1. FROM UNTYPED TO TYPED UNIVERSES

1.1. Organizing Untyped Universes

Para compreendermos os diferentes tipos que uma Linguagem de Programação contém, primeiramente devemos nos perguntar: “Por que os tipos são necessários para as LPs”. Conseguimos responder está pergunta através da evolução da matemática e da computação do universo não tipado para o tipado, vamos analisar e dividir este universo em 4 partes:

a) *Strings de bits na memória do computador:*

O mais concreto dos tipos. Considerado “não tipado” porém representado como cadeias de bits: caracteres, números, ponteiros, dados estruturados, programas.

O pedaço de memória bruta, geralmente não temos como dizer o que está sendo representado este significado é dado por um interpretador externo.

b) $Expressões - S\ em\ LISP\ puro:$

Baseado nas cadeias de bits as Expressões-S o LISP é um pouco mais estruturado (átomos e células).

c) $Expressões - \lambda\ em\ calculo\ \lambda:$

No cálculo λ, tudo é uma função. Números, estruturas de dados e até cadeias de bits podem ser representados por funções apropriadas

d) *Conjunto da teoria de conjuntos:*

Neste tópico , tudo é um elemento ou um conjunto de elementos e / ou outros conjuntos.

A partir da classificação do universo não tipado podemos inferir tipos para integrarem gerando um universo tipado por meio da categorização objetos de acordo com seu uso e comportamento.

1.2. Static and Strong Typing

Um dos principais objetivos de uma LP ao implementar um universo tipado é eliminar inconsistências. O compilador conferir se o tipo representado está de acordo para evitar incoerências e erros intencionais ou não no sistema como se fosse uma camada protetora.

A técnica descrita acima e realizada para facilitar a execução das operações esperadas em objetos de dados além de erros no programa em funcionamento serem todos lógicos, uma vez que erros de tipos não são passados desapercebido pelo compilador. No entretanto o programa impõe ao programador uma disciplina de programação que torna os programas mais estruturados e fáceis de ler.

1.3. Kinds of Polymorphism:

Linguagens tipadas convencionais, como: Pascal C Java tem a ideia que cada valor e variável pode ser interpretado como sendo de um e apenas um

tipo. Esse tipo de classificação nas Linguagens de Programação tem o nome de Linguagens monomórficas.

Entretanto as funções polimórficas são funções que os operandos (parâmetros reais) podem ter mais de um tipo. O Cientista da computação **Christopher S. Strachey** em 1967 distinguiu informalmente o Polimorfismo entre dois tipos principais.

- Polimorfismo universal
  - Parametric.
  - Inclusion
- Ad-Hoc.
  - Overloading
  - Coercion

1.4. The Evolution of Types in Programming Languages

As mais diversas Lps foram evoluindo ao longo dos anos FORTRAN trabalhava com a distinção de pontos flutuantes e inteiros. ALGOL 60 tornou esta distinção explícita introduzindo declarações de identificador redundantes para variáveis reais e booleanas inteiras.

Pascal fornece uma extensão mais limpa de tipos para registros e ponteiros de arrays, bem como tipos definidos pelo usuário.

LISP, que descreve o interpretador de linguagem LISP em LISP, mas é muito mais complexa. Ele descreve uma coleção de mais de 75 tipos de objetos de sistema relacionados por uma hierarquia de herança.

1.5. Type Expression Sublanguages

À medida que o conjunto de tipos de uma linguagem de programação se torna mais rico e seu conjunto de tipos definíveis torna-se infinito, torna-se útil definir o conjunto de tipos por uma sublinguagem de expressão de tipo.

A capacidade de expressar relações entre tipos envolve alguma capacidade de realizar cálculos em tipos para determinar se eles satisfazem a relação desejada. Tais cálculos poderiam, em princípio, ser tão poderosos quanto os cálculos executados em valores.

1.6. Preview of Fun

Fun é uma linguagem baseada em cálculo lambda que enriquece o cálculo X tipado de primeira ordem com recursos de segunda ordem projetados para modelar polimorfismo e linguagens orientadas a objetos.

As extensões sintáticas da expressão de tipo sublinguagem determinadas por esses recursos podem ser resumidas da seguinte forma:

Type ::= . . . (QuantifiedType QuantifiedType ::= V

A. Type 1 Universal Quantification 3

A. Type 1 Existential Quantification V

AEType. Type 1 3Adype. Type Bounded Quantification

Tipos universalmente quantificados são eles próprios tipos de primeira classe e podem ser parâmetros reais em tal substituição. A quantificação existencial enriquece os recursos de primeira ordem, permitindo tipos de dados abstratos com representação oculta.

2. THE X-CALCULUS

2.1 The Untyped X-Calculus

A evolução de universos não tipados para universos tipados pode ser ilustrada pelo cálculo X, inicialmente desenvolvido como uma notação não tipada para capturar a essência da aplicação funcional de operadores a operandos. As expressões no cálculo X têm a seguinte sintaxe (usamos diversão em vez do X tradicional para trazer à tona a correspondência com as notações da linguagem de programação):

2.2 The Typed X-Calculus

O cálculo X digitado é como o cálculo X, exceto que cada variável deve ser explicitamente digitada quando introduzida como uma variável limitada. Assim, a função sucessora no cálculo X digitado tem a seguinte forma:

value succ = fun (x: Int) x+1

2.3 Basic Types, Structured Types, and Recursion

O cálculo X digitado é geralmente acrescido de vários tipos de tipos básicos e estruturados.Formados por tipos basicos Unit Bool Int Real