Các kiểu dữ liệu

Sự tiến hóa của những ngôn ngữ lập trình hiện đại được kết hợp chặt chẽ với sự phát triển (và chính thức hóa) của các khái niệm về kiểu dữ liệu. Ở cấp độ ngôn ngữ máy, tất cả các giá trị không định kiểu (có nghĩa là, chỉ là các mẫu bit). Tuy nhiên, những lập trình viên ngôn ngữ Assembler, thường nhận ra sự khác biệt cơ bản giữa các địa chỉ (xem là định vị) và dữ liệu (được coi là tuyệt đối).Do đó, họ nhận ra rằng sự kết hợp một số các địa chỉ và dữ liệu (ví dụ, tổng của hai địa chỉ) được định nghĩa đầy đủ.

Tuy nhiên ,quan điểm về kiểu của ngôn ngữ Assembler là không hoàn thiện, bởi vì nó xem kiểu như là một thuộc tính của một dữ liêu thay vì một thuộc tính của ô có chứa dữ liệu. Đó là, có hay không một thao tác luôn có ý nghĩa có thể được xác định chỉ trong thời gian chạy khi các giá trị toán hạng thực tế là có sẵn. Trình tiện ích Assembler có thể sẽ nhận ra sự vô hiệu của một biểu thức mà khi cộng hai nhãn, trong khi nó sẽ chấp nhận một chuỗi mã mà tính toán chính xác phép cộng đó! Điểm yếu này đã dẫn đến sự ra đời của cấu trúc được gán thẻ bao gồm (trong thời gian chạy) thông tin kiểu với dữ liệu. Cấu trúc này có thể phát hiện lỗi các nhãn được thêm, do sự thêm các lệnh có thể phát hiện ra các toán hạng của nó là hai địa chỉ. Thật không may, các thông tin kiểu bao gồm dữ liệu thường giới hạn bởi kiến trúc của máy. Lập trình viên- các kiểu dữ liệu được khai báo không thể nhận được sự kiểm tra chính xác tự động như là các kiểu dữ liệu có sẵn.

FORTRAN và sau đó là ngôn ngữ bậc cao được phát triển trên ngôn ngữ assaembler bằng cách kết hợp các thông tin về kiểu với các vị trí đang nắm giữ dữ liệu hơn là bản thân dữ liệu. Nói chung, ngôn ngữ lập trình kết hợp thông tin về kiểu với tên kiểu nó có thể là biến hoặc các tham số hình thức. Khi một thuộc tính như kiểu được kết hợp với một tên kiểu, chúng tôi nói rằng các tên kiểu ràng buộc với thuộc tính. Xác định kiểu diễn ra tại thời gian biên dịch thường được gọi là kiểu tĩnh, và diễn ra trong thời gian chạy chương trình được gọi là kiểu động. Ngôn ngữ kiểu tĩnh là những ràng buộc để xác định các kiểu tại thời gian biên dịch. Kể từ khi các kiểu được biết đến tại thời gian biên dịch, trình biên dịch có thể phát hiện một loạt các lỗi kiểu (ví dụ, một nỗ lực để nhân hai biến Boolean).

Ngôn ngữ bậc cao trước Pascal thường bị giới hạn khái niệm về các kiểu dữ liệu vì chúng bị giới hạn bởi phần cứng của máy (số nguyên, số thực, gấp đôi chính xác số nguyên và số thực, và ngăn chặn các vị trí tiếp giáp nhau). Hai đối tượng đã có kiểu khác nhau nếu có các đoạn mã khác nhau thao tác chúng. Pascal và các ngôn ngữ sau đó đã lấy một cách tiếp cận khá khác nhau, dựa trên khái niệm của các kiểu dữ liệu trừu tượng. Trong Pascal, các lập trình viên có thể tạo ra hai đối tượng có kiểu khác nhau ngay cả khi chúng có cùng một cách biểu diễn và sử dụng cùng một đoạn mã. Các quy tắc về kiểu đã chuyển từ tập trung vào những gì có ý nghĩa vào các máy tính để có ý nghĩa với các lập trình viên.

Khái niệm về kiểu dữ liệu : kiểu dữ liệu không chỉ là một tập hợp các đối tượng dữ liệu mà còn là một tập hợp các phép toán có thể thao tác trên các đối tượng dữ liệu này.

Ngày nay, khi ta nói đến kiểu dữ liệu thực chất là nói đến kiểu dữ liệu trừu tượng. Kiểu dữ liệu trừu tượng là một tập hợp các đối tượng dữ liệu và tập hợp các phép toán, thao tác trên các đối tượng dữ liệu đó..

Kiểu dữ liệu trừu tượng có thể được định nghĩa bởi ngôn ngữ hoặc do người lập trình định nghĩa.

Ví dụ: kiểu dữ liệu trừu tượng do ngôn ngữ định nghĩa: kiểu integer trong pascal: tập các đối tượng dữ liệu là tập các số nguyên từ -32768 đến 32767; tập hợp các phép toán bao gồm các phép toán một ngôi (+,-), các phép toán 2 ngôi (+, -,*, div, mod), các phép toán quan hệ (<, <=, =, …).

Ví dụ: kiểu lập trình do người lập trình định nghĩa: ngăn xếp, hàng đợi, danh sách…..

1. Ngôn ngữ có kiểu động

Ngôn ngữ có kiểu động: không kiểm tra kiểu trong quá trình biên dịch mà kiểm tra kiểu trong quá trình thực thi chương trình.

Ví dụ: funtion log(() nhận tham số đầu vào là một số và trong chwong trình gọi hàm này như sau: log (“zoo”) truyền vào một chuỗi. Nếu là kiểm tra kiểu tĩnh thì, trình dịch sẽ đưa ra 1 thông báo “này chờ chút, không thể truyền một chuỗi vào hàm vì nó cần một số” và chương trình sẽ không được dịch. Còn với kiểu động thì chương trình vẫn dịch tốt nhưng khi thức thi sẽ báo lỗi **runtime.**

➔Như vậy các ngôn ngữ định kiểu mạnh thường chạy chậm hơn các ngôn ngữ định kiểu tĩnh vì kiểu phải được kiểm tra lúc chạy chương trình.

Nó có thể bỏ qua ràng buộc xác định kiểu cho đến khi chạy chương trình, dẫn tới ngôn ngữ có kiểu động. Biên dịch các ngôn ngữ (như SNOBOL, APL, và awk) thường chỉ gán các kiểu tại thời gian chạy. Những ngôn ngữ này không khai báo kiểu; định danh kiểu có thể tự động thay đổi. Đây điểm khác nhau của ngôn ngữ bậc cao với các ngôn ngữ có ít kiểu, như Bliss hay BCPL, mà chỉ có một loại kiểu dữ liệu, ô hoặc từ.

Bỏ qua ràng buộc về định danh một kiểu để có ý nghĩa với chi phí hiệu quả,! trong khi chạy mã chương trình phải xác định kiểu của biến để thao tác giá trị thích hợp. Ví dụ, trong ngôn ngữ kiểu động, các mảng không cần phải đồng nhất. Như một ví dụ về mất hiệu quả, ngay cả trong ngôn ngữ có kiểu tĩnh, các giá trị của các kiểu lựa chọn yêu cầu một số thời gian chạy kiểm tra để đảm bảo rằng các biến thức dự kiến sẽ có mặt

2. Kiểu mạnh

Một trong những thành tựu chủ yếu của Pascal là sự nhấn mạnh nó dựa vào định nghĩa các kiểu dữ liệu. Nó đã xem việc tạo các kiểu của các lập trình viên- khai báo các kiểu dữ liệu như là một phần của phát triển chương trình. Pascal đã đưa ra khái niệm về kiểu mạnh để bảo vệ các lập trình viên tránh khỏi những lỗi về kiểu.

Một ngôn ngữ kiểu mạnh cung cấp các quy tắc cho phép các trình biên dịch xác định kiểu của từng giá trị (kiểu của biến và biểu thức) .phép gán và truyền tham số hình thức của các kiểu tương đương là không hợp lệ, ngoại trừ một số trường hợp chuyển đổi kiểu tự động. Điều cơ bản là

các kiểu khác nhau đại diện cho các thuộc tính khác nhau, do đó, chúng phải được kiểm tra cẩn thận, rõ ràng chính xác.

Ví dụ:

>> x = “hello world”

>> Print (x) – hello world

>> x=10

=> => print (x) – 10

Biến x không lưu trữ giá trị của biến mà chỉ lưu trữ một reference đến một đối tượng trong vùng nhớ mà thôi. Bì thế khi gán x=”hello world” thì x tham chiếu đến một đối tượng string, x=10 thì x tham chiếu đến một đối tượng có kiểu integer. Tại hai vùng nhớ vẫn tồn tại 2 đối tượng: hello world và 10. Tại vùng nhớ lưu trữ đối tượng 10, ta không thể lưu trữ một string vào đó để thế chỗ. ➔ với kiểu strongtyped : tại một vùng nhớ chỉ lưu trữ được một kiểu giá trị dữ liệu mà thôi.

*** Các kiểu mạnh hay yếu, kiểu tĩnh hay động nó phụ thuộc vào các loại ngôn ngữ lập trình. ví dụ : ngôn ngữ pascal định kiểu mạnh, ngôn ngữ định kiểu tĩnh : Java, C++, C..***

3.Sự tương đương kiểu

Ý nghĩa của sự tương đương cấu trúc:

- Phép gán
- Truyền tham số hình thức trong lời gọi chương trình con
- Dùng kiểu này như một kiểu khác: ví dụ: w:array [1..5] of real ta có thể dùng biến w như kiểu integer mà không bị lỗi kiểu.

Ví dụ:

```
TYPE Vect1 = ARRAY[1..10] OF REAL;
Vect2 = ARRAY[1..10] OF REAL;
VAR x,z : Vect1;
y : Vect2;
PROCEDURE Sub(a:Vect1);
.....
END; { Sub }
BEGIN { Chương trình chính }
x := y;          // phép gán.
Sub(y);          //truyền tham số trong gọi chương trình con.
......
END.
```

Khái niệm của kiểu mạnh dựa trên một định nghĩa chính xác khi mà các kiểu là tương đương. Điều ngạc nhiên là định nghĩa ban đầu của pascal không có dịnh nghĩa của các kiểu tương đương. Vấn dề này có thể được trình bày bởi yêu cầu kiểu kiểm tra xem T1 và T2 có tương đương hay không ở trong ví dụ 3.1

Ví dụ 3.1:

Type T1,T2 =array [1..10] of real;

T3 =array [1..10] of real;

Cấu trúc tương đương chỉ ra rằng 2 kiểu là tương đương nếu, sau khi tất cả các định danh kiểu được thay thế bởi định nghĩa của chúng và có cùng một cấu trúc. Đây là định nghĩa đệ quy bởi vì những định nghĩa về định danh kiểu bản thân chúng phải bao gồm các định danh kiểu. Điều này là mơ hồ vì nó đưa ra cái mà có cấu trúc giống nhau. Mọi người đồng ý rằng T1,T2 và T3 có cấu trúc tương đương. Tuy nhiên, không ai đồng ý rằng bản ghi đòi hỏi tên trường đồng nhất để hoặc mảng yêu cầu phạm vi chỉ số giống hệt nhau để có cấu trúc giống nhau. Trong ví dụ 3.2 T4 , T5, T6 đồng nhất với T1 trong một vài ngôn ngữ nhưng không phải là tất cả.

