Insper

APS - Lógica da Computação
Texscript

Caio Emmanuel

Dezembro/2022

Motivação

Eu sempre quis uma linguagem de programação que pudesse escrever o mais próximo possível de fórmulas matemáticas. Onde eu pudesse usar símbolos matemáticos como o próprio símbolo de desigual (≠), ou o símbolo de igual fosse realmente uma comparação e não uma atribuição. O símbolo de união (U) poderia simbolizar concatenação, etc…

Uma linguagem que eu sempre gostei para escrever fórmulas (e a mais popular) é o LaTex. Entretanto, ele é puramente declarativo, ou seja, ainda que eu possa declarar fórmulas e formatar estas, não existe nenhuma lógica por trás. Por isso eu criei o TexScript que tenta adicionar uma lógica por trás do LaTex e executar código tex.

Características

O maior ponto do texscript é que podemos escrever o mais próximo possível da linguagem LaTex e portanto descrever o código como um conjunto de fórmulas.

Por exemplo, para atribuir funções em matemática fazemos:

funcao : domínio → contra-domínio

Que em LaTex seria:

$funcao : conjunto x conjunto ... \to conjunto x conjunto ...$

Em texscript o mesmo seria apenas:

funcao : argumento x argument... bloco_de_codigo

Isso porque a segunda parte, o contradomínio, traz a ideia de retorno esperado da função, que nesse caso poderia ser o tipo, mas o texscript não é fortemente tipada, já que LaTex, claramente, não existem tipos

Os blocos são delimitados por **\[** e **\]** ao contrário do popular **$$** dos blocos de LaTex. Isso porque esses delimitadores são aceitos pela maioria dos compiladores de Tex e o duplo cifrão traria problemas para identificar abertura e fechamento de blocos dentro de blocos.

A atribuição é feita através de ← (**\gets)** ao contrário do símbolo =, isso pois esse símbolo agora vai ter seu papel original: comparar duas fórmulas.

As condições não mudam, infelizmente, isso porque as estruturas matemáticas que representam condições (como funções por partes ou operadores lógicos) em LaTex todos trazem problemas para serem reconhecidos.

O while agora vai ser uma somatória (Σ, **\sum**), pois passa a ideia de uma iteração.

A divisão será feita com a função **\frac** do LaTex. Da seguinte forma:
\frac{NUMERADOR}{DENOMINADOR}

E a última característica principal é que agora os símbolos vão ser escritos como o LaTex escreve estes, por exemplo, negação será **\neg**, and será **\land**, or será **\lor**, concatenação será **\cup**, multiplicação será **\cdot**.

Curiosidades

- Apesar de não ser perfeito, o código LaTex usado para rodar a lógica é bem parecido com o que monta as fórmulas.
- Como foi visto, muitas fórmulas em LaTex têm uma contra-barra no início (\ ), isso foi um problema já que o Python reserva vários tokens que se iniciam com contra-barra também (\n, \t, \s, \g, \f…). E por isso, foi preciso introduzir uma camada a mais de pré-processamento para substituir essa aparições por outros caracteres. Isso não muda nada no tokenizer, além de termos que ler um caractere que não é exatamente o que o usuário digitou.
- LaTex não se dá muito bem com texto corrido, então os nomes das funções aparecem esquisitos em vários compiladores.

Exemplos

Reescrevi os exemplos da disciplina em tex e aqui estão os resultados.

Veja código fonte, compilado em LaTex e passado no compilador:

```
Main
\[
  % single var
  x_1 \gets 1
  Write(x_1)
\]
```

Main

$x_1 \leftarrow 1$

$\text{Write}(x_1)$

```
❯ python main.py assets/codes/basic_test.tex
1
```

```
Main
\[
  % All bool and int operations
  y \gets 2
  z \gets (y \eq 2)
  Write(y + z)
  Write(y - z)
  Write(y \cdot z)
  Write(\frac{y}{z})
  Write(y \eq z)
  Write(y < z)
  Write(y > z)
\]
```

Main

$$y \leftarrow 2$$

$$z \leftarrow (y2)$$

$$Write(y+z)$$

$$Write(y-z)$$

$$Write(y \cdot z)$$

$$Write(\frac{y}{z})$$

$$Write(yz)$$

$$Write(y<z)$$

$$Write(y>z)$$

```
❯ python main.py assets/codes/test_int_operations.tex
3
1
2
2
0
0
1
```

Aqui eu não achei um bom compilador que não misturasse as funções com fórmulas, resultado no próximo slide

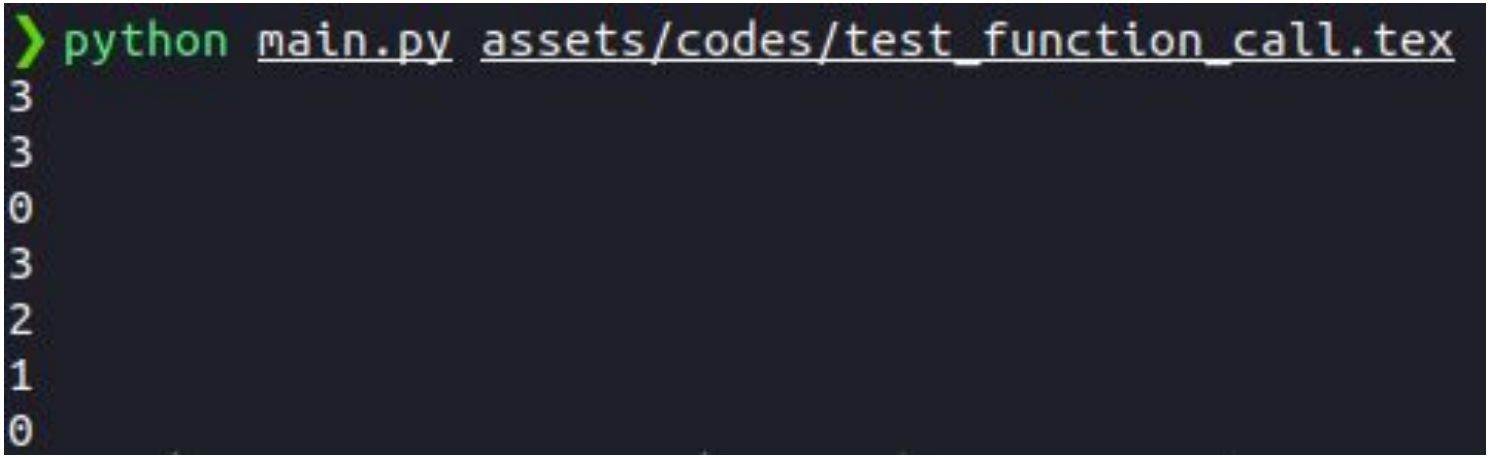
```
soma : x \times y
\[
    return x + y
\]

Main
\[
x_1 \gets 2
x_1 \gets soma(1, x_1)

x_1 \gets Read()
If ((x_1 > 1) \land \neg(x_1 < 1)) \[
    x_1 \gets 3
\]
Else \[
    x_1 \gets (-20 + 30) \cdot 4 \cdot 3 \cdot 40 % teste de comentario
\]

Write(x_1)
x_1 \gets Read()
If ((x_1 > 1) \land \neg(x_1 < 1))
    x_1 \gets 3
Else
    x_1 \gets \frac{((-20+30) \cdot 12)}{40}

Write(x_1)
\sum ^ {(x_1 > 1) \lor (x_1 \eq 1)} \[
    x_1 \gets x_1 - 1
    Write(x_1)\]
\]
```


❯ python main.py assets/codes/test_function_call.tex
3
3
0
3
2
1
0