CTeX: Computable TeX

Weicheng Zhao, Hu Zheng, Rachel Liu, Unal Yigit Ozulku

December 18th, 2021

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 contents. Meanwhile, they also need to use another programming language to compute or verify the expressions. CTeX aims to combine the process of writing mathematical expressions and computing them together, to reduce the duplicate work.

CTeX is purely functional, as mathematical expressions are functional. It is created using a limited subset of the mathematical symbols and syntax in LaTeX. LaTeX is only used for typesetting and has no semantics itself. The semantics come from mathematics. In addition, ambiguity is everywhere in mathematical expressions. To limit the complexity and make the implementation feasible, CTeX has restrictions on the expressions to make each of them rigorously defined and have a unique meaning in mathematics.

Language Tutorial

Installation and Basic Usage

First, download the code for the compiler along with all of the tests from GitHub:

```
$ git clone https://github.com/cs4115-CTeX/CTeX
```

Then, prepare the environment for compiling and run the CTeX compiler. We will take Ubuntu 20.04 x64 as an example, starting with installing OCaml (>=4.13), llvm 10 and other dependencies.

```
$ sudo apt install ocaml llvm-10.0 llvm-10.0-dev llvm-runtime m4 opam
menhir
$ opam init && eval `opam config env`
$ opam install menhir llvm.10.0.0 ocamlfind ocamlbuild oUnit
```

You also need to install pdfLaTeX to render the TeX file, if you have not installed yet, you can run the following command.

```
$ sudo apt install texlive-latex-base texlive-fonts-recommended
texlive-fonts-extra texlive-latex-extra
```

Then your environment should be ready to compile the CTeX compiler. At the root directory of the CTeX working directory, run the following command.

```
$ make clean
$ make
```

Then the CTeX compiler would be built and all tests would run. After compiling the CTeX compiler successfully, you could use the following ccommand to build and run a .ctex file or generate .pdf file from it.

```
$ make [build|run|pdf] ctex=[.ctex files]
```

Tutorial of CTeX

In most cases, you could just write CTeX as if you are in the math environment of TeX or LaTeX, but there are some limits. For example, superscripts in CTeX are always explained as power, also you could not use most of the complicated functions that LaTeX provided. For all differences and limitations, please refer to the language manual part.
Let's start with printing out the magic number 42. In CTeX, the original comment symbol "`%`" in LaTeX is used as print function, so that if you want to write some functions in LaTeX then evaluate them given some data using CTeX, you will not worry about editing the source file and messing up the pdf output. You can still use "`%%`" to indicate a comment in CTeX, thus it would be considered as a comment for both CTeX and LaTeX.

```
% 42 %% is a magic
```

Compiling and running the above CTeX file would get you an output "42".

You could also define your own functions in CTeX, just like what you would write in maths. For example, to write a function that computes the GCD of two numbers, you could use the following code.

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

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

Compiling and running the above CTeX file would get you an output "21".

Language Manual

Lexical Conventions

Token

There are four types of tokens in CTex languages: identifiers, operators, constants and other symbols. Blanks, horizontal and vertical tabs, newlines, formfeeds, and comments as described below are ignored except as they serve to separate tokens.

Comments

The characters `%%` introduce a comment, which terminates at the end of the line. Comments do not nest.

Identifiers

An identifier could be:

- A single letter or a single Greek letter, for example, "`a`" and "`\alpha`". Uppercase and lowercase letters and Greek letters are all supported. Upper and lower case (Greek) letters are considered different. No restriction on identifiers' length.
- Specific style operators with a sequence of letters or Greek letters between curly braces, for example, "`\mathrm{abc}`" and "`\mathbb{\alpha\beta}`".
- Anything follows the two cases above with an underscore "_" and a single digit, a single letter, a single Greek letter, a sequence of digits between curly braces or a sequence of letters or Greek letters between curly braces following. For example, "`x_a`" and "`\mathbb{\alpha}_2`".

Empties are allowed between curly braces but unnecessary blanks will be removed, which means "`\mathbb{\theta a}`" and "`\mathbb{\theta  a}`" are the same identifier.

Specific style operators that could be used in identifying an identifier in case 2 are:

```
\mathrm \mathit \mathbf \mathsf \mathtt \mathfrak \mathcal \mathbb
\mathscr
```

Acceptable Greek letters in CTeX are as follows:

```
\alpha \beta \Gamma \gamma \Delta \delta \epsilon \varepsilon \zeta
\eta \Theta \theta \vartheta \iota \kappa \varkappa \Lambda \lambda
\mu \nu \Xi \xi \Pi \pi \varpi \rho \varrho \Sigma \sigma \varsigma
```

```
\tau \Upsilon \upsilon \Phi \phi \varphi \chi \Psi \psi \Omega \omega
```

All other expressions appearing after a backslash are not acceptable.

Constants

There are two kinds of constants, Integer Constant and Floating Constant.

An integer constant should consist of a sequence of digits and would always be considered decimal. All integer constants will be considered as integer type.

A floating constant consists of an integer part, a decimal point and a fraction part. All floating constants will be considered as a floating type.

There is a special kind of literal in CTeX that is a single digit, called “short”. Though it will be parsed the same as integer constants, it does not need curly brackets as a component in some expressions.

Operators

Operators in CTeX are as follows.

```
^ _ ( ) / + - = < >
\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
```

Other Symbols

Besides what is mentioned above, there are following symbols mostly used in the CTeX language to separate codes.

