào sơ đồ khối và các *quyết định* là các cung đi ra từ một cấu trúc lựa chọn.

#### a) Phủ các lệnh (các đỉnh)

Một phép thử phủ là phủ (trùm) hết các lệnh của một chương trình nếu làm cho mỗi lệnh của nó được thực hiện. Đây là một tiêu chuẩn tối thiểu: Người ta không xét những thử nghiệm mà mỗi lệnh của chương trình không được thực hiện ít nhất một lần.

Chú ý rằng tiêu chuẩn này không phải luôn luôn thỏa mãn bằng một chương trình có thể chứa các lệnh mà không thể được thực hiện.

#### b) Phủ các quyết định (các cung)

Một phép thử phủ các quyết định nếu trong khi thực hiện, mỗi cung của sơ đồ tổ chức của chương trình được duyệt qua ít nhất một lần: nghĩa là nếu mỗi phép chọn được thực hiện ít nhất một lần cho mỗi giá trị có thể (thuê hay fals e trong trường hợp ghép rẽ nhánh logic).

Như vậy, tiêu chuẩn này không phải luôn cần phải thỏa mãn.

Ví dụ: `if A > 0 then if A ≥ 0 then...else...`

### c) Phủ các điều kiện

Ta xét một chương trình chứa cấu trúc rẽ nhánh logic gồm các lệnh not, and và or. Một phép thử phủ các điều kiện nếu việc thực hiện chương trình kéo theo sự tính giá trị của biểu thức này cho mọi giá trị logic có thể. Như vậy một biểu thức có hai toán hạng P, Q sẽ được tính toán với :

	A	B
	true	true
	true	false
	false	true
	false	false

Phép phủ các điều kiện cho phép củng cố phép phủ các quyết định. Ví dụ có thể phủ các quyết định bằng cách thực hiện phép lựa chọn P và Q với :

$P = \text{true}$ ,  $Q = \text{true}$  và  $P = \text{false}$ ,  $Q = \text{false}$ ,

điều này không cho phép phân biệt phép rẽ nhánh A or B.

### d) Phủ các lộ trình thực hiện chương trình (path testing)

Một phép thử phủ các lộ trình chạy chương trình nếu gây ra việc thực thi mỗi lộ trình thực hiện chương trình. Không tồn tại phép thử như vậy nếu chương trình có vô hạn lộ trình thực hiện trong trường hợp tổng quát. Thông thường người ta xây dựng phép thử phủ các lộ trình thực hiện có số lượng  $\leq$  một hằng đã cho.

### e) Xác định dữ liệu cho phép phủ lộ trình thực hiện đặc biệt

Giả thiết rằng với mọi lệnh P của chương trình và mọi quyết định S, có thể tính  $ptpre(P, S)$ , điều kiện đầu yếu nhất ứng với P và S. Người ta có thể với mọi lộ trình của chương trình, tính được một công thức E sao cho các dữ liệu của chương trình thỏa mãn E nếu và chỉ nếu việc thực hiện của chương trình đi theo lộ trình đã chọn.

Đặc biệt, E không là sai nếu và chỉ nếu lộ trình đã chọn là lộ trình thực thi. Như vậy chỉ cần tìm ra các dữ liệu làm thỏa mãn E để có phép thử phủ lộ trình đã chọn. Điều này có thể thực hiện bằng ta, hay chứng minh một cách sáng tạo công thức  $xE$ .

Phương pháp này được dùng để định nghĩa phép thử phủ các quyết định của một chương trình :

- Lựa chọn một tập hợp các lộ trình phủ các quyết định.
- Với mỗi lộ trình, tính điều kiện đầu yếu nhất tương ứng (hoặc một điều kiện đầu mạnh hơn).
- Tìm các dữ liệu thỏa mãn các điều kiện điều này.

### f) Phủ các luồng dữ liệu

Với mỗi biến của chương trình, người ta gọi định nghĩa là một trường hợp của biến đó, một giá trị được gán cho biến (ví dụ :  $x:=1$ , `readln(x)` ...). Người ta gọi sử dụng là một trường hợp mà giá trị của biến được sử dụng (ví dụ :  $y:=x+y$  đối với biến  $x$ ).

Trong các sử dụng, người ta phân biệt các sử dụng trong các lệnh không phải là lựa chọn, gọi là C- sử dụng, với C : calculus, các sử dụng trong các lệnh lựa chọn, gọi là P- sử dụng, với P : Predicate.

Một phép thử là phủ các C-sử dụng nếu với mỗi biến  $x$ , gây ra việc thực thi mới lộ trình giữa một định nghĩa  $x$  và một C-sử dụng đầu tiên của  $x$ .

Một phép thử là phủ các P-sử dụng nếu, với mỗi biến  $x$  gây ra việc thực thi mỗi lộ trình giữa một định nghĩa  $x$ , và một giá trị lựa chọn.

### II.4.2. Các phương pháp dựa trên đặc tả

Những phương pháp này còn được gọi là thử nghiệm chức năng (functional testing), hay thử nghiệm này, người ta không chú ý đến chương trình, mà chỉ làm việc với đặc tả chức năng của chương trình. Người ta có thể thiết kế tập dữ liệu thử trước khi viết chương trình.

#### a) Các thử nghiệm toàn thể (Exhaustive Testing)

Người ta thử nghiệm chương trình với tất cả dữ liệu có thể về mặt lý thuyết, điều này chỉ làm được nếu tập hợp dữ liệu thử là hữu hạn. Thực tế, ngay cả khi tập hợp dữ liệu là hữu hạn thì thời gian thực hiện chương trình cho các thử nghiệm toàn thể là quá lớn trong phần lớn trường hợp.

Ví dụ :

1. Tính  $\sqrt{x}$ , với  $x$  nguyên giữa 0 và  $2^{31}$

Với thời gian một thử nghiệm là 1s, khi đó mất  $2^{31} = 2147483648$  s.

Một năm có  $365 \times 24 \times 3600$ s = 31536000s.

Vậy thời gian một thử nghiệm toàn thể là  $\approx 68$  năm.

2. Thử nghiệm phép cộng các số nguyên giữa 0 và  $2^{31}$

Thời gian thử nghiệm một phép cộng là 1  $\mu$ s. Số lượng dữ liệu là :

$$2^{31} \times 2^{31} = 2^{62} \approx 9.22 \times 10^{18}$$

Thời gian thử nghiệm toàn thể là trên 292 471 năm.

#### b) Các thử nghiệm bởi các lớp tương đương (Equivalence partitioning)

Nguyên lý : Phân hoạch tập hợp dữ liệu thành một số hữu hạn lớp và lựa một phân tử (hay một mẫu phân tử) trong mỗi lớp. Người ta đặt trong cùng một lớp các

dữ liệu được cho là phù hợp với chương trình theo cách đặc tả. Những dữ liệu này có thể cùng gây ra khiếm khuyết trong cùng tình huống.

Chú ý cần thử nghiệm các dữ liệu nằm ở phạm vi giáp ranh giữa các lớp tương đương để phát hiện các lỗi sai kiểu  $\leq$  thường lẫn với  $<$ , v.v . . .

### **c) Thử nghiệm định hướng bởi cú pháp (Syntax Controlled Testing)**

Khi dữ liệu là tập hợp các chuỗi ký tự (các ngôn ngữ lập trình), chúng được đặc tả bởi các ôtomat hữu hạn, hoặc bởi các văn phạm phi ngữ cảnh.

Ví dụ, nếu phần mềm được thử nghiệm có tính tương tác qua lại, như các hệ điều hành, thì tập hợp dãy các hành động có thể được định nghĩa bởi một ôtomat hữu hạn.

Người ta có thể định nghĩa các tập dữ liệu thử phủ các trạng thái đạt được, các cung, các lộ trình có độ dài bị chặn, v.v . . .

Khi tập hợp dữ liệu được định nghĩa bởi một văn phạm vi ngữ cảnh, người ta có thể xây dựng phép thử phủ các quy tắc của văn phạm (mỗi quy tắc được áp dụng ít nhất một lần để tiến hành một thử nghiệm).

Chú ý rằng lúc này, người ta chỉ có thể nhận được dữ liệu đúng, việc nhận được các dữ liệu sai bởi cùng phương pháp cần thiết phải viết một văn phạm sản sinh ra tập hợp các dữ liệu sai, điều này lại là một vấn đề hóc búa (vì rằng bù của một ngôn ngữ PNC chưa chắc đã là PNC).

### **d) Các thử nghiệm ngẫu nhiên (Random Testing)**

Đây là tập dữ liệu thử sử dụng các dữ liệu lấy ngẫu nhiên, tuân theo luật xác suất, chẳng hạn luật đồng đều, để tiến hành nhưng thường là kém hiệu quả.

## **II.4.3. Kết luận**

Hiện nay, người ta thường xây dựng phép thử nghiệm bằng cách phối hợp các thử nghiệm chức năng và thử nghiệm cấu trúc : người ta bắt đầu thử nghiệm chức năng trước (ngay khi đặc tả yêu cầu), sau đó hoàn thiện phép thử nghiệm bởi các tiêu chuẩn cấu trúc (bao bọc các lệnh, bao bọc các quyết định...) khi có được chương trình.

## **II.4.4. Các tiêu chuẩn kết thúc thử nghiệm**

Vấn đề đặt ra là *khi nào thì kết thúc thử nghiệm ?* hay cụ thể hơn là *xác định phạm vi thử nghiệm như thế nào ?*

Nếu kết thúc thử nghiệm sớm thì có thể chưa tìm hết lỗi trong chương trình. Còn nếu kết thúc muộn quá thì lại nâng cao giá thành sản phẩm. Sau đây là một số tiêu chuẩn :

1. *Dừng khi không còn gây ra được khiếm khuyết.*

Thường thì một chương trình lớn bao giờ cũng có lỗi, tiêu chuẩn này tỏ ra phi thực tế, hơn nữa mâu thuẫn với mục đích của các thử nghiệm phá hủy.

2. *Dừng khi thời gian (hay kinh phí) gia hạn cho thử nghiệm đã hết.*

Để tiêu chuẩn này có hiệu lực thì phải định lượng được tập hợp các dữ liệu thử trước khi tiến hành thử nghiệm.

3. *Căn cứ vào kinh nghiệm của các dự án tương tự đã hoàn tất.*

Một phép thử nghiệm bao bọc các quyết định (hay 80% của các cung) không gây ra khiếm khuyết. Vấn đề : Lựa chọn tùy tiện của tiêu chuẩn.

4. *Thử nghiệm chừng 70 sai sót không được phát hiện hay sau một thời hạn 3 tháng không xảy ra.*

Vấn đề : Ước lượng số lượng sai sót trong chương trình, ước lượng tỷ lệ % các sai sót được phát hiện bởi thử nghiệm, ước lượng tỷ lệ % sai sót phạm phải trong các giai đoạn phát triển phần mềm và tại giai đoạn thử nghiệm mà những sai sót này được phát hiện.

5. *Thử nghiệm đến khi số lượng sai sót tìm thấy không còn giảm theo một cách có ý nghĩa nữa.*

Vấn đề : Làm sao ước lượng được số sai sót đã giảm theo cách có ý nghĩa ?

6. *Phương pháp các đột biến (Mutant method)*

Người ta thay đổi chương trình bằng cách đưa vào các lỗi. Các chương trình bị thay đổi được gọi là các "đột biến". Một phép thử là "tốt" nếu diệt được 100% (95%, v.v . . .) các "đột biến" đó.

Vấn đề : Các sai sót đưa vào có phải là những sai sót thực tiễn (có thực)?

## II.5. Các phép thử nghiệm thống kê

### II.5.1. Mở đầu

Các phép thử nghiệm thống kê (Statistical Testing) nhằm để đo độ tin cậy (reliability) của phần mềm, nghĩa là đo xác suất chạy ổn định và đúng đắn trong những điều kiện sử dụng cho trước. Các thử nghiệm phá hủy không cho phép đánh giá được tính tin cậy của một chương trình vì rằng các thử nghiệm phá hủy không tính đến các điều kiện sử dụng như phương pháp này.

Người ta gọi *khiếm khuyết* (failure) là những hiện tượng bất thường xảy ra làm hệ thống đang thực thi dẫn đến những hiệu quả không phù hợp với đặc tả ban đầu. Một khiếm khuyết có thể xảy ra do phần cứng hoặc do một sai sót trong chương trình. Sau đây, người ta chỉ quan tâm đến những khiếm khuyết do lỗi phần mềm gây nên.

Trong những điều kiện sử dụng đã cho, sự xuất hiện thường xuyên các khiếm khuyết do các sai sót khác nhau gây ra là rất biến động : một số sai sót gây ra thường xuyên các khiếm khuyết, những sai sót khác thì rất hiếm, có thể không bao giờ xảy ra trên thực tế.

Việc thực thi một phần mềm với một dữ liệu cố định trước là một quá trình có tính xác định gây ra hoặc là một kết quả đúng, hoặc là một khiếm khuyết. Nếu người ta ở trong những điều kiện sử dụng chương trình, mỗi dữ liệu có thể được của chương trình sẽ cho một xác suất nào đó.

Tập hợp các dữ liệu cùng xác suất sử dụng như vậy được gọi là một mẫu sử dụng (use pattern) của chương trình. Từ một mặt cắt sử dụng đã cho, người ta định nghĩa xác suất một lần chạy cho một kết quả đúng và xác suất một khiếm khuyết, còn được gọi là tỷ suất khiếm khuyết.

Với một mô hình độc lập với thời gian, người ta định nghĩa độ tin cậy (reliability) của một chương trình như là xác suất của sự kiện " lần chạy sau của chương trình là đúng ", nghĩa là 1, xác suất của một khiếm khuyết.

Với một mô hình phụ thuộc thời gian, người ta định nghĩa độ tin cậy như là một xác suất của sự kiện "chương trình chạy đúng đắn trong thời gian t". Lúc này độ tin cậy là một hàm của thời gian.

Các mô hình phụ thuộc vào thời gian thường được sử dụng cho các phần mềm tương hỗ (như là các hệ điều hành). Tiếp theo đây, người ta sẽ chỉ khai triển các mô hình độc lập với thời gian.

Khi xuất hiện một khiếm khuyết, nếu là một khiếm khuyết về phần cứng, thì phải sửa chữa, nếu là một khiếm khuyết về phần mềm thì phải chạy trình sửa lỗi debugger.

Sửa chữa các hư hỏng thuộc về phần cứng là để thay thế những chi tiết hư hỏng, thiết lập lại sự vận hành ổn định của thiết bị như trước. Còn chạy trình debugger là để sửa các lỗi về thiết kế, tăng độ tin cậy của phần mềm.

Thường người ta sử dụng đại lượng liên quan đến độ tin cậy là số lần sử dụng trung bình cho đến khi xảy ra khiếm khuyết (đối với mô hình độc lập với thời gian), hoặc sử dụng sau một thời gian trung bình nào đó đến khi xảy ra khiếm khuyết (đối với mô hình phụ thuộc vào thời gian).

Đại lượng liên quan đến độ tin cậy MTTF (Mean Time To Failure : thời gian trung bình để xảy ra khiếm khuyết) được tính như sau :

Trong mô hình độc lập với thời gian :

Độ ổn định = xác suất một lần chạy đúng.  
= 1 - xác suất một khiếm khuyết.



MTTF = số lần sử dụng trung bình cho đến khi xảy ra  
 khiếm khuyết.  
 = 1 / xác suất một khiếm khuyết.  
 = 1 / (1 - Độ ổn định).

Tỷ suất khiếm khuyết là nghịch đảo của MTTF.

### **II.5.2. Ước lượng độ ổn định của một phần mềm**

Để ước lượng độ ổn định hay khả năng vận hành thông suốt (reliability) của một phần mềm, người ta căn cứ vào kết quả của các phép thử nghiệm thống kê, nghĩa là các thử nghiệm ngẫu nhiên tùy theo mẫu sử dụng đã chọn.

#### **a) Phương pháp trực tiếp**

Giả thiết rằng trong khi tiến hành n phép thử, người ta gặp d khiếm khuyết. Ta có thể ước lượng độ ổn định của phần mềm đang xét bởi biểu thức :

$$1 - d / n$$

Phương pháp này chỉ có thể đưa ra một ước lượng tốt về độ ổn định nếu số các khiếm khuyết d là có nghĩa (chẳng hạn độ tin cậy là 1 nếu khi thử nghiệm không xảy ra khiếm khuyết nào, điều này không có nghĩa).

Hơn nữa, nếu sau khi thử nghiệm, mà chạy trình debugger, thì chương trình sẽ bị thay đổi và việc ước lượng sẽ chỉ còn hợp thức một cách có điều kiện khi giả thiết về chất lượng của quá trình debugger.

#### **b) Phương pháp thử nghiệm giả thuyết (Hypothesis Testing)**

Vấn đề là xây dựng một tập hợp các phép thử nghiệm mà kết quả được ấn định trước cho phép khẳng định hay bác bỏ độ ổn định của phần mềm đang xét có một giá trị R với một độ tin cậy x%. R và x thoả mãn :

$$0 < R < 1 \text{ và } 0 < x < 100$$

Các tham số R và x cũng như quy cách về kết quả được cố định trước. Người ta nói chương trình là được kiểm nghiệm nếu có độ ổn định R.

