Projeto da Linguagem de Programação

Lucas Virgili

Sumário

1 Fase 1 **1**
1.1 Domínio . . . 1
1.1.1 Introdução . . . 1
1.1.2 Descrição do domínio . . . 2
1.2 Proposta da linguagem de programação . . . 3
1.3 Elementos essencias à linguagem . . . 3

2 Fase 2 **4**
2.1 Tipos de dados . . . 4
2.1.1 Tipos primitivos . . . 4
2.1.2 Tipos compostos . . . 4
2.2 Expressões . . . 4
2.2.1 Literais . . . 4
2.3 Funções . . . 5
2.4 Ordem de avaliação . . . 5
2.5 Comandos . . . 5
2.5.1 Declaração . . . 5
2.5.2 Atribuição . . . 5
2.6 Estruturas de controle de fluxo . . . 6
2.6.1 Condicionais . . . 6
2.6.2 Repetição . . . 6
2.7 Vinculação . . . 6
2.7.1 Forma de vinculação . . . 6
2.7.2 Tempo de vinculação . . . 6
2.8 Sistema de tipos . . . 7

1 Fase 1

1.1 Domínio

1.1.1 Introdução

Há aproximadamente 2400 anos, Zeno de Elea abalou as fundações da matemática da época através da proposição de diversos paradoxos. Um deles é muito famoso:

> Um corredor nunca pode terminar uma corrida, já que para isso, ele primeiro tem que andar metade do percurso, em seguida um quarto, depois um oitavo, e assim por diante, *ad infinitum*.

Após 2000 anos, matemáticos dos séculos XII e XIII deram início à teoria de séries infinitas. Nessa teoria, a noção usual de soma, válida para conjuntos finitos, é expandida para coleções infinitas.

Dessa forma, o "paradoxo" de Zeno foi selecionado, já que, naquele contexto, a soma que representa as "etapas" que o corredor deve percorrer é conhecida

$$\frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{2^n} + \ldots \tag{1}$$

e seu resultado é 1.

1.1.2 Descrição do domínio

A teoria de séries e sequências não tem seu uso limitado a mostrar como gregos mortos estavam errados; ela é extremamente usada. Por exemplo, em análise, utiliza-se sequências de funções para demonstrar os teoremas de convergência monótona de Lesbegue. Esse teorema é importantíssimo em probabilidade, por exemplo.

Definimos sequências e séries abaixo.

Sequência: Uma função f cujo domínio é o conjunto dos inteiros positivos $1, 2, 3, \ldots$ é uma sequência infinita. O valor $f(n)$ é o enésimo *termo* da sequência.

Série: Dada uma sequência, podemos gerar uma nova sequência somando termos sucessivos. Logo, se temos uma sequência

$$a_1, a_2, \ldots, a_n, \ldots \tag{2}$$

Podemos gerar as seguintes "somas parciais":

$$s_1 = a_1, s_2 = a_1 + a_2, s_3 = a_1 + a_2 + a_3 \tag{3}$$

e assim continuarmos até a enésima soma parcial:

$$s_n = a_1 + a_2 + a_3 + a_4 + \ldots + a_n = \sum_{i=1}^{n} a_i \tag{4}$$

A sequência s_n das somas parciais é chamada de *série infinita* ou, simplesmente *série*, e é denotada por

$$a_1 + a_2 + a_3 + \ldots, \text{ ou } \sum_{i=1}^{\infty} a_i \tag{5}$$

Informalmente, dizemos que uma sequência converge se existe uma quantidade L para a qual a sequência se aproxima o quanto quisermos[1]. Uma série, então, converge se sua sequência s_n converge.

[1]Formalmente, se para qualquer $\epsilon > 0$, existe um um número positivo N tal que $|f(n) - L| < \epsilon$ para qualquer $n \geq N$.

É comum, enquanto estamos trabalhando com sequências ou séries, escrevermos programas em uma linguagem como C para avaliar se uma sequência ou série converge.

1.2 Proposta da linguagem de programação

Para este projeto, propomos desenvolver uma linguagem de programação que permita a declaração de sequências e séries, bem como analisar a convergência das mesmas através de métodos conhecidos.

Por exemplo, será possível para o programador definir uma sequência e operar sobre elas:

```
seq s
s(n) = 1 / n ^ 2
s(3) = 0.125
series(s, 3) = 0.875 ## calcula s_3
sequence_converges(s, 0.000001)
## -> (true, 0)
## Se a sequencia converge com uma precisao de 10 ^ (-5)


series_converges(s, 0.00001)
## -> (true, 1)
## Se a serie converge com uma "precisao" de 10 ^ (-5)
```

1.3 Elementos essencias à linguagem

Os seguintes elementos são fundamentais para a linguagem:

1. **Declarações**
   Nesta linguagem, as "variáveis" serão as sequências. Como visto no exemplo acima, o programador poderá declarar sequências utilizando a palavra reservada **seq**.