Symbol	**Meaning**
`\\`	End a statement or use as separators between different cases
`&`	Separator of case expression and then expression in the case statement
`{}`	Used to bracket parameters of some operators and a sequence of letters in definition of an identifier, also used in some expressions.
`%`	Print the result of the following expression

	until the end of the line
`\begin{cases}`	Start of the case statement
`\end{cases}`	End of the case statement
`\begin{split}`	Start of the statement closure.
`\end{split}`	End of the statement closure.
`,`	Separate argument list when defining or calling a function, also used in some expressions

Syntax

In this part, we will introduce expressions and their syntax in CTeX by giving out the formal definition of every kind of expression.

Arithmetic Conversions

When a description of an arithmetic operator below uses the phrase “the numeric arguments are converted to a common type”, this means that the operator implementation for built-in types works as follows:

If either argument is a floating point number, the other is converted to floating point; otherwise, both must be integers and no conversion is necessary.

Some additional rules apply for certain operators.

Atoms

Atoms are the most basic elements of expressions. The simplest atoms are identifiers or literals. Forms enclosed in parentheses, brackets or braces are also categorized syntactically as atoms. The syntax for atoms is:

```
short_atom ::= short
long_atom ::= literal | parenth_form
paren_no_id ::= "(" expr_no_id ")"
atom ::= short_atom | long_atom
atom_closure ::= "{" expr_calc "}"
```

Identifiers

An identifier occurring as an atom is a name. See section Identifiers for lexical definition. When the name is bound to an object, evaluation of the atom yields that object.

```
ident ::= ID | ID "_" DIGIT

paren_id ::= "(" ident ")"
```

Literals
CTeX only supports integer and floating numeric literals:

```
short ::= digit
literal ::=  integer | floatnumber
```

Evaluation of a literal yields an object of the given type with the given value. The value may be approximated in the case of floating points.

Parenthesized form
A parenthesized form is an expression enclosed in parentheses or similar math operator `\labs \rabs` for absolute value, `\lfloor \rfloor` for floor and `\lceil \rceil` for ceiling:

```
parenth_form ::=
           "(" expr ")"
          | "\|" expr_calc "\|"
          | "\lfloor" expr_calc "\rfloor"
          | "\lceil" expr_calc "\rceil"
```

A parenthesized expression yields the single expression that makes up the expression list.

The Power Operator

The power operator binds more tightly than unary operators on its left, and binds less tightly than unary operators on its right. The syntax is:

```
expr_pow ::=  atom | atom "^" atom_closure | ident "^" atom_closure
```

The Log-Like Functions Operators

The log-like function operators include `\lg \ln \log \sqrt \sin \cos \tan \arcsin \arccos \arctan \sinh \cosh \tanh \cot \sec \csc \coth`. They have the same priority. The syntax is:

```
log_like_ops ::= "\lg" | "\ln" | "\log"
| "\sqrt" | "\sin" | "\cos" | "\tan" | "\arcsin"
| "\arccos" | "\arctan" | "\sinh" | "\cosh"
| "\tanh" | "\cot" | "\sec" | "\csc" | "\coth"
log_op := "\log" "_" atom_closure
expr_log ::=  expr_pow
    | log_like_ops expr_log
    | log_op atom_closure expr_log
    | log_like_ops ident
    | log_op atom_closure ident
    | log_like_ops paren_no_id
    | log_op atom_closure paren_no_id
    | log_like_ops paren_id
    | log_op atom_closure paren_id
```

Unary arithmetic operations

All unary arithmetic operations have the same priority:

```
expr_unary ::=
      expr_impl_mult
    | paren_id
    | paren_no_id
    | "-" expr_unary
    | "+" expr_unary
    | "-" expr_spec
    | "+" expr_spec
    | "-" ident
    | "+" ident
```

Implicit multiplication

Implicit multiplication declares the situation like `x(x+y)` in math. It's given higher priority than multiplicative operators. The situation of "ident paren_id" is specially processed as there is a chance that it could be a function call, a function definition or an implicit multiplication, it could not be determined until the type checker gets to know where the ident is a function or a variable. So it is pushed later to the type checker.

```
expr_impl_mult ::=
    expr_log
```

```
    | expr_impl_mult expr_log
    | expr_impl_mult ident
    | expr_impl_mult paren_no_id
    | expr_impl_mult paren_id
    | ident expr_log
    | ident ident
    | ident paren_no_id
    | paren_no_id expr_log
    | paren_no_id ident
    | paren_no_id paren_no_id
    | paren_no_id paren_id
    | paren_id expr_log
    | paren_id ident
    | paren_id paren_no_id
    | paren_id paren_id

expr_spec ::= ident paren_ident
```

Multiplicative Operators

The multiplicative operators `* \codt \times / \div \pmod` group left-to-right. The `* \codt \times` operators yield the product of its arguments.

```
mult_op: "*" | "\cdot" | "\times"
```

The `/ \div` operator yields the quotient of its arguments.

```
div_op: "/" | "\div"
```

The `\mod` operator yields the remainder from the division of the first expression by the second.

```
expr_mult ::=
    expr_unary
    | expr_mult mult_op expr_unary
    | expr_mult div_op expr_unary
    | expr_mult "\mod" expr_unary
    | expr_mult mult_op ident
```

```
    | expr_mult div_op ident
    | expr_mult "\mod" ident
    | expr_mult mult_op expr_spec
    | expr_mult div_op expr_spec
    | expr_mult "\mod" expr_spec
    | ident mult_op expr_unary
    | ident div_op expr_unary
    | ident "\mod" expr_unary
    | ident mult_op ident
    | ident div_op ident
    | ident "\mod" ident
    | ident mult_op expr_spec
    | ident div_op expr_spec
    | ident "\mod" expr_spec
    | expr_spec mult_op expr_unary
    | expr_spec div_op expr_unary
    | expr_spec "\mod" expr_unary
    | expr_spec mult_op ident
    | expr_spec div_op ident
    | expr_spec "\mod" ident
    | expr_spec mult_op expr_spec
    | expr_spec div_op expr_spec
    | expr_spec "\mod" expr_spec
```