Cho  $c = x/100$ , ta có :

$1 - c =$  xác suất cho một sản phẩm có độ ổn định thấp hơn độ ổn định R.

# Đặc tả phần mềm

## I. Mở đầu đặc tả phần mềm

### I.1. Khái niệm về đặc tả phần mềm

#### I.1.1. Đặc tả phần mềm là gì ?

Đặc tả (specification) được định nghĩa trong từ điển tiếng Việt (1997) : "*Mô tả thật chi tiết một bộ phận đặc biệt tiêu biểu để làm nổi bật bản chất của toàn thể*".

Theo Computer Dictionary của Microsoft Press<sup>®</sup> (1994), đặc tả là *sự mô tả chi tiết : Về mặt phần cứng, đặc tả cung cấp thông tin về các thành phần, khả năng và yếu tố kỹ thuật của máy tính. Về mặt phần mềm, đặc tả mô tả môi trường hoạt động và chức năng của chương trình.*

Theo IBM Dictionary of Computing (1994), đặc tả là (1) *một dạng thức văn bản chi tiết cung cấp các mô tả xác định về một hệ thống nhằm để phát triển hay hợp thức hoá.* (2) Trong lĩnh vực phát triển hệ thống, đặc tả là *mô tả cách thiết kế, cách bố trí thiết bị và cách xây dựng chương trình cho hệ thống.*

Như vậy, đặc tả là sự mô tả các đặc trưng nhằm diễn đạt các yêu cầu và các chức năng của một sản phẩm phần mềm cần thiết kế. Đặc tả liên quan đến các đối tượng, các khái niệm và các thủ tục nào đó cần đến khi phát triển chương trình. Đặc tả có các đặc trưng :

- Tính chính xác (Correctness)
- Tính trừu tượng (Abstraction)
- Tính chặt chẽ về mặt Toán học (Rigorousness)

#### I.1.2. Các phương pháp đặc tả

Người ta thường sử dụng 3 phương pháp đặc tả : đặc tả phi (không) hình thức, đặc tả hình thức và đặc tả hỗn hợp.

**Đặc tả phi hình thức** (informal specification) được diễn đạt bằng ngôn ngữ tự nhiên và toán học. Tuy phương pháp đặc tả này không chặt chẽ nhưng dễ hiểu và dễ diễn đạt. Ta thường sử dụng khi cần phát biểu các bài toán, các yêu cầu ban đầu.

**Ví dụ 1 :**

1. Tìm nghiệm của phương trình  $f(x) = 0$  với  $f(x)$  là một đa thức có bậc cho trước sao cho với giá trị thực  $x$  thì  $f(x)$  có giá trị bằng 0.
2. Biến đổi ma trận vuông  $A$  cấp  $n \times n$  về dạng tam giác trên, nghĩa là ma trận  $A$  có các phần tử nằm phía trên đường chéo chính thì bằng 0.

**Đặc tả hình thức** (formal specification) được diễn đạt bằng ngôn ngữ đại số và logic toán, rất chặt chẽ, chính xác và không mập mờ (non-ambiguous).

**Ví dụ 2**

1. Tìm nghiệm của phương trình  $f(x) = 0$  với  $f(x)$  là một đa thức có bậc cho trước sao cho với giá trị thực  $x$  thì  $f(x)$  có giá trị bằng 0.
2. Biến đổi ma trận vuông  $A$  cấp  $n \times n$  về dạng tam giác trên, nghĩa là ma trận  $A$  có các phần tử nằm phía trên đường chéo chính thì bằng 0.

Các tính chất của đặc tả hình thức

- đặc tả mô tả những cái phải làm nhưng không phải mô tả làm như thế nào.
- Lập trình thể hiện tường minh việc lựa chọn cách khai triển : nghiên cứu thuật giải, cách viết công thức...
- Đặc tả cho phép diễn tả đầy đủ một vấn đề, giảm tối thiểu tính phức tạp của hệ thống đang xét.
- Đặc tả phải cho phép kiểm tra được quá trình phát triển phần mềm (chất lượng và tính tin cậy)

Đặc tả hình thức liên quan đến :

- Cấu trúc dữ liệu và các hàm (kiểu dữ liệu)
- Thời gian
- Thao tác
- Đơn thể hay đối tượng.

Tính đại số căn cứ trên việc định nghĩa các kiểu dữ liệu, tính hiệu quả đại số được xác định bởi các công cụ toán học, đại số và logic.

**Đặc tả hỗn hợp** (Mixing Specification) phối hợp giữa hai phương pháp : hình thức và phi hình thức. Thường mô tả phi hình thức nhằm làm giải thích rõ hơn, dễ hiểu hơn một khi mô tả hình thức quá phức tạp.

**1.1.3. Các thí dụ minh họa**

Mô tả các cấu trúc dữ liệu :

Cho ma trận vuông  $A$  cấp  $n \times n$ ,  $n \geq 1$  :

$A = \{a_{ij} \mid i = 1..n, j = 1..n\}$  gồm các phần tử  $a_{ij}$  ở hàng  $i$ , cột  $j$

Bốn đỉnh (góc) của ma trận  $A$  là  $a_{11}$ ,  $a_{1n}$ ,  $a_{nn}$  và  $a_{n1}$

Đường chéo chính là vector  $d1 = \{a_{ii} \mid i = 1..n\}$

Đường chéo phụ là vector  $d2 = \{a_{i, n-i+1} \mid i = 1, n\}$

Phần tử  $a_{ij}$  đối xứng với  $a_{ji}$  qua đường chéo chính  $d1$

Phần tử  $a_{ij}$  đối xứng với  $a_{n-j+1}$  qua đường chéo phụ  $d2$

Ma trận tam giác trên :

$$A_0 = \{ a_{ij} \mid a_{ij} \neq 0, \forall i = 1..n, j = i..n \wedge a_{ij} = 0, \forall i = 2..n, j = 1..i - 1 \}$$

Ma trận tam giác dưới :

$$A^0 = \{ a_{ij} \mid a_{ij} \neq 0, \forall i = 1..n, j = 1..i \wedge a_{ij} = 0, \forall i = j..n - 1, j = 2..n \}$$

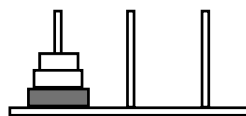
## 1.2. Đặc tả và lập trình

Trong những trường hợp có thể, người ta hướng đặc tả về một ngôn ngữ lập trình nào đó. Ví dụ về đặc tả đệ quy cho bài toán tháp Hà nội (Tower of Hanoi).

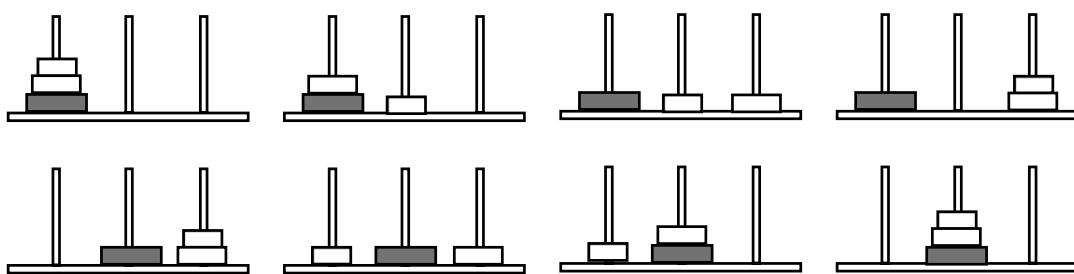
Cho chồng  $n$  đĩa  $n = 64$  xếp thành hình tháp ở cột A (lớn nhất dưới cùng và nhỏ dần lên trên). Hãy chuyển chồng  $n$  đĩa này qua cột B theo nguyên tắc sau :

1. Mỗi lần chỉ di chuyển một đĩa từ cột này qua cột kia
2. Không đặt đĩa to lên đĩa nhỏ
3. Lấy vị trí cột C để đặt tạm các đĩa trung gian

Sau đây là bài toán Tháp Hà nội với  $n = 3$  đĩa.



Hình 5.1. Chồng đĩa trước khi chuyển



Hình 5.2. Chồng đĩa sau khi chuyển (với 7 lần xếp)

**a) Cách giải phi hình thức**

Chuyển  $n - 1$  đĩa từ A qua C lấy B làm cột trung gian, sau đó chuyển đĩa dưới cùng từ A sang B. Tiếp tục chuyển  $n - 1$  đĩa từ C qua B lấy A làm cột trung gian theo cách trên.

**b) Cách giải hình thức bằng đặc tả**

Gọi thủ tục chuyển  $n$  đĩa từ A qua B lấy C làm trung gian ( $n > 0$ ) là :

**Hà\_nội (n, A, B, C)**

và thủ tục chuyển một đĩa từ A qua B là :

**Chuyển\_một\_đĩa (A, B) .**

Khi đó, ta có đặc tả :

```
Hà_nội (n, A, B, C) = if n > 0 then begin
                        Hà_nội (n - 1, A, C, B);
                        Chuyển_một_đĩa (A, B);
                        Hà_nội (n - 1, C, B, A)
                    End
```

ta dễ dàng viết các thao tác trên thành thủ tục như sau :

```
Procedure ChuyểnCột(n, A, B, C: TênCột);
Begin
    if n>0 then begin
        ChuyểnCột(n-1, A, C, B);
        Chuyển_một_đĩa_từ_A_sang_C;
        ChuyểnCột(n-1, B, A, C);
    End
End;
```

Thao tác *Chuyển\_một\_đĩa\_từ\_A\_sang\_C*; được viết thành lệnh :

```
Writeln('Chuyển một đĩa từ ', A:1, ' -> ', C:1);
```

Thêm biến đếm  $i$  để tính số bước chuyển đĩa, chương trình đầy đủ như sau :

```
Program HanoiTower;
Type TênCột = 1 .. 3;
Var i, N: Integer;
Procedure ChuyểnCột(n, A, B, C: TênCột);
Begin
    if n>0 then begin
        ChuyểnCột(n-1, A, C, B);
        i:=i+1;
        Writeln(i:3, 'Chuyển một đĩa từ ', A:1, ' -> ', C:1);
        ChuyểnCột(n-1, B, A, C);
    End
End;
Begin
    Write('Số đĩa cần chuyển : ');
```

```

Readln(N) ;
i:=0;
ChuyểnCột;
Readln
End.

```

Chạy chương trình trên sẽ cho kết quả như sau :

```

Số đĩa cần chuyển : 4
1.Chuyển một đĩa từ 1 -> 2
2.Chuyển một đĩa từ 1 -> 3
3.Chuyển một đĩa từ 2 -> 3
4.Chuyển một đĩa từ 1 -> 2
5.Chuyển một đĩa từ 3 -> 1
6.Chuyển một đĩa từ 3 -> 2
7.Chuyển một đĩa từ 1 -> 2
8.Chuyển một đĩa từ 1 -> 3
9.Chuyển một đĩa từ 2 -> 3
10.Chuyển một đĩa từ 2 -> 1
11.Chuyển một đĩa từ 3 -> 1
12.Chuyển một đĩa từ 2 -> 3
13.Chuyển một đĩa từ 1 -> 2
14.Chuyển một đĩa từ 1 -> 3
15.Chuyển một đĩa từ 2 -> 3

```

Trong trường hợp tổng quát n đĩa, số bước chuyển đĩa sẽ là :

$$2^0 + 2^1 + \dots + 2^n = 2^n - 1 \text{ lần.}$$

Với n=64, giả sử thời gian để chuyển một đĩa là t giây, thì thời gian để chuyển hết 64 đĩa của bài toán Tháp Hà nội sẽ là :

$$(2^{64} - 1) \times t = 1.8446744074E+19 \times t \text{ giây.}$$

Một năm có  $365 \times 24 \times 60 \times 60 = 31\,536\,000$  giây, giả sử  $t = 10^{-2}$  giây thì số năm cần để chuyển 64 đĩa là :

$$(1.8446744074E+19 / 31536000) \times 10^{-2} = 5.8494241735E+11 \approx 5.8 \text{ tỷ năm !}$$

**Bài tập :** 1, 2, 3, 4, 5 trang 140-141 (Nguyễn Xuân Huy).

## II. Đặc tả cấu trúc dữ liệu

### II.1. Cấu trúc dữ liệu cơ sở vectơ

#### II.1.1. Dẫn nhập

Cho một cuốn từ điển. Cần tra cứu một từ ở một trang nào đó bất kỳ :

Duyệt lần lượt các từ, từ đầu từ điển, cho đến khi gặp từ cần tra cứu, gọi là tìm tuần tự (giống tệp tuần tự)

Nếu từ điển đã được sắp xếp ABC, có thể tìm ngẫu nhiên một từ, sau đó tùy theo từ đã gặp mà tìm phía trước hay phía sau từ đó từ cần tra cứu.

Có thể xem từ điển là một vectơ cho phép tìm kiếm ngẫu nhiên một từ.

Trong tin học, bộ nhớ máy tính cũng xem là một vectơ gồm các ô nhớ lưu trữ dữ liệu

### II.1.2. Đặc tả hình thức

Cho một tập giá trị E và một số nguyên  $n \in \mathbb{N}$ .

Một vectơ là một ánh xạ V từ khoảng  $I \subset \mathbb{N}$  vào E.

$V : I \rightarrow E, I = [1..n]$ , n là số phần tử của V, hay kích thước.

V có thể rỗng nếu  $n = 0$

Ký hiệu vectơ bởi  $(V[1..n], E)$  hoặc  $E : V[1..n]$ , hoặc V nếu không có sự hiểu nhầm.

Một phần tử của vectơ là cặp  $(i, V[i])$  với  $i \in [1..n]$ , để đơn giản ta viết  $V[i]$ .

Một vectơ có thể được biểu diễn bởi tập các phần tử của nó :

$(V[1], V[2], \dots, V[n])$  hay  $(x_1, x_2, \dots, x_n)$  nếu  $x_i = V[i]$ , là các giá trị (trực tiếp)

Vectơ con : Ta gọi thu hẹp của V trên một khoảng liên tiếp của  $[1..n]$  là vectơ con của  $V[1..n] : V[i..j], j > i$ , rỗng nếu  $i > j$

Ví dụ :  $V[1..5] = (7, 21, -33, 6, 8)$

Các vectơ con :  $V[2..4] = (21, -33, 6)$

$V[1..3] = (7, 21, -33)$  v.v...

## II.2. Truy nhập một phần tử của vectơ

Cho  $V[1..n]$ . Với  $\forall i \in [1..n]$ , phép truy nhập  $V[i]$  sẽ cho giá trị phần tử có chỉ số i của V. Kết quả không xác định nếu  $i \notin [1..n]$

Ví dụ :  $V[1..5] = (7, 21, -33, 6, 8)$

$V[2] = 21, V[4] = 6$  nhưng  $V[0], V[7]...$  không xác định.

Vectơ được sắp xếp thứ tự (SXTT)

Ta nói :

- Vectơ rỗng ( $n = 0$ ) là vectơ được SXTT.
- Vectơ chỉ gồm 1 phần tử ( $n = 1$ ) là vectơ được SXTT.
- Vectơ  $V[1..n], n > 1$  là vectơ được SXTT nếu

$\forall i \in [1..n - 1], \forall [i] \leq \forall [i + 1]$

Có thể định nghĩa đệ qui 3 :

$V[1..i]$  được SXTT,  $V[i] \leq V[i + 1] \Rightarrow V[1..i + 1]$  được SXTT, với  $i \in [1..n - 1]$

Một số ký hiệu khác :

$a \in V[1..n] \Leftrightarrow \exists j \in [1..n], a = V[j]$

$a \notin V[1..n] \Leftrightarrow \forall j \in [1..n], a \neq V[j]$

$a < V[1..n] \Leftrightarrow \forall j \in [1..n], a < V[j]$

Ta cũng có cho các phép so sánh  $\leq, >, \geq, =$  và  $\neq$ .

