Resumo para prova 3 de LP

Sistemas de Tipos

1. **O Projeto de um Verificador de Tipos**

   O verificador de tipos é um componente que usa as regras da linguagem para atribuir tipos às construções sintáticas do programa. Ele garante que os tipos estão sendo usados corretamente, evitando erros de tipo durante a execução do programa.

2. **Monomorfismo**

   Monomorfismo refere-se a um sistema de tipos onde cada variável, constante, resultado de função e parâmetro formal deve ter um tipo específico definido.

   ### Vantagens

   - A verificação de tipos é simples e direta.

   ### Desvantagens

   - Dificulta a reutilização de código, pois algoritmos genéricos não podem ser facilmente implementados.

   ### Exemplo

   - No Pascal, algumas funções como write e eof não seguem o monomorfismo estritamente, introduzindo polimorfismo e sobrecarga.

3. **Sobrecarga**

   Sobrecarga é a capacidade de associar múltiplas funções ou operadores ao mesmo identificador, dependendo do contexto de uso.

Tipos de sobrecargas

- Independente de Contexto: Funções são diferenciadas apenas pelo tipo de seus parâmetros.
- Dependente de Contexto: Além dos parâmetros, o tipo de retorno ou contexto é usado para diferenciar as funções.

Exemplo

Em uma linguagem que permite sobrecarga, x := 7 / 2 pode resultar em um número real enquanto n := 7 / 2 pode resultar em um número inteiro, dependendo dos tipos de x e n.

4. **Polimorfismo**

   Polimorfismo permite que funções ou tipos operem de forma uniforme em diferentes tipos de dados.

   Polimorfismo Paramétrico

   Usa variáveis de tipo em vez de tipos específicos.

   Exemplo

   Em ML, a função fun segundo (x: σ, y: τ) = y pode operar com qualquer tipo para σ e τ.

5. **Tipos Parametrizados**

   Tipos parametrizados são tipos que aceitam outros tipos como parâmetros.

   Exemplo em Pascal: Tipos como file, set e array são tipos parametrizados predefinidos.

   Exemplo em ML: type τ par = τ * τ define um par de qualquer tipo.

6. **Politipos**

   Os politipos são tipos que contêm uma ou mais variáveis de tipo. Eles são uma forma avançada de tipagem que permite a criação de funções e estruturas de dados muito mais flexíveis e reutilizáveis.

Definição

```
•   Politipo: Um tipo que contém variáveis de tipo, repr
esentando tipos desconhecidos ou genéricos. Por exemplo,
a assinatura de uma função \\sigma \\times \\tau \\right
arrow \\tau onde \\sigma e \\tau são variáveis de tipo.
```

Derivação de Tipos

```
•   Um politipo gera uma família de tipos específicos at
ravés da substituição de suas variáveis de tipo por tipo
s concretos.
•   Exemplo: O politipo \\tau \\rightarrow \\tau pode re
presentar funções como Inteiro -> Inteiro, String -> Str
ing, Boolean -> Boolean, etc.
```

Exemplo de Politipos

```
•   Lista(\\tau): Uma lista genérica que pode conter ele
mentos de qualquer tipo \\tau.
•   Tipo \\tau \\rightarrow \\tau: Pode ser visto como a
interseção de todas as funções que tomam um tipo \\tau e
retornam o mesmo tipo \\tau.
```

7. Inferência de Tipo

   Inferência de tipo é o processo pelo qual o compilador deduz o tipo de uma expressão automaticamente, sem necessidade de anotação explícita pelo programador.

   Inferência de Tipo Monomórfico

   • **Exemplo:** Em ML, ao escrever fun par(n) = (n mod 2 = 0), o compilador infere que n é do tipo integer e o retorno é boolean.

   Inferência de Tipo Polimórfico

   • **Exemplo:** Em ML, ao escrever fun length(l) = case l of nil ⇒ 0 | cons(h, t) ⇒ 1 + length(t), o compilador infere que l é do tipo Lista(\(\tau\)) e a função retorna um integer.