Ví dụ 3.2:

Type

T4 = array [2 . . 11] of real ------ same length

T4 = array [2 .. 10] of real ------ compatible index type

T4 = array [blue .. red] of real ------ incompatible index type

Thử nghiệm câú trúc tương đương không phải luôn là quan trọng, bởi vì các kiểu đệ quy có thể thực hiện được điều đó. trong ví dụ 3.3, các kiểu TA và TB là những cấu trúc tương đương, như là TC và TD, mặc dù sự mở rộng của chúng là vô hạn.

Type

```
TA = pointer to TA;
TB = pointer to TB;
TC =
        Record
                Data : integer;
                Next : pointer to TC;
        End;
TD =
        Record
                Data : integer;
                Next : pointer to TD;
        End;
```

Trái lại với cấu trúc tương đương, sự tương đương về tên phát biểu rằng 2 biến là cùng kiểu nếu chúng được khai báo cùng tên kiểu, như là integer hoặc một vài kiểu đã khai báo. Khi một biến được khai báo sử dụng một hàm tạo kiểu (đó là một biểu thức trả về một kiểu), kiểu của nó được cấp cho một tên gọi mới nội tại, với mục đích về sự tương đương tên gọi. Những phương thức khởi tạo kiểu bao gồm những từ khoá array, record và pointer to. Bởi vậy, sự tương đương về kiểu nói rằng T1 và T3 ở trên là khác nhau, giống như TA và TB. Có giải thích khác nhau có thể có khi nhiều biến được khai báo bằng cách sử dụng một hàm tạo kiểu duy nhất, chẳng hạn như T1 và T2 ở trên. Ada khá chặt chẽ; Nó gọi T1 và T2 là khác nhau. Các chuẩn hiện tại cho Pascal dễ dàng hơn; nó gọi

T1 và T2 là giống nhau. Hình thức của sự tương đương tên gọi này cũng được gọi là sự tương đương khai báo.

Sự tương đương tên gọi dường như là thiết kế tốt hơn bởi vì thực tế chỉ có 2 kiểu dữ liêu chia sẻ cùng một cấu trúc không có nghĩa là chúng đại diện cho cùng một trừu tượng . T1 có thể đại diện cho 10 thành viên của Milwaukee Brewers, trong khi T3 có thể đại diện cho điểm trung bình của 10 học sinh trong một khóa học ngôn ngữ lập trình tiên tiến. Với giải thích này, chúng ta chắc chắn không muốn T1 và T3 được coi như la tương đương.

Tương đương tên có một điểm yếu là khi một kiểu không có tên như trong khai báo trực tiếp:

VAR w : ARRAY[1..10] OF REAL;

Biến w có kiểu riêng nhưng là kiểu không có tên. Như vậy w không thể được dùng như là một đối số cho một phép toán mà phép toán đó đòi hỏi một đối số của một kiểu có tên

Tuy nhiên, có những lý do tốt để sử dụng cấu trúc tương đương, mặc dù những kiểu không liên quan cũng có thể ngẫu nhiên trở thành tương đương. Những ứng dụng viết ra giá trị của chúng và cố gắng đọc chúng vào sau (có thể dưới sự kiểm soát của một chương trình khác) xứng đáng cùng một loại kiểu an toàn sở hữu bởi các chương trình mà chỉ thao tác các giá trị nội bộ. Modula-2+, được sử dụng tên tương đương, kết quả đầu ra cả tên kiểu và cấu truc của kiểu cho mỗi giá trị ngăn ngừa độc giả vô tình sử dụng tên giống nhau với nghĩa khác nhau. Chưa xác định kiểu được gán một tên ở bên trong. Những lỗi sai sót xuất hiện nếu lập trình

viên chuyển mã về, bởi vì trình biên dịch tạo ra tên nội bộ khác cho một kiểu vô danh. Modula-3, mặt khác, sử dụng cấu trúc tương đương. Nó đưa ra cấu trúc của kiểu (nhưng không phải là tên của nó) với mỗi giá trị đưa ra. Không có nguy cơ là sắp xếp lại chương trình sẽ dẫn đến sự không tương thích kiểu dữ liệu được viết bởi một phiên bản trước của chương trình.

Ví dụ:

```
TYPE
    Meters = INTEGER;

    Liters = INTEGER;

VAR     Len : Meters;

        `Vol : Liters;
```

Các biến Len và Vol có kiểu tương đương cấu trúc và do đó một lỗi như phép cộng Len + Vol sẽ không được tìm thấy bởi phép kiểm tra kiểu tĩnh. Khi có nhiều lập trình viên làm việc chung trong một chương trình thì tương đương kiểu không cố ý có thể gây nên các lỗi rất nghiêm trọng như trong ví dụ nói trên.

Một ngôn ngữ có thể cho phép chuyển nhượng(chỉ định) mặc dù kiểu của biểu thức và kiểu biến đích (nói đến) không phải là tương đương, chúng chỉ cần được chỉ định tương thích. Ví dụ, dưới cái tên

tương đương, hai kiểu mảng có thể có cấu trúc tương tự nhưng được tương đương bởi vì chúng được tạo ra bởi các trường hợp khác nhau của các nhà xây dựng kiểu mảng. Tuy nhiên, ngôn ngữ có thể cho phép chuyển nhượng(chỉ định) nếu các kiểu được đóng (kết thúc) đủ, ví dụ, nếu chúng có cấu trúc tương đương. Trong một đặc điểm(đặc trưng) tương tự, hai loại có thể tương thích đối với bất kỳ hoạt động(thao tác) nào, như là thêm vào, mặc dù chúng không phải là kiểu tương đương. Nó thường là phản đối (không phân minh) hay không để nói một ngôn ngữ sử dụng tên tương đương nhưng có những quy định lỏng lẻo để tương thích hoặc để nói rằng nó sử dụng cấu trúc tương đương. Tôi sẽ tránh được việc sử dụng "tương thích" và chỉ cần nói về tính tương đương.

Quy tắc của Modula-3 để xác định (quyết định) khi hai kiểu có cấu trúc tương đương là khá phức tạp. Nếu mỗi giá trị của một kiểu này là một giá trị của kiểu thứ hai sau đó kiểu đầu tiên được coi là "kiểu phụ"(kiểu con) của kiểu thứ hai. Ví dụ, một bản ghi kiểu TypeA là một kiểu con của một bản ghi kiểu TypeB chỉ khi các trường của chúng có tên giống nhau và trình tự giống nhau, và tất cả các kiểu của các trường của TypeA là những kiểu con của bản sao của chúng trong TypeB. Một kiểu mảng TypeA là một phân (kiểu phụ)nhóm của một loại mảng TypeB nếu chúng có cùng một số kích thước kích về size (mặc dù phạm vi của các chỉ số có thể khác nhau) và chỉ số và các kiểu thành phần giống nhau. //Ngoài ra còn có các quy tắc cho các mối quan hệ kiểu phụ giữa thủ tục và các kiểu con trỏ. Nếu hai kiểu này là kiểu con(kiểu phụ) của nhau, chúng là tương đương. Yêu cầu chuyển nhượng(chỉ định) giá trị được gán được là một kiểu phụ(con) của biến đích. Sau khi Pascal trở nên phổ

biến, một điểm yếu trong hệ thống kiểu của nó trở nên rõ ràng. Ví dụ, với các mã trong hình 3.4

Type Natural = 0 .. maxint;

bạn có thể mong đợi các số tự nhiên (là một dãy các số nguyên) là tương đương với số nguyên, do đó số tự nhiên và số nguyên có thể được thêm vào hoặc được gán. Mặt khác, với mã của hình 3.5,

```
type
    feet   = 0 .. maxint;
    meters = 0 .. maxint;
```

bạn có thể mong chờ feet và mét được inequivalent. Hóa ra (hiện có)trong Pascal mà các miền phụ của một kiểu hiện có (có) (hoặc một kiểu định danh được định nghĩa như là kiểu định danh khác) là tương đương (hạn chế phạm vi có thể). Nhưng tôi không muốn feet và m là tương đương.

Người phát triển Pascal (đặc biệt là Ada) đã cố gắng để khái quát các quy tắc kiểu để cho phép các kiểu có nguồn gốc từ một kiểu hiện có để được coi như là tương đương. Trong nhiều ngôn ngữ, có ngôn ngữ có thể khai báo một kiểu là một kiểu phụ của một kiểu hiện có, trong trường hợp này kiểu phụ và kiểu ban đầu là kiểu tương đương. Một ngôn ngữ khác cũng có thể khai báo một kiểu xuất phát từ một kiểu hiện có, trong trường hợp các kiểu có nguồn gốc và bản gốc không phải là kiểu tương đương. Để bổ sung cho feet và meter như là tương đương, bởi vậy tôi có thể tạo ra các kiểu như sau :

Ví dụ: 3.6

```
Type                                                    1
    Feet =derived integer range 0.. maxint;             2
    Meters =derived integer range 0.. maxint;           3
Variable                                                4
    Imperial_length :feet;                              5
    Matric_length     :meters;                          6
Begin                                                   7
    Metric_length:=metric_length *2;                    8
End;                                                    9
```

Để đảm bảo rằng các giá trị của một kiểu có nguồn gốc được lưu trữ bởi một chương trình và được đọc bởi kiểu của chúng, Modula-3 quy từng kiểu có nguồn gốc với một chuỗi chữ. giá trị nhãn hiệu (branded)chỉ có thể được đọc vào các biến với cùng một (same brand)nhãn hiệu. Nói cách khác, các lập trình viên có thể kiểm soát được các kiểu có nguồn gốc được coi là có cấu trúc tương đương với nhau.

Có một vấn đề nhỏ trong dòng 8 trong hình 3.6. phép * được định nghĩa trên integer và real, nhưng tôi cố ý tạo cho meters một kiểu mới khác biệt với số nguyên. Tương tự, 2 là một kiểu integer, không phải meters. Ada giải quyết vấn đề này bằng chồng toán tử, thủ tục, và các chữ kết hợp với một kiểu dẫn xuất. Đó là, khi meters đã được tạo ra, một tập mới của thao tác số học và thủ tục (như sqrt) được tạo ra để lấy giá trị của kiểu meters. Tương tự, các chữ số nguyên cũng được thừa nhận như các chữ meters. Biểu thức ở dòng 8 là hợp lệ, nhưng metric_length * rial_length liên quan tới không phù hợp kiểu (3-mismatch)

Trình biên dịch xác định phiên bản của chồng toán tử, thủ tục, và chữ để sử dụng. Trực giác, nó sẽ thử kết hợp tất cả các khả năng có thể của sự thông dịch, và nếu có một sự thoả mãn chính xác tất cả các quy tắc về kiểu , biểu thức là hợp lệ và được định nghĩa tốt. Đương nhiên, một trình biên dịch thông minh sẽ không thử tất cả các khả năng có thể, con số có thể tăng theo cấp số nhân trong chiều dài của biểu thức.Thay vào đó, trình biên dịch xây dựng một tập các cây con, mỗi cây thay thế cho một khả năng thông dịch quá tải. Khi gốc của cây biểu thức được đạt tới, hoặc là một giải pháp overload duy nhất đã được tìm thấy, hoặc trình biên dịch biết rằng không có giải pháp duy nhất có thể [Baker 82]. (Nếu không có thủ tục quá tải thích hợp có thể được tìm thấy, nó vẫn có thể ép các kiểu của các tham số thực sự để kiểu được chấp nhận bởi một thủ tục khai báo.