Additive Operators

The additive operators `+` and `-` group left-to-right. **`expression + expression`** yields the sum of the two expressions. “expression - expression” yields the difference of the operands.

```
expr_add ::=
    expr_mult
    | expr_add "+" expr_mult
    | expr_add "-" expr_mult
    | expr_add "+" ident
    | expr_add "-" ident
    | expr_add "+" expr_spec
    | expr_add "-" expr_spec
    | ident "+" expr_mult
    | ident "-" expr_mult
```

```
    | ident "+" ident
    | ident "-" ident
    | ident "+" expr_spec
    | ident "-" expr_spec
    | expr_spec "+" expr_mult
    | expr_spec "-" expr_mult
    | expr_spec "+" ident
    | expr_spec "-" ident
    | expr_spec "+" expr_spec
    | expr_spec "-" expr_spec
```

Frac-like Operations

The frac-like function operators include `\frac` and `\binom`. The syntax is:

```
frac_op ::= "\frac" | "\binom"
expr_frac ::= frac_op atom_closure atom_closure
```

Functional Expressions

Functional expressions evaluate a function call that was defined before in the program by the user. To distinguish between function call and function definition, we check each argument, when it is not an ident, it should not be a function definition, if all of the arguments are ident, we push back to the statement parts to determine what it is. Please note that the situation with only one argument has to wait until we go to the type checker, then it is possible to tell whether it is a function call or an implicit multiplication.

```
expr_no_id ::=
      expr_add
    | ident "(" ")"
    | call_expr ")"
    | "\gcd" "(" arg_list ")"
    | "\max" "(" arg_list ")"
    | "\min" "(" arg_list ")"
    | "\frac" atom_closure atom_closure
    | "\binom" atom_closure atom_closure

fun_def ::=
      ident "(" ident "," ident
    | fun_def "," ident
```

```
call_expr ::=
      ident "(" ident "," expr_no_id
    | ident "(" expr_no_id "," expr_no_id
    | ident "(" expr_no_id "," expr_spec
    | ident "(" expr_no_id "," ident
    | ident "(" expr_spec "," expr_no_id
    | ident "(" expr_spec "," expr_spec
    | ident "(" expr_spec "," ident
    | ident "(" ident "," expr_spec
    | fun_def "," expr_no_id
    | call_expr "," expr_calc
```

Calculating Expressions

Calculating expressions are illogical expressions, which calculate the value of the expressions.

```
expr_calc ::= ident | expr_spec | expr_no_id
```

Comparisons

All comparison operations in CTeX have the same priority, which is lower than that of any arithmetic operations.

```
expr_comp ::=
      expr_calc
    | expr_comp "<" expr_calc
    | expr_comp ">" expr_calc
    | expr_comp "\leq" expr_calc
    | expr_comp "\leq" expr_calc
    | expr_comp "=" expr_calc
    | expr_comp "\neq" expr_calc
    | expr_comp "\nmid" expr_calc
    | expr_comp "\mid" expr_calc
```

Logical Expressions

In CTeX, boolean operations are evaluated from left to right, but they do not yield outputs. Expressions in CTeX are defined as expr_logic.

```
expr_logic_atom ::=
      expr_comp
    | "(" expr_logic ")"
    | "\neg" expr_logic_atom

expr_logic ::=
      expr_logic_atom
    | expr_logic "\wedge" expr_logic_atom
    | expr_logic "\vee" expr_logic_atom
```