Để xét các thuật toán xử lý vectơ, ta sử dụng mô tả dữ liệu :

Const  $n = 100$  ;

Type

Vectơ = array [ 1..n ] of T ;

{T là kiểu của các phần tử của vectơ}

### II.3. Các thuật toán xử lý vectơ

Duyệt vectơ

Cho  $V[1..n]$ , thuật toán duyệt vectơ được viết đệ quy như sau :

```

Procedure scan (V: vectơ; i, n: integer);
  Begin
    if i <= n then begin
      Operation (V[i]);
      Scan (V, i + 1, n) {i := i + 1; nếu bỏ đệ qui}
    end
  end;

```

#### II.3.1. Truy tìm tuần tự một phần tử của vectơ (sequential search)

##### a) Vectơ không được sắp xếp thứ tự

Lập luận giả sử đã xử lý  $i - 1$  ( $1 < i \leq n + 1$ ) phần tử đầu của  $V$  và khẳng định rằng phần tử  $\notin V[1..i - 1]$

Xảy ra hai trường hợp :

$i = n + 1$  : phần tử  $\notin V[1..n]$ , kết thúc, phần tử  $\notin V$

$i \leq n$  : lại xảy ra hai trường hợp :

$V[i] =$  phần tử : phần tử  $\notin V[1..i]$ , kết thúc, phần tử  $\notin V$

$V[i] \neq$  phần tử : phần tử  $\notin V[1..i]$ , tiếp tục  $i := i + 1$

và cho phép khẳng định lại phần tử  $\notin V[1..i - 1]$

Ta viết thuật toán không đệ qui như sau :

```

function check(V: Vectơ; n: integer; phần tử: T): Boolean;
{ (n > 0)  $\Rightarrow$  (check, phần tử  $\notin V$ ) }  $\vee$  (not check, phần tử  $\notin V$ )
Var i: integer;

```



```

begin
  i := 1; {phần tử  $\notin \forall [1..i - 1], i \leq n$ }
  while ( $\forall [i] <> \text{phần tử}$ ) and ( $i < n$ ) do
    {phần tử  $\notin \forall [1..i], i < n$ }
    i := i + 1; {phần tử  $\notin \forall [1..i - 1], i \leq n$ }
  {(( $\forall [i] = \text{phần tử}$ )  $\vee$  ( $i = n$ ), phần tử  $\notin \forall [1..i - 1],$ 
   $i \leq n$ )  $\Rightarrow$  ( $\forall [i] = \text{phần tử}, \text{phần tử} \notin V$ )
   $\vee$  ( $\forall [i] \neq \text{phần tử}, \text{phần tử} \notin V$ )}
```

Ta có thể viết lại thuật toán dưới dạng đệ quy như sau :

```

function check(V:Vectơ, i,n:integer; phần tử: T): Boolean;
{n  $\geq 0 \Rightarrow$  check, phần tử  $\in \forall [i..n]$  )
 $\vee$  (Not check, phần tử  $\notin \forall [i..n]$  )}
begin
  if i > n then check := false
  else if  $\forall [i] = \text{phần tử}$  then check := true
  else check := check (V, i + 1, n, phần tử)
end;
```

Khi gọi hàm, i có thể nhận giá trị bất kỳ, từ 1..n, đặc biệt i = 1

Trường hợp duyệt vectơ từ phải qua trái, ta không cần dùng biến i nữa :

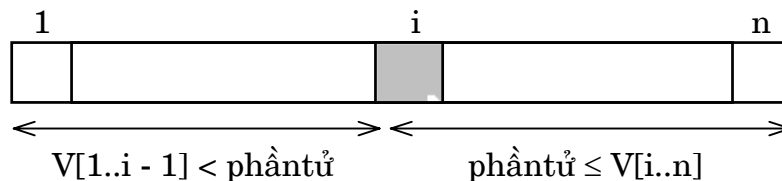
```

function check (V:Vectơ; n: integer; phần tử: T): Boolean;
{n  $\geq 0 \Rightarrow$  (check, phần tử  $\in V$ )  $\vee$  (Not check, phần tử  $\notin V$ )}
```

### b) Vectơ được sắp xếp thứ tự

Ta cần tìm chỉ số  $i \in [1..n]$  sao cho thỏa mãn :

$V[1..i - 1] < \text{phần tử} \leq V[i..n]$



Hình 5.3. Vectơ được sắp xếp thứ tự

Vấn đề là kiểm tra đẳng thức phần tử =  $V[i]$  không trong  $V$  đã được sắp xếp ?

Lập luận :

Giả sử đã xử lý  $i - 1$  ( $1 \leq i \leq n + 1$ ) phân tử đầu của  $V$  và  $V[1..i - 1] <$  phân tử đã được khẳng định : xảy ra hai trường hợp :

$i = n + 1$  : kết thúc  $V[1..n] <$  phân tử, phân tử  $\notin V$

$i \leq n$  : lại có hai trường hợp mới :

$V[i] \geq$  phân tử : kết thúc, đã tìm được  $i$  sao cho

$V[i..i - 1] <$  phân tử  $\leq V[i..n]$

chỉ còn phải kiểm tra phân tử =  $V[i]$  ?

$V[i] <$  phân tử : có nghĩa  $V[1..i] <$  phân tử, tiếp tục thực hiện :  $i := i + 1$  để có lại khẳng định  $V[1..i - 1] <$  phân tử

Ta có thuật toán như sau :

```
function checknum(V:vectơ; n:nguyên; phântử:T): Boolean;
{ V được SXTT, n > 0  $\Rightarrow$  (checknum, phântử  $\in V$ )  $\vee$ 
  (Not checknum, phântử  $\notin V$ ) }
Var i: integer ;
begin
if phântử > V[n] then
  checknum := false { not checknum, phântử  $\notin V$  }
else begin { phântử  $\leq V[n]$  }
  i := 1 ; { V[1..i - 1] < phântử }
  while (V[i] < phântử) do { V[1..i] < phântử }
    i := i + 1 ; { V[1..i] < phântử }
    { V[1..i - 1] < phântử, V[i]  $\geq$  phântử }
    checknum := (V[i] = phântử)
  { (checknum, phântử  $\in V$ )  $\vee$  ( $\neg$ checknum, phântử  $\notin V$ ) }
end { (checknum, phântử  $\in V$ )  $\vee$  ( $\neg$ checknum, phântử  $\notin V$ ) }
end;
```

### II.3.2. Tìm kiếm nhị phân (Binary search)

#### a) Phương án 1

Giả sử vectơ  $V[1..n]$  ( $n > 1$ ) đã được sắp xếp thứ tự :

$\forall i \in [1..n - 1], V[i] \leq V[i + 1]$

Ta chia  $V$  thành 3 vectơ con  $V[1..m - 1]$ ,  $V[m..m]$  và  $V[m + 1..n]$  được sắp xếp thứ tự sao cho :

$V[1..m - 1] \leq V[m] \leq V[m + 1..n]$

Xảy ra 3 trường hợp :

$\in V[1..m - 1]$  nếu phân tử  $< V[m]$

phân tử =  $V[m]$

$\in V[m + 1..n]$  nếu phân tử  $> V[m]$

lúc này ta trở lại bài toán đã xét : tìm phân tử trong vectơ  $V[1..m - 1]$  hoặc  $V[m + 1..n]$ . Kết thúc nếu phân tử =  $V[m]$

Một cách tổng quát, lần lượt xác định dãy các vectơ có  $V_1, V_2, \dots, V_k$  sao cho mỗi  $V_i$  có kích thước nhỏ hơn kích thước của vectơ con trước đó  $V_{i-1}$

Đề ý rằng nếu chọn  $V_1 = V[1..n], V_2 = V[2..n], \dots, V_k = V[k..n]$ , ta đi đến phép tìm kiếm tuần tự đã xét ở trên.

Ta sẽ chọn  $m$  là vị trí giữa (nếu  $n$  lẻ) để cho  $V[1..m - 1]$  và  $V[m + 1..n]$  có kích thước bằng nhau, hoặc chọn  $m$  sao cho chúng hơn kém nhau một phân tử.

Khi đó kích thước của các vectơ thuộc dãy  $V_1, V_2, \dots, V_k$  sẽ lần lượt được chia đôi tại mỗi bước :  $n, n/2, \dots, n/2^{k-1}$ .

Như vậy, sẽ có tối đa  $\lceil \log_2 n \rceil$  vectơ con khác rỗng.

Ví dụ : nếu  $n = 9000$ , số vectơ con khác rỗng tối đa sẽ là 13, vì  $2^{13} = 8192$

Xây dựng thuật toán :

Sau một số bước, ta có vectơ con  $V[\text{inf}.. \text{sup}]$  sao cho :

$V[1..\text{inf} - 1] < \text{phân tử} < V[\text{sup} + 1..n]$

Xảy ra hai trường hợp :

- $\text{inf} > \text{sup}$  ( $\text{inf} = \text{sup} + 1$ )

$(V[1..\text{inf}-1] < \text{phân tử} < V[\text{sup}+1..n], \text{inf} = \text{sup}+1) \Rightarrow (\text{phân tử} \notin V, \text{kết thúc})$

- $\text{inf} \leq \text{sup}$  :  $m = (\text{inf} + \text{sup}) \text{ div } 2$

khi đó ta có  $V[\text{inf}..m - 1] \leq V[m] \leq V[m + 1..\text{sup}]$

Tồn tại 3 khả năng như sau :

- Phân tử =  $V[m]$  : kết thúc, phân tử  $\in V$
- Phân tử  $< V[m]$  : tiếp tục tìm kiếm trong  $V[\text{inf}..m - 1]$   
lấy  $\text{sup} := m - 1$  để có lại khẳng định phân tử  $< V[\text{sup} + 1..m]$
- Phân tử  $> V[m]$  : tiếp tục tìm kiếm trong  $V[m + 1..\text{sup}]$   
lấy  $\text{inf} := m + 1$  để có lại khẳng định  $V[1..\text{inf} - 1] < \text{phân tử}$

Như vậy cả hai trường hợp :  $V[1..\text{inf} - 1] < \text{phân tử} < V[\text{sup} + 1..n]$

Khởi đầu, lấy  $\text{inf} := 1$  và  $\text{sup} := n$

Ta có thuật toán như sau :

```
function binary (V:vectơ; n:integer; phân tử:T): Boolean;
{V được SXTT  $\Rightarrow$  (binary, phân tử $\in V$ )  $\vee$  (not binary, phân tử $\notin V$ )}
Var inf, sup, m : integer ;
OK : Boolean ;
begin
  OK : false ; {not OK, phân tử  $\notin V$ }
```

```

inf := 1 ; sup := n ;
{  $\forall [1..inf - 1] < \text{phântử} < \forall [sup+1..n]$  }
while (inf ≤ sup) and (not OK) do begin
  m := (inf + sup) div 2 ;
  if  $\forall [m] = \text{phântử}$  then OK := true {OK,  $\text{phântử} \in V$ }
  else {not OK}
    if  $\forall [m] < \text{phântử}$  then inf := m+1 {  $\forall [1..inf-1] < \text{phântử}$  }
    else sup := m - 1 ; {  $\forall [sup + 1..n] > \text{phântử}$  }
    { ( $\forall [1..inf - 1] < \text{phântử} < \forall [sup + 1..n]$  , not OK)
      ∨ (tìm thấy,  $\text{phântử} \in V$ ) }
  end;
{ (inf = sup + 1) ∨ (tìm thấy),
  (¬ tìm thấy,  $\forall [1..inf - 1] < \text{phântử} < \forall [sup + 1..n]$ ) ∨
  (tìm thấy,  $\text{phântử} \in V$ ) ⇒
  (¬ tìm thấy,  $\forall [1..inf - 1] < \text{phântử} < \forall [inf..n]$ ) ∨
  (tìm thấy,  $\text{phântử} \in V$ ) ⇒
  (¬ tìm thấy,  $\text{phântử} \notin V$ ) ∨ (tìm thấy,  $\text{phântử} \in V$ ) }
Nhị phân := OK
end ;

```

**Viết chương trình trên dưới dạng đệ quy :**

```

function NhịPhân(V:Vectơ; inf, sup:integer; phântử:T):boolean;
{ (V được SXTT ⇒ (NhịPhân,  $\text{phântử} \in V$ ) ∨ ¬ NhịPhân,  $\text{phântử} \notin V$ ) }
Var m : integer ;
begin
  if inf > sup then NhịPhân := false
  else begin
    m := (inf + sup) div 2 ;
    if  $\forall [m] = \text{phântử}$  then NhịPhân := true
    else if  $\forall [m] < \text{phântử}$  then
      NhịPhân := NhịPhân(V, m+1, sup, phântử)
    else NhịPhân := NhịPhân(V, inf, m - 1, phântử)
  end
end ;

```

Hàm này có thể được gọi với các giá trị inf, sup bất kỳ, thông thường được gọi bởi dòng lệnh :

NhịPhân (V, 1, n, phântử)

## **b) Phương án 2**

Có thể tìm ra những phương án khác cho thuật toán tìm kiếm nhị phân. Chẳng hạn, thay vì kiểm tra đẳng thức  $\forall [m] = \text{phântử}$ , ta kiểm tra khẳng định :

$$\forall [1..inf - 1] < \text{phântử} \leq \forall [inf..n]$$

Ssau đó kiểm tra  $\text{phântử} = \forall [inf]$  để có câu trả lời.

Mặt khác, có thể thay đổi giá trị trả về của hàm tìm kiếm nhị phân bởi vị trí của phân tử trong vectơ, bằng 0 nếu phân tử  $\notin V$ .

Nếu  $\text{inf} = 1$ , khẳng định có dạng  $\forall [1..n] < \text{phân tử} \leq V[\text{sup}..n]$  và được viết gọn  $\text{phân tử} \leq V[1..n]$ .

```
function NhịPhân(V:vectơ;n:integer;phân tử: T): integer ;
{ (V được SXTT, n > 0)  $\Rightarrow$  (m  $\in$  [ 1..n
NhịPhân = m,  $\forall [m] = \text{phân tử}$ )  $\vee$  (NhịPhân = 0, phân tử  $\notin V$ )}
Var m, inf, sup : integer ;
begin
  if phân tử > V[n] then nhị phân := 0
  else begin
    inf := 1 ; sup := n ;
    {  $\forall [1..inf - 1] < \text{phân tử} \leq V[\text{sup}..n]$  }
    while inf < sup do begin
      m := (inf + sup) div 2 ;
      if phân tử  $\leq V[m]$  do sup := m {phân tử  $\leq V[\text{sup}..n]$ }
      else inf := m + 1 {  $\forall [1..inf - 1] < \text{phân tử}$  }
      {  $\forall [1..inf - 1] < \text{phân tử} \leq V[\text{sup}..n]$  }
    end ;
    { (inf = sup,  $\forall [1..inf - 1] < \text{phân tử} \leq V[\text{inf}..n]$ 
 $\Rightarrow$  ( $\forall [1..inf - 1] < \text{phân tử} \leq V[\text{inf}..n]$ ) }
    if phân tử = V[inf] then NhịPhân:= inf
    else NhịPhân:= 0
  end
end ;
```

### III. Đặc tả đại số : mô hình hóa phát triển phần mềm

(Phần này chỉ phục vụ tham khảo)

#### III.1. Mở đầu

Đặc tả đại số không mô tả các yếu tố liên quan đến thời gian thực thi cũng như trạng thái.

Ngôn ngữ đặc tả trạng thái liên quan đến :

- Ngữ nghĩa (Semantic)
- Cú pháp (syntax)
- Các thuộc tính (Properties)

Hình vẽ

Ngữ nghĩa của các đặc tả đại số liên quan đến :

- Dấu kí (signature) của một kiểu đại số trừu tượng
- Hạng (term) với các biến
- Phương trình và các tiên đề
- Các mô hình đặc biệt ...

Cú pháp của đặc tả đại số

Ví dụ :

Xây dựng kiểu string cho các xâu ký tự cùng các phép toán thông dụng trên xâu như sau :

- Tạo xâu rỗng mới (phép toán new)
- Ghép xâu (append)
- Thêm một ký tự vào xâu (add to)
- Lấy độ dài xâu
- Kiểm tra xâu rỗng (is empty)
- Kiểm tra hai xâu bằng nhau không (=)
- Trích ký tự đầu tiên của xâu (frist)

Đề định nghĩa kiểu string, người ta còn sử dụng các kiểu sau :

- char : kiểu của ký tự
- nat : kiểu của số nguyên
- bool : kiểu giá trị logic

Tên các tập hợp và các phép toán trên tập hợp xác định một ký dấu (signature).  
Như vậy một dấu kí được xây dựng từ :

- Tên các kiểu đặc tả

- Tên các phép toán với việc chỉ rõ miền xác định (domain) và miền trị (range)  
như sau :

tên phép toán : miền xác định  $\rightarrow$  miền trị

Ta xây dựng dấu kí từ kiểu string như sau

```
Adt String ;
Use char, Not, Bool ;
Sorts string ;
Operations
new :  $\rightarrow$  string ;
append _ _ : String, string  $\rightarrow$  string ;
add _ to _ : char, string  $\rightarrow$  string ;
# _ : String  $\rightarrow$  not ;
is empty ? _ string  $\rightarrow$  bool ;
_ = _ : string, string  $\rightarrow$  bool ;
frist _ : string  $\rightarrow$  char ;
```

Tên xuất hiện trong một dấu kí gồm hai loại là có ích (interest) và bổ trợ (auxiliary) tùy theo vai trò của chúng. Ví dụ :

- String là có ích

- Char, not và bool là bổ trợ

Cú pháp (cp)

Cp đặc tả đại số sử dụng trong ví dụ trên được chia ra thành các khối : đầu, giao tiếp và thân của đặc tả. Mỗi khối gồm một số khai báo ngăn cách nhau bởi các từ khóa (có gạch chân)

Đối với khối giao tiếp (interface), người ta sử dụng các khái niệm tiền tố (prefix), trung tố (infix) và hậu tố (postfix) như sau :

Tiền tố : tên của phép toán được đặt trước dãy các tham biến

Ví dụ : appenend \_ \_ : string, string  $\rightarrow$  string ;

Từ đó người ta có thể viết các hạng dưới dạng :

append x y hay

append (x y) hay

(append x y)

Trung tố : cho phép định nghĩa toán tử hay vị từ

Ví dụ \_ = \_ : string, string  $\rightarrow$  bool ;

từ đó có thể viết các hạng dưới dạng :

$x = y$  hay  $(x = y)$

Hỗn hợp : cho phép viết các biểu thức bất kỳ một cách mềm dẻo như `add_to_` :  
`char, string → string`

từ đó có thể viết các hạng dưới dạng :

`add c to append (x y)`

Trong nhiều trường hợp trên đây, các cặp dấu ngoặc dấu được dùng để phân cách các hạng với nhau

### III.2. Phân loại các phép toán

Các phép toán được chia ra thành 2 loại :

Loại quan sát được (operations)

Loại phát sinh (generator operations)

Loại quan sát được có các dạng sau :

Kiểu có ích [và kiểu hỗ trợ] → Kiểu hỗ trợ

Ví dụ : `_ = _ : string, string → bool;`

`# _ : string → not ;`

`is empty ? : string → bool ;`

`first _ : string → char ;`

Loại phát sinh có dạng :

Kiểu có ích [và kiểu hỗ trợ] → Kiểu có ích

Ví dụ :

`new : _ → string ;`

`add_to _ : char, string → string ;`

Ở đây, phép toán `new` tạo ra một xâu rỗng, còn phép toán `add _ to _` thêm một ký tự vào xâu.