Tuy nhiên, ép kiểu thường gây ngạc nhiên cho các lập trình viên và dẫn đến nhầm lẫn.)

Khái niệm về kiểu con có thể được khái quát bằng cách cho phép mở rộng và cắt giảm các kiểu hiện có [Paaki 90]. Ví dụ, kiểu mảng có thể được mở rộng bằng cách tăng thêm chỉ mục và giảm bằng cách giảm đi các chỉ mục. Các kiểu liệt kê có thể được mở rộng bằng cách thêm các hằng số liệt kê mới và giảm bằng cách loại bỏ các hằng số liệt kê. Kiểu bản ghi có thể được mở rộng bằng cách thêm vào các bản ghi mới và giảm bằng cách loại bỏ các bản ghi . (Oberon cho phép mở rộng các kiểu bản ghi.) Mở rộng các kiểu bản ghi tương tụ với khái niệm xây dựng các lớp con trong lập trình hướng đối tượng, thảo luận trong chương 5.

Các kiểu kết quả có thể được chuyển đổi về (with) các kiểu ban đầu cho các mục đích gán và truyền tham số. Sự biến đổi có thể là do đổi kiểu hoặc bằng cách ép kiểu. Trong cả hai trường hợp, sự biến đổi có thể bỏ qua các phần tử mảng và các trường bản ghi mà không cần thiết trong các kiểu mục tiêu(target) và có thể thiết lập các phần tử và các trường chỉ được biết đến trong các kiểu mục tiêu (target) đến một giá trị lỗi. Nó có thể tạo ra một lỗi thực thi nếu giá trị liệt kê không được biết đến trong các kiểu mục tiêu.(target).

Ưu điểm của sự mở rộng và giảm nhiều kiểu giống như là lớp con trong các ngôn ngữ hướng đối tượng,đượoc thảo luận trong chương 5: kiểu mới có thể sử dụng các phần mềm đã được phát triển cho các kiểu hiện có; chỉ những trường hợp mới thì mới cần phải được giải quyết cụ thể trong phần mềm mới. Một mô-đun mở rộng hoặc giảm một kiểu đưa vào không bắt buộc các module này đưa các kiểu vào được biên dịch lại.

Chú thích: (3) Trong Ada, một lập trình viên cũng có thể chồng toán tử, do đó, người ta có thể khai báo một thủ tục mà có một đơn vị ma trận và đơn vị một đế quốc, chuyển đổi chúng, và sau đó nhân chúng.

4. Kích thước

Ví dụ liên quan đến meters (bộ đếm giờ) và feet (đơn vị đo dặm ở anh, số nhiều là foot) chỉ ra rằng những kiểu đơn không ngăn chặn được lỗi chương trình. Tôi muốn ngăn chặn việc nhân 2 giá trị feet và gán kết quả trả lại vào một biến có kiểu feet bởi vì kiểu của kết quả trả về là bình phương feet, không phải là feet.

Ngôn ngữ AL dành cho lập trình thao tác cơ khí (các trình điều khiển), dưa ra một kiểu như là thuôc tính của biểu thức được gọi là kích thước nhằm ngăn chặn lỗi. khái niệm này được dề xuất lần đầu tiên bởi C.A.R. Hoare. Và khái niệm đó được mở rộng từ đó. Nghiên cứu gần đây đã chỉ ra làm thế nào để cài đặt các kích thước vào trong ngôn ngữ máy[Kennedy 94].(đa hình trong ML sẽ được bàn luận kỹ lưỡng sau ở trong chương này). AL có 4 khai báo kích thước cơ bản: time (thời gian), angle (góc), distance (khoảng cách) và mass (khối lượng). mỗi kích thước cơ bản có khai báo hằng số tương ứng là giây (second), cm (centimeter), gram. Tập giá trị của những hằng số này tương ứng với một tập đơn vị khác; người lập trình chỉ cần biết các hằng là nhất quán với nhau. Ví dụ: 60 giây = 1 phút. Kích thước mới có thể được khai báo và được xây dựng từ những kích thước cũ. AL không hỗ trợ cho các lập trình viên khai bao các kích thước cơ bản. nhưng như vậy một sự mở rộng có thể là hợp lý. Mặt hữu ích khác của các dimension cơ bản có thể (tất nhiên) là về dòng điện hiện nay (đơn vị được đo, cho khoảng cách, trong ampe), nhiệt độ (độ kelvin), cường độ tỏa sáng (lumens), tiền tệ (đồng bảng anh). Nhìn lại, góc có thể là một sự lựa chọn tốt cho một chiều hướng cơ bản, nó tương đương với tỷ lệ của hai khoảng cách: Khoảng cách dọc theo một cung và bán kính của vòng tròn. Ví dụ dưới đây cho thấy kích thước được sử dụng :

Ví dụ:

Dimension:

Diện tích:= độ dài * độ dài;

Vận tốc = quãng đường/ thời gian;

```
Constant:
        Mile:=5280 *foot; // foot đã được khai báo
        Acre:= mile*mile/640 ; //1 mẫu ở Anh
Variable:
        d1, d2 : distance real; //khoảng cách thực
        a1 : area ral
        v1 : velocity real;
begin
        d1 := 30  * foot;
        a1 :=d1 * (2 * mile) + (4 * acre);                13
        v1 :=a1 / (5 * foot * 4 * minute);
        d2 := 40;--- invalid: dimension error
        d2 :=d1+v1;--- invalid:dimension error
        write(d1/foot,"d1 in feet ", v1*hour/mile, "v1 in miles per hour");
end;
```

Dòng 13, a1 là vùng gồm 4 mẫu anh (acre) cộng với 30 feet nhân với 2 miles. Dòng 14 trình dịch có thể kiểm tra biểu thức ở vế phải có là kích thước của vận tốc, tức là bàng quãng đường/ thời gian, mặc dù nó là khó khăn cho một con người để tìm ra một giải thích đơn giản của biểu thức. Trong những ngôn ngữ này còn thiếu 1 tính năng của kích thước, những kiểu dữ liệu trừu tượng, có thể được thay thế, sẽ được giới thiệu trong phần tiếp theo. Sẽ xem xét kỹ hơn trong phần bài tập.

---

5. Kiểu dữ liệu trừu tượng (Abstract Data Type)

Khi thiết kế thuật toán với một dãy các hành động trên các đối tượng dữ liệu, chúng ta cần sử dụng sự trừu tượng hoá dữ liệu (data abstraction).

Sự trừu tượng hoá dữ liệu có nghĩa là chúng ta chỉ quan tâm tới một tập các đối tượng dữ liệu (ở mức độ trừu tượng) và các phép toán có thể thực hiện được trên đối tượng dữ liệu đó.

Sử dụng sự trừu tượng hoá dữ liệu trong thiết kế thuật toán là phương pháp luận thiết kế rất quan trọng. Nó có các ưu điểm sau:

- Đơn giản hoá quá trình thiết kế, giúp ta tránh được sự phức tạp liên quan tới biểu diễn cụ thể của dữ liệu .
- Chương trình sẽ có tính mođun (modularity). Chẳng hạn, một hành động trên đối tượng dữ liệu phức tạp được cài đặt thành một mođun (một hàm). Chương trình có tính mođun sẽ dễ đọc, dễ phát hiện lỗi, dễ sửa, …

Sự trừu tượng hoá dữ liệu được thực hiện bằng cách xác định các kiểu dữ liệu trừu tượng (Abstract Data Type). Kiểu dữ liệu trừu tượng (ADT) là một tập các đối tượng dữ liệu cùng với các phép toán có thể thực hiện trên các đối tượng dữ liệu đó.

Ví dụ. Tập các điểm trên mặt phẳng với các phép toán trên các điểm mà chúng ta đã xác định tạo thành ADT điểm.

ADT là stack. Thủ tục mà thao tác ngăn xếp được push, pop, và empty. Cho dù thực hiện sử dụng một mảng, một danh sách liên kết, hoặc một tập tin dữ liệu là không thích hợp cho đối tượng và có thể được ẩn.

Một kiểu dữ liệu trừu tượng gồm 2 phần: đặc tả và cài đặt.

- Các đặc tả một phần có chứa khai báo dự định sẽ được hiển thị cho khách hàng của mô-đun, nó có thể bao gồm các hằng số, các loại, các biến, và tiêu đề thủ tục.

- Các phần cài dặt có chứa các cơ quan thủ tục (có nghĩa là, phần triển khai thực hiện) cũng như các tờ khai khác được private tới các module.

5.1 Đặc tả kiểu dữ liệu trừu tượng

Nhớ lại rằng, một ADT được định nghĩa là một tập các đối tượng dữ liệu và một tập các phép toán trên các đối tượng dữ liệu đó. Do đó, đặc tả một ADT gồm hai phần: đặc tả đối tượng dữ liệu và đặc tả các phép toán.

- Đặc tả đối tượng dữ liệu. Mô tả bằng toán học các đối tượng dữ liệu. Để mô tả chúng, chúng ta cần sử dụng sự trừu tượng hoá (chỉ quan tâm tới các đặc tính quan trọng, bỏ qua các chi tiết thứ yếu).
- Đặc tả các phép toán. Việc mô tả các phép toán phải đủ chặt chẽ, chính xác nhằm xác định đầy đủ kết quả mà các phép toán mang lại,

nhưng không cần phải mô tả các phép toán được thực hiện như thế nào để cho kết quả như thế. Cách tiếp cận chính xác để đạt được mục tiêu trên là khi mô tả các phép toán, chúng ta xác định một tập các tiên đề mô tả đầy đủ các tính chất của các phép toán. Chẳng hạn, các phép toán cộng và nhân các số nguyên phải thoả mãn các tiên đề: giao hoán, kết hợp, phân phối, …. Chúng ta sẽ mô tả mỗi phép toán bởi một hàm (hoặc thủ tục), tên hàm là tên của phép toán, theo sau là danh sách các biến. Sau đó chỉ rõ nhiệm vụ mà hàm cần phải thực hiện.

Ví dụ. Sau đây là đặc tả ADT số phức. Ở đây, chúng ta sẽ đặc tả các ADT khác theo khuôn mẫu của ví dụ này.

Mỗi số phức là một cặp số thực (x, y), trong đó x được gọi là phần thực (real), y được gọi là phần ảo (image) của số phức.