Operator Precedence

The precedence of expression operators is the same as the order of the following table (highest precedence first):

```
short_atom:
    DIGIT
long_atom:
      LIT_INT
    | LIT_FL
    | "\labs" expr_calc "\rabs"
    | "\lfloor" expr_calc "\rfloor"
    | "\lceil" expr_calc "\rceil"
ident:
    | ID
    | ID “_” DIGIT
paren_no_id:
    "(" expr_no_id ")"
paren_id:
    "(" ident ")"
atom: short_atom | long_atom
atom_closure: "{" expr_calc "}"

expr_pow:
      atom
    | atom "^" atom_closure
    | ident "^" atom_closure

log_like_op:
```

```
     "\lg" | "\ln" | "\sqrt" | "\sin" | "\cos" | "\tan"
     | "\arcsin" | "\arccos" | "\arctan" | "\sinh" | "\cosh"
     | "\tanh" | "\cot" | "\sec" | "\csc" | "\coth"
log_op:
    "\log" "_" atom_closure
expr_log:
      expr_pow
    | log_like_ops expr_log
    | log_op atom_closure expr_log
    | log_like_ops ident
    | log_op atom_closure ident
    | log_like_ops paren_no_id
    | log_op atom_closure paren_no_id
    | log_like_ops paren_id
    | log_op atom_closure paren_id

expr_impl_mult:
    expr_log
    | expr_impl_mult expr_log
    | expr_impl_mult ident
    | expr_impl_mult paren_no_id
    | expr_impl_mult paren_id
    | ident expr_log
    | ident ident
    | ident paren_no_id
    | paren_no_id expr_log
    | paren_no_id ident
    | paren_no_id paren_no_id
    | paren_no_id paren_id
    | paren_id expr_log
    | paren_id ident
    | paren_id paren_no_id
    | paren_id paren_id

expr_spec:
    ident paren_id

expr_unary:
      expr_impl_mult
```

```
    | paren_id
    | paren_no_id
    | "-" expr_unary
    | "+" expr_unary
    | "-" expr_spec
    | "+" expr_spec
    | "-" ident
    | "+" ident

mult_op: "*" | "\cdot" | "\times"
div_op: "/" | "\div"
expr_mult:
    expr_unary
    | expr_mult mult_op expr_unary
    | expr_mult div_op expr_unary
    | expr_mult "\mod" expr_unary
    | expr_mult mult_op ident
    | expr_mult div_op ident
    | expr_mult "\mod" ident
    | expr_mult mult_op expr_spec
    | expr_mult div_op expr_spec
    | expr_mult "\mod" expr_spec
    | ident mult_op expr_unary
    | ident div_op expr_unary
    | ident "\mod" expr_unary
    | ident mult_op ident
    | ident div_op ident
    | ident "\mod" ident
    | ident mult_op expr_spec
    | ident div_op expr_spec
    | ident "\mod" expr_spec
    | expr_spec mult_op expr_unary
    | expr_spec div_op expr_unary
    | expr_spec "\mod" expr_unary
    | expr_spec mult_op ident
    | expr_spec div_op ident
    | expr_spec "\mod" ident
    | expr_spec mult_op expr_spec
    | expr_spec div_op expr_spec
```

```
    | expr_spec "\mod" expr_spec

expr_add:
    expr_mult
    | expr_add "+" expr_mult
    | expr_add "-" expr_mult
    | expr_add "+" ident
    | expr_add "-" ident
    | expr_add "+" expr_spec
    | expr_add "-" expr_spec
    | ident "+" expr_mult
    | ident "-" expr_mult
    | ident "+" ident
    | ident "-" ident
    | ident "+" expr_spec
    | ident "-" expr_spec
    | expr_spec "+" expr_mult
    | expr_spec "-" expr_mult
    | expr_spec "+" ident
    | expr_spec "-" ident
    | expr_spec "+" expr_spec
    | expr_spec "-" expr_spec

expr_no_id:
      expr_add
    | ident "(" ")"
    | call_expr ")"
    | "\gcd" "(" arg_list ")"
    | "\max" "(" arg_list ")"
    | "\min" "(" arg_list ")"
    | "\frac" atom_closure atom_closure
    | "\binom" atom_closure atom_closure

call_expr:
      ident "(" ident "," expr_no_id
    | ident "(" expr_no_id "," expr_no_id
    | ident "(" expr_no_id "," expr_spec
    | ident "(" expr_no_id "," ident
    | ident "(" expr_spec "," expr_no_id
```

```
    | ident "(" expr_spec "," expr_spec
    | ident "(" expr_spec "," ident
    | ident "(" ident "," expr_spec
    | fun_def "," expr_no_id
    | call_expr "," expr_calc

expr_calc:
      ident
    | expr_spec
    | expr_no_id

arg_list: expr_calc | arg_list "," expr_calc

expr_comp:
      expr_calc
    | expr_comp "<" expr_calc
    | expr_comp ">" expr_calc
    | expr_comp "\leq" expr_calc
    | expr_comp "\leq" expr_calc
    | expr_comp "=" expr_calc
    | expr_comp "\neq" expr_calc
    | expr_comp "\nmid" expr_calc
    | expr_comp "\mid" expr_calc

expr_logic_atom:
      expr_comp
    | "(" expr_logic ")"
    | "\neg" expr_logic_atom

expr_logic:
      expr_logic_atom
    | expr_logic "\wedge" expr_logic_atom
    | expr_logic "\vee" expr_logic_atom
```

Statements

We define 9 kinds of statements as below. The first 7 kinds of statements are simple statements, while the last 2 kinds are complicated statements. A statement ends in double backslashes.

Statements can compound together into a statement list within curly braces as a single statement to be used in complicated statements. A function call or definition with all arguments are ident, could only be determined what it is until now. Please note that if a “expr_spec” appears at the position before a “=” followed by a statement, we could determine that it is a function definition and need not to wait until the type checker.

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

stmt_list ::= stmt | stmt_list stmt
```

Scope

If and only if the assignment statement or function definition statement is a part of a complicated statement and the identifier has not been bound outside the complicated statement, the assignment statement or function definition statement will make the identifier local. Otherwise, it will make the identifier be able to use and refer to the whole program globally, even in other complicated statements.

Expression Statements

Expression statements are used (mostly interactively) to compute and write a value, or (usually) to call a procedure (a function that returns no meaningful result). Other uses of expression statements are allowed and occasionally useful. The syntax for an expression statement is:

```
expression_stmt ::= expr_calc "\\"
```

An expression statement evaluates the expression.

Assignment Statements

An assignment statement evaluates the expression and assigns the result to the identifier. The identifier would be rebound if it was already bound. Assignment statements are used to (re)bind names to values:

```
assignment_stmt ::=  identifier "=" expr_calc "\\"
```

Print Statements

Print statements are used to print the evaluation result of an expression to standard output. It will also output a newline `\n` implicitly after outputting the result.

```
print_stmt ::= "%" expr_calc EOL
```

Function Definitions

A function definition defines a user-defined function.

```
fun_def ::=
      ident "(" ident "," ident
    | fun_def "," ident

Func_def_stmt ::=
    ident "(" ")" "=" stmt
  | fun_def ")" "=" stmt
  | expr_spec "=" stmt
```

A function definition is an executable statement. Its execution binds the function name. The function definition does not execute the function body.

Case Statements

The case statement is used for conditional execution.

```
case_stmt ::= "\begin{cases}" suite_list "\end{cases}"
suite ::= stmt "&" expr_logic "\\"
```

It selects exactly one of the suites by evaluating the expressions one by one until one is found to be true; then the statement in that suite is executed and no other part of the case statement is executed or evaluated. If all expressions are false, then none of the suites would be executed.

Split Statements

The split statement is used for multiple statements.

