CTeX Programming Language Proposal

Unal Yigit Ozulku (uyo2000), Weicheng Zhao (wz2578), Hu Zheng (hz2709)

Introduction

The CTeX programming language is a functional programming language based on a subset of the mathematical syntax in LaTeX. The goal of this language is to make mathematical expressions written in LaTeX in papers computable.

In most cases, researchers tend to use LaTeX to write mathematical content. Meanwhile, they have to use a different programming language to actually compute or verify the expressions, which requires duplicate effort. So we aim to reduce such repeated work as much as possible.

The language is purely functional, as mathematical expressions are functional.

We create our language using a limited subset of the mathematical symbols and syntax in LaTeX. However, LaTeX is only used for typesetting and has no semantics itself. The semantics come from mathematics. Meanwhile, ambiguity is everywhere in mathematical expressions. To limit the complexity and make the implementation feasible, we impose some restrictions on the expressions to define them rigorously and have a unique meaning in mathematics.

Our focus is mathematics, not LaTeX. We will only focus on the math part of LaTeX. Our code is indeed renderable LaTeX, but rendering is not part of our job. To render the source code, just use any LaTeX engine.

Features

We try not to focus on types in computer science. Since the syntax is mainly a subset of LaTeX, which means that, in most cases, users could write the code in CTeX like in LaTeX, we will focus on our restrictions and modifications to the math part of LaTeX, which we are going to elaborate on.

We try to minimize the number of special characters. So if a symbol can both be represented by a Unicode character and a control sequence, we choose to use the latter as the canonical expression. For example, "divide" can be `|` or `\mid`, thus we choose to regard only the latter as the "divide" symbol mathematically.

Our identifiers can only be a single character, which is the same as the situation in mathematics. For example, `aa` will be regarded as `a` times `a` and not a new identifier `aa`.

However, you can still use common styling control sequences like `\mathfrak` to form new identifiers. For example,

```
a=1 \\
\mathfrak{a}=2 \\
```

is valid since they are different identifiers.

Superscripts `x^i` are mostly regarded as power.

Subscripts can be used to either form a new identifier or form a sequence, which is basically a function. But these cases are exclusive. For example,

```
x_1=1 \\
x_i=i \\ %% will err since x in x_1 is already used to form an identifier
```

```
x_i=i \\
x_1=1 \\ %% will evaluate to true, since x_1 is regard as a member of the sequence
x_2=1 \\ %% will evaluate to false, since x_2 is 2
```

The subscript is character is however not affected by identifiers defined under the same name:

```
i=1 \\
x_i=i \\
x_2=1 \\ %% evaluates to false
x_2=2 \\ %% evaluates to true
```

The program consist of five types of statements:

1. Regular statement, e.g. `x=1+2`
2. Function definition, e.g. `f(x)=x^x`, `x_i=i^3`
3. Case statement, e.g. `\begin{cases} b & b \mid a \\ 0 & b \nmid a \\ \end{cases}`
4. Print. We use the LaTeX comment symbol `%` for print, e.g., `%1` will print the number 1. We choose the comment symbol because the print statement typically doesn't appear in

mathematical expressions, so we don't want it rendered.

5. Comment. We use `%%` for comment, essentially escaping the print statement.

Interesting Programs

Summation

```
s = \sum_{i = 1}^{100} i \\
% s %% evaluates to 5050
```

The rendered equation (not our job, just render the above with a LaTeX engine):

$$s = \sum_{i=1}^{100} i$$

Piecewise Function

```
\gcd(a,b) =
\begin{cases}
  \gcd(b,a\mod b) &b \neq 0 \\
  a& b=0 \\
\end{cases} \\

% gcd(105,63) %% evaluates to 21
```

The rendered equation (not our job, just render the above with a LaTeX engine):

$$\gcd(a,b) = \begin{cases} \gcd(b,a \mod b) & b \neq 0 \\ a & b = 0 \end{cases}$$

Number Theory

```
f(a,b)=
\begin{cases}
  b & b \mid a \\
  0 & b \nmid a  \\
\end{cases} \\
```

```
\sigma(n)=\sum_{i=1}^n f(n,i) \\

% \sigma(12) %% evaluates to 28
% \sigma(28) %% evaluates to 56
```

The rendered equation (not our job, just render the above with a LaTeX engine):

$$f(a,b) = \begin{cases} b & b \mid a \\ 0 & b \nmid a \end{cases}$$
$$\sigma(n) = \sum_{i=1}^{n} f(n,i)$$