Các tiên đề được xây dựng từ các phép toán dùng cho các kiểu hỗ trợ giả sử được định nghĩa như sau :

`true : → bool ;`

`false : → bool ;`

`not _ : bool → bool ;`

`_ and _ : bool, bool → bool ;`

`_ or _ : bool ; bool → bool ;`

`0 : → not ;`



```

1 : → not ;
succ : not → not ;
_ + _ : not, not → not ;
_ - _ : not, not → not ;
_ * _ : not, not → not ;
_ / _ : not, not → not ;
_ = _ : not, not → bool;
a : → char ;
b : → char ;
...
_ = _ : char, char → bool ;

```

### III.3. Hạng và biến

Trong đặc tả đại số, các biến được định kiểu và có thể nhận giá trị tùy ý tùy theo kiểu đã định nghĩa. Ví dụ : khai báo kiểu  $x : \text{string}$  ;  $y : \text{string}$  ;  $c : \text{char}$  ; định nghĩa các biến  $x, y, c$  để sử dụng trong các hạng sau đây :

```

add c to x = append (x y)
append (is empty ? (new), add x to x)

```

Hạng là một biểu thức nhận được từ việc tổ hợp liên tiếp các phép toán của signature (dấu kí). Một hạng là hợp thức nếu hạng đó thỏa mãn các phép toán đã sử dụng (kiểu và vị trí). Qui tắc quy nạp được dùng để xây dựng tập hợp các hạng + có kiểu  $s$  được viết  $t : s$  được định nghĩa như sau :

$$\frac{+ : s_1, s_2, \dots, s_n \rightarrow s \wedge t_1 : s_1, t_2 : s_2, \dots, t_n : s_n}{(f t_1 t_2 \dots t_n) : s}$$

$(f t_1 t_2 \dots t_n) : s$

trong đó sử dụng quy tắc khai báo kiểu biến

$x : s$

Từ đó, hạng hợp thức trong hai hạng từ ví dụ vừa xét là

```

add c to x = append (x y)

```

### III.4. Phép thế các hạng

Phép thế (substitutions) là một phép toán trên các hạng cho phép thay thế các biến (có mặt) trong các hạng bởi các hạng khác. Tập hợp các biến FV xuất hiện trong một hạng được định nghĩa một cách đệ quy như sau :

$$FV(ft_1 t_2 \dots t_n) = FV(t_1) \cup FV(t_2) \cup \dots \cup FV(t_n)$$

$$FV(x) = \{x\}$$

$$\text{Ví dụ : } FV(\text{append}(\text{is empty?}(\text{new}), \text{add c to x})) = \{x, c\}$$

Phép thế trong một hạng t cho các thành phần chứa biến x bởi hạng u, ký hiệu  $t[u/x]$ , được định nghĩa như sau :

Với  $x \in FV(t)$  thì

$$(f t_1 t_2 \dots t_n)[u/x] = (f t_1[u/x] t_2[u/x] \dots t_n[u/x])$$

$$y[u/x] = u \quad y = x$$

$$= y \quad y \neq x$$

$$\text{Ví dụ : } \text{append}(\text{is empty?}(\text{new}), (\text{add c to x}))[(\text{add c to y})/x]$$

$$= \text{append}(\text{is empty?}(\text{new}), (\text{new}), (\text{add c to}(\text{add c' to y})))$$

Mô tả các thuộc tính qua các phương trình

Các tiên đề sử dụng trong đặc tả được xây dựng theo logic vị trí bậc 1 dạng phương trình (pt)

Một phương trình hợp thức có vế trái và vế phải cùng kiểu hạng :

$$AX \text{ spec} = \{t = t' \mid t : s \wedge t' : s\}$$

Trong ví dụ về đa kiểu string, phép toán `is empty?` được định nghĩa theo phương trình :

$$\text{is empty?}(\text{new}) = \text{true};$$

Có nghĩa một chuỗi vừa mới tạo ra là rỗng - sau đó, việc thêm một ký tự mới vào chuỗi sẽ cho kết quả là false :

$$\text{is empty?}(\text{add c to x}) = \text{false};$$

Tính đệ quy của phương trình :

$$\text{append}(x, \text{add c to y}) = \text{add c to}(\text{append}(x, y));$$

chỉ ra rằng việc ghép một chuỗi với chuỗi được tạo ra bằng cách thêm một ký tự vào chuỗi này thì cũng có giá trị như ghép hai chuỗi trước rồi sau đó thêm một ký tự vào chuỗi kết quả. Điều đó hợp lý vì ta có tính chất của phương trình :  $\text{append}(x, \text{new}) = x$  ;

nghĩa là ghép một chuỗi nào đó với chuỗi rỗng cũng cho ra kết quả chính chuỗi đó

Ta có các tiên đề về chuỗi ký tự như sau :

Axioms

$$\text{is empty?}(\text{new}) = \text{true};$$

$$\text{is empty?}(\text{add c to x}) = \text{false};$$

$$\# \text{ new} = 0;$$

$$\# (\text{add c to x}) = x(x) = + 1;$$

$\text{append}(x, \text{new}) = x$  ;  
 $\text{append}(x, \text{add } c \text{ to } y) = \text{add } c \text{ to } \text{append}(x \ y)$  ;  
 $(\text{new} = \text{new}) = \text{true}$  ;  
 $\text{add } c \text{ to } x = \text{true}$  ;  
 $(\text{add } c \text{ to } x = \text{new}) = \text{false}$  ;  
 $(\text{new} = \text{add } c \text{ to } x) = \text{false}$  ;  
 $(\text{add } c \text{ to } = \text{add } d \text{ to } y) = (c = d) \text{ and } (x = y)$  ;

where ...

... where

$x, y$  : string ;

$c, d$  : char ;

end string ;

Các tiên đề điều kiện

Các tiên đề điều kiện tích cực (positive conditional axioms) là mở rộng của các phương trình, chúng là các mệnh đề Horn về tính bằng nhau, có dạng :

$$t_1 = t_1' \wedge t_2 = t_2' \wedge \dots \wedge t_n = t_n' \Rightarrow t = t'$$

Ví dụ :  $\text{is empty?}(x) = \text{false} \Rightarrow \text{first}(\text{add } c \text{ to } x) = \text{first}(x)$  ;

$\text{is empty?}(x) = \text{true} \Rightarrow \text{first}(\text{add } c \text{ to } x) = c$  ;

### III.5. Các thuộc tính của đặc tả

Đặc tả đặt ra hai vấn đề sau đây :

- Hợp thức hóa
- Luồng năng chứng bác (completude) của đặc tả

#### III.5.1. Mô hình lập trình (triển khai)

Các mô hình lập trình mô tả cách thức triển khai của đặc tả. Có nghĩa các chương trình trừu tượng sẽ kiểm chứng các thuộc tính đã trình bày trong đặc tả (thiết lập). Tập hợp các mô hình đặc tả với các phép toán kèm theo được ký hiệu Mod (spec).

Khái niệm lập trình dẫn đến quan hệ thỏa mãn ký hiệu  $\mathcal{U}$  xác định tính triển khai đúng đắn của đặc tả. Ta có :

$$M \in \text{Mod}(\text{spec}) \Leftrightarrow (\forall t, t' : s \text{ và } t = t' \in \text{Ax spec ta có } M \mathcal{U} t = t')$$

Với mọi tiên đề :  $t = t'$  của Ax spec

$$\text{Mod}(\text{spec}) \mathcal{U} t = t' \Leftrightarrow \forall M \in \text{Mod}(\text{spec}),$$

$M \cup t = t'$

### III.5.2. Mô hình đặc biệt

Những mô hình chấp nhận được bởi một đặc tả rất phong phú. Sau đây là một ví dụ về mô hình cho đặc tả kiểu Bool :

Hình vẽ

trong hai mô hình A và B ở trên, đặc tả kiểu Bool thỏa mãn với các quy ước có giá trị là các dấu x, các phép toán biểu diễn bởi các quan hệ giữa miền xác định và miền trị.

Chú ý rằng A chứa các giá trị vô ích tương tự như việc sử dụng 1 byte cho kiểu Bool, còn B chứa vừa đủ (tối thiểu) giá trị cần thiết tương tự như sử dụng 1 bit cho kiểu Bool.

### III.5.3. Mô hình đồng dư

Mô hình này là một thương đại số các hạng đồng dư định nghĩa bởi các quy tắc sau đây :

-  $t = t'$  là tiên đề khi đó  $t \sim t'$

- Phản xạ :  $t \sim t$

- Đối xứng :  $t \sim t' \Rightarrow t' \sim t$

- Bắc cầu :  $t = t' \wedge t' \sim t'' \Rightarrow t \sim t''$

- Khả thế : (cấu thành - substitutivite)

$t_1 \sim t_1' \wedge t_2 \sim t_2' \wedge \dots \wedge t_n \sim t_n' \Rightarrow (ft_1 t_2 \dots t_n) \sim (ft_1' t_2' \dots t_n')$

- Thay thế : cho x là biến, u là hạng  $t \sim t' \Rightarrow t [u/x] \sim t' [u/x]$

Quá trình khai triển một đặc tả

Khai triển một đặc tả là vấn đề khó khăn. Những định nghĩa về cú pháp các chức năng mong đợi không là khó khăn nhưng tính đúng đắn của chúng lại không kiểm chứng được dễ dàng.

## III.6. Phép chứng minh trong đặc tả đại số

Mục đích phần này là chỉ ra cách chứng minh (chứng minh) các thuộc tính trong các đặc tả đại số. Một thuộc tính cần chứng minh có dạng một định lý, chẳng hạn dạng một phương trình.

Giả sử ta cần chứng minh thuộc tính sau đây trong đặc tả các số nguyên tự nhiên dương Not :

$\text{succ}(0) = \text{succ}(\text{succ}(0)) = \text{succ}(\text{succ}(0))$

?  $\text{succ}(0) + \text{succ}(0) = \text{succ}(\text{succ}(\text{succ}(0)))$

Tiên đề :  $\text{succ}(x) + y = \text{succ}(x + y)$

Quy tắc thay thế với  $s = \{x = 0, y = \text{succ}(\text{succ}(0))\}$

$\text{succ}(0) + \text{succ}(\text{succ}(0)) = \text{succ}(0 + \text{succ}(\text{succ}(0)))$

Tiên đề :  $0 + x = x$  và quy tắc thay thế với  $s = \{x = \text{succ}(\text{succ}(0))\}$

$0 + \text{succ}(\text{succ}(0)) = \text{succ}(\text{succ}(0))$

Quy tắc thay thế với phép succ trên (2)

$\text{succ}(0 + \text{succ}(\text{succ}(0))) = \text{succ}(\text{succ}(0))$

Quy tắc bắc cầu cho (1) và (3)

5.

Định lý đã được chứng minh. Cần chú ý rằng thuộc tính này là hợp thức cho mọi quá trình đặc tả số tự nhiên Not.

### III.6.1. Lý thuyết tương đương

Lý thuyết tương đương (của một đặc tả) được xây dựng từ các tiên đề của đặc tả, là tập hợp các định lý hợp thức qua các quy tắc sau đây :

- Phản xạ :  $t = t$

- Đối xứng :  $t = t' \Rightarrow t' = t$

- Bắc cầu :  $t = t' \wedge t' = t'' \Rightarrow t = t''$

- Khả thế :  $t = t' \wedge t_2 = t_2' \wedge \dots \wedge t_n = t_n' \Rightarrow$

$(ft_1, t_2, \dots, t_n) = (ft_1', t_2', \dots, t_n')$

- Phép thế : cho  $x$  là biến và  $u$  là hạng

$t_1 = t_1' \wedge t_2 = t_2' \wedge \dots \wedge t_n = t_n' \Rightarrow t = t'$

khi đó  $t_1 [u/x] = t_1' [u/x] \wedge \dots \wedge t_n [u/x] = t_n' [u/x]$

$\Rightarrow t [u/x] = t' [u/x]$

- Phép cắt :  $\text{Cond}_1 \wedge (u = u') \wedge \text{cond}_2 \Rightarrow t = t'$

và  $\text{cond} \Rightarrow x = x'$ , khi đó :

$\text{cond}_1 \wedge \text{cond} \wedge \text{cond}_2 \Rightarrow t = t'$

Các qui tắc của lý thuyết tương đương thể hiện các thuộc tính bằng nhau (phản xạ, đối xứng và bắc cầu), thuộc tính hàm (khả thế), các biến (phép thế) và thay thế các vế bằng nhau (phép cắt). Các quy tắc này xác định phép suy diễn  $\hat{E}$  EQ, định lý sau đây minh họa kích thước và tính rõ của phép suy diễn

Định lý : lý thuyết tương đương

với một đặc tả spec,  $\forall t = t', Ax \text{ spec } \hat{E} \text{ EQ } t = t'$

$\Leftrightarrow \text{Mod}(Ax \text{ spec}) \cup t = t'$

Công thức Mod  $(Ax \text{ spec}) \text{ U } t = t'$  chỉ ra rằng phương trình là hợp thức trong mọi cách lập trình có thể. Tuy nhiên có thể xảy ra một số trường hợp đặc biệt khi mô hình không hợp lý, lúc đó có thể  $\text{true} = \text{false}$ .

Ta có thể chứng minh rằng thuộc tính

$$\text{succ}(\text{succ}(0)) - \text{succ}(\text{succ}(0)) = 0$$

là hợp thức (valid), những thuộc tính

$$\text{succ}(\text{succ}(\text{succ}(0))) - \text{succ}(\text{succ}(0)) = 0$$

Không là hợp thức trong đặc tả đang xét, vì rằng sau khi suy diễn, ta nhận được  $\text{succ}(0) = 0$  là không hợp thức.

Ta có thể thấy rằng  $x - x = 0$  không chứng minh được trong ngữ cảnh đang xét mặc dầu định lý này tỏ ra hiển nhiên trong đặc tả. Từ đó, ta có thể bổ sung thêm một số giả thiết cho mô hình để tăng khả năng chứng minh có thể.

### III.6.2. Khái niệm về lý thuyết quy nạp

Như đã chỉ ra, ta cần thêm các định lý tổng quan hơn để có khả năng suy diễn từ các tiên đề của đặc tả, chẳng hạn như  $x + y = y + x$  là không có tính suy diễn trong lý thuyết tương đương.

Ta sẽ thêm vào các qui tắc sử dụng trong lý thuyết tương đương một quy tắc mới như sau :

- Qui nạp : giả sử  $G$  là công thức sao cho  $x$  là một biến tự do, nếu với mọi  $t$ ,  $G[t/x]$  là suy diễn được thì  $G$  cũng suy diễn được cho  $t$ . Quy tắc này chỉ rõ rằng có thể kết luận rằng nếu việc chứng minh một định lý là hợp thức cho mọi trường hợp, định nghĩa bởi một hạng, bởi một biến thì định lý cũng hợp thức cho công thức được lượng hóa một cách phổ dụng trên biến này.

Tương tự đối với định lý tương đương, định lý sau đây cho kết quả thuyết phục cho việc suy diễn quy nạp đối với đặc tả hữu hạn.

Định lý 3.2 : Lý thuyết quy nạp

Với một đặc tả đại số spec

$$\forall t = t', Ax \text{ spec } \hat{E} \text{ Ind } t = t'$$

$$\Leftrightarrow \text{Mod}_{\text{Gen}}(Ax \text{ spec}) \text{ U } t = t'$$

Ta sẽ minh họa nguyên lý này bởi một ví dụ trên các giá trị logic xây dựng từ các phép toán true, false và not. Ta muốn chứng minh rằng :

$$\text{not}(\text{not}(b)) = b$$

- trường hợp cơ sở :

$$? \text{not}(\text{not}(\text{true})) = \text{not}(\text{false}) = \text{true} ;$$

$$2. \text{Not}(\text{not}(\text{false})) = \text{not}(\text{true}) = \text{false} ;$$

- Không quy nạp :

$\text{not}(\text{not}(b)) = b$  suy ra  $\text{not}(\text{not}(\text{not}(b))) = \text{not}(b)$

quy tắc khả thể với not cho  $\text{not}(\text{not}(b)) = b$

$\text{not}(\text{not}(\text{not}(b))) = \text{not}(b)$

Nhờ quy tắc quy nạp mà thuộc tính mong muốn được chứng minh. Như vậy lý thuyết quy nạp cho phép chứng minh tính giao hoán của phép cộng trong Not qua  $x + y = y + x$  việc chứng minh cần quy nạp hai lần trên  $x$  và  $y$ .