```
split_stmt ::= "\begin{split}" stmt_list "\end{split}"
```

It returns the value of the last statement after running previous statements.

Top Level and Full Grammar Specification

The CTeX compiler will get its input from the file. The full grammar of CTeX is shown below.

```
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

stmt_list: stmt | stmt_list stmt

expression_stmt: expr_calc "\\"
assignment_stmt: identifier "=" expr_calc "\\"
print_stmt: "%" expr_calc EOL
fun_def:
      ident "(" ident "," ident
    | fun_def "," ident
case_stmt: "\begin{cases}" suite_list "\end{cases}"
split_stmt: "\begin{split}" stmt_list "\end{split}"
suite: stmt "&" expr_logic "\\"
suite_list: suite | suite_list suite
```

Project Plan

Planning and Specification Process

We meet every week after the class to discuss any problems confronted during the previous week, report the progress and arrange the work for the next week. During the meeting, each of us could express our thoughts and solve problems together. Based on the milestone required by the class schedule, we adopted agile ways to plan our development, that is to say, we would create a basically runnable architecture and try to find out any problems in it. Then we would try to find the reason during the weekly meetings and arrange one of us to fix it. We also communicated and discussed a lot via Slack during the week.

Development Process

As mentioned above, we adopted an agile way of working. When working on the language manual, we made a very basic version, and each of us would try to find any problems or mistakes in it and leave a comment so we could discuss how to fix the problem. During the development of the compiler, each of us made a basically working part of the compiler, then we put them together and tried to find bugs and problems. Every time when any of us found an issue then he/she could just add a test case that would fail, then one of us would be assigned to fix this problem, make the test run successfully, while not failing any other test cases. Then a pull request would be created and others would review the modifications, and add more test cases that try to fail it. If it keeps working, then the pull request would be merged into the master branch, which relies heavily on the automatic check provided by Github Action.

Testing Process

We believed everyone should write test cases for all of the features or modifications they were working on to prove that everything was working as expected. And everyone should also write test cases for other's code so that we would not overlook any problems because of the developer's mental set.

We added automated test suites at the very beginning of our development and used Github Action as our Continuous Integration tool. So that it would be triggered at every Pull Request to the master branch and check all the tests automatically. Only when the Pull Request does not fail the check, it could be merged into the master branch.

Programming Style Guide

The codebase that is used to implement CTeX is mostly OCaml (for compiler), C code (for some functions) and shell scripts (for test suites). We tried to write clean, readable, modular, OCaml code.

We use OCamlFormatter to format all of our OCaml code, with a provided .ocaml format file within our code base, so that we can keep a uniform style when writing the code.
We also require most of the modification to be developed on a branch first and only when Github Actions do not fail when checking the code, the branch would be merged into the master branch of our code base.

Project Timeline

The project timeline we followed is shown as below:

Date	Milestone
10.8	Project proposal complete
10.31	LRM and parser complete; scanner and ast tree start
11.6	Ast, scanner, code generation complete; debug for Hello World
11.12	Hello World program complete
11.19	More functions added; start to write test
11.26	Complete adding functions; fix bugs; change ast tree order
12.3	Fix parser, SAST, code generations, type checker
12.18	Project complete
12.21	Report and slides finish

Roles & Responsibilities

- Manager: Rachel Liu
- System Architect: Hu Zheng
- Language Guru: Weicheng Zhao
- Test Designer: Unal Yigit Ozulku

Every member of the team contributed to the compiler and tests as well as the writing of proposal, LRM and Final Report. We collaborated on many features, doing different parts and helping each other out when stuck. The general division-of-labor are as follows:

Weicheng Zhao:

- Initial version of the type checker and SAST tree.
- Code generation.
- Test suites and build tools.

- Continuous integration.
- Additional C function calls.
- Pretty print of AST and SAST.

Hu Zheng:

- Project idea and proposal.
- Scanner and pretty print of tokens..
- Unit test suite for scanner.
- Subscripted identifiers in parser and type checker
- Examples and test cases.

Rachel Liu:

- Parser and AST.
- Edit on type checker.
- Pretty print of AST and SAST.
- Project report and slides.

Unal:

- Test cases and examples.
- Documentation.
- Edit to the testing suite.

Project Log

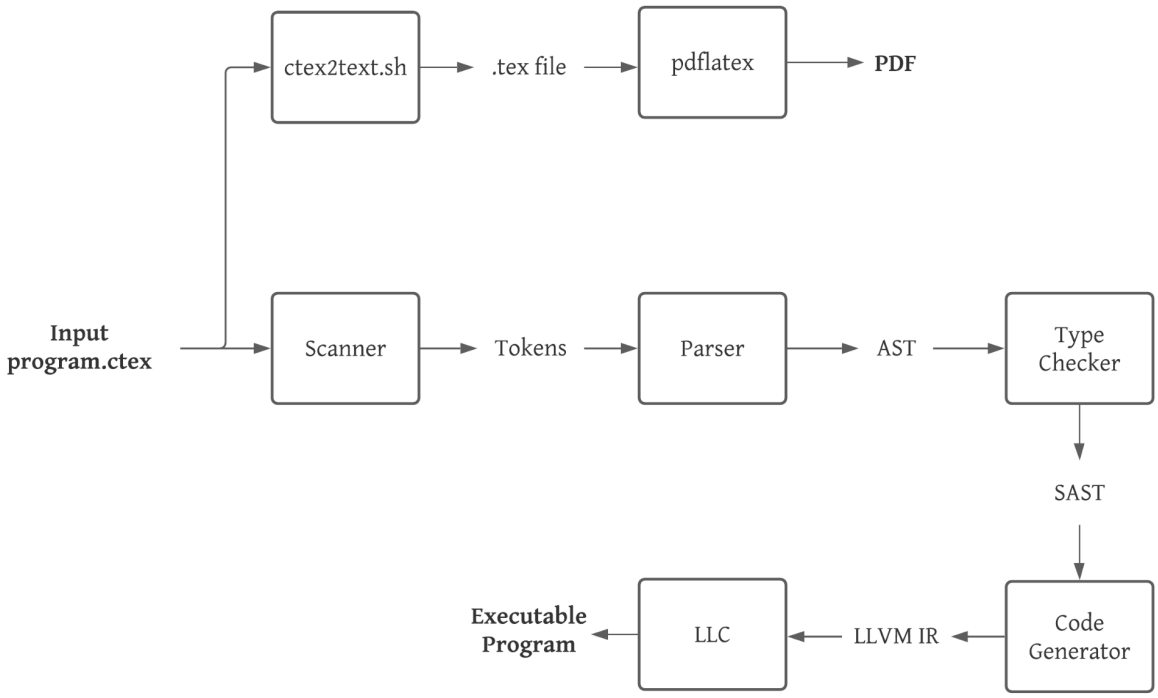
Please see appendix for a complete commit log listing.

Software Development Environment

- Operating Systems: MacOS, Ubuntu 20.04
- Languages: OCaml, C, Shell Scripts
- Text Editor: VS Code, IntelliJ IDEA
- Version Control: GitHub
- Documentation: Codi MD, Google Docs, Github Issue, Notion
- Continuous Integration: Github Action

Architectural Design

The architecture of the CTeX compiler is outlined in the following diagram:



PDF Generation

This part is the combination of a simple shell script and a LaTeX engine.

ctext2text.sh

This is a simple shell script that adds a bunch of LaTeX environment declarations and wrappers to generate source code that can be directly rendered with a LaTeX engine.

pdflatex

The LaTeX engine (We chose pdflatex. But it may be any working engine) then compile the source code into readable PDF.

Executable Generation

Scanner

The scanner is mostly straightforward. For simple tokens like `+` or `\begin{cases}`, it simply lists the rule and returns the corresponding token.

When encountering `%%`, which is a sign of comments, it enters another entrypoint `ctex_read_comment`. The only thing done there is to discard anything and resume the main entrypoint `ctex_read_token` on newline.

When encountering `\`, which is a sign of commands or greek letters, it tries to read the next immediate word and decides if it's a valid command or greek letter, matching against a predefined set, as we aren't allowing custom commands. In this way, we are able to reject invalid commands at the scanner phase.

We also deal with compound identifiers formed with formatting environments, e.g. `\mathrm{ident\alpha\beta}` in the scanner. When encountering such environments, we enter a new entrypoint ctex_read_ident. There we only accept a restricted set of rules such as those for letters and greek letters. On encountering `}`, we finish up the compound identifier and get back to the main entrypoint.

Since '\', '{', and '}' are illegal or represent special meaning in the assembly code, the scanner would transform all such characters into underlines so that they would not mess up the .S file.

Parser

The peculiarity of our grammar prevents us from using the builtin operators and precedences. We built our own generation rules to precisely control the precedences. The details can be found at the LRM part.

There are a few problems whose solutions are worth mentioning.

First,

Type Checker

The type checker mainly focuses on inferring the type of every variable, expression and statements. Since users of CTeX need not to point out the type explicitly like what people always do in mathematics, the type checker needs to infer the types. The rules are as follows:

- The initial bind of a variable would determine the type of the variable by the type at the right side of the "=" symbol.
- All arguments of user-defined functions are inferred as float type, and the return type is inferred from the type of the statement which is the body of the function definition.