8. **Coerção**

   Coerção é a conversão implícita de um tipo para outro, realizada automaticamente pela linguagem de programação.

   Definição

   - **Coerção:** Um mapeamento implícito de valores de um tipo para outro tipo diferente.
   - **Exemplo:** Em Ada, a coerção deve ser explícita, como em Float(x) para converter x para um tipo Float.

9. **Subtipos e Heranças**

   Subtipos permitem que um tipo herde operações e propriedades de outro tipo, promovendo a reutilização de código e a flexibilidade na programação.

   Definição

   - **Subtipo:** Um tipo S é subtipo de T se S for um subconjunto de T. Valores de S podem ser usados onde valores de T são esperados.
   - **Herança:** Associada à Programação Orientada a Objetos (POO), permite que tipos derivados herdem operações de seus tipos base.

   Exemplo em Pascal

   - Pascal permite subtipos através da definição de intervalos. Por exemplo, um intervalo de inteiros de 1 a 10 pode ser considerado um subtipo do tipo inteiro.

Conclusão

Entender sistemas de tipos, incluindo conceitos como monomorfismo, sobrecarga, polimorfismo, politipos, inferência de tipo, coerção e subtipos, é essencial para escrever programas robustos e eficientes. Cada um desses conceitos contribui para a flexibilidade e a segurança da linguagem de programação, permitindo que os desenvolvedores criem códigos mais reutilizáveis e menos propensos a erros.

Sequenciadores

Sequenciadores são construções que alteram o fluxo de controle normal de um programa, permitindo a criação de fluxos de controle mais genéricos e flexíveis.

1. Saltos (Jumps)

Saltos são transferências explícitas de controle de um ponto do programa para outro, geralmente utilizando o comando goto seguido de um rótulo (label). Por exemplo:

```
L1:
…
goto L1
```

2. Escapes

Escapes são sequenciadores que terminam a execução de um comando textualmente fechado, transferindo o controle diretamente para fora do comando. Permitem fluxos de controle com uma única entrada e múltiplas saídas.

Exemplo em C:

```
while (x <= 100) {
if (x < 0) {
printf("Erro: valor negativo");
break;
}
...
scanf("%d", &x);
}
```

Exemplo em Ada:

```
outer_loop: loop
  inner_loop: loop
    exit outer_loop when condition;
  end loop inner_loop;
end loop outer_loop;
```

Escapes importantes incluem return, que termina a execução de um procedimento, e halt, que termina a execução do programa inteiro.

3. Exceções

Exceções tratam de condições anômalas ou erros que ocorrem durante a execução de um programa, como divisão por zero ou erros de entrada/saída. Em vez de interromper a execução, o controle é transferido para um tratador de exceções.

Exemplo em Java:

```
try {
  // código a ser executado
} catch (NumberFormatException nfe) {
  // tratamento da exceção
} finally {
  // código a ser executado mesmo que uma exceção seja lançad
}
```

Tratamento de Exceções:

- PL/I foi a primeira linguagem a incorporar tratamento de exceções.
- Ada melhorou o conceito, permitindo associar tratadores específicos para diferentes exceções e comandos.

Exemplo em Java:

```
public class ExemploDeExcecao {
  public static void main(String[] args) {
    String var = "ABC";
    try {
```

```
      Integer i = new Integer(var);
      System.out.println("A variável i vale " + i);
    } catch (NumberFormatException nfe) {
      System.out.println("Não é possível atribuir a string " +
                         "A seguinte mensagem foi retornada:\
    }
  }
}
```

Este exemplo ilustra o uso do bloco try-catch para capturar e tratar exceções específicas, garantindo a robustez do programa.

Conclusão

Sequenciadores, incluindo saltos, escapes e exceções, são ferramentas poderosas que permitem o controle preciso do fluxo de execução em um programa. No entanto, seu uso deve ser cuidadoso para evitar a criação de código difícil de entender e manter. A compreensão desses conceitos é crucial para escrever programas eficientes e robustos.