### III.6.3. Chứng minh tự động bởi viết lại

Việc chứng minh bởi viết lại (demonstration by rewriding) là một kỹ thuật cho phép chứng minh tự động. Đó là quá trình ước lượng các hạng bằng cách viết lại một cách hệ thống các hạng thành các dạng chuẩn (dạng không thể ước lượng được nữa) và phép chứng minh các thuộc tính. Nguyên lý sử dụng là hướng đến các phương trình đặc tả theo quy tắc viết lại và áp dụng liên tiếp các quy tắc này trên các hạng đã ước lượng.

Ví dụ : từ tiên đề  $\text{not true} = \text{false}$  ta sẽ dẫn đến quy tắc  $\text{not true} \text{ ??? } \text{false}$ . Quy tắc này được dùng để chứng minh tính bằng nhau của dạng  $t = t'$ . Sự bằng nhau là hợp thức nếu hai vế của chúng được viết lại thành duy nhất một hạng không thể ước lượng được nữa.

Định lý 3.3 Chứng minh bởi viết lại

Với một đặc tả  $\text{spect}$ ,  $t$ ,  $t'$  là các hạng nếu

$t \text{ ??? } \dots \text{ ??? } \text{to}$  và  $t' \text{ ??? } \dots \text{to}$  thì :

$Ax \text{ spec } \hat{E} \text{ EQ } t = t'$

Ở đây ta sử dụng ký hiệu  $t \text{ ??? } * t'$  cho dãy  $t \text{ ??? } \dots \text{ ??? } t$  hay  $t$  là một dạng chuẩn của hạng, nghĩa là một hạng không thể thu gọn.

Cần chú ý rằng đẳng thức tạo ra bởi viết lại không bắt buộc đồng nhất với đẳng thức nhận được từ hệ thống suy diễn  $\hat{E} \text{ EQ}$  (không hoàn toàn).

Để có thể thực hiện các phép chứng minh theo lý thuyết trước đây, ta cần nhận được một hệ thống viết lại hội tụ tương đương với hệ thống sinh bởi các tiêu đề.

Giải pháp đầu tiên là hướng tới các phương trình, Nếu hệ thống nhận được là đi đến đích (mọi hướng suy dẫn khác nhau có thể đều dẫn về cùng kết quả) và kết thúc (sau một số hữu hạn bước viết lại trước khi nhân được dạng chuẩn). Từ đó các thuộc tính chứng minh được tương đương với các phương trình xuất phát.

Chẳng hạn để đặc tả Bool, ta cần nhận được bằng cách hướng các tiên đề từ trái qua phải :

$\text{not}(\text{true}) \text{ ??? } \text{false}$

not (false) ??? true

true and b ??? b

false and b ??? false

true or b ??? true

false or b ??? b

false xor b ??? not (b)

Ví dụ : sử dụng các quy tắc trên để viết lại hạng sau đây :

not (false or (true and false))

not (true and false) not (false or false)

not (false)

true

Tuy nhiên, nguyên lý hướng về viết lại không đủ để chứng minh mọi thuộc tính tương đương. Ta có thể minh họa điều đó trong đặc tả các số tự nhiên một cách đơn giản như sau :

Interface

Sort not ;

Operations

0 :  $\rightarrow$  not ;

\_ \_ : not  $\rightarrow$  not ;

\_ + \_ : not not  $\rightarrow$  not ;

Body

Axions

a x 1 0 + x = x ;

a x 2 : x + (- x) = 0 ;

...

Từ đặc tả trên, ta có thể xây dựng các quy tắc :

0 + x ??? x

x + (- x) ??? 0

- 0 ??? 0



Cần chú ý rằng trong trường hợp này, việc hướng các quy tắc từ trái qua phải chưa đủ, vì nếu muốn chứng minh  $- 0 = 0$  thì phải cần áp dụng tiên đề 1 từ phải qua trái, sau đó áp dụng tiên đề 2 từ trái qua phải, như vậy sẽ không tương ứng với việc lựa chọn định hướng ???.

Rõ ràng việc định hướng là một cơ chế chứng minh chưa đầy đủ, đặc biệt đối với các tiên đề về các phép tính sinh. Cơ chế này có thể đầy đủ trong nhiều tình huống thực tế.

Trong trường hợp các tiên đề không định hướng như phép giao hoán, các kỹ thuật đặc tả được phát triển để thực hiện viết lại (hệ viết lại kiểu modun kết hợp - giao hoán)

Các phép toán phát sinh (xây dựng)

Theo định nghĩa, một mô hình được phát sinh bởi một tập hợp con  $w$  các phép toán nếu mọi giá trị của mô hình này nhân được bởi một hạng được xây dựng từ các bộ sinh  $w$ . Định nghĩa này cho phép, theo định lý quy nạp, chỉ xem xét các bộ sinh trong các chứng minh bằng cách chỉ chứng minh quy nạp chúng (bởi vì mọi hạng đạt được bởi các việc tổ hợp các bộ sinh)

### **III.6.4. Phân cấp trong đặc tả đại số**

Các mô hình phân thể làm thỏa mãn các tiên đề và các hạn chế do các ràng buộc đơn thể chủ yếu dựa trên khả năng buộc đơn thể. Các ràng buộc đơn thể chủ yếu dựa trên khả năng không bị xáo trộn giữa các mô hình ở mức phân cấp thấp hơn khi sử dụng một đặc tả. Nguyên lý này cho phép sử dụng việc phân cấp các đặc tả trong các giai đoạn khai triển, đặc biệt khi làm mịn (refinement). Có nghĩa là các mô hình phải được lập trình độc lập với nhau.

Các kiểu xáo trộn có thể xuất hiện trong một đặc tả đại số là :

- "junk" (mảng) : các giá trị được thêm vào bởi việc dùng các đơn thể, ràng buộc về tính đầy đủ hạn chế kiểu xáo trộn này.

- "cofusion" (trộn lẫn) : các giá trị bị thay đổi (collapse), do ràng buộc về sự hiện hữu phân cấp làm ảnh hưởng đến đặc tả.

Lớp các mô hình phân cấp được ký hiệu bởi Hitol (spec) đó là những mô hình thỏa mãn quy tắc về sự hạn chế các mô hình trên các mô hình con bảo toàn được ngữ nghĩa của chúng.

Tính rõ ràng (Completeness)

Ta sẽ minh họa tình huống "junk" bằng một ví dụ

Adl Không\_hoàn\_toàn ;

In terface

Use Not, Bool ;

Operation

$f : \text{not} \rightarrow \text{bool} ;$

Body

Axioms

$f(\text{succ}(x)) = \text{false} ;$

where

$x : \text{not} ;$

End ;

Ví dụ trên không rõ ràng khi thêm định nghĩa hàm  $f$  trên các đơn thể về  $\text{not}$  và  $\text{bool}$ . Các tiên đề về đơn thể của hàm  $f$  sẽ làm xáo trộn các kiểu đã định nghĩa. Ta thấy một giá trị mới  $f(0)$  kiểu sẽ không được xác định vì không có tiên đề nào chỉ ra  $f(0)$  là  $\text{true}$  hay  $\text{false}$ .

## IV. Đặc tả hay cách cụ thể hóa sự trừu tượng

(Specification or How to Make Abstractions Real)

### IV.1. Đặc tả phép thay đổi bộ nhớ

Giả sử ta cần đặc tả phép thay đổi nội dung một bộ nhớ. Để đơn giản hóa mà không làm mất tính được biểu diễn bởi sơ đồ sau :

1	0
2	0
3	0

bộ nhớ chỉ gồm 3 địa chỉ nhớ chứa 3 giá trị lúc đầu đều là 0 giả lệnh, làm thay đổi nội dung của bộ nhớ được ký hiệu bởi **chg** ( $x, y$ )

Ví dụ lệnh **chg** (2, 5) làm thay đổi địa chỉ thứ 2 từ giá trị 0 thành 5 :

1	0	<b>chg</b> (2, 5)	1	0
2	0		2	5
3	0		3	0

Như vậy, lệnh **chg** ( $x, y$ ) đã thay đổi nội dung của địa chỉ  $x$  thành giá trị  $y$  và giữ nguyên nội dung của các địa chỉ còn lại.

Gọi  $q$  là bộ nhớ lúc đầu và  $q'$  là bộ nhớ nhận được sau khi thực hiện lệnh **chg** ( $x, y$ ), ta có thể chuyển sơ đồ trên thành dạng điều kiện như sau :

$$\boxed{q \text{ chg}(x, y) q'} \quad (1.1)$$

Ta nói lệnh **chg** ( $x, y$ ) đã chuyển bộ nhớ  $q$  thành  $q'$ .

Định nghĩa ngữ nghĩa của lệnh **chg** chính là đặc tả, nhờ viết điều kiện tương đương với điều kiện (1.1)

Ví dụ sự kết hợp các điều kiện sau đây để đặc tả phép toán thay đổi **chg** :

$$q'(x) = y$$

$$q'(a) = q(a) \text{ nếu } a \neq x \quad (1.2)$$

Như vậy ta đã ngầm ẩn thừa nhận rằng  $q$  và  $q'$  chỉ định các hàm. Trong ví dụ này, với bộ nhớ 3 địa chỉ, miền xác định của hàm là tập hợp  $\{1, 2, 3\}$  tổng quát các địa chỉ và miền giá trị là các giá trị có thể lưu trữ được trong bộ nhớ đang xét, chẳng hạn là tập hợp các số nguyên  $\{-2^{31}, \dots, 2^{31}\}$  (1.3)

Một cách tổng quát, gọi  $A$  là tập hợp các địa chỉ,  $V$  là tập hợp các giá trị, ta có:

$$q \in A \rightarrow V \quad (1.4)$$

Như vậy một biểu thức dạng  $q(a)$  chỉ có nghĩa nếu  $a \in A$ , từ đó  $q(a)$  chỉ có nghĩa nếu  $a \in A$ , từ đó  $q(a) \in V$ . Do bộ nhớ  $q'$  nhận được từ  $q$  sau khi thực hiện lệnh **chg**,  $q'$  cũng thỏa mãn điều kiện (1.4)

$$q' \in A \rightarrow V$$

Tuy nhiên điều kiện (1.2) không phải luôn luôn có nghĩa vì rằng biểu thức  $q'(x)$  đòi hỏi  $x \in A$ . Ta thấy điều kiện (1.1) dẫn đến (1.2) nhưng ngược lại không hoàn toàn đúng.

Ta có thể hạn chế các điều kiện (1.2) để có được điều kiện tương đương như sau

$$x \in A$$

$$y \in V$$

$$q'(x) = y \quad (1.5)$$

$$q'(a) = q(a) \text{ với } x \neq a \text{ và } x \in A$$

Trong (1.5), hai điều kiện đầu được gọi là điều kiện đầu (preconditions), hai điều kiện sau được gọi là các điều kiện sau (postconditions)

Ta cần kiểm tra (1.5) là chấp nhận được, nghĩa là có điều kiện (1.4) là bất biến (invariant). Muốn vậy ta cần chứng minh định lý sau đây :

$$((1.4) \text{ và } (1.5)) \text{ kéo theo } (1.4)' \quad (1.6)$$

Định lý về tính chấp nhận được (plausibility)

Ta có thể nói hai điều kiện tương đương (1.1) và (1.5)

$$q \text{ **chg** } (x, y) \text{ } q'$$

$$x \in A$$

$$y \in V \quad (1.7)$$

$$q'(x) = y$$

$$q'(a) = q(a) \text{ với } x \neq a \text{ và } a \in A$$

Người ta nói  $q$  xác định trạng thái (state) của hệ thống, một điều kiện như (1.4) là một bất biến của hệ thống, (1.7) là đặc tả lệnh làm chuyển (tiến triển) trạng thái của hệ thống. Ở đây ta sử dụng các biến có đánh dấu nháy để chỉ trạng thái của hệ thống sau khi chuyển đổi. Ta cũng nói một đặc tả là chấp nhận được (plausible) nếu đặc tả đó bảo toàn bất biến của hệ thống.

Nhận xét

Dễ dàng chứng minh định lý (1.6) nhưng cũng dễ dàng bước lại định lý bằng cách xét phản ví dụ sau :

1	0	1	0	
2	0	<b>chg</b> (2, 5)	2	5
3	0	3	0	
9	4	0		

Rõ ràng ví dụ trên thỏa mãn các điều kiện (1.4) và (1.5) nhưng không thỏa mãn (1.4)'

Thực ra, lệnh **chg** (2, 5) thay đổi nội dung địa chỉ  $x$  (cho giá trị  $y$ ) nhưng vẫn giữ nguyên các địa chỉ khác, nghĩa là kích thước bộ nhớ không thay đổi.

## IV.2. Hàm

Trên đây, ta đã vận dụng quan điểm toán học về hàm, sau đây ta tiếp tục làm rõ một số khái niệm và tính chất của quan điểm này.

Cho hai tập hợp  $X$  và  $Y$ , biểu thức  $X \rightarrow X$  biểu diễn tập hợp các hàm toàn phần (total) với miền xác định (nguồn) là  $X$  và miền trị (đích) là  $Y$ . Tương tự, biểu thức  $X \rightarrow Y$  biểu diễn tập hợp các hàm bộ phận (partical) từ nguồn  $X$  vào đích  $Y$ . Sự khác nhau giữa chúng là ở chỗ, một hàm bộ phận không hoàn toàn xác định cho mọi giá trị của nguồn  $X$ .

Nếu  $f$  là một hàm bộ phận thì ký hiệu  $\text{dom}(f)$  (domain) là tập hợp con của  $X$  mà  $f$  xác định. Trong trường hợp hàm toàn phần, miền xác định và nguồn là đồng nhất. Ví dụ hàm  $q$  ở mục trên là hàm toàn phần với nguồn  $X = \{1, 2, 3\}$  và đích  $Y = \{-2^{31}, \dots, 2^{31}\}$ , ta có thể viết :

$$q \in \{1, 2, 3\} \rightarrow \{-2^{31}, \dots, 2^{31}\}$$

Tuy nhiên đây là một hàm bộ phận từ tập số nguyên  $Z$  vào chính nó vì các tập hợp nguồn và đích của  $q$  đều là tập hợp con của  $Z$ .

$$q \in Z \rightarrow Z$$

lúc này  $\text{dom}(q) = \{1, 2, 3\}$

Ta cũng có thể xây dựng tập hợp con của đích chứa các giá trị xác định từ tập hợp con của nguồn, gọi là ran ( $f$ ) (range)

Khi một hàm được định nghĩa, ta có thể liệt kê các thành phần của hàm. Ví dụ, ta có :

$$\begin{array}{l} 1 \quad 0 \quad 0 \\ 2 \quad 0 \quad \text{chg}(2, 5) \quad 5 \\ 3 \quad 0 \quad 0 \end{array}$$

tương ứng với hai hàm :

$$q = \{ 1 \rightarrow 0, 2 \rightarrow 0, 3 \rightarrow 0 \}$$

$$q' = \{ 1 \rightarrow 0, 2 \rightarrow 5, 3 \rightarrow 0 \}$$

Để nhận được các dom và các ran từ các hàm, người ta tập hợp các phần tử đặt ở bên trái và bên phải của mũi tên tương ứng :

$$\text{dom}(q) = \{1, 2, 3\} \quad \text{dom}(q') = \{1, 2, 3\}$$

$$\text{ran}(q) = \{0\} \quad \text{ran}(q') = \{0, 5\}$$

Một cách tổng quan, ta có kết quả sau :

$$\text{dom}(\{x \rightarrow y\}) = \{x\}$$

$$\text{ran}(\{x \rightarrow y\}) = \{y\} \quad (2.1)$$

$$f \in X \rightarrow Y \quad \text{kéo theo} \quad \text{dom}(f) = X$$

Cho hàm  $f \in X \rightarrow Y$  và một tập hợp con  $S \subseteq X$ , người ta ký hiệu  $f \setminus S$  là hàm nhận được bằng cách loại khỏi dom ( $f$ ) các phần tử của  $S$ . Đây là phép hạn chế tương ứng với định nghĩa sau :

$$\text{dom}(f \setminus S) = \text{dom}(f) - S$$

$$(f \setminus S)(x) = f(x) \text{ với } x \in \text{dom}(f) - S$$

### HÌNH VẼ

$$\text{Ta có : } \{1 \rightarrow 5, 2 \rightarrow 8, 3 \rightarrow 6\} \setminus \{1, 2\} = \{3 \rightarrow 6\} \setminus \{1, 2\} = \{3 \rightarrow 6\}$$

Cho hai hàm có cùng nguồn và cùng đích nhưng có các dom rời nhau. Ký hiệu  $f \cup g$  là hợp của hai hàm theo định nghĩa sau :

$$\text{dom}(f \cup g) = \text{dom}(f) \cup \text{dom}(g)$$

$$f(x) \text{ nếu } x \in \text{dom}(f)$$

$$(f \cup g)(x) = g(x) \text{ nếu } x \in \text{dom}(g) \quad (2.4)$$

Bằng cách dùng hai phép toán trên đây, ta có thể định nghĩa sự chồng lên (overload) của một hàm bởi một hàm khác. Ta ký hiệu  $f + g$  tác động lên hai hàm  $f$  và  $g$  có cùng nguồn và cùng đích mà lần này, các dom không nhất thiết rời nhau :

$$\text{Ta có : } f + g = (f \setminus \text{dom}(g)) \cup g \quad (2.5)$$

Từ (2.3) ta có :

$$\begin{aligned} (f \setminus \text{dom}(g))(x) & \quad \text{nếu } x \in \text{dom}(f \setminus \text{dom}(g)) \\ (f + g)(x) = & \quad g(x) \quad \text{nếu } x \in \text{dom}(g) \end{aligned} \quad (2.6)$$

hay

$$\begin{aligned} (f + g)(x) = f(x) & \quad \text{nếu } x \in \text{dom}(f) - \text{dom}(g) \\ & \quad g(x) \quad \text{nếu } x \in \text{dom}(g) \end{aligned}$$

### HÌNH VẼ

$$\begin{aligned} \text{Ví dụ : } & \{ 1 \rightarrow 0, 2 \rightarrow 0, 3 \rightarrow 0 \} + \{ 2 \rightarrow 5 \} \\ & = (\{ 1 \rightarrow 0, 2 \rightarrow 0, 3 \rightarrow 0 \} \setminus \{ 2 \} \cup \{ 2 \rightarrow 0 \}) \\ & = \{ 1 \rightarrow 0, 3 \rightarrow 0 \} \cup \{ 2 \rightarrow 5 \} \\ & = \{ 1 \rightarrow 0, 2 \rightarrow 5, 3 \rightarrow 0 \} \end{aligned}$$

Lý do cơ bản để đưa vào các phép toán  $\setminus$ ,  $\cup$  và  $+$  thay vì sử dụng một cách hệ thống những định nghĩa của chúng (2.3), (2.4) và (2.7) trong việc hình thức hóa là ở chỗ ta có thể chứng minh dễ dàng dãy các tính chất đại số sẽ sử dụng về sau.