Trên các số phức, có thể thực hiện các phép toán sau:

1. Create (a, b). Trả về số phức có phần thực là a, phần ảo là b.
2. GetReal (c). Trả về phần thực của số phức c.
3. GetImage (c). Trả về phần ảo của số phức c.
4. Abs (c). Trả về giá trị tuyệt đối (mođun) của số phức c.
5. Add (c_1,c_2). Trả về tổng của số phức c_1 và số phức c_2.
6. Multiply (c_1, c_2). Trả về tích của số phức c_1 và số phức c_2.
7. Print (c). Viết ra số phức c dưới dạng a + i b trong đó a là phần thực, b là phần ảo của số phức c.

5.2 Cài đặt kiểu dữ liệu trừu tượng

Cài đặt ADT có nghĩa là biểu diễn các đối tượng dữ liệu bởi các CTDL và cài đặt các hàm thực hiện các phép toán trên dữ liệu.

Trong giai đoạn đặc tả, chúng ta chỉ mới mô tả các phép toán trên các đối tượng dữ liệu, chúng ta chưa xác định các phép toán đó thực hiện nhiệm vụ của mình như thế nào. Trong chương trình, để sử dụng được các phép toán của một ADT đã đặc tả, chúng ta cần phải cài đặt ADT đó trong một ngôn ngữ lập trình.

Công việc đầu tiên phải làm khi cài đặt một ADT là chọn một CTDL để biểu diễn các đối tượng dữ liệu.

Ví dụ. Chúng ta có thể biểu diễn một số phức bởi cấu trúc trong C + +

```
struct      complex
{
        float    real;
        float    imag;
 }
```

Sau khi đã chọn CTDL biểu diễn đối tượng dữ liệu, bước tiếp theo chúng ta phải thiết kế và cài đặt các hàm thực hiện các phép toán của ADT.

Trong giai đoạn thiết kế một hàm thực hiện nhiệm vụ của một phép toán, chúng ta cần sử dụng sự trừu tượng hoá hàm (functional abstraction). Sự trừu tượng hoá hàm có nghĩa là cần mô tả hàm sao cho người sử dụng biết được hàm thực hiện công việc gì, và sao cho họ có

thể sử dụng được hàm trong chương trình của mình mà không cần biết đến các chi tiết cài đặt, tức là không cần biết hàm thực hiện công việc đó như thế nào.

Sự trừu tượng hoá hàm được thực hiện bằng cách viết ra mẫu hàm (function prototype) kèm theo các chú thích.

Mẫu hàm gồm tên hàm và theo sau là danh sách các tham biến. Tên hàm cần ngắn ngọn, nói lên được nhiệm vụ của hàm. Các tham biến cần phải đầy đủ: các dữ liệu vào cần thiết để hàm có thể thực hiện được công việc của mình và các dữ liệu ra sau khi hàm hoàn thành công việc.

Bước tiếp theo, chúng ta phải thiết kế thuật toán thực hiện công việc của hàm khi mà đối tượng dữ liệu được biểu diễn bởi CTDL đã chọn. Việc cài đặt hàm bây giờ là chuyển dịch thuật toán thực hiện nhiệm vụ của hàm sang dãy các khai báo biến địa phương cần thiết và các câu lệnh. Tất cả các chi tiết mà hàm cần thực hiện này là công việc riêng tư của hàm, người sử dụng hàm không cần biết đến, và không được can thiệp vào. Làm được như vậy có nghĩa là chúng ta đã thực hành nguyên lý che dấu thông tin (the principle of information hiding) - một nguyên lý quan trọng trong phương pháp luận lập trình môđun.

Một kiểu dữ liệu trừu tượng Stack có thể được lập trình.

Module Stack; 1

export 2

```
        Push, Pop, empty, StackType, MaxStackSize;                3
constant                                                          4
        MaxStackSize = 10;                                        5
type                                                              6
        private Stack Type=                                       7
        record                                                    8
                size : 0 .. MaxStackSize: = 0;                    9
                data: aray 1 .. MaxStackSize of integer;          10
        end;                                                      11
    - - Chi tiết bị bỏ qua cho các thủ tục sau                    12
        procedure Push (reference ThisStack: StackType;           13
                readonly what: integer);                          14
        procedure Pop (reference ThisStack): integer;             15
        procedure empty (readonly ThisStack): Boolean;            16
end; - -Stack
```

6. Nhãn, thủ tục, types as first-class values

- Một giá trị là bất cứ một cái gì mà có thể được thao tác bởi một chương trình.

- Một kiểu là một tập các giá trị với các thao tác có thể được áp dụng giống nhau cho mỗi giá trị trong tập.

Biểu đồ phân biệt lớp giá trị đầu tiên, thứ hai và thứ ba.

Thao tác	Class giá trị		
	Đầu tiên	Thứ hai	thứ ba
Truyền giá trị như một tham số	Có	Có	Không có
Trả lại giá trị như một tham số	Có	Không	Không
Gán giá trị vào một biến	Có	Có	Không

=>Một giá trị mà tất cả các thao tác đều chấp nhận được gọi là first-class value (lớp giá trị đầu tiên).

=>Một giá trị mà tất cả các thao tác đều không được chấp nhận, trừ thao tác truyền giá trị như một tham số thì được gọi là second-class value (lớp giá trị thứ hai).

=>Một giá trị mà tất cả các thao tác đều không được chấp nhận gọi là third-class value (lớp giá trị thứ ba).

Ví dụ:

❖ Trạng thái các giá trị trong Pascal:

- Các giá trị first-class (thứ nhất): các giá trị chân lý, các ký tự, kiểu liệt kê, các số nguyên, số thực, con trỏ.

- Các giá trị lower-class(lớp dưới): có thể được truyền như là các tham số, nhưng nó không được lưu trữ hay trả ra, hoặc được sử dụng như là các thành phần trong các giá trị khác:

+ Các giá trị hỗn hợp (bản ghi, mảng, tập hợp, tệp): không thể được trả ra.

+ Thủ tục và hàm trừu tượng.

+ Tham chiếu tới các biến (trừ trường được che dấu thành con trỏ).

❖ Trong ngôn ngữ ML:

- Tất cả các giá trị trong ngôn ngữ ML đều thuộc lớp giá trị thứ nhất.
- Cái mà chúng ta có thể làm được trong ML mà không làm được trong pascal:

+ Tạo ra một bản ghi gồm 2 chức năng.

+ Viết 1 hàm nhận 1 hàm f:int→int và trả ra thành phần của f là chính nó .

+ Viết 1 biểu thức mà giá trị của nó tham chiếu tới một giá trị .

Ngôn ngữ khác nhau trong cách họ thao tác với các nhãn, thủ tục, và các kiểu.

Ví dụ: Thủ tục lớp giá trị thứ 3 trong Ada, lớp giá trị thứ 2 trong Pascal, và lớp giá trị đầu tiên giá trị trong C và Modula-2.

Nhãn nói chung là lớp giá trị thứ 3 , nhưng chúng là những lớp giá trị thứ hai trong Algol-60.

Cho phép nhãn hiệu và thủ tục được cấp giá trị đầu tiên là phức tạp. Như vậy giá trị có thể được lưu trữ trong các biến và được gọi tại một thời điểm khi các trung tâm ngăn xếp không còn chứa các bản ghi kích hoạt mà chúng hướng vào.

```
Variable                                                    1
    ProcVar: procedure ();                                  2
  procedure outer ();                                       3
      variable OuterVar: integer;                           4
      procedure inner ();                                   5
          begin - -inner                                    6
              write (OuterVar);                             7
          end;- - inner                                     8
      begin - Outer                                         9

        ProcVar: = inner; - xong nhiệm vụ đc giao           10

    end; - Outer                                            11

begin - chương trình chính                                  12
      outer ();                                             13
      ProcVar ();                                           14
```

end; **15**

Bởi thời gian bên trong được gọi (như giá trị của biến thủ tục ProcVar ở dòng 14), môi trường tham khảo của các cá thể của bên ngoài có được ngừng hoạt động, bởi vì ngoài đã quay trở lại. Vấn đề này được gọi là dangling-procedure problem. Ngôn ngữ có lập trường khác nhau chú ý đến vấn đề lơ lửng thủ tục:

1. Với bất kỳ chương trình mà cố gắng để gọi một bao đóng với một con trỏ lơ lửng là sai lầm, nhưng không cố gắng để tạo ra lỗi.
2. Ngăn chặn tình hình xấu phát sinh từ các hạn chế về ngôn ngữ. Cấp đầu thủ tục không cần một môi trường tham chiếu nonlocal. Không xử lý ngôn ngữ nhãn như các giá trị hạng nhất.
3. Ngăn chặn tình hình xấu từ phát sinh do thực hiện tốn kém.

Nhãn như giá trị lớp đầu là đáng sợ vì lý do khác: chúng có thể được lưu trữ trong một biến và nhiều lần gọi. Do đó, các thủ tục mà trau chuốt một nhãn (có nghĩa là, nó xác định các nhãn) có thể trở lại nhiều hơn một lần, bởi vì nhãn có thể được gọi nhiều lần.Nhiều lần- quay lại thủ tục chắc chắn sẽ gây nhầm lẫn.

7. ML(META LANGUAGE)

Giới thiệu

ML là một siêu ngôn ngữ,(là ngôn gnữ cơ sở để viết một số ngôn ngữ khác).ML gồm Ocam ML,ML standar,XML,… ML được viết năm 1973 bởi Robin Milner.So với các ngôn ngữ lập trình khác thì ML có các ưu điểm chính sau:

- ML là một ngôn ngữ tương tác-Nó giống như một cuộc đối thoại (hỏi-trả lời).
- ML có kiểu tính.Kiểu của tất cả các định danh sẽ được trình biên dịch kiểm tra trong luc biên dịch chương trình.
- ML là một ngôn ngữ kiểu mạnh.Nghĩa là nó không cho phép dùng các giá trị của các kiểu này như là một kiểu khác.Chúng rất chặt chẽ trong việc phát hiện dùng sai kiểu.
- ML là một ngôn ngữ đa hình(tìm hiểu ở phần sau)

7.1.Biểu thức.

ML không làm việc trên các câu lệnh như một số ngôn ngữ lập trình khác. Đơn vị mà nó thao tác là các biểu thức.

Biểu thức có thể là một biểu thúc số học:

VD: in(3+5)*2;

Out:16:int

Có thể là một biểu thức xâu.

VD: in:”this is it”;

Out:”this is it”:string

Có thể là một biểu thức điều kiện.Cấu trúc như sau:

If(biểu thúc boolean) then..else..

VD: in: if true then 3 else 4;

Out:3:int;

Lưu ý:

Tất cả các giá trị trong ML là các giá trị first-class,bao gồm:

- Tất cả các giá trị nguyên thuỷ:số nguyên,số thực,xâu.
- Các giá trị hỗn hợp:bản ghi,kiểu danh sách ,kiểu mảng

Không có cấu trúc lặp trong ML(for-do, while-do,repeat-until) vì cấu trúc lặp hoàn toàn phụ thuộc vào biến mà trong ML không sử dụng biến.

7.2.Khai báo toàn cục.

Trong ML không sử dụng các biến mà là các định danh(identifiers). Định danh là các hằng tên,do đó giá trị của nó không thay đổi(trừ khi nó được khai báo lại).

Trong phạm vi toàn cục thì các định danh được khai báo ở mức cao nhất.

Chú ý rằng khai báo không phải là một biểu thức.Khai báo thiết lập ràng buộc về giá trị cho định danh thay vì trả ra giá trị.