- All results of non-logical operators, except "\gcd", "\binom" and "\mod", would have float type if any one of the operands are float, otherwise, is inferred as integer.
- All results of logical operators would have bool type.
- If any suite of the case statements is inferred to be float, then the result of the case statement will be float, otherwise, integer.
- If there are recursive function calls in the function definition, then the return type of the function will be inferred as float, and would change based on the result of the evaluation of the function body.

The type checker also resolves all nested function definitions, by adding all function declarations found in the AST tree to a function declaration list. Since CTeX does not require an entry point, type checker would also pack all statements into a statement closure as the body of the main function.

Since there might be references to the previous bound variable inside the nested function definition, thus, when there is a function definition, the type checker would check the current local bound, and add all of them into the frame list in the function declaration, which is used in the code generation.

The type checker also checks the existence of the variables that are about to be used. To maintain such a lookup map for bound variables and defined functions, the type checker would update it after every evaluation of a statement, and would inherit the previous map, modifies and saves as a new one in the function declaration.

Code Generator

The code generator translates the SAST tree into LLVM IR, by iterating through the list of function declarations and translating every function. Before that, the code generator needs to declare all mathematical functions adopted from GNU Library for possible use.

When generating code for a function, it would follow the type inferred by the type checker and add cast instructions as needed to be operands or results. For example, the operands of "\mod" should be all integers, and if any of the operands are float, a cast instruction would be inserted.

The code generator uses frame to solve the usage of the out of scope bound in the nested function definition. Thus, when there is a non-empty frame list in a function, the first argument of this function is the frame of the function and the real arguments start from the second.

When confronting a function definition, the code generator would allocate a struct for the frame and store all variables in the frame list of this function, and save the frame into the map of local variables. Then when confronted with a function call, it could be retrieved from the map and put into the first parameter.

Since the frame variable is kept as the scope of the function definition in the local variables map, and would be inherited to be transported via the frame if there is any nested function definition so it would not mess up the scope. Also since we directly store the value, thus any modification afterwards to frame variables would not affect the frame itself, which provides reasonable actions similar to lambda functions in other languages.

LLC

LLC is to generate assembly code and then we could use assembler to generate the executable files. Since we use a lot of math functions from GNU libraries, “-lm” option is needed to indicate that the math library should be linked.

Test Plan

Test Suite and Automation

We have 3 different test suites for the CTeX project, 2 of them are unit test suites for the scanner and parser. There is also an integrated end-to-end test suite for the whole compiler. Different test suites provide us with a handful of tools to locate and isolate the bug and fix it. All 3 suites are written in shell scripts mainly. All tests are found within the tests directory, and executing the Makefile in this directory will run all three test suites.

The convention we followed was that the passing .ctex tests would be named `test_[description of test].ctex`, and failed tests would be named using the convention `bad_[testname.ctex]`. All passing tests are accompanied by a .reference file that includes the expected output for its accompanying test (e.g. if test_chained_functions.ctex is supposed to be accepted by the compiler, it would be accompanied by a test_chained_functions.reference files with its expected output.)

Test Suite for the Scanner

The test suite for the scanner compiles the scanner and compares its output with the reference files in the test cases. To let the scanner output a list of tokens, which is not supposed to do so, we use a shell script to inject some OCaml code into the scanner source before the suite compiles and tests it.

For each .ctex test file in the `tests/test_scanner/test_cases` directory, the file is compiled and compared against the .reference file with the same name. Passing cases are compared with their expected output, and failed cases are compared against their expected compiler error messages. Test cases with matching .reference files are passed with “OK”, and cases without a .reference counterpart, as well as cases where the .reference file does not match with the output are failed with “FAILED”.

For each failed test case, two additional files are generated in a test_tmp dictionary: `[test_name].out` and `[test_name].diff:` the former contains the output generated by the compiler, while the latter contains all differences between the .out file and the .reference file.

There is also an additional unit test suite powered by OUnit for the scanner, added later mainly to resolve the problem of invalid names in assembly. GCC does not allow names in assembly to contain backslash. As a result, we are required to get rid of the backslashes in the scanner phase. The previous test cases only compare the token type. So we added a unit test suite to compare the content in memory.

Test Suite for the Parser

We use menhir's interpreter mode to implement the test suite for the parser, by comparing the output of the menhir given a list of tokens that the scanner outputs (by reusing the injection code in the scanner test suite.) The menhir would output the CST of the given tokens so that we can check if the results are consistent with our language manual.

For each .ctex test file in the `tests/test_e2e/test_cases` directory, the file is compiled and compared against the .reference file with the same name. Passing cases are compared with their expected output and are passed with "OK", while cases where the .reference file does not match with the output are failed with "FAILED". For the parser testing suite, test cases that aren't supposed to be accepted by the parser (i.e. cases named `bad_[testname.ctex]`) are not accompanied by .reference files.

End-to-End Test Suite

The e2e test suite would compile the whole compiler thus it could compile and run the given test cases, and compare the outputs of the executable file with the references. It also converts the CTeX source file into TeX file and tries to use pdfLaTeX to generate the pdf file from it, so that we could check if a runnable CTeX file is also renderable as a TeX file.

For each .ctex test file in the `tests/test_e2e/test_cases` directory, the file is compiled and compared against the .reference file with the same name. Passing cases are compared with their expected output, and failed cases are compared against their expected compiler error messages. Test cases with matching .reference files are passed with "OK", and cases without a .reference counterpart, as well as cases where the .reference file does not match with the output are failed with "FAILED".

For each failed test case, three additional files are generated in a test_tmp dictionary:
`[test_name].out:` contains the output generated by the compiler
`[test_name].diff:` contains all differences between the .out file and the .reference file.
`[test_name].ll:` LLVM IR file

Examples

- Accepted example for the scanner (test_gcd.ctex)

test_gcd.ctex

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

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

test_gcd.reference

