CTeX

- make mathematical expressions in LaTeX computable

Weicheng Zhao

Hu Zheng

Rachel Liu

Unal Yigit Ozulku

The Team

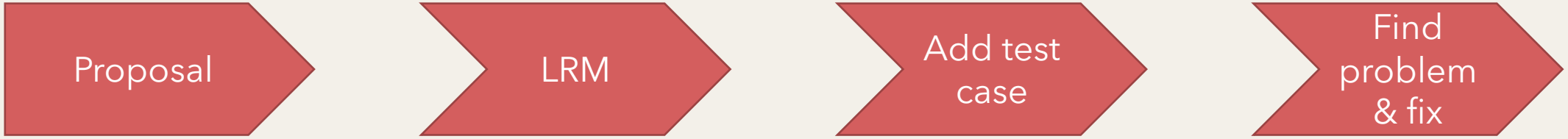
Hu
Systems Architect

Weicheng
Language Gurus

Unal
Tester

Rachel
Manager

Project Plan



Motivation: avoid duplicate work

- LaTeX is widely used to write mathematical contents
- LaTeX only provides writing mathematical expressions function
- Need another tool to do the computation

About CTeX

- A functional programming language based on a subset of the mathematical syntax in LaTeX
- Aim to combine the process of writing mathematical expressions and computing them together
- Have restrictions on the expressions

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

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

From the Math world to CTeX

- Try to keep the syntax consistent with mathematics
- Use number, variable and function as if in math
- Support implicit multiply
  - Let xy, $2x$ valid multiplication in CTeX
  - $x2$ is also multiplication in CTeX, though uncommon in math
  - Priority problems, such as $\sin 2x$ and $|\, x \,|\, y \,|\, z \,|$
  - Not fit in the shorthand tools in Yacc
- Limit the semantic meanings of every symbols
  - | is used to represent absolute value and divisible.
  - Let | used as the symbol of absolute and “\mid” for divisible.

From TeX to CTeX

- Adopt ways that TeX used to type in symbols
- Caes environment for IF statement
- Split environment for statement closure
- “\\” for end of a statement
- “%” for print so that it would not appear when being rendered
- “%%” for starting comment to keep compatible with TeX
- EOL(“\n”) is used as the end of print and comment for compatibility
- No other complicated usage in TeX

Type	Content
Identifiers	• A single letter or a single Greek letter • Specific style operators • Anything follows the 2 cases above with “_”
Operators	^ _ ( ) / + - = < > \cdot \times \div \frac \leq \geq \neq \mid \nmid \neg \binom \arccos \arcsin \arctan \cos \cosh \cot \coth \csc \exp \mod \gcd \vee \wedge \lg \ln \log \sqrt \max \min \sec \sin \sinh \tan \tanh \left| \right| \lfloor \rfloor \lceil \rceil
Constants	• Integer • Float
Other symbols	“\\”, “&”, “{}”, “,” “\begin{cases}”, “\end{cases}” “”\begin{split}”, “\end{split}”

Tokens in CTeX

Syntax

- Use the syntax tree to control the precedence
  - Atom
  - Power Operator
  - Log-Like Functions Operators
  - Implicit multiplication
  - Unary arithmetic operations
  - Multiplicative Operators
  - Additive Operators
  - Frac-like Operations
  - Functional Expressions
  - Calculating Expressions
  - Comparisons
  - Logical Expressions

$-e^y$ to be $-(e^y)$

$\sin x \cos y$ to be $(\sin x) \times (\cos y)$

$a - b$ should not be $a \times (-b)$

Boolean are only supposed to appear in Case statement.