Cấu trúc khai báo:val tên định danh= giá trị ràng buộc.Xem ví dụ sau:

In:a=3 and

b=5 ;

Trong ví dụ trên ta có 2 định danh là a và b được đặt ở mức trên cùng của chương trình.Các định danh được ngăn cách nhau bởi từ khoá and.Kể từ sau khi được khai báo chũng sẽ được sử dụng(trừ khi ta khai báo lại chúng) trong to àn chương trình.

7.3.Khai báo địa phương(cục bộ).

Khi khai báo được đặt trong 1 khối(block) let-end ,ta có khai báo cục bộ.

Ví dụ:

In: let

Val a=3 and b=5

In

(a+b) div 2

End;

Out:4:int

Chỉ trong thân của khói đó các định danh này mới được phép truy xuất.(trừ khi ra ngoài khối đó ta khai báo lại định danh).

7.4.Danh sách.

Danh sách là một sự tương ứng,nghĩa là các phần tử trong danh sách bắt buộc phải cùng kiểu.Kiểu của phần tử của danh sách có thể là kiểu thực ,kiểu nguyên hay kiểu xâu.

Các phần tử của danh sách đượC đặt trong dấu [],cách nhau bơi dấu phẩy.Chiều dài của danh sach là số phần tử của danh sách đó.

Ví dụ:

[] ||danh sách rỗng(ko có phần tử nào,chiều dài danh sách =0)

[1,2] ||danh sách nguyên gồm 2 phần tử nguyên,chiều dài danh sách là 2

Các phép toán trong danh sách:

Null ||trả ra giá trị true nếu tham số của hàm là nil và ngược lại.

Hd(head) ||trả về phần tử đầu tiên của một danh sách khác rỗng.

Tl(tail) ||trả về danh sách còn lại sau khi đã lấy đi ohần tủe đàu tiên của một danh sách

khác rỗng

@ ||các danh sách móc nối.

Ví dụ:

Biểu thức	Giá trị trả về

Null	true
null[1,2,3]	false
hd[1,2,3]	1
tl[1,2,3]	[2,3]
[]@[1,2]	[1,2]

7.5.Hàm

Hàm là khái niệm cơ bản trong các ngôn ngữ hàm.Lập trình theo hàm(gọi tắt là lập trình hàm) là lập trình dựa trên việc tính toán giá trị của hàm.

Khái niệm hàm trong lập trình hàm khác với hàm trong lập trình mệnh lệnh. Các hàm trong LTH không gây ra hiệu ứng phụ trong chương trình.Do đó kết quả của 1 hàm không phụ thuộc vào thời điểm hàm được gọi mà chỉ phụ thuộc vào cách gọi hàm như thế nào đối với các tham số mà thôi.Còn càc hàm trong LTML ,kết quả trả về có thể khác nhau đối với cùng 1 đối số,phụ thuộc vào trạng thái hàm đượoc gọi.

Ví dụ: Trong ngôn ngữ lập trình mệnh lệnh,kết quả của biểu thức f x+f x có thể khác với kết quả 2*f x,vì lời gọi đầu tiên có thể làm thay đổi x hoặc 1 biến nào đó được tiếp cận bởi f.Trong ML,kết quả của hai biểu thức trên là như nhau.

7.6. Đa hình.

Tính đa kiểu là yếu tố rất quan trọng trong các ngôn ngữ hàm .Các ngôn ngữ hàm có một hệ thống kiểu dữ liệu hoàn toàn khác với các ngôn ngữ mệnh lệnh. Trong các ngôn ng ữ m ệnh lệnh (nh ư Pascal, C, Ada..), tính định kiểu tĩnh chặt chẽ (static strong typing) bắt buộc người l ập trình phải mô tả kiểu cho m ỗi bi ến dùng đến. Sau khi khai báo, ng ười s ử d ụng không được thay đổi kiểu d ữ liệu của biến trong khi chạy chương trình. Trong ngôn ngữ hàm,với mỗi tham biến hình thức,một hàm đa kiểu có thể chấp nhận lời gọi tương ứng với nhiều tham số thực sự có các kiểu khác nhau.

Ví dụ: danh sách rỗng [] có thể coi là một danh sách có kiểu đa hình ,vì có thể xem nó như một danh sách c ác s ố nguyên h ặc danh sách các kí tự ASCII.

7.7.Suy luận kiểu.

Việc khai báo kiểu hàm là không cần thiết. Các ngôn ngữ hàm thường có khả năng suy diễn kiểu tự động nhờ một bộ kiểm tra kiểu (type checker).

Suy đoán kiểu là một cơ chế mà ở đó các đặc tả về kiểu thường có thể bị loại bỏ hoàn toàn (nếu có thể),nhằm giúp cho trình biên dịch dễ dàng dự đoán được kiểu của các giá trị từ ngữ cảnh mà các giá trị đó được sử dụng. Thí dụ một biến được gán giá trị 1 thì trình dịch loại suy đoán kiểu không cần khai báo riêng rằng đó là một kiểu integer.

Nhận x ét:

Ưu điểm của ML:

- Ngôn ngữ có tính mềm dẻo nhờ sử dụng hàm đa kiểu.
- Chương trình không phụ thuộc vào phàn cứng của máy tính,nâng cao tính sáng tạo của người lập trình.Người lập trình không cần quan tâm cài đặt thế nào trong máy.
- Do không phụ thuộc nhiều vào các biến toàn cục nên việc lập trình sẽ dễ hơn lập trình mệnh l ệnh.
- Không gây ra các hiệu ứng phụ

Nhược điểm của ML: Thiếu lệnh gán và không sử dụng biến nên có khó khăn trong việc mô tả cấu trúc dữ liệu và khó thực hiện quá trình vào ra dữ liệu.

Do không phụ thuộc vào phần cứng nên làm hạn chế giao tiếp giữa ngươi sử dụng và hệ điều hành.

7.8.Các hàm bậc cao.(Higher-Order Functions)

HOF tuy là một kĩ thuật rất cơ bản trong lập trình FP nhưng nó chỉ mới xuất hiện từ C# 2.0. Higher Order Function xem hàm cũng là dạng dữ liệu cho nên tham số của hàm cũng là một hàm.

"Filter", "Map" và "Reduce" được xem là ba Higher Order Function rất hữu ích cho các phép toán trong dãy (List, Aray, IEnumerable). Tuy nhiên Higher Order Function trong C# 2.0 chỉ hỗ trợ cho kiểu dữ liệu List và Array.

ML hỗ trợ các hàm bậc cao.Các hàm này chứa các hàm khác như là các tham số hay xem các hàm như là các kết quả.Các các hàm bậc cao đặc biệt hữu ích trong việc ứng dụng từng phần .trong đó một lời gọi chi cung cấp 1 vài tham số của hàm . Chúng mang lại tính hiệu quả và là nền tảng của lập trình hàm

7.9.Các kiểu ML.

Kiểu của một biểu thức chỉ ra tập hợp các giá trị mà nó có thể tạo ra.Các kiểu này gồm các kiểu nguyên thuỷ do ngôn ngữ đ ịnh nghĩa sẵn (nguyên,thực,logic,xâu)và các kiểu được định nghĩa(danh sách,hàm,con trỏ,chương trình con).Một kiểu ML chỉ nhận thông tin về các thuộc tính là cái mà có thể đựoc dùng để tính thời gian biên dịch và không phân biệt đựoc sự khác nhau giữa các giá trị có các cáu trúc giống nhau. Theo cách khác các kiểu ML có thể có quan hệ cấu trúc rõ ràng trong các giá trị.Phần bên phải của 1 cặp phải có kiểu giống như kiểu của phần bên trái của cặp.Một hàm phải trả ra một giá trị có kiểu giống với kiểu của các tham số của hàm.

Các tên kiểu có thể được dùng để biểu diễn các kiểu hợp.Các kiểu hợp hầu hết là có ích như các kiểu của các hàm,mặc dù 1 vài biểu thức không phải là hàm,như [] -kiểu của ‘a list cũng là kiểu hợp.Một ví dụ điển hình của một hàm hợp là hd,của kiểu ‘a list->’a.Kiểu của hd chỉ ra rằng nó có thể nhận bất kì một danh sách nào và kiểu của kết quả là giống như kiểu của các phần tử của danh sách.

Mỗi kiểu thể hiện một miền kiểu.Miền kiểu là tập hợp tất cả các giá trị mà kiểu có thể nhận.Cho ví dụ,int*int thể hiện miền của cặp

nguyên,và int->int thể hiện miền của tất cả các hàm nguyên.Một biểu thức có thể có tới vài kiểu;.Cho ví dụ,hàm đồng nhất fn x=>x có kiểu int->int,vì nó ánh xạ bất kì biểu thức nguyên nào tới chính nó;

Người lập trình có thể viết thêm vào một kiểu của biểu thức tới dữ liệu của biểu thức theo thứ tự để biểu thị sự ép kiểu

Một phép ép kiểu giới hạn tham chiếu kiểu bởi ML bằng ép các biểu thức hợp hay các hàm hợp,

Trong quá trình kiểm tra kiểu,nếu không có sự tương thích giữa kiểu thực của đối số và kiểu đang được mong đợi của phép toán ấy thì có hai lựa chọn có thể:

- Sự tương thích kiểu bị báo lối;
- Hoặc một sự chuyển đổi kiểu tự động được thi hành để đổi kiểu của đối số thực tế thành kiểu đúng với yêu cầu.

7.10 Kiểu được định nghĩa

Người lập trình có thể tạo ra các kiểu cấu trúc mới trong 1 kiểu đã được khai báo.Một sự khai báo kiểu tạo ra 1 tên kiểu mới.Cấu trúc này bắt đầu bằng tên,theo sau là từ khoá OF

Ngoài các kiểu nguyên thuỷ được định nghĩa bởi ngôn ngữ, người lập trình còn có thể định nghĩa các kiểu của riêng mình. Định nghĩa một kiểu dữ liệu mới bao gồm việc xác định các yếu tố sau:

- Tên của kiểu.
- Sự biểu diễn bộ nhớ cho các đối tượng dữ liệu của kiểu.
- Tập hợp các phép toán (các chương trình con) thao tác trên các đối tượng dữ liệu của kiểu.

Ví dụ trong Pascal ta xét định nghĩa kiểu như sau:

```
TYPE
    RealVect = ARRAY[1..10] OF real;
```

Sau đó ta có thể dùng phép khai báo biến:

```
VAR
    A: RealVect;
    B,C:RealVect;
```

Ưu điểm của định nghĩa kiểu:

- Làm cho việc viết chương trinh trở nên ngắn gọn, sáng sủa hơn.
- Khi cần thay đổi cấu trúc dữ liệu, chỉ cần thay đổi một lần ở mức định nghĩa kiểu chứ không cần phải thay đổi nhiều lần ở mức khai báo từng biến riêng biệt.

Chúng ta thấy rằng kiểu do người dùng định nghĩa chính là một kiểu dữ liệu trừu tượng.