```
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
```

- Accepted example for the parser

test_associativity.ctex

```
a = 20 \\
\lg \sin a + 10\\
```

test_associativty.reference

```
Ready!
ACCEPT
[program:
  [stmt_list:
    [stmt_list:
      [stmt:
        ID
```

```
        EQ
        [expr_calc:
          [expr_no_id:
            [expr_add:
              [expr_mult:
                [expr_unary:
                  [expr_impl_mult:
                    [expr_log: [expr_pow: [atom: [long_atom:
LIT_INT]]]]
                  ]
                ]
              ]
            ]
          ]
        ]
        DBS
      ]
    ]
    [stmt:
      [expr_calc:
        [expr_no_id:
          [expr_add:
            [expr_add:
              [expr_mult:
                [expr_unary: [expr_impl_mult: [expr_log: LG
[expr_log: SIN ID]]]]
              ]
            ]
            ADD
            [expr_mult:
              [expr_unary:
                [expr_impl_mult:
                  [expr_log: [expr_pow: [atom: [long_atom:
LIT_INT]]]]
                ]
              ]
            ]
          ]
        ]
```

```
      ]
      DBS
    ]
  ]
  EOF
]
```

- Accepted example for the compiler (end-to-end)

test_collatz.ctex

```
f(n) =
\begin{cases}
  n / 2 \\ & n \mod 2 = 0 \\
  3 n + 1 \\ & n \mod 2 = 1 \\
\end{cases}

g(n) =
\begin{cases}
  1 \\ & f(n) = 1 \\
  g(f(n)) \\ & f(n) \neq 1 \\
\end{cases}