Sau đây là một số tính chất :

$$\begin{aligned} (f \cup g) \setminus S &= (f \setminus G) \cup (g \setminus S) \\ (f + g) \setminus S &= (f \setminus S) + (g \setminus S) \\ (f \setminus S) \setminus T &= f \setminus (S \cup T) \\ f \cup g &= g \cup f \\ (f \cup g) \cup h &= f \cup (g \cup h) \\ \text{dom}(f \cup g) &= \text{dom}(f) \cup \text{dom}(g) \quad (2.8) \\ \text{ran}(f \cup g) &= \text{ran}(f) \cup \text{ran}(g) \\ (f + g) + h &= f + (g + h) \\ \text{dom}(f + g) &= \text{dom}(f) \cup \text{dom}(g) \end{aligned}$$

Bây giờ ta có thể sửa chữa đặc tả đã nhận được ở cuốn mục trước (1.7). Đặc tả này rõ ràng đơn giản hơn :

$$\begin{aligned} & q \text{ chg } (x, y) \text{ } q' \\ & x \in A \\ & y \in V \\ & q' = q + \{ x \rightarrow y \} \end{aligned}$$

Định lý về tính chấp nhận được (plausibility) bây giờ dễ dàng được chứng minh bởi phép tính hình thức đơn giản (simple formal calculus).

Theo (2.1) và (2.8), ta có :

$$\text{dom}(q') = \text{dom}(q) \cup \{x\}$$

$$\text{ran}(q') \subset \text{ran}(q) \cup \{y\}$$

Nhưng ta có giả thiết (1.4) và giả thiết (2.8) và theo (2.2)

$$\text{dom}(q) = A \text{ và } \{x\} \subset A$$

$$\text{ran}(q) \subset V \text{ và } \{y\} \subset V$$

Như vậy :

$$\text{dom}(q') = A$$

$$\text{ran}(q') \subset V$$

Từ đó theo (2.2) thì  $q' \in A \rightarrow V$

### IV.3. Hợp thức hóa và phục hồi

Trong mục 1, ta đã đặc tả cách hoạt động của một bộ nhớ trong đó, ta có thể thay đổi nội dung của nó. Bây giờ, ta sẽ tiếp tục phát triển ví dụ này bằng cách đặt ra hai yêu cầu bổ sung ta muốn rằng những thay đổi trên bộ nhớ có đặc tính tạm thời, nghĩa là ta có thể thay đổi trở lại nhờ một phép toán thích hợp, mặt khác ta có hợp thức hóa (validation) bằng cách trả lại những thay đổi trước đó.

Ta gọi not (restart) và vld (validate) là những thao tác mới. Những yêu cầu bổ sung vừa nêu có thể biểu diễn hình thức bằng cách kết hợp các điều kiện sau đây :

$$q \text{ vld } q_1$$

$$q_1 \text{ op } q_2$$

...

$$q_{n-1} \text{ op } q_n$$

$$q_n \text{ rst } q'$$

Trong đó op (operations) là một thao tác dạng chung (x, y) hay rdm, dẫn đến điều kiện :

$$q' = q$$

với q và n bất kỳ. Vả lại, yêu cầu về tính "trong suốt" của phép toán hợp thức hóa dẫn đến điều kiện :

$$q' = q$$

Cách giải quyết hiển nhiên nhất mang tính ý niệm là làm tăng gấp đôi bộ nhớ. Như vậy trạng thái của hệ thống bây giờ được đặc trưng bởi hai biến p và q như sau :

$$p \in A \rightarrow V$$

$$q \in A \rightarrow V \quad (3.1)$$

Bộ nhớ  $q$  đóng vai trò bộ nhớ trước đó, còn bộ nhớ  $p$  dùng để khôi phục trạng thái cũ khi cần khởi động lại. Sau đây là đặc tả của 3 thao tác cho hệ thống với mod thay thế chg :

$$(p, q) \text{ mod } (x, y) ((p', q'))$$

$$p' = p \quad (3.2)$$

$$q \text{ chg } (x, y) q'$$

Ta thấy thao tác thay đổi bộ nhớ xuất hiện như một sự mở rộng, ở mức đặc tả, của phép toán chg :

$$(p, q) \text{ rst } (p', q')$$

$$q' = p$$

$$q' = q \quad (3.3)$$

$$(p, q) \text{ vld } (p', q')$$

$$p' = q$$

$$q' = q \quad (3.4)$$

Dễ dàng chứng minh rằng cả ba phép toán này đều chấp nhận được, nghĩa là sau khi thực hiện chúng, ta có (3.1)'. Nói cách khác,  $p$  và  $q$  được thay thế bởi  $p'$  và  $q'$  trong (3.1), với giả thiết rằng (3.1) đã được kiểm chứng trước khi thực hiện chúng.

Rốt cuộc, ta phải chứng minh rằng việc kết hợp các điều kiện sau đây :

$$(p, q) \text{ vld } (p_1, q_1)$$

$$(p_1, q_1) \text{ op } (p_2, q_2)$$

...

$$(p_{n-1}, q_{n-1}) \text{ op } (p_n, q_n)$$

$$(p_n, q_n) \text{ rst } (p', q')$$

$$\text{dẫn đến } q' = q$$

Thật vậy, theo (3.4), ta có :

$$p_1 = q$$

và theo (3.2) và (3.4) do op là một trong hai phép toán mod (x, y) hoặc rst, ta có

:

$$p_n = \dots p_1$$

Cuối cùng, theo (3.3) :  $q' = p_n$

Rõ ràng phép hợp thức hóa là trong suốt vì dẫn đến  $q' = q$  theo (3.4)

Sau đây là một ví dụ về các phép toán trên :



```

    q    p
1 0    0
2 0    0
3 0    0
mod (1, 1)
1 1    0
2 0    0
3 0    0
mod (2, 2)
1 1    0
2 2    0
3 0    0
mod (1, 3)
1 3    0
2 2    0
3 0    0
vld
1 3    3
2 2    2
3 0    0
mod (3, 1)    q    q
1 3    3
2 2    2
2 1    0
not
1 3    3
2 2    2
3 0    0

```

Phần tiếp theo sẽ làm mịn mô hình này, nghĩa là đưa vào các biến trạng thái mới để thể hiện các ràng buộc về phần cứng và phần mềm.

#### IV.4. Bắt đầu triển khai thực tiễn

Về mặt thực tiễn, có nhiều cách để triển khai hệ thống đã được đặc tả trong mục trước. Thật vậy, có nhiều yếu tố kỹ thuật có thể ảnh hưởng đến cách triển khai ; chẳng hạn kích thước không gian V các giá trị đóng vai trò quan trọng : giả sử rằng các phần tử của tập hợp A tương ứng với các địa chỉ của các trang trong một hệ thống có bộ nhớ phân trang (paging). Trong trường hợp này, mỗi "giá trị" sẽ tương ứng với nội dung của một trạng thái có kích thước điển hình là 1 bytes. Nếu những trang này dùng để thể hiện một bộ nhớ ảo 4 Mbytes, thì ta sẽ thấy rằng có 4096 trang và bởi vậy thời gian dùng để thực hiện các sao chép cần thiết cho các thao tác khởi động và hợp thức hóa có thể tỏ ra nặng nề.

Trong trường hợp vừa nêu (ta sẽ triển khai thực tiễn trong mục này), cách giải quyết là sử dụng cách gián tiếp : hai là bộ nhớ p và q tạo thành trạng thái của hệ thống đã xét trong mục trước, sẽ được thay thế bởi hai bảng (hai hàm) a và n (a: ancient, n : new) chứa các con trỏ tới bộ nhớ m. Hệ thống mới sẽ được đặc trưng bởi các thành phần sau :

$$\begin{aligned}
 a &\in A && \rightarrow && D \\
 n &\in A && \rightarrow && D \\
 m &\in D && \rightarrow && V \quad (4.1)
 \end{aligned}$$

Trong đó D là tập hợp các địa chỉ của bộ nhớ m. Sau đây là sơ đồ minh họa hệ thống mới này :

	n	a	m	
1	2	2	1	2
2	4	6	2	5
3	5	5	3	5
	4	8		
	5	1		
	6	4		

Trong ví dụ trên, cũng trong các ví dụ trên về sau, ta tiếp tục sử dụng các giá trị V như trước.

Bằng cách kết hợp các bảng a và n với m, ta nhận được các bảng p và q của hệ thống cũ :

q	p	
1	5	5
2	8	4
3	1	1

Như vậy, ta có thể thấy rằng hai hệ thống không độc lập với nhau : hai hệ thống mới hiện thực hóa hệ thống cũ và hệ thống cũ thể hiện sự thay đổi biến như sau

$$p(x) = m(a(x))$$

$$q(x) = m(n(x)) \text{ với } x \in A \quad (4.2)$$

Ta có thể kiểm chứng ngay được rằng những thay đổi của biến là có ý nghĩa (rõ ràng vì các hàm  $a$ ,  $n$  và  $m$  là toàn phần) và chặt chẽ với bất biến (3.1) chỉ rõ rằng  $p$  và  $q$  đều thuộc tập hợp  $A \rightarrow V$

Phép biến đổi mod đặc tả bởi các điều kiện (3.2) sẽ được triển khai bởi một phép toán mới  $\text{mod}_1$ . Phép  $\text{mod}_1$  sẽ ghi một giá trị mới, một địa chỉ mà không thuộc vào miền trị (range) của  $n$  cũng như miền trị của  $a$ , nói cách khác, địa chỉ này chỉ phụ thuộc vào tập hợp :

$$\text{ran}(n) \cup \text{ran}(a)$$

hay rõ hơn, thuộc tập hợp  $D - (\text{ran}(n) \cup \text{ran}(a))$ , tập hợp được đặt tên là  $L$  (Liberty).

Nếu  $L \neq \emptyset$ , một điều kiện trước được đặt ra, thì ta có thể chọn một địa chỉ bất kỳ  $u$ , với đặc tả sau :

$$(a, n, m) \text{ mod}_1 (x, y) (a', n', m')$$

$$x \in A$$

$$y \in V$$

$$L \neq \emptyset$$

$$a' = a$$

$$n' = n + \{x \rightarrow u\}$$

$$m' = m + \{u \rightarrow y\} \quad (4.3)$$

trong đó

$$L = \text{ran}(n) \cup \text{ran}(a)$$

$$u \in L$$

Ta cần chứng minh rằng đặc tả (4.3) là chấp nhận được, nghĩa là :

$$((4.1) \text{ và } (4.3)) \text{ kéo theo } (4.1)' \text{ (4.4)}$$

Mệnh đề (4.4) trên đây là hiển nhiên. Tiếp theo ta cần chứng minh phép  $\text{mod}_1$  phù hợp (đúng) với phép mod đã đặc tả ở (3.2) như sau :

$$((4.2), (4.2)' \text{ và } (4.3)) \text{ kéo theo } (3.2) \text{ (4.5)}$$

Mệnh đề này không hiển nhiên, tương ứng với sơ đồ giao hoán dưới đây :

### HÌNH VẼ

Nói cách khác, nếu các giá trị của các biến  $(a, n, m)$  và  $(a', n', m')$  thỏa mãn đặc tả (4.3), thì khi trở về các biến cũ  $(p, q)$  và  $(p', q')$  (các biến đã bị thay đổi trở thành các biến  $(a, n, m)$  và  $(a', n', m')$  bởi các phép biến đổi (4.2) và (4.2)') các giá trị của các biến  $(p, q)$  và  $(p', q')$  thỏa mãn đặc tả (3.2).

Nói gọn lại, (4.3) hiện thực (3.2)

Các phép toán mới khởi tạo lại các hợp thức hóa mô tả trong các đặc tả sau đây rõ ràng chấp nhận được và phù hợp :

$$(a, n, m) \text{ rst}_1 (a', n', m')$$

$$a' = a$$

$$n' = n \quad (4.6)$$

$$m' = m$$

$$(a, n, m) \text{ vld}_1 (a', n', m')$$

$$a' = n$$

$$n' = n \quad (4.7)$$

$$m' = m$$

Nhìn vào các công thức, ta thấy đã không chép lại các giá trị nhưng chỉ có các địa chỉ có thể có theo một sự tiết kiệm đáng kể về thời gian nhưng không lớn lắm về không gian nhớ. Giả thiết với 4096 trang và địa chỉ của D là 2 bytes, mỗi bảng a và n sẽ chiếm 8 Kbytes.

Hệ thống mới hoạt động qua ví dụ sau :

	n	a	m		
1	1	1	1	0	
2	2	2	2	0	
3	3	3	3	0	
4	0				
5	0				
6	0				
$\text{mod}_1 (1, 1)$					
1	4	1	1	0	$L = \{ 4, 5, 6 \}$
2	2	2	2	0	$u = 4 \in L$
3	3	3	3	0	(chọn $u = \min (L)$ )
4	1				
5	0				
6	0				

## IV.5. Phép hợp thành (cấu tạo)

Ở mục trước, ta đã sử dụng một cách phi hình thức phép hợp thành (composition operation) của  $p$  và  $q$  theo  $a$ ,  $m$  và  $n$ . Trong mục này, ta tiếp tục định nghĩa phép hợp thành một cách chặt chẽ hơn.

