Universidade de São Paulo
Instituto de Matemática e Estatística
IME

Trabalho 2 - Os Conceitos da Linguagem Modula-3

Sistemas de tipos e Abstrações
MAC316

Integrantes do grupo:
João Antonio Mazzer Mantovani (nº USP 10812022),
Lucas Henrique Bahr Yau (nº USP 9763832),
Patrícia da Silva Rodrigues (nº USP 11315590),
Sabrina Araújo da Silva (nº USP 12566182),
Samantha Miyahira (nº USP 11797261)

Outubro
2022

Conteúdo

1 Sistemas de tipos (monomórficos, polimórficos e sobrecarregados) e verificação de tipos utilizados na linguagem

1.1 Sistemas de tipos monomórficos e polimórficos

Em linguagens com tipos **monomórficos**, as variáveis e abstrações são definidas somente para apenas um tipo. Por exemplo, se definirmos em C uma variável do tipo CHAR, ela não poderá receber outros valores de tipos que não sejam CHAR, e se definirmos uma pilha para receber INT, não será possível inserir um CHAR nesta pilha. Linguagens muito conhecidas por possuírem um sistema de tipos monomórficos são C e Pascal.

Em linguagens com tipos **polimórficos**, as variáveis e abstrações podem conter/manipular diferentes tipos. Por exemplo, uma lista em Python pode receber qualquer tipo: uma STRING, uma FUNÇÃO ou OBJETOS, e no escopo das funções os parâmetros podem ser tratados de forma igual. Outro exemplo é a operação de inverter uma lista em Python, que pode ser feita independentemente dos tipos dos dados que ela contém. O polimorfismo aumenta a expressividade da linguagem, permitindo, por exemplo, utilizar uma mesma estrutura de dados para diversos tipos diferentes.

Em Modula-3, estruturas de dados como listas, filas e árvores são polimórficas, ou seja, para cada estrutura é possível utilizar um mesmo conjunto de operações para qualquer tipo. Por exemplo, um procedimento para empilhar ou desempilhar elementos de uma pilha pode ser usado tanto para inteiros como para reais, dando flexibilidade à linguagem e permitindo a reutilização de código.

1.2 Sistemas de tipos sobrecarregados

Os sistemas de tipos sobrecarregados são aqueles em que um mesmo operador ou função pode ser utilizado para diferentes tipos dentro da mesma variável. Essa sobrecarga pode ocorrer de duas formas distintas: dependente de contexto, onde o resultado da operação depende da declaração explícita do tipo, ou independente de contexto, onde o resultado da operação é computado e automaticamente associado a um tipo.

Em Modula-3, há a sobrecarga de operadores aritméticos e de funções. No caso dos operadores aritméticos, essa sobrecarga depende de contexto. Por exemplo, a divisão de um INTEGER com um FLOAT é uma expressão ilegal, devendo um dos operadores ser convertido para o tipo do outro. Algumas operações aritméticas, como +, -, * e / também possuem sobrecarga dependende de contexto com o tipo SET.

Há também as operações com sobrecarga independente de contexto. É o caso das operações lógicas NOT, OR e AND. Essas sempre retornam um valor booleano de acordo com a verificação sobre os operandos.

1.3 Verificação de tipos

A verificação de tipos é a atividade de assegurar que os operandos sejam de tipos compatíveis. Um tipo é equivalente (ou compatível) quando ele é válido para o operador ou tem permissão, segundo as regras da linguagem, para ser convertido automaticamente para um tipo válido (coerção de tipos). Um erro de tipo geralmente é o uso indevido de operadores com operandos.

Em Modula-3, sendo a vinculação de tipos feita estaticamente, a verificação de tipos também é estática, ou seja, as variáveis do programa são verificadas em tempo de compilação. Além disso, a linguagem é *type safe*, ou seja, um objeto sempre é do tipo esperado e o compilador consegue determinar se uma operação pode ser feita nele. Caso haja uma falha, haverá um erro e uma dica de como resolver o problema.

2 Abstrações

2.1 Abstração de processos

Sob o ponto de vista computacional, uma abstração de processos deve "esconder"os passos computacionais necessários para solucionar o problema. Comportamentos com efeito colateral, ou seja, em que alteram um estado em outra parte do programa, podem ser abstraídos por procedimentos.

2.1.1 Procedimentos

As abstrações de procedimentos são realizadas por uma série de comandos que provoca mudanças sucessivas nos valores de suas variaveis.

Um procedimento em Modula-3 consiste em três itens:

- o *corpo*, que é uma declaração,
- a *assinatura*, que especifica os argumentos formais do procedimento, o tipo de resultado e o conjunto de levantamentos (o conjunto de exceções que o procedimento pode gerar),
- o *ambiente*, que é o escopo com relação ao qual os nomes de variáveis no corpo serão interpretados.

Em Modula-3, existem duas formas de declaração de procedimentos:

PROCEDURE id sig = B id

PROCEDURE id sig

No qual id é um identificador, sig é uma assinatura de procedimento e B é um bloco. Em ambos os casos, o tipo de id é o tipo de procedimento determinado por sig. A primeira forma é permitida apenas em módulos; a segunda forma é permitida apenas em interfaces.

Um procedimento que retorna um resultado é chamado de procedimento de função; um procedimento que não retorna um resultado é chamado de procedimento adequado. Um procedimento de nível superior é um procedimento declarado no escopo mais externo de um módulo. Qualquer outro procedimento é um procedimento local.