2. **Operadores**
   Sejam a_n e b_n duas sequências convergentes. É fácil mostrar que a série

   $$\sum_{n=1}^{\infty}(\alpha a_n + \beta b_n E) \tag{6}$$

   também converge e seu limite é dado por

   $$\alpha \sum_{n=1}^{\infty} a_n + \beta \sum_{n=1}^{\infty} b_n \tag{7}$$

   Assim, podemos multiplicar séries convergentes por constantes numéricas e também podemos somar e subtrair séries convergentes. Logo, a linguagem irá fornecer os operadores soma $(+)$ e subtração $(-)$ entre séries e o operador produto $(*)$ entre uma constante e uma série.

3. **Funções**
   A linguagem oferecerá ao programador as seguintes funções:

Função	O que ela calcula
series(sequencia, n)	calcula a enésima soma parcial de sequencia
sequence_converges(sequenca, precisao)	diz se sequência converge com precisão precisao
series_converges(serie, precisao)	diz se a série converge com precisão precisao

2 Fase 2

2.1 Tipos de dados

2.1.1 Tipos primitivos

Os seguintes tipos de dados primitivos serão inclusos na linguagem:

- **Inteiros**: Como vimos na definição de sequências acima, estas são indexadas por inteiros positivos. Logo, é natural que a linguagem ofereça inteiros para o programador.
- **Reais**: O uso de números não inteiros também é fundamental. Assim, a linguagem utilizará ponto flutuante para representar tais valores.
- **Booleanos**: Como vimos no exemplo acima, pretende-se que as funções de convergência retornem um par (booleano, limite) e, caso o valor booleano seja verdadeiro, limite é o valor para qual a série ou sequência converge. Dessa forma, é necessário que haja o tipo booleano para armazenar os valores verdadeiro ou falso.

2.1.2 Tipos compostos

Não parece necessário oferecer nenhum tipo composto na linguagem.

2.2 Expressões

2.2.1 Literais

Para denotarmos o valor dos tipos descritos acima, usaremos as notações usuais: 1 denota o inteiro de valor um e 1.0 denota o valor real um.

2.3 Funções

A linguagem oferecerá as funções trigonométricas usuais, como $\sin, \cos, \tan$, bem como logaritmos, exponenciação e raíz quadrada.

Não só, os operadores aritméticos usuais, $+, -, /$ e $*$ serão fornecidos para manipularmos inteiros e reais. Além disso, como dito acima, séries convergentes continuam convergentes quando multiplicadas por um número ou somadas a outra série convergente. Logo, os operadores $+$ e $-$ serão também definidos sobre séries e o operador $*$ será definido sobre um escalar e uma série.

2.4 Ordem de avaliação

A ordem de avaliação será a usual da matemática: com maior precedência, temos operações delimitadas por parênteses, seguido de aplição de funções, depois a multiplicação e divisão com igual precendência e, por fim, por soma e subtração.

Por exemplo a expressão

$$1 + 5 * 2\hat{}2 \tag{8}$$

será avaliada como 21. Por sua vez,

$$7 + 4 * 3 \tag{9}$$

é igual a 19. Já

$$(7 + 4) * 3 \tag{10}$$

é avaliada como 33.

2.5 Comandos

2.5.1 Declaração

Haverá três comandos de declaração: um para sequências, **seq**, um para variáveis inteiras e um para variáveis reais:

```
seq s; # declara uma sequencia s
int i; # declara um inteiro a
real pi; # declara um valor real pi
```

2.5.2 Atribuição

Atribuição será feita com o operador $=$. Contudo, quando declararmos uma sequência, precisamos identificar qual é a "incógnita":

```
real pi;
pi = 3.14;

seq s;
s(n) = pi * (1 / n)
```

2.6 Estruturas de controle de fluxo

2.6.1 Condicionais

A princípio, não vemos necessidade em incluir condicionais "usuais" como if... then ... else na linguagem.

2.6.2 Repetição

Dado que é bastante comum queremos calcular "os primeiros 10 termos da sequência" ou "os termos para n valendo de 10 a 20", a linguagem incluirá um comando "take", cujo uso está exemplificado abaixo:

```
seq s;
s(n) <- 1 / ( n ^ 2 )

take(s, 1, 10); # Imprime os 10 primeiros termos se s
take(s, 10, 20); # Imprime os termos para n de 10 a 20
```

Incluimos o comando take como um comando de repetição pois em uma linguagem como C, um while seria análogo:

```c
double s(double n) {
    return 1 / pow(n, 2);
}

int main() {
    // equivalente a take(s, 1, 10)
    int i = 1;
    while ( i <= 10 ) {
        printf("%lf ", s(i));
    }
    printf("\n");
    return 0;
}
```

2.7 Vinculação

2.7.1 Forma de vinculação

A linguagem só irá permitir associação explícita, como visto nos exemplos acima.

2.7.2 Tempo de vinculação

A vinculação será estática, já que não vemos necessidade em alterar a mesma durante a execução do programa.

2.8 Sistema de tipos

A linguagem terá sobrecarga independente do contexto, já que o operador soma, por exemplo, pode ser tanto usado para somar valores numéricos como para somar séries.

Haverá coerção para tipos válidos quando necessário. Por exemplo, a função sin recebe um valor real. Se um inteiro for passado como parâmetro, ele será promovido. Contudo, um erro será disparado caso somemos um inteiro a uma série, por exemplo.

Por fim, o único erro de tipos possível será se declararmos uma variável numérica e depois tentarmos atribuir a ela uma sequência.