Phép hợp thành, ký hiệu là  $\circ$ , tác động lên hai toán hạng là hai hàm, chẳng  $f$  và  $g$ , thuộc về các tập hợp  $X \rightarrow Y$  và  $Y \rightarrow Z$  tương ứng, sao cho hàm  $f \circ g$  thuộc về tập  $X \rightarrow Z$

Phép hợp thành có thể được định nghĩa như sau :

$$\text{dom}(f \circ g) = \{ x \in \text{dom}(g) / g(x) \in \text{dom}(f) \}$$

$$(f \circ g)(x) = f(g(x)) \text{ với } x \in \text{dom}(f \circ g) \quad (5.1)$$

Ví dụ :

$$\{ 2 \rightarrow 6, 5 \rightarrow 8 \} \circ \{ 1 \rightarrow 2, 4 \rightarrow 3, 7 \rightarrow 5 \} = \{ 1 \rightarrow 6, 7 \rightarrow 8 \}$$

Từ định nghĩa trên có thể suy ra ngay rằng nếu các hàm  $f$  và  $g$  đều toàn phần (nói cách khác, nếu  $\text{dom}(g) = X$  và  $\text{dom}(f) = Y$ ), thì hàm  $f \circ g$  cũng là toàn phần. Một cách tổng quan hơn, nếu miền trị của  $g$  nằm trong miền xác định của  $f$  thì hai hàm  $g$  và  $f \circ g$  có cùng miền xác định. Ta có thể dễ dàng xây dựng một số luật kiểu đại số để nối liền phép hợp thành với các phép hội và hạn chế đã xét ở mục 2. Sau đây là một số luật như vậy :

$$(f \cup g) \circ h = (f \circ h) \cup (g \circ h)$$

$$f \circ (g \cup h) = (f \circ g) \cup (f \circ h) \quad (5.2)$$

$$S \cap \text{rang}(g) = \emptyset \text{ kéo theo } (f \setminus S) \circ g = f \circ g \quad (5.3)$$

$$\text{dom}(f) \cap \text{ran}(g) = \emptyset \text{ kéo theo } f \circ g = \{ \} \quad (5.4)$$

Biểu thức  $\{ \}$  chỉ định hàm rỗng

$$f \circ (g \setminus S) = (f \circ g) \setminus S \quad (5.5)$$

$$\{ x \rightarrow y \} \circ \{ x \rightarrow u \} = \{ x \rightarrow y \} \quad (5.6)$$

Từ các luật trên, ta có thể đơn giản hóa phép thay đổi biến đã định nghĩa ở (4.2) như sau :

$$p = m \circ a$$

$$q = m \circ n \quad (5.7)$$

Và ta có thể chứng minh dễ dàng định lý phù hợp (4.5) bằng cách sử dụng các tính chất trên. Thật vậy, ta cần chứng minh hai đẳng thức sau đây :

$$m' \circ n' = (m \circ n) + \{ x \rightarrow y \}$$

$$m' \circ o' = m \circ a$$

Nghĩa là :

$$(m + \{ u \rightarrow y \}) \circ (n + \{ x \rightarrow u \}) = (m \circ n) + \{ x \rightarrow y \}$$

$$(m + \{ u \rightarrow y \}) \circ a = m \circ a$$

với các giả thiết :

$$u \notin \text{ran}(n) \text{ nghĩa là } \{ u \} \cap \text{ran}(n) = \emptyset \quad (5.8)$$

$$u \notin \text{ran}(a) \text{ nghĩa là } \{ u \} \cap \text{ran}(a) = \emptyset \quad (5.9)$$

Ta có kết quả phụ như sau :

$$(m \setminus \{ u \}) \circ (n \setminus \{ x \}) = ((m \setminus \{ u \}) \circ n) \setminus \{ x \} \text{ theo (5.5) } = (m \circ n) \setminus \{ x \}$$

theo (5.3) và (5.8). Nhưng :

$$\{ x \rightarrow y \} \circ (n \setminus \{ x \}) = \{ \}$$

theo (5.4), (2.2) và (5.8). Và ta cũng có :

$$(m \setminus \{ u \}) \circ \{ x \rightarrow u \} = \{ \}$$

theo (5.4) và (2.2)

$$\{ u \rightarrow y \} \circ \{ x \rightarrow u \} = \{ x \rightarrow y \}$$

theo (5.6)

Như vậy theo (5.2) và (2.5) :

$$(m + \{ u \rightarrow y \}) \circ (m + \{ x \rightarrow u \}) = (m \circ n) \setminus \{ x \} \cup \{ x \rightarrow y \} = (m \circ n) + \{ x \rightarrow y \}$$

Mặt khác ta có :

$$m \setminus \{ u \} \circ a = m \circ a \text{ theo (5.3)}$$

$$\{ u \rightarrow y \} \circ a = \{ \} \text{ theo (5.4)}$$

Như vậy :

$$(m + \{ u \rightarrow y \}) \circ a = m \circ a \text{ theo (2.4) và (5.3)}$$

## IV.6. Triển khai thứ hai

Mục này sẽ tối ưu cách triển khai đầu tiên đã trình bày trong mục 4 bằng cách xây dựng tập hợp L các địa chỉ tự do của D, tập hợp mà ta đã chọn tùy ý một phần tử u trong đặc tả phép toán  $\text{mod}_1$  ở (4.3).

Ý tưởng thiết kế cách triển khai thứ hai này nằm ở chỗ giữ lại trạng thái của mỗi địa chỉ của D mà địa chỉ này có thể thuộc về một (và chỉ một mà thôi) trong 4 tập hợp rời nhau như sau :

$$RN \_ RN$$

$$RA \cap RN$$

$$RN \_ RA$$

$$RA \cup RN = L$$

trong đó  $RA = \text{ran}(a)$ ,  $RN = \text{ran}(n)$

Tùy theo một địa chỉ  $d$  của  $D$  thuộc về một trong bốn tập hợp trên, ta nói trạng thái tương ứng sẽ là :

old (cũ)  $d \in \text{ran}(a)$

common (chung)  $d \in \text{ran}(a)$  và  $d \in \text{ran}(n)$

new (mới)  $d \in \text{ran}(a)$

free (tự do)  $d \notin \text{ran}(a)$  và  $d \notin \text{ran}(n)$

Khi một địa chỉ tự do được chọn, khi một thay đổi xảy ra, địa chỉ đó chuyển qua trạng thái mới ; về địa chỉ quá tải trong bảng  $n$ , nếu địa chỉ đó không phân chia bên trong bảng  $n$  (nghĩa là nếu hàm  $n$  là đơn ánh và điều này được giả thiết là luôn đúng), khi đó, địa chỉ sẽ trở nên tự do nếu đang ở trạng thái mới hoặc chuyển sang trạng thái cũ nếu đang ở trạng thái chung.

Khi một phép hợp thức hóa hay khởi động lại các địa chỉ tự do hay chung vẫn như cũ. Các địa chỉ mới chuyển thành tự do khi một sự khởi động lại và là trường hợp chung khi hợp thức hóa.

Cuối cùng, các địa chỉ cũ chuyển thành tự do khi hợp thức hóa và trở thành chung khi khởi động lại. Chú ý rằng các địa chỉ cũ không liên quan đến sự thay đổi. Sơ đồ dưới đây tóm tắt một cách phi hình thức những chuyển đổi khác nhau này.

Mục đích để hình thức hóa phương pháp này, ta đưa vào một biến mới  $s$  định nghĩa trạng thái của mỗi địa chỉ của  $D$ .

$$s \in D \rightarrow \{\text{fr}, \text{nw}, \text{cm}, \text{ol}\} \quad (6.1)$$

Ta có bất biến sau đây :

$$RA - RN = \text{adr}(\text{ol})$$

$$RA \cap RN = \text{adr}(\text{cm}) \quad (6.2)$$

$$RN - RA = \text{adr}(\text{nw})$$

$$RA \cup RN = \text{adr}(\text{fr})$$

### HÌNH VẼ

Trong đó :

$$RA = \text{ran}(a)$$

$$RN = \text{ran}(n)$$

$$\text{adr}(z) = \{x \in D / s(x) = Z\}$$

với  $Z \in \{\text{fr}, \text{nw}, \text{cm}, \text{ol}\}$

Cuối cùng bất biến thứ ba chỉ rõ rằng cả hai hàm  $n$  và  $a$  đều đơn ánh, nghĩa là hai địa chỉ của  $A$  phân biệt sẽ luôn luôn tương ứng với các địa chỉ của  $D$  phân biệt qua các hàm này.

Tập hợp các hàm từ  $A$  vào  $D$  như vậy được ký hiệu

bởi  $A \downarrow D$  như vậy

$$n \in A \cup D$$

$$a \in A \cup D$$

Bây giờ sẽ là định nghĩa ba hàm chuyển tiếp lần lượt là  $f$ ,  $g$  và  $h$  sử dụng khi thay đổi (cho các địa chỉ của  $D$  liên quan), khởi động lại thay cho hợp thức hóa (cho mọi địa chỉ của  $D$ ) :

$$f = \{fr \rightarrow nw,$$

$$nw \rightarrow fr, \text{ thay đổi}$$

$$cm \rightarrow ol\}$$

$$g = \{fr \rightarrow fr,$$

$$nw \rightarrow fr,$$

$$cm \rightarrow cm, \text{ khởi động lại}$$

$$ol \rightarrow cm\}$$

$$h = \{fr \rightarrow fr,$$

$$nw \rightarrow cm,$$

$$cm \rightarrow cm, \text{ hợp thức hóa}$$

$$ol \rightarrow fr\}$$

Ta có đặc tả của 3 phép toán mới  $\text{mod}_2$ ,  $\text{rst}_2$ , và  $\text{vld}_2$  xuất hiện như là các mở rộng tương ứng từ mục 4 :

$$(a, n, m, s) \text{ mod}_2 (x, y) (a', n', m', s')$$

$$(a, n, m) \text{ mod}_1 (x, y) (a', n', m') \quad (6.5)$$

$$s' = s + \{u \rightarrow f(s(u)), v \rightarrow f(s(v))\}$$

xem (4.3)

trong đó :

$$L = \{ Z \in D \mid s(z) = fr \}$$

$$u \in L$$

$$v = n(x)$$

$$(a, n, m, s) \text{ rst}_2 (a', n', m', s') \quad (6.6)$$

$$s' = gos \quad \text{xem (4.6)}$$

$$(a, n, m, s) \text{ vld}_2 (a', n', m', s') \quad (6.7)$$

$$(a, n, m) \text{ vld}_1 (a', n', m') \quad \text{xem (4.7)}$$

$$s' = hos$$

Sau đây là một quá trình chuyển đổi của hệ thống

CHÙA

Chúng minh



Mặc dù trong mục trước ta đã kiểm chứng kỹ lưỡng đặc tả hệ thống và nhận được kết quả thỏa mãn, nhưng chưa đảm bảo được tính đúng đắn của đặc tả trong mọi trường hợp.

Để đi đến một kết quả tổng quát, ta cần phải chứng minh không phải cho một trường hợp đặc biệt nào đó mà phải cho các dữ liệu tượng trưng thỏa mãn những giả thiết nào đó, các phép toán đã đặc tả là phù hợp và chấp nhận được.

Việc chứng minh tính phù hợp của các phép toán đặc tả ở (6.5), (6.6) và (6.7) so với các phép toán đặc tả ở (4.3), (4.6) và (4.7) là hiển nhiên vì rằng trong cách lập các công thức thì các phép toán (4.3), (4.6) và (4.7) một cách tương ứng.

Trái lại, việc chứng minh tính chấp nhận được phức tạp hơn. Trước hết ta cần chứng minh ba nhóm định lý bất biến sau đây :

((6.1) và (6.5)) kéo theo (6.1)' (7.1)

((6.2) và (6.5)) kéo theo (6.2)' (7.2)

((6.3) và (6.5)) kéo theo (6.3)' (7.3)

((6.1) và (6.6)) kéo theo (6.1)' (7.4)

((6.2) và (6.6)) kéo theo (6.2)' (7.5)

((6.3) và (6.6)) kéo theo (6.3)' (7.6)

((6.1) và (6.7)) kéo theo (6.1)' (7.7)

((6.2) và (6.7)) kéo theo (6.2)' (7.8)

((6.3) và (6.7)) kéo theo (6.3)' (7.9)

Đối với 3 định lý ở nhóm 1, ta có thể dẫn đến các giả thiết cho các điều kiện sau đây :

$a \in A \cup D$

$n \in A \cup D$

$u \notin RN$  (7.10)

$u \notin RA$

$v \in RN$

Trong đó  $u$  và  $v$  được định nghĩa ở (6.5) và  $RA, RN$  được định nghĩa ở (6.2) chứng minh (7.1)

Một khó khăn nhỏ là hàm chuyển tiếp  $f$  được định nghĩa ở (6.4) là hàm bộ phận. Cần chứng minh rằng  $s'$  được định nghĩa đúng, nghĩa là các biểu thức  $f(s(u))$  và  $f(s(v))$  trong (6.5) có nghĩa, nói cách khác ta có :

$$s(u) \in \text{dom}(f)$$

$$s(v) \in \text{dom}(f)$$

điều này hiển nhiên vì rằng theo (7.10), ta có :

$$s(u) = fr$$

$$s(v) = \{nw, cm\}$$

và theo (6.4) ta có

$$\text{dom}(f) = \{fr, nw, cm\}$$

Để chứng minh (7.2) và (7.3) ta cần kết quả sau đây liên quan đến sự quá tải của một hàm đơn ánh thừa nhận mà không chứng minh :

$$f \in X \rightarrow Y \text{ kéo theo } f' \in X \rightarrow Y$$

$$x \in \text{dom}(f) \text{ ran}(f') = r'$$

$$y \notin Y - \text{ran}(f) \text{ } y \neq f(x)$$

Trong đó :

$$f' = f + \{x \rightarrow y\}$$

$$r' = (\text{ran}(f) - \{f(x)\}) \cup \{y\}$$

chứng minh (7.2) : Theo (7.10), (7.11) và (6.5) ta có

$$RA' = RA \text{ (vì rằng } a' = a \text{ theo (4.3))}$$

$$RN' = (RN - \{V\}) \cup \{u\} \text{ theo 7.11}$$

$$u \neq v \text{ theo 7.10}$$

HÌNH VẼ

Xảy ra hai trường hợp :

$$1/ v \in RA, \text{ nghĩa là } s(v) = cm$$

Khi đó :

$$RA' - RN' = (RA - RN) \cup \{v\}$$

$$RA' \cap RN' = (RA \cap RN) - \{v\}$$

$$RN' - RA' = (RN - RA) \cup \{u\}$$

$$RA' \cup RN' = (RA \cup RN) - \{u\}$$

HÌNH VẼ

$$2/ v \notin RA, \text{ nghĩa là } s(v) = nw$$

Khi đó :

$$RA' - RN' = RA - RN$$

$$RA' \cap RN' = RA \cap RN$$

$$RN' - RA' = ((RN - RA) - \{v\}) \cup \{v\}$$

$$RN' \cup RA' = ((RN \cup RA) - \{u\}) \cup \{v\}$$

### HÌNH VẼ

Như vậy các chuyển tiếp từ  $s(u)$  và  $s(v)$  như sau

$fr \rightarrow nw$  với  $u$

$cm \rightarrow ol$  với  $v$  trong trường hợp 1/

$nw \rightarrow fr$  với  $v$  trong trường hợp 2/

Các chuyển tiếp này tương ứng với các chuyển tiếp đã chỉ ra bởi hàm  $g$  định nghĩa ở (6.4)

## IV.7. Triển khai thực hiện lần thứ ba

Lần này, ta giả thiết rằng xảy ra các sai sót cần phải dự phòng nhờ hệ thống hợp thức hóa và khởi động lại.

Giả sử các bảng  $a$ ,  $n$ ,  $m$  và  $s$  được cài đặt trên các thiết bị nhớ khác như sau :

### HÌNH VẼ

Từ cách tổ chức này, ta muốn dự phòng các sai sót tác động lên bộ nhớ trong bằng cách khởi động lại từ đĩa. Ở đây, ta đã cài đặt các bảng  $s$  và  $s$  trong bộ nhớ với mục đích tăng tính hiệu quả của phép thay đổi bộ nhớ là nhanh nhất có thể.

Với mục đích trên, việc khởi động lại làm thay đổi bảng  $s$  từ chính nó (thực tế là  $s' = gos$  theo (6.6)) không còn có tác dụng nữa vì rằng ta giả thiết rằng bộ nhớ trung tâm là  $s$  sẽ không còn nữa.

Một giải pháp là gấp đôi bảng  $s$  lên đĩa cho mỗi lần hợp thức hóa. Xây dựng bảng mới  $t$  được tương ứng với bất biến như sau :

$$t \in D \rightarrow \{fr, cm\} \quad (8.1)$$

Chú ý rằng ta không cần mọi giá trị các trạng thái địa chỉ đĩa vì rằng  $t$  chỉ dùng để tái sinh lại bảng  $s$  khi khởi động lại, từ đó ta có bất biến bổ sung như sau :

$$\{x \in D \mid t(x) = cm\} = \text{ran}(a) \quad (8.2)$$

Ta có các đặc tả mối như sau :

$$(a, n, m, s, t) \text{ mod}_3 (x, y) (a', n', m', s', t')$$

$$(a, n, m, s) \text{ mod}_2 (x, y) (a', n', m', s') \quad (8.3)$$

$$t' = t \quad \text{xem (6.5)}$$

$$(a, n, m, s, t) \text{ rst}_3 (a', n', m', s', t')$$

$$(a, n, m) \text{ rst}_1 (a', n', m') \quad (8.4)$$

$$s' = t \quad \text{xem (4.6)}$$

$$\begin{aligned}
 t' &= t \\
 (a, n, m, s, t) &\text{vld}_3 (a', n', m', s', t') \quad (8.5) \\
 (a, n, m, s) &\text{vld}_2 (a', n', m', s') \text{ xem (6.7)} \\
 t' &= s'
 \end{aligned}$$

Để dàng chứng minh rằng cả ba đặc tả trên là phù hợp và chấp nhận được. Mặt khác ta thấy rằng phép khởi động lại tái sinh bộ nhớ trong từ đĩa và chỉ từ đĩa mà thôi.

Triển khai lần thứ tư và lần thứ năm

Ta sẽ mã hóa các giá trị fr, nw, cm và ol như hàm đơn ánh k như sau :

$$\begin{aligned}
 k &= \{(0, 0) \rightarrow \text{fr}, \\
 &(0, 1) \rightarrow \text{ol}, \\
 &(1, 0) \rightarrow \text{cm}, \quad (9.1) \\
 &(1, 1) \rightarrow \text{nw}\}
 \end{aligned}$$

Sau đó ta biểu diễn các hàm s và t nhờ ba chuỗi bit như sau :

$$\begin{aligned}
 b \in D &\rightarrow \{0, 1\} \text{ để biểu diễn } s \\
 c \in D &\rightarrow \{0, 1\} \\
 d &\rightarrow \{0, 1\} \text{ để biểu diễn } t \quad (9.2)
 \end{aligned}$$

Cuối cùng, thay đổi các biến tương ứng với các điều kiện sau :

$$\begin{aligned}
 s(x) &= k(b(x), c(x)) \\
 t(x) &= k(d(x), 0) \quad \text{với } x \in D \quad (9.3)
 \end{aligned}$$

Giả sử là phép bù (đảo ngược bit) như sau

$$0 = 1, 1 = 0 \quad (9.4)$$

Ta thấy có thể mã hóa hàm f đã định nghĩa ở (6.4) nhờ hai phép bù và hàm h cũng đã định nghĩa ở (6.4) nhờ phép sao chép và đặt về 0 tương ứng với hàm :

$$Z \in D \rightarrow \{0\} \quad (9.5)$$

Ta có 3 đặc tả mới như sau :

$$\begin{aligned}
 (a, n, m, b, c, d) &\text{mod}_4 (x, y) (a', n', m', b', c', d') \\
 (a, n, m) &\text{vld}_1 (a', n', m') \\
 b' &= b \quad (9.8) \\
 c' &= z \quad \text{xem (4.7)} \\
 d' &= b
 \end{aligned}$$

Bây giờ ta chỉ cần tóm tắt lại những gì đã làm cho đến lúc này, nghĩa là một mặt, sao chép lại các đặc tả (4.3), (4.6) và (4.7) vào bên trong của (9.6), (9.7) và (9.8), mặt khác, nhóm các bất biến (4.1), (6.1), (6.2), (6.3), (8.1), (8.2) và (9.2)

Điều này làm được bằng cách khử các biến trở thành dư thừa (chứa s và t) bởi các phép thay đổi biến (9.3).