Kiểm tra kiểu dẫn tới sự so sánh giữa kiểu dữ liệu của đối số thực đã được cho của một phép toán và kiểu dữ liệu của đối số mà phép toán

đó cần đến. Nếu kiểu giống nhau thì đối số được chấp nhận và phép toán được tiến hành, nếu kiểu khác nhau, thì một lỗi được xem xét hoặc một sự cưỡng bức chuyển đổi kiểu được dùng để đổi kiểu của đối số thực thành kiểu thích hợp.

Vấn đề ở đây là cần phải xác định hai kiểu như thế nào thì được coi là "giống nhau" hay tương đương. Xét ví dụ sau đây:

```
TYPE
    Vect1 = ARRAY[1..10] OF REAL;
    Vect2 = ARRAY[1..10] OF REAL;
VAR x,z : Vect1;
    y : Vect2;
PROCEDURE Sub(a:Vect1);
.....
END; { Sub }
BEGIN { Chương trình chính }
    x := y;
    Sub(y);
    ......
END.
```

Vấn đề ở đây là các biến x, y và a có cùng kiểu do đó lệnh gán x := y và lời gọi chương trình con Sub(y) là đúng hay chúng có khác kiểu.

Có hai cách giải quyết cho vấn đề này: tương đương tên và tương đương cấu trúc.

1/ Tương đương tên

Hai kiểu dữ liệu được xem là tương đương chỉ khi chúng có tên giống nhau. Như vậy các kiểu Vect1 và Vect2 ở trên là khác kiểu mặc dù đối tượng dữ liệu có chung một cấu trúc. Lệnh gán x := y và lời gọi chuong trình con Sub(y) là không hợp lệ. Tương đương tên là phương pháp được dùng trong Ada và Pascal. Tương đương tên có một điểm yếu là khi một kiểu không có tên như trong khai báo trực tiếp:

VAR w : ARRAY[1..10] OF REAL;

Biến w có kiểu riêng nhưng là kiểu không có tên. Như vậy w không thể được dùng như là một đối số cho một phép toán mà phép toán đó đòi hỏi một đối số của một kiểu có tên.

2/ Tương đương cấu trúc

Hai kiểu dữ liệu được xem là tương đương nếu chúng xác định các đối tượng dữ liệu có cấu trúc bên trong giống nhau. Thông thường thuật ngữ "cấu trúc bên trong giống nhau" có nghĩa là giống nhau về sự biểu diễn bộ nhớ được dùng cho cả hai lớp đối tượng dữ liệu. Ví dụ Vect1 và Vect2 là tương đương cấu trúc bởi vì mỗi một đối tượng dữ liệu của kiểu Vect1 và mỗi một đối tượng dữ liệu của kiểu Vect2 có chung số phần tử có kiểu tương đương.

Quản lý bộ nhớ đối với các đối tượng dữ liệu của cả hai kiểu này là giống nhau, do đó công thức truy nhập giống nhau có thể được sử dụng để lựa chọn các phần tử và nói chung sự cài đặt tại thời gian thực hiện của các kiểu dữ liệu là giống hệt nhau.

Tương đương cấu trúc không có các bất tiện như tương đương tên nhưng nó lại có những vấn đề khác, chẳng hạn như hai biến có thể tương đương cấu trúc một cách không cố ý mặc dù người lập trình đã khai báo chúng một cách tách biệt như trong ví dụ sau:

```
TYPE

    Meters = INTEGER;

    Liters = INTEGER;

VAR  Len : Meters;

    Vol : Liters;
```

Các biến Len và Vol có kiểu tương đương cấu trúc và do đó một lỗi như phép cộng Len + Vol sẽ không được tìm thấy bởi phép kiểm tra kiểu tĩnh. Khi có nhiều lập trình viên làm việc chung trong một chương trình thì tương đương kiểu không cố ý có thể gây nên các lỗi rất nghiêm trọng như trong ví dụ nói trên.

8.MIRANDA

Miranda là một ngôn ngữ lập trình hàm bậc cao. Được thiết kế bởi David Turner thuộc đại học Kent . Có sử dụng một số khái niệm từ ML

và Hope .Nó là 1 kiểu dữ liệu mạnh ,kiểu suy ra từ ngữ cảnh, cung cấp những kiểu dữ liệu trừu tượng và có những hàm mở rộng.

Miranda là một trình thông dịch chạy trên UNIX .Có đặc điểm là thuần túy hàm. Miranda lần đầu tiên được phát hành vào năm 1985, như một thông dịch viên nhanh trong C cho hệ điều Unix , với phiên bản tiếp theo vào năm 1987 và 1989. Sau này ngôn ngữ lập trình Haskell cũng có nhiều điểm tương tự vớ ngôn ngữ Miranda.

Lĩnh vực ứng dụng

Việc sử dụng chính của Miranda là:

- tạo mẫu nhanh
- giảng dạy lập trình chức năng như một ngôn ngữ đặc tả
- nghiên cứu lập trình chức năng
- một mục đích chung công cụ lập trình

Cú pháp hàm trong Miranda:

<tên hàm> ::<miền xác định> -><miền giá trị>

<tên hàm>[<danh sách tham đối>]=<biểu thức> [<điều kiện>]

Một ví dụ đơn giản về ngôn ngữ Miranda:

z = sq x / sq y

sq n = n *

x = a + b

y = a – b

a = 10

b = 5

8.1.Cấu trúc danh sách

Đây là cấu trúc dữ liệu quan trọng nhất của các ngôn ngữ lập trình hàm trong Miranda là bằng văn bản với dấu ngoặc vuông và dấu phẩy,

ví dụ như:

week_days = ["Mon","Tue","Wed","Thur","Fri”]

days = week_days ++ ["Sat","Sun"]

♦Một số các phép toán trên danh sách

product [...] : tính tích

fac n = product [1..n]

sum [...] : tính tổng

resul = sum [1, 3..100]

hd [...] : trả về phần tử đầu tiên

hd [1, 3, 100] --> 1

tl [...] : trả về danh sách các PT thứ 2 trở đi

---

tl [1, 3, 2, 5] --> [3, 2, 5]

: [...] : chèn 1 PT vào đầu danh sách

0 : [1, 3, 2, 5] --> [0, 1, 3, 2, 5]

♦ Viết lại một số hàm

length L = 0, if L = []

length L = 1 + length (tl L), otherwise

concat L1 L2 = L2, if L1 = []

concat L1 L2 = L1, if L2 = []

concat L1 L2 = (hd L1) : concat (tl L1) L2, otherwis

8. 2. Phép so khớp

–Một hàm có thể định nghĩa bằng nhiều biểu thức vế phải khác nhau

– Cú pháp tổng quát :

<pattern> = <expression>, <condition>

– **Ví dụ** : tính dãy Fibonacci

fib 0 = 0

fib 1 = 1

fib (n+2) = fib (n+1) + fib n

Ví dụ : tính độ dài chuỗi

Length [] = 0

length (a : L) = 1 + (length L)

– **Ví dụ** : tính tổng của mảng

sum [] = 0

sum [a : X] = a + sum X

– **Ví dụ** : tính tích của mảng

product [] = 1

product [a : X] = a * product X

8.3. Phương pháp currying (tham đối hoá từng phần - partial parametrization)

– Cho hàm n biến : f(x1, x2,..., xn)

– Có thể viết lại :

x1-->(x2-->...(xn-->f(x1..xn))...))

–Một hàm nhiều hơn 1 tham đối thì có thể tham đối hoá từng phần

triple x = 3 * x hoặc triple = multi 3

8.4. Kiểu và tính đa kiểu (polymorphic)

– Các ngôn ngữ hàm thường không sử dụng định kiểu

– Cho phép định nghĩa các hàm đa kiểu với lời gọi có các tham đối có các kiểu dữ liệu khác nhau

♦ Có 3 loại kiểu nguyên thủy đó là: kiểu số; kiểu bool và kiểu kí tự

Kiểu số :gồm có kiểu số nguyên và kiểu số thực.

Kiểu bool: có 2 giá trị là True và False;

Kiểu kí tự: gồm các kí tự trong ASCII.Các kí tự được đặt trong dấu ngoặc đơn

– Ví dụ

```
pair x y = [x, y]
```

– Sử dụng

```
pair 1 2 --> [1, 2]

pair true false --> [true, false]
```

8.5. Tính hàm theo kiểu khôn ngoan

– Thông thường khi tính giá trị một hàm, các thamđối được tính giá trị trước --> tính giá trị của hàm mult (fac 3) (fac 4) --> mul 6 24 --> 144

– Rút gọn biểu thức --> đơn giản hơn : rút gọn theo thứ tự áp dụng mult (fac 3) (fac 4) --> (fac 3) * (fac 4)

– **Ví dụ**

cond b x y = x, if b

cond b x y = y, otherwise

– **Ví dụ** : tính fac với cond

fac n = cond (n = 0) 1 (n * fac (n-1))

--> nếu tham đối thứ 3 luôn được tính thì hàm trong Miranda chỉ tính giá trị tham đối khi cần

Ex : fac 1

--> cond (1=0) 1 (1*fac (1-1)) II gọi fac

--> if (1=0) then 1 else (1*fac (1-1)) II gọi cond

--> if false then 1 else (1*fac (1-1)) II tính 1=0

--> 1*fac (1-1)

--> 1*(cond (1-1=0) 1 ((1-1)*fac ((1-1)-1)))

8.6. Một vài ví dụ :

– Loại bỏ những phần tử trùng nhau

uniq [] = []

uniq (a:(a:L) = uniq (a:L)

uniq (a:L) = a : uniq L

Ex :

uniq [3, 3, 4, 6, 6, 6, 6, 7]

--> [3, 4, 6, 7]

uniq ['a', 'b', 'b', 'c', 'c']

--> ['a', 'b', 'c']

---

9 ♦ RUSSELL

Ngôn ngữ Russell có trước ML nhưng hoàn toàn tương tự trong hương vị nói chung [Demers 79; Boehm 86]. Nó được phát triển để khám phá những ngữ nghĩa của các loại, đặc biệt, để cố gắng làm cho giá trị các loại hạng nhất. Russell là một ngôn ngữ mạnh mẽ, suy luận các loại từ ngữ cảnh, cung cấp các kiểu dữ liệu trừu tượng, và có bậc cao chức năng.

Russell khác với ML trong một số cách nhỏ.Mặc dù nó là tĩnh nghiên cứu, khai báo chức năng mới không ghi đè những cái cũ cùng tên nếu loại khác nhau ; thay vào đó, tên trở thành quá tải, số lượng và loại tham biến thực tế được dùng để phân biệt chức năng nghĩa là trong bất kỳ ngữ cảnh cụ thể nào.(Redeclaration của mã nhận dạng ngoại trừ chức năng không được cho phép chút nào.) Chức năng có thể được tuyên bố

được gọi như tiền tố, hậu tố, hoặc trung tố điều hành. Chức năng lấy hơn hai tham biến vẫn có thể được gọi với điều hành trung tố ; một số đã cho của tham biến được đặt vào trước điều hành, và phần còn lại đằng sau.ML chỉ cho phép ký hiệu trung tố cho chức năng nhị phân.Để ngăn chặn hiệu ứng phụ với sự có mặt của biến (loại ref), chức năng không nhập vào mã nhận dạng ánh xạ tới biến.