Uma declaração de tipo de procedimento tem o formato:

TYPE T = PROCEDURE sig
onde sig é uma especificação de assinatura, que tem a forma:
$(\text{formal}_1; ...; formal_n) : R\ RAISES\ S$

Os nomes dos parâmetros e os padrões afetam o tipo de um procedimento, mas não seu valor. Por exemplo, considere as declarações:

```
PROCEDURE P(txt: TEXT := "P") =
    BEGIN
    Wr.PutText(Stdio.stdout, txt)
    END P;
VAR q: PROCEDURE(txt: TEXT := "Q") := P;
```

Agora P = q é TRUE, mas P() imprime "P"e q() imprime "Q". A interpretação dos parâmetros padrão é determinada pelo tipo de procedimento, não pelo seu valor; a atribuição q := P altera o valor de q, não seu tipo.

Suponha que temos um tipo de array em nosso programa:

TYPE DataArray = ARRAY [0 .. size - 1] OF REAL;

e também um tipo de procedimento

TYPE Modifier = PROCEDURE (VAR R: REAL);

Então podemos ter vários arrays do tipo DataArray e também vários procedimentos do tipo Modifier. Suponha que em vários pontos do programa precisemos aplicar algum procedimento do tipo

Modifier a cada um dos reais em um desses arrays e que em diferentes pontos do programa precisemos aplicar procedimentos diferentes a um desses arrays. Este processo de aplicação de algum procedimento para cada elemento de algum array pode ser abstraído de nosso sistema declarando um procedimento como segue:

```
PROCEDURE Apply(VAR A: DataArray; Proc: Modifier) =
BEGIN
    FOR i := 0 TO size - 1 DO
      Proc(A[i]);
    END;
END Apply;
```

2.2 Parâmetros

A passagem de parâmetro se dá ou por cópia ou por referência. A passagem por cópia implica na criação de um novo endereço de memória responsável por armazenar a cópia do valor passado. A passagem por referência não cria um novo endereço de memória, pois o que será manipulado será o mesmo endereço de memória da variável original que foi referenciada.

Um tipo de referência é rastreado ou não rastreado. Em Modula-3, O tipo REF T é o tipo de todas as referências rastreadas para variáveis do tipo T; o tipo UNTRACED REF T é o tipo de todas as referências não rastreadas a variáveis do tipo T.

```
NULL <: REF T <: REFANY
NULL <: REF T NÃO RASTREADA <: ENDEREÇO
```

A seguir temos um exemplo da utilização de parâmetro na linguagem Modula-3:

Palindromo

```
MODULE Palindromo;
IMPORT Text;
PROCEDURE isPalindromo(string: TEXT): BOOLEAN =
  VAR len := Text.Length(string);
  BEGIN
    FOR i:= 1 TO len DIV 2 - 1 DO
      IF Text.GetChar(string, i)  Text.GetChar(string, (len - i - 1)) THEN
        RETURN FALSE;
      END;
    END;
    RETURN TRUE;
  END isPalindrome;
END Palindrome.
Parâmetro passado: string
Tipo: TEXT
```

2.3 Abstração de tipos

A abstração de tipos tem a finalidade de diminuir a complexidade dos problemas por meio de módulos que podem conter os vários tipos de abstrações: de dados e de processos (tipos, variáveis, constantes, funções, etc.). O encapsulamento está relacionado com o agrupamento de abstrações dentro dos módulos. Além disso, o conceito de ocultação de informação é utilizado para diferenciar as abstrações que são usadas apenas dentro dos módulos com as que devem ser usadas externamente, isto é, informações visíveis das não-visíveis externamente.

2.3.1 Tipos Genéricos

Os tipos genéricos são tipos de dados abstratos que correspondem aos módulos genéricos em Modula-3. São representados com o prefixo INTERFACE ou MODULE com a palavra-chave GENERIC. Podemos exemplificar esse tipo com uma pilha (stack):

```
GENERIC INTERFACE GenericStack(Element);
(* Here Element.T is the type to be stored in the generic stack. *)
TYPE
    T = Public OBJECT;
    Public = OBJECT
    METHODS
        init(): TStack;
        format(): TEXT;
        isEmpty(): BOOLEAN;
        count(): INTEGER;
        push(elm: Element.T);
        pop(VAR elem: Element.T): BOOLEAN;
    END;
END GenericStack.
```

fonte: https://en.wikipedia.org/wiki/Modula-3#Object-oriented

2.3.2 Orientação a Objetos em Modula-3

As técnicas utilizadas em orientação a objetos podem ser utilizadas em Modula-3 por meio de módulos e genéricos. Uma classe é um tipo objeto no Modula-3 e é declarado como OBJECT o qual possui a mesma sintaxe que RECORD. Entretanto, OBJECT é um tipo de referência e RECORD não, visto que é similar a structs em C.

Em Modula-3, é convenção utilizar T como tipos exportados, ou seja, um tipo público que expõem os métodos dados. Um exemplo está logo abaixo:

```
INTERFACE Person;

TYPE T <: Public;
   Public = OBJECT
   METHODS
      getAge(): INTEGER;
      init(name: TEXT; age: INTEGER): T;
   END;

END Person.
```

fonte: https://en.wikipedia.org/wiki/Modula-3#Object-oriented

3 Referências

- http://lucacardelli.name/papers/modula3typesystem.a4.pdf
- http://csis.pace.edu/ bergin/M3text/Ch2M3.html#RTFToC4