Khi sao chép, ta thấy rằng đặc tả (4.3) chứa điều kiện trước  $L \neq \emptyset$  không dễ gì tính được. Để khắc phục nhược điểm này ta đưa vào một biến mới  $w$  là một số nguyên

$$w \in \mathbb{N} \quad (9.9)$$

$w$  chứa các phần tử của tập hợp  $L$  (cardinality)

$$w = |RA \cup RN|$$

Ta thừa nhận ngầm rằng các tập hợp  $D$  và  $A$  đều là hữu hạn. Khi hợp thức hóa và khởi động lại, bộ đếm  $w$  được khởi tạo giá trị  $|D| - |A|$  (vì rằng  $a$  và  $n$  đều đơn ánh) là một hằng số dương của hệ thống. Khi có sự thay đổi, bộ đếm  $w$  tăng lên khi và chỉ khi địa chỉ  $v$ , quá tải trong  $n$ , đang ở trạng thái  $cm$ , nghĩa là nếu  $b(v) = 1$  và nếu  $c(v) = 0$

Với sự mở rộng mới này, bất biến của hệ thống lúc này sẽ là :

$$a \in A \rightarrow D \quad (6.3)$$

$$n \in A \rightarrow D \quad (6.3)$$

$$m \in D \rightarrow V \quad (4.1)$$

$$b \in D \rightarrow \{0, 1\} \quad (9.2)$$

$$c \in D \rightarrow \{0, 1\} \quad (9.2)$$

$$d \in D \rightarrow \{0, 1\} \quad (9.2)$$

$$d \in D \rightarrow \{0, 1\} \quad (9.2)$$

$$w \in \mathbb{N} \quad (9.9)$$

$$RA - RN = \{x \in D / b(x) = 0 \text{ và } c(x) = 1\} \quad (6.2)$$

$$RA \cap RN = \{x \in D / b(x) = 1 \text{ và } c(x) = 0\} \quad (6.2)$$

$$RN - RA = \{x \in D / b(x) = 1 \text{ và } c(x) = 1\} \quad (6.2)$$

$$RA \cup RN = \{x \in D / b(x) = 0 \text{ và } c(x) = 0\} \quad (6.2)$$

$$RA = \{x \in D / d(x) = 1\} \quad (9.2)$$

$$W = |RA \cup RN| \quad (9.10)$$

Trong đó

$$RA = \text{ran}(a)$$

$$RN = \text{ran}(n)$$

Sau đây là các đặc tả được tóm tắt bằng cách thay thế các danh sách dài các biến bởi hai biến trạng thái  $state$  và trạng thái có dấu nháy ( $'$ )  $state'$

$$state \text{ mod}_5 (x, y) \text{ state}'$$

$$x \in A$$

$$y \in V$$

$$w \neq 0$$

$$a' = a$$

$$n' = n + \{x \rightarrow u\}$$

$$\begin{aligned}
m' &= m + \{u \rightarrow y\} \\
b' &= b + \{u \rightarrow b(u) \vee v \rightarrow b(v)\} \quad (9.12) \\
c' &= c + \{u \rightarrow c(u), v \rightarrow c(v)\} \\
d' &= d \\
(b(v) = 1 \ \& \ c(v) = 0) &\Rightarrow w' = w - 1
\end{aligned}$$

Trong đó

$$u \in \{Z \in D \mid b(z) = 0 \ \& \ c(z) = 0\}$$

$$v = n(x)$$

state  $\text{rst}_5$  state'

$$a' = a$$

$$n' = a$$

$$m' = m$$

$$b' = d \quad (9.13)$$

$$c' = z$$

$$d' = d$$

$$w' = |D| - |A|$$

state  $\text{vdl}_5$  state'

$$a' = n$$

$$n' = n$$

$$m' = m$$

$$b' = b \quad (9.14)$$

$$c' = z$$

$$d' = b$$

$$w' = |D| - |A|$$

Trong đó  $Z \in D \rightarrow \{0\}$

Sau đây là quá trình biến đổi cụ thể

## IV.8. Đặc tả làm gì ?

Sau đây ta sẽ trả lời câu hỏi về mục đích (cho ai, cho cái gì) của các công việc mà ta đã làm. Vai trò đầu tiên của một đặc tả là cho phép mở ra các tranh luận về đề tài đặc tả đề tài đặc tả để cập đến. Thực tế, khác với một chương trình, một đặc không phải viết ra để máy tính có thể hiểu được mà để cho những NSD có thể hiểu và tin tưởng vào tính đúng đắn của nội dung đã đặc tả.

Từ đặc tả lúc đầu ở mục 3 cho đến các đặc tả tiếp theo ở các mục 4, 6, 8 và 8, ta đã sử dụng các ràng buộc mỗi lúc một mang tính thực tiễn. Ta đã khẳng định được tính đúng đắn của đặc tả bởi các chứng minh định lý phù hợp và chấp nhận được : bảo toàn tính bất biến.

Bây giờ vấn đề là sử dụng các đặc tả để lập trình. Trong mục này, ta sẽ lập trình cho các trường hợp đặc tả (9.12), (9.13) và (9.14), bằng cách sử dụng ngôn

ngữ giả Pascal. Ta đưa vào các quy tắc để tiết lập mối liên hệ giữa đặc tả và lập trình.

Quy tắc 1 :

Khi một đặc tả chứa các điều kiện trước, chương trình tương ứng sẽ là một hàm. Theo nghĩa của Pascal, mỗi giá trị sai của điều kiện trước sẽ trả về biến trạng thái state một giá trị phân biệt.

Chương trình tương ứng với đặc tả (9.12) là như sau :

```
if not (x in A) then
state := bad - x
else if not (y in v) then
state := bad - y
else if w = 0 then
state := no more place
else begin
State := OK ;
Modification {gọi thủ tục}
end ;
```

Quy tắc 2 :

Khi một đặc tả chứa các biến phụ, ta có thể mở mọi thủ tục chứa các biến này như là các biến cục bộ. Thủ tục này bắt đầu bởi các lệnh khởi động

Thủ tục Modification như sau :

```
procedure Modification ;
var u, v : D
begin
u := 1 ; {chọn u là địa chỉ bé nhất của L}
while (b (u) ≠ 0) or (c (u) ≠ 0) do
u := u + 1 ; (10.2)
v := n (x)
{tiếp tục thân thủ tục}
end ;
```

Quy tắc 3 :

Các điều kiện sau khác nhau nếu có dạng  $a' = \dots$  (trong đó dấu ba chấm ... chỉ định một biểu thức khởi động chứa các biến có đánh dấu nháy), thì có thể được chuyển thành phép gán qua các quy tắc hỗ trợ như sau :

- Loại bỏ các điều kiện sau dạng đẳng thức.

Ví dụ :  $d' = d$

- Thay thế dấu = bởi dấu gán bằng :=

- Thực hiện phép tối ưu khi một hàm là quá tải

- Loại bỏ các dấu nhảy '
- Thay thế một điều kiện sau bởi cấu trúc điều kiện if... else :

Các điều kiện sau không bị loại bỏ của đặc tả (9.12) như sau :

```

if not (x in A) then
    state := bad - x
else if not (y in V) then
    state := bad - y(10.1)
else if w = 0 then
    state := no-more-place
else begin
    state := O.K ;
    Modication {gọi giá trị thủ tục}
end ;

```

Quy tắc 2 :

Khi một đặc tả chứa các biến phụ, ta có thể mở một thủ tục chứa các biến này như là các biến cục bộ. Thủ tục này bắt đầu bởi các lệnh khởi động.

Thủ tục Modication như sau :

```

Procedure Modication ;
var u, v : D
begin
    u := 1 ; {chọn u là địa chỉ bé nhất của L}
    while (b(u) ≠ 0) or (c(u) ≠ 0) do
        u := u + 1 ; (10.2)
        v := n(x)
    {tiếp tục thân thủ tục}
end ;

```

Quy tắc 3 :

Các điều kiện sau khác nhau nếu đều có dạng a' = ... (trong đó dấu chấm ... chỉ định một biểu thức không chứa các biến có đánh dấu nhảy), thì có thể được chuyển thành phép gán qua các quy tắc hỗ trợ như sau :

- Loại bỏ các điều kiện sau dạng đẳng thức.

Ví dụ : d' = d

- Thay thế dấu = bởi dấu gán bằng :=
- thực hiện phép tối ưu khi một hàm là quá tải.
- Loại bỏ các dấu nhảy '
- Thay thế một điều kiện sau bởi cấu trúc điều kiện if ... else :

Các điều kiện sau không loại bỏ của đặc tả (9.12) như sau :

```

n(x) := u ;
m(u) := y ; (10.3)

```



$b(u) := b(u) ; b(v) := b(v) ;$

$c(u) := c(u) ; c(v) = c(v) ;$

if  $(b(v) = 1)$  and  $(c(v) = 0)$  then  $w := w - 1$

Quy tắc 4 :

Một cách hệ thống các hệ chương trình đã viết được bởi các quy tắc đảm bảo tính "song song" đưa vào từ đặc tả. Từ các đoạn chương trình (10.1), (10.2) và (10.3) ta nhận được công thức đầy đủ hơn như sau :

## Một số đề thi

### I. Đặc tả (Specification)

1. Cho ma trận vuông  $A$  cấp  $n \times n$ . Viết đặc tả thể hiện : **a)** Mỗi phần tử trên đường chéo chính là phần tử lớn nhất trên cùng hàng đi qua phần tử đó. **b)** Mỗi phần tử trên đường chéo phụ là phần tử nhỏ nhất trên cùng cột đi qua phần tử đó.
2. Một xâu (string)  $w$  được gọi là *đối xứng* (palindrome) nếu  $w = w^R$  hay đọc xuôi ngược đọc ngược đều như nhau ( $w^R$  là xâu đảo ngược của  $w$ ). Ví dụ các xâu *omo*, *mannam*, ... đều là *đối xứng*. Viết đặc tả thể hiện các xâu *đối xứng*.
3. Đa thức cấp  $n$  được viết dưới dạng Toán học là :

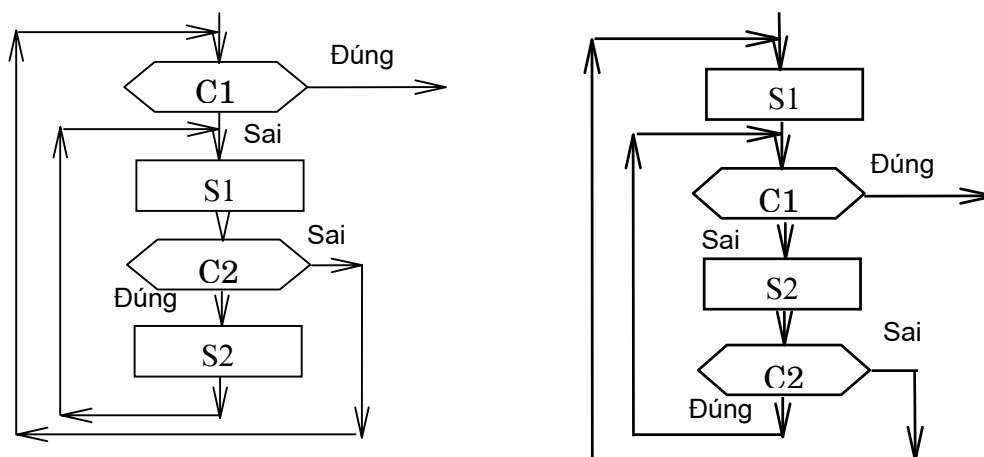
$$P_n(x) = a_0 + a_1x^1 + a_2x^2 + \dots + a_nx^n$$

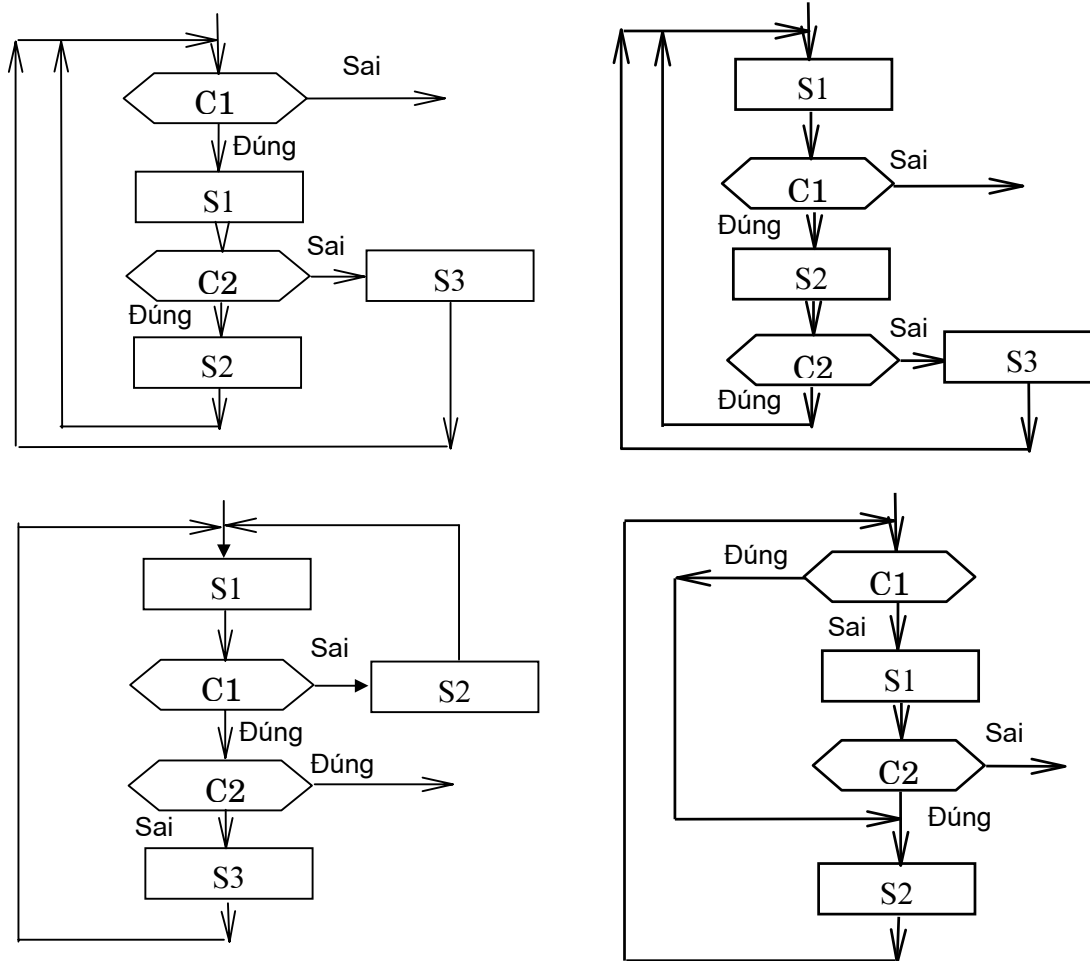
Viết đặc tả thể hiện phép cộng, so sánh hai đa thức  $P_n(x)$  và  $Q_m(x)$ , nhân đa thức với một hằng số  $a \times P_n(x)$  và nhân hai đa thức  $P_n(x) \times Q_m(x)$ .

4. Các phân số (hay số hữu tỷ) được biểu diễn bởi danh sách  $(n, d)$ , với  $n$  là tử số và  $d$  là mẫu số, là những số nguyên ( $d \neq 0$ ). Viết đặc tả xây dựng các hàm xử lý phân số: rút gọn, trừ, chia và so sánh hai phân số, cộng, nhân hai phân số và chuyển đổi phân số thành số thực.

### II. Lập trình cấu trúc (Structured programming)

1. Viết lệnh bằng giả ngữ (phỏng Pascal), chỉ sử dụng tối đa ba cấu trúc tuần tự, điều kiện if và lặp (while-repeat), theo các sơ đồ khối dưới đây :





2. Viết lệnh bằng giả ngữ (phỏng Pascal), chỉ sử dụng tối đa ba cấu trúc tuần tự, điều kiện if và lặp (while□repeat), theo sơ đồ khối dưới đây :

### III. Thử nghiệm chương trình (Testing)

Giả sử các chương trình đã cho ở phần trên đây là các đơn thể gọi đến các đơn thể con S1, S2 và S3). Trình bày một phương pháp để thử nghiệm đơn thể gọi.