Danh mục của Russell là không chuẩn ; những gì ML gọi là loại chữ ký ở Russell ; kiểu dữ liệu trừu tượng (tập hợp các chức năng) là loại ở Russell.Vì thế Russell thành công trong làm loại giá trị hạng nhất, nó không hoàn thành hoàn toàn nhiều như chúng tôi sẽ mong.Cú pháp của Russell hoàn toàn khác với ML. Để nhất quán, tôi sẽ tiếp tục sử dụng cú pháp thuật ngữ và ML như tôi đã thảo luận về Russell.

Sự khác biệt chủ yếu giữa Russell và ML đó là trong Russell thì loại dữ liệu trừu tượng là những giá trị hạng nhất, giống như giá trị, con trỏ, và chức năng. Vậy là, các kiểu dữ liệu trừu tượng có thể được thong qua như các tham biến, gửi trả từ chức năng, và lưu trữ trong định danh.Giá trị kiểu dữ liệu trừu tượng cũng có thể được thao tác sau khi chúng đã được xây dựng.

Cụ thể hơn, Russell coi kiểu dữ liệu trừu tượng như là một tập hợp các chức năng có thể được đưa vào đối tượng của miền cụ thể.Kiểu dữ liệu trừu tượng boolean bao gồm chức năng nullary đúng và sai, toán tử nhị phân như bằng và và hoặc, và còn có cả câu lệnh nữa như nếu và trong khi, trong đó có các thành phần boolean.Thao tác của kiểu dữ liệu trừu tượng tức là xoá bỏ hoặc chèn chức năng trong định nghĩa của nó.

Đường viền giữa dữ liệu và chương trình trở thành hoàn toàn làm mờ đi nếu chúng ta nhìn nhận theo cách này.Rốt cuộc, chúng tôi không được dùng để kiểm soát các cấu trúc như trong khi chức năng lấy hai tham biến, boolean và câu lệnh, và gửi trả câu lệnh.Chúng tôi thường không xem xét câu lệnh để được dữ liệu chút nào, vì nó không thể được đọc, ghi, hoặc chế tác.

Các thành phần của một kiểu dữ liệu trừu tượng có thể rất khác nhau

khác. Tôi có thể khai báo một kiểu dữ liệu trừu tượng của tôi hình bao gồm các Boolean giả cũng như các số nguyên 3 (cả nullary chức năng). Nếu tôi muốn phân biệt đó là có nghĩa là sai, tôi có thể hội đủ điều kiện đó bằng cách nói bool.false hoặc My-Type.false. (Toán tử là một thành phần được chiết xuất từ một loại dữ liệu trừu tượng.)

Tôi có thể khai báo một kiểu dữ liệu trừu tượng đơn giản các số nguyên nhỏ như trong Hình 3,65.

Figure 3.65

```
val SmallInt =                                                    1

        type  New = fn : void -> SmallInt – constructor           2

        and ":=" =      fn : (SmallInt ref, SmallInt) -> SmallInt  3

    -- assignment                                                 4

and     ValueOf = fn : SmallInt ref -> SmallInt – deref           5
```

```
and       alias = fn : (SmallInt ref, SmallInt  ref) -> bool        6

-- pointer equality                                                 7

and       "<" = fn : (SmallInt,SmallInt) -> Boolean                 8

... -- other comparisons, such as <= , =, >, >=,        ≠           9

and       "-" = fn : (SmallInt, SmallInt) -> SmallInt               10

... -- other arithmetic, such as +, *, div, mod                     11

and       "0" : SmallInt – constant                                 12

... -- other constants 1, 2, ... , 9                               13

-- the rest are built by concatenation                             14

and       "^" = fn : (SmallInt,SmallInt) -> SmallInt                15

-- concatenation                                                   16

;                                                                   17
```

Tôi sử dụng khoảng trống ở hàng thứ 2 để cho biết rằng chức năng mới là

nullary. Khai báo chức năng là ghi nhớ tất cả sự thi hành của chúng.

Nói chung, xây dựng một kiểu dữ liệu trừu tượng với hình thức viết tắt mở rộng ra đanh sách như thế. Ví dụ, đó là cách ngắn gọn về kê

khai các liệt kê, bản ghi, lựa chọn, và bản sao mới của kiểu dữ liệu trừu tượng có sẵn. Danh sách được tạo ra bằng hình thức viết tắt chứa chức năng với cơ quan định sẵn.

`Kể từ khi các loại dữ liệu trừu tượng có thể được thông qua như các tham số, các lập trình viên có thể xây dựng các chức năng đa hình về giá trị của các loại dữ liệu trừu tượng. Nó được phổ biến để vượt qua cả hai loại có chứa thông số giá trị và loại có chứa các thông số chức năng. Hình 3,66 cho thấy rõ làm thế nào để khai báo một hàm đa hình Boolean nhất nếu một giá trị đã cho là nhỏ nhất trong các kiểu dữ liệu trừu tượng của nó.

Figure 3.66

Khai báo một hàm đa hình Boolean được cho là nhỏ nhất trong các kiểu dữ liệu của nó

```
val      least =                                                    1

fn                (value : bool, bool) => value = false             2

| (value : SmallInt, SmallInt) => value = SmallInt."0"      3

| (value : Other, Other :                type) => false;           4
```

Dòng 2 được áp dụng khi các tham số đầu tiên là Boolean và tham số thứ hai để chỉ ra. Nó trả về đúng chỉ khi tham số đầu tiên có giá trị false.

Dòng 3 được áp dụng khi các tham số đầu tiên là của Smallint loại. Áp dụng dòng 4 cho tất cả các loại khác, miễn là các loại tham số đầu tiên

phù hợp với giá trị tham số thứ hai. Gọi ít nhất ("chuỗi", int) sẽ không gây ra được, không ai trong số đó có các lựa chọn thay thế phù hợp.

Thao tác trên một kiểu dữ liệu trừu tượng bao gồm thêm, thay thế, và xóa các chức năng của mình. Các lập trình viên phải cung cấp một cơ thể cho tất cả các chức năng thay thế. Ví dụ, tôi có thể xây dựng một phiên bản của kiểu số nguyên để đếm số lần chuyển nhượng đã được thực hiện trên các giá trị của nó (hình 3,67).

Figure 3.67 val

```
    InstrumentedInt =                                   1

record (Value : int, Count : int)                       2

-- "record" expands to a list of functions              3

adding                                                  4

Alloc =             fn void =>                          5

let                                                     6

val                         x = InstrumentedInt.new     7

in                                                      8

count x := 0;                                           9

x -- returned from Alloc                                10

and                                                     12
```

```
Assign = fn                                                    13
(IIVar : InstrumentedInt ref,                                  14
IIValue : InstrumentedInt) ->                                  15
                (       -- sequence of several statements      16
count IIValue := count IIValue + 1;                            17
                )                                              18
                        Value IIVar := Value IIValue;          19
    and                                                        20
        GetCount = fn (IIValue : InstrumentedInt) ->           21
 count IIValue                                                 22
    and                                                        23
        new = InstrumentedInt.Alloc -- new name                24
    and                                                        25
        ":=" = InstrumentedInt.Assign -- new name              26
    And                                                        27
        ValueOf = ValueOf Value                                28
    and                                                        29
```

```
Alloc, Assign, -- internal functions                30

Value, Count, -- fields (also functions);           31
```

Hai kiểu dữ liệu trừu tượng được coi là để có cùng loại nếu chúng chứa tên cùng chức năng (theo bất kỳ trật tự nào) với tham biến tương đương và kết quả loại.Định nghĩa này là dạng lỏng lẻo của tính tương đồng cấu trúc.

10. Dynamic typing in statically typed

LANGUAGES

Có vẻ là kì lạ để bao gồm gõ động và gõ tĩnh trong ngôn ngữ , nhưng có tình huống trong đó kiểu đối tượng không thể được dự đoán vào thời gian biên dịch.Thực ra, có tình huống trong đó chương trình có thể tạo ra loại mới trong tính toán của nó.

Một đề nghị thanh lịch để thoát ra từ các loại tĩnh là giới thiệu một loại tên predeclared động [Abadi 91]. Phương pháp này được sử dụng rộng rãi trong Amber [Cardelli 86].Giá trị của loại này được xây dựng bởi các chức năng đa hình predeclared làm cho động. Chúng được thực hiện như một cặp có chứa một giá trị và mô tả một loại, như thể hiện trong hình 3,68 (trong một cú pháp ML-like).

Figure 3.68

```
Val A = makeDynamic 3;                              1

Val B = makeDynamic "a string";                     2
```

```
Val C = makeDynamic A;                    3
```

Giá trị đặt trong A là 3, và mô tả kiểu của nó là int. Giá trị đặt trong B là "một chuỗi", và mô tả kiểu của nó là chuỗi. Các giá trị được đặt trong C là cặp đại diện A, và mô tả kiểu của nó là động.

Giá trị của loại động có thể được thao tác bên trong biểu thức phân biệt loại cơ bản và gán tên cục bộ đến giá trị thành phần, như trong hình 3.69.

Figure 3.69

```
 val rec          Stringify =   fn      Arg : dynamic =>          1

typecase      Arg                                                2

of            s : string => ’"’ + s + ’"’                        3

|             i : int => integerToString(i)                       4

|             f : ’a -> ’b => "function"                          5

|             (x, y) => "(" + (Stringify makeDynamic x) +         6

", " + (Stringify makeDynamic y) + ")"                            7

|             d : dynamic => Stringify d                          8

|             _ => "unknown";                                     9
```

Stringify là chức năng lấy động - gõ tham biến arg và gửi trả chuỗi phiên bản tham biến. Nó phân biệt thể loại arg trong biểu thức khay chữ

in với mẫu hình để giữ loại và để gán mã nhận đạng cục bộ các thành phần của loại.

Hình 3.70 cho Ví dụ phức tạp hơn thấy rằng xếp lồng khay chữ in biểu thức.Chức năng áp đụng lấy hai tham biến động và gọi cái thứ nhất với cái thứ nhì khi tham biến, kiểm tra ứng dụng như vậy là hợp lệ.

Figure 3.70

```
val rec  Λpply =                                                    1
fn              Function : dynamic =>                               2
fn                   Parameter : dynamic =>                         3
typecase                  Function                                  4
of                        f : 'a -> 'b =>                           5
typecase                       Parameter                            6
of                             p : 'a => makeDynamic f(p);          7
```

Dòng 5 rõ ràng kết gán 'a và ' b sao cho có thể được sử dụng sau này trong dòng 7 khi chương trình kiểm tra tính tương đồng.Dòng 7 cần gọi makeDynamic sao cho gửi trả giá trị của áp dụng (ấy là, động) được biết đến trình biên dịch.Ở mỗi biểu thức khay chữ in, nếu thực tế tại quá trình thi hành không phải là bảo vệ, lỗi đánh máy đã xảy ra.Tôi có thể sử dụng câu lệnh nâng cao hiện bằng một thứ tiếng với ngoại lệ xử lý.