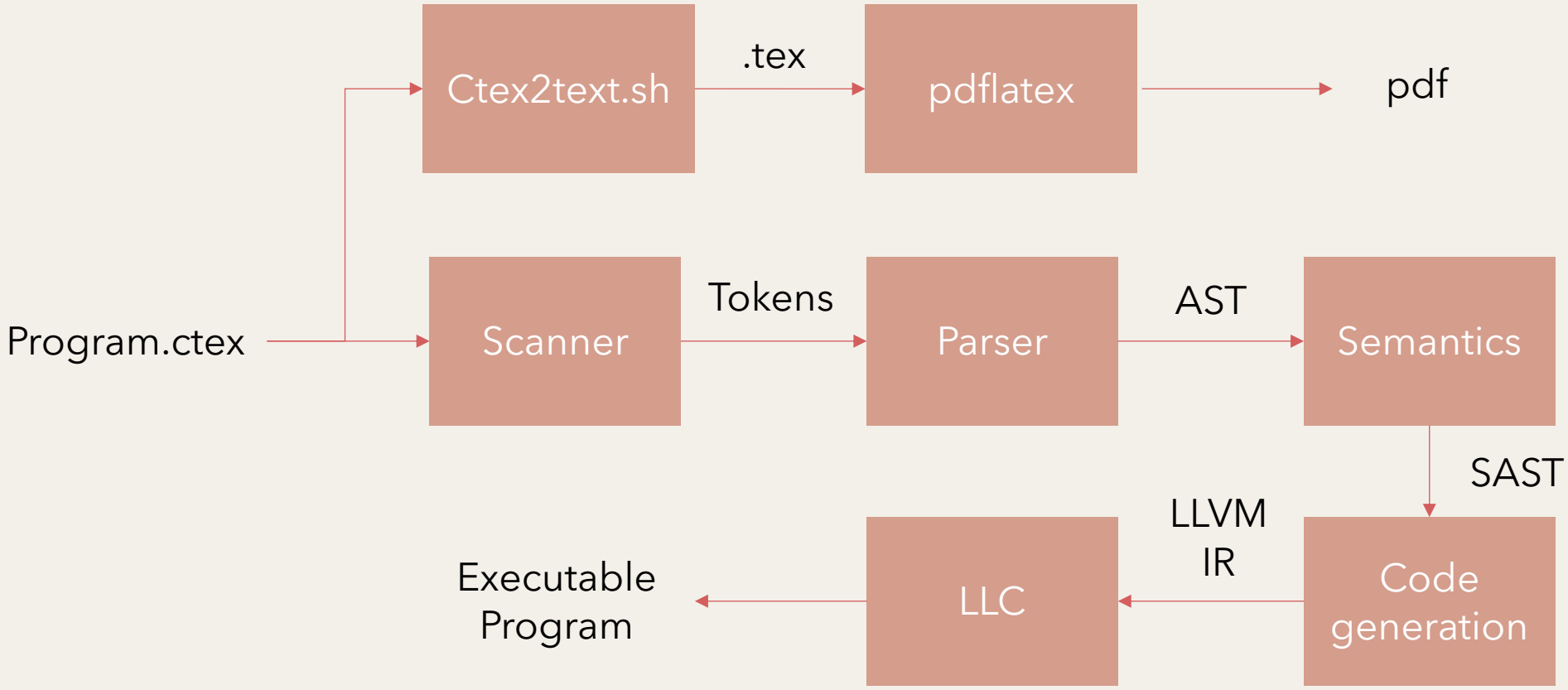
Statements

- Statement changes local bound, while expression not.
- 6 kinds of statements
  - Expression
  - Assign
    - No explicitly declaration as there is not in math
    - Considered as statements rather than expression
  - Print
  - Function Definition
  - Case statement
  - Statement closure
- Last 3 kinds of statements are complex because they contain statement or a list of statement

What about $f(a)$

- $f \times a$ or a function f call with arguments a, or even whether the user is about to define such a function?
- Distinguish between implicit multiply and function call
  - Push to type checker which has information what f is.
- Eliminate shift reduce conflicts related to function definition and call.
  - The length of the arguments list is unknown
  - Not until the "=" after the argument list can we get to know whether it is a definition or a function call. It is LR(*).
  - Start from argument list with a length of 2. (1 is processed specially as mentioned above, so is 0)
    - If there is an argument is not identifier, it could not be a function definition.
    - We could exit earlier
    - If all identifier, then just take a look at the symbol after the ")" we could know whether it is a definition or a call.

Architectural Design



Tests

Three testing suites:
- Scanner
- Parser
- end-to-end testing (compiler)

Comparing sample code to expected output (.reference) and rejected to code to its expected error

Over 100 tests in the final repository

e.g. test_gcd.ctex tested for the scanner

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

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

```
ID LPRN ID COMMA ID RPRN EQ LCASE ID LPRN ID COMMA ID MOD ID RPRN AMP ID NEQ
DIGIT DBS ID AMP ID EQ DIGIT DBS RCASE PCT ID LPRN LIT_INT COMMA LIT_INT RPRN
EOL EOF
```

Demo


file: stmt_list EOF
stmt:
expression_stmt
| assignment_stmt
| print_stmt
| ident "(" ")" "=" stmt
| fun_def ")" "=" stmt
| expr_spec "=" stmt
| fun_def ")" "\\"
| case_stmt
| split_stmt

Thanks for listening