Loại động không vi phạm mạnh đánh máy.Trình biên dịch vẫn biết mỗi giá trị, vì tất cả các loại sẽ không rõ nếu không được gộp lại với nhau như loại động.Kiểm tra quá trình thi hành được cần đến chỉ ở tính bảo vệ của biểu thức khay chữ in.Trong mỗi nhánh, một lần nữa loại tĩnh lại được biết đến.

Nó có thể cho phép lập thời gian cưỡng chế của các loại động. Nếu một giá trị động được sử dụng trong một nơi mà các trình biên dịch không có bất kỳ ý nghĩa áp dụng, nó hoàn toàn có thể cung cấp một typecase phân biệt ý nghĩa mà nó biết làm thế nào để xử lý, như trong hình 3,71.

Figure 3.71

```
in:  write makeDynamic (4 + makeDynamic 6)          1

out: 10 : int                                       2
```

Trong dòng 1, toán tử + không quá tải đối với số nguyên cộng với giá trị động. Trình biên dịch nhận ra thực tế này và chèn một typecase rõ ràng để xử lý một trong những nghĩa mà nó biết, số nguyên cộng với số nguyên. Các predeclared viết chức năng không thể xử lý các loại hình động, vì vậy typecase khác được lắp cho tất cả các loại mà nó có thể xử lý. Nói cách khác, nhập vào được mở rộng để có thể hiện trong hình 3,72.

Figure 3.72

```
typecase    makeDynamic 4 +                         1

typecase    makeDynamic 6                           2
```

```
of        i : int => I                    3
end;                                      4
of                                        5
i : int => write I                        6
r : real => write r                       7
...                                       8
end;                                      9
```

Các biểu hiện typecase trong dòng 2-4 có kiểu int, do đó + trong dòng 1 cũng là xác định. Để viết tham số cho một loại thời gian biên dịch, tôi phải viết chức năng vào typecase ngoài (ở dòng 6-8). Viết chức năng vào các biểu thức typecase tiềm ẩn có thể dẫn đến một sự bùng nổ mã.

Nó là tốt hơn để cưỡng chế trong thời gian chạy, khi các loại thực tế được biết đến với từng loại hình động. Các chương trình trong hình 3,72 sẽ sử dụng số nguyên và in các số nguyên. Thời gian chạy cưỡng chế vẫn hoàn toàn an toàn, mặc dù một số lỗi sẽ không được phát hiện cho đến khi thời gian chạy.

11. Final comments

Các cuộc thảo luận về các loại có nguồn gốc và kích thước là một phần của một vấn đề lớn hơn về việc làm thế nào để hạn chế ngôn ngữ lập trình cần phải ngăn nắp phù hợp với mỹ thuật lập trình.

Một cách để nhìn câu hỏi này [Gauthier 92] là để nhận thấy mặt này thế giới thực là rất giới hạn gõ, một sinh viên của vật lý nhận ra. Không nên thêm cam và táo, nhiều kém hơn vôn và calo. Mặt khác, bộ nhớ của hầu hết các máy tính là hoàn toàn không định kiểu, tất cả mọi thứ được đại diện bởi các bit (tổ chức thành các byte đều không định kiểu hoặc từ). Các ngôn ngữ lập trình đại diện cho một nền tảng cho việc mô tả thế giới thực thông qua máy tính, do đó, nó nằm đúng ở đâu đó giữa những thái cực. Nó cần cân bằng loại bảo mật một cách giản dị.Loại bảo mật yêu cầu mỗi loại có giá trị riêng của mình để trùng khớp thế giới thực. Ví dụ, danh sách chính xác ba yếu tố là khác danh sách của bốn yếu tố. Số nguyên hạn chế, ngay cả con số khác nhau từ các số nguyên không bị giới hạn..Tinh giản hoá yêu cầu loại dễ xác định và loại được hiệu quả đã kiểm tra, tốt nhất là ở thời gian biên dịch. Nó không phải là dễ dàng để bao gồm cân nhắc độ dài hoặc số lý thuyết trong loại mô tả các danh sách và số nguyên, tương ứng. Nó phần lớn là vấn đề về sở thích cá nhân nơi nền tảng này nên trên quang phổ xếp loại từ giới hạn gõ, sử dụng gõ mạnh và có lẽ cung cấp loại với chiều, với gõ động và dễ dàng chuyển đổi kiểu dữ liệu. Những người ủng hộ quan điểm phong cách hạn chế với niềm tự hào cho rõ ràng các chương trình của họ và thực tế là đôi khi họ chạy đúng thời gian đầu tiên. Kiểu hạn chế làm họ hãnh diện đến độ rõ nét của chương trình của họ và việc đôi khi chúng chạy đúng lần đầu.

Nhiều người thích kiểu ràng buộc của ngôn ngữ ada, nhưng một số lại thích kiểu tự do của ngôn ngữ C.Việc thay đổi ngôn ngữ lập trình như là thay đổi một hương vị nào đó, kinh nghiệm của tôi thì ngôn ngữ lập trình như là ngôn ngữ máy,thậm chí cả ngôn ngữ assembler.Sau đó tôi thưởng thức tính phức tạp của SNOBOL. Algol là một mắt mở thực sự

với sự kiểm soát các loại khai báo và cấu trúc của nó. Với tôi một chương trình thanh lịch là một chương trình trong đó nó có thể được đọc lần đầu tiên với một người dùng mới. Tôi thích dùng C nặng bởi vì nó được thực hiện rộng rãi và tôi cần đến cổng các chương trình của tôi trên máy. ML là một ngôn ngữ cho thấy làm thế nào để làm cho chức năng hạng nhất giá trị và làm thế nào để đối phó với đa hình dạng và vẫn còn được gõ mạnh. Loại suy luận làm giảm các lập trình viên của tờ khai nhập cẩn thận. Miranda mở rộng những ý tưởng này với danh sách vô hạn và đánh giá lười biếng. (Ngoài ra còn có một biến thể lười biếng của ML với phần mở rộng tương tự.) Russell thậm chí cho phép một số loại được chế tác theo những cách khá đơn giản. Không gì trong số những ngôn ngữ này thực sự cho phép loại các giá trị tự nhận là hạng nhất. Như một phần mở rộng có thể sẽ yêu cầu kiểm tra kiểu thời gian chạy hoặc mất gõ mạnh. (Các bài tập tìm hiểu khái niệm này.)

Loại hệ thống là một lĩnh vực nghiên cứu hoạt động. Lồng ghép các yếu tố thàn. Ngôn ngữ đa hình như ML, ví dụ, đang được nghiên cứu [Kennedy 94].

Các biến thể SML của ML bao gồm một gia hạn bậc cao thử nghiệm hệ thống mô-đun, trong đó phần chung có thể được tham số bởi (Có thể chung chung) mô-đun khác.

Mặc dù tôi đã cố ý tránh các vấn đề về cú pháp trong chương này, tôi
muốn chỉ ra rằng cú pháp thiết kế chắc chắn ảnh hưởng đến tính dễ dàng mà các lập trình viên có thể học và sử dụng một ngôn ngữ. So sánh, ví dụ, giống loại C và ML :

C	ML
int z	z : int
int (*a)(char)	a : (char -> int) ref
int (*((*b)(int)))(char)	b : (int -> ((char -> int) ref))) ref
int (*c)(int (*)(char))	c : ((char -> int) ref -> int) ref

Mặc dù kiểu biểu thức của C được ngắn hơn, tôi tìm thấy nó khó khăn để tạo ra và để hiểu.

Xem lại bài tập

3,1 khác biệt giữa cách Modula-2 + và Modula-3 kiểu xử lý tương đương đối với các loại có nguồn gốc là gì?

3.2 Nếu hai loại là tương đương nhau,thì nhất thiết cấu trúc phải tương đương?

3,3 Bạn có thể xét cấu trúc tương đương First và Secon trong hình 3,73 không? Tại sao có hoặc tại sao không?

Figure 3.74

```
Type                                    1
First =                                 2
Record                          3
A : integer;                    4
B :             record          5
B1, B2 : integer;               6
end;                            7
end;                            8
Second =                        9
Record                          10
A:              record          11
A1, A2 : integer;               12
end;                            13
B : integer;                    14
        end;                    15
```

3.4 Làm thế nào các kiểu dữ liệu trừu tượng có thể được sử dụng để thực hiện kích thước?

3.5 Tại sao instantiation của một module chung một hoạt động thời gian biên dịch, không phải là một hoạt động thời gian chạy?

3,6 Có những kiểu sắp xếp gì của các loại tương đương mà ML không sử dụng - tên tương đương, kết cấu tương đương, hoặc cái gì khác?

Bài tập

3.7 Có thể đếm giá trị của loại TA trong hình 3.3 (trang 57).

3,8 Cho rằng First và Secon không có cấu trúc tương đương, trong bài tập **3.9** đề xuất một thuật toán để kiểm tra cấu trúc tương đương

3.10 Nói thêm về phạm vi đến ML

3.11 Viết một thủ tục tích lũy trong ML có thể được sử dụng để tổng hợp một danh sách, như đề nghị trên trang 75

3.12 Hiển thị sử dụng hợp lệ của leftProjection, được giới thiệu trong hình 3.52 (trang 88).

3.13 Chương trình quicksort trong ML.

3.14 Hiển thị kiểu dữ liệu như thế nào trong ML có thể cho tôi những ảnh hưởng của f1: ((alt int bool) -> 'a) -> (' a * 'a), như đề xuất trên trang 82.

3.15 Sử dụng các loại năng động trong một khuôn khổ ML tuyên bố một chức năng BuildDeep như vậy mà BuildDeep 2 chức năng sản xuất một kiểu int động ->(Int -> int), BuildDeep 3 tạo ra một chức năng của kiểu int

động ->(Int -> (int -> int)), và vv. Các chức năng sản xuất phải trả lại tất cả các thông sô tổng quát của chúng.

3.16 Tổng hợp các loại ở ML là những giá trị đích thực đầu tiên của lớp. Đó là,tôi có thể xây dựng mọi thứ của loại, hoặc các loại -> int. Quyết định những gì được xây dựng trong các chức năng vào loại loại nên được. Cố gắng giữ ngôn ngữ mạnh mẽ gõ

3.17 Mở rộng ML để có một kiểu biểu thức. Đưa ra các chức năng hợp lý có sử dụng kiểu.Các chức năng này nên có ý nghĩa thời gian chạy (không chỉ là thời gian biên dịch).

3.18 Các loại (trong ý nghĩa ML) ít nhất trong hình 3,66 là gì (trang 96)?

3.19 Io có thể được xây dựng (xem Chương 2) đại diện trong ML?

3.20 Hiển thị hai loại trong Russell được tương đương, nhưng không phải là tương đương với cấu trúc.

3.21 Có phải tất cả các loại trong Russel đều là loại cấu trúc tương đương?

3.22 Russell ngăn ngừa các tác dụng phụ với sự hiện diện của các biến (các loại ref) bằng cách cấm nhập định danh các chức năng từ ánh xạ tới các biến.
Tại sao quy tắc này quan trọng ?

3.23 Russell cũng cấm một khối từ xuất giá trị của loại một tuyên bố tại địa phương trong khối đó. Tại sao quy tắc này quan trọng trong

Russell? Nên ML cũng có một quy tắc? Có thể quy định này được thi hành tại thời gian biên dịch.