% f(15) %% evaluates to 46
% f(14) %% evaluates to 7
% f(f(14)) %% evaluates to 22

%% Collatz conjecture:

% g(14) %% evaluates to 1
% g(114) %% evaluates to 1
% g(543) %% evaluates to 1
% g(17) %% evaluates to 1
% g(7) %% evaluates to 1
% g(55) %% evaluates to 1
```

test_collatz.reference

```
46
7
22
1
1
1
1
1
1
```

test_collatz.ll

```
; ModuleID = 'CTeX'
source_filename = "CTeX"

@fmt_float = private unnamed_addr constant [4 x i8] c"%g\0A\00",
align 1
@fmt_float.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00",
align 1
@fmt_float.2 = private unnamed_addr constant [4 x i8] c"%g\0A\00",
align 1
@fmt_float.3 = private unnamed_addr constant [4 x i8] c"%g\0A\00",
align 1
@fmt_float.4 = private unnamed_addr constant [4 x i8] c"%g\0A\00",
align 1
@fmt_float.5 = private unnamed_addr constant [4 x i8] c"%g\0A\00",
align 1
@fmt_float.6 = private unnamed_addr constant [4 x i8] c"%g\0A\00",
align 1
@fmt_float.7 = private unnamed_addr constant [4 x i8] c"%g\0A\00",
align 1
@fmt_float.8 = private unnamed_addr constant [4 x i8] c"%g\0A\00",
align 1

declare i32 @printf(i8*, ...)

declare i32 @abs(i32)
```

```
declare double @fabs(double)

declare double @pow(double, double)

declare double @log10(double)

declare double @log(double)

declare double @sqrt(double)

declare double @sin(double)

declare double @cos(double)

declare double @tan(double)

declare double @asin(double)

declare double @acos(double)

declare double @atan(double)

declare double @sinh(double)

declare double @cosh(double)

declare double @tanh(double)

declare i32 @binom(i32, i32)

declare i32 @gcd(i32, i32)

define double @f(double %n) {
entry:
  %n1 = alloca double
  store double %n, double* %n1
  %case_result = alloca double
  %n2 = load double, double* %n1
  %mod_tmp_e1 = fptoui double %n2 to i32
```

```
  %mod_tmp = urem i32 %mod_tmp_e1, 2
  %eq_tmp = icmp eq i32 %mod_tmp, 0
  br i1 %eq_tmp, label %then, label %else

merge:                                            ; preds =
%merge_tail, %then
  %cond_result = load double, double* %case_result
  ret double %cond_result

then:                                             ; preds = %entry
  %n3 = load double, double* %n1
  %div_tmp = fdiv double %n3, 2.000000e+00
  store double %div_tmp, double* %case_result
  br label %merge

else:                                             ; preds = %entry
  %n4 = load double, double* %n1
  %mod_tmp_e15 = fptoui double %n4 to i32
  %mod_tmp6 = urem i32 %mod_tmp_e15, 2
  %eq_tmp7 = icmp eq i32 %mod_tmp6, 1
  br i1 %eq_tmp7, label %then_tail, label %else_tail

merge_tail:                                       ; preds =
%else_tail, %then_tail
  br label %merge

then_tail:                                        ; preds = %else
  %n8 = load double, double* %n1
  %impl_tmp = fmul double 3.000000e+00, %n8
  %add_tmp = fadd double %impl_tmp, 1.000000e+00
  store double %add_tmp, double* %case_result
  br label %merge_tail

else_tail:                                        ; preds = %else
  br label %merge_tail
}

define double @g(double %n) {
entry:
```

```
  %n1 = alloca double
  store double %n, double* %n1
  %case_result = alloca double
  %n2 = load double, double* %n1
  %f_result = call double @f(double %n2)
  %eq_tmp = fcmp oeq double %f_result, 1.000000e+00
  br i1 %eq_tmp, label %then, label %else

merge:                                            ; preds =
%merge_tail, %then
  %cond_result = load double, double* %case_result
  ret double %cond_result

then:                                             ; preds = %entry
  store double 1.000000e+00, double* %case_result
  br label %merge

else:                                             ; preds = %entry
  %n3 = load double, double* %n1
  %f_result4 = call double @f(double %n3)
  %neq_tmp = fcmp one double %f_result4, 1.000000e+00
  br i1 %neq_tmp, label %then_tail, label %else_tail

merge_tail:                                       ; preds =
%else_tail, %then_tail
  br label %merge

then_tail:                                        ; preds = %else
  %n5 = load double, double* %n1
  %f_result6 = call double @f(double %n5)
  %g_result = call double @g(double %f_result6)
  store double %g_result, double* %case_result
  br label %merge_tail

else_tail:                                        ; preds = %else
  br label %merge_tail
}

define i32 @main() {
```

```
entry:
  %f_result = call double @f(double 1.500000e+01)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
([4 x i8], [4 x i8]* @fmt_float, i32 0, i32 0), double %f_result)
  %f_result1 = call double @f(double 1.400000e+01)
  %printf2 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
([4 x i8], [4 x i8]* @fmt_float.1, i32 0, i32 0), double %f_result1)
  %f_result3 = call double @f(double 1.400000e+01)
  %f_result4 = call double @f(double %f_result3)
  %printf5 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
([4 x i8], [4 x i8]* @fmt_float.2, i32 0, i32 0), double %f_result4)
  %g_result = call double @g(double 1.400000e+01)
  %printf6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
([4 x i8], [4 x i8]* @fmt_float.3, i32 0, i32 0), double %g_result)
  %g_result7 = call double @g(double 1.140000e+02)
  %printf8 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
([4 x i8], [4 x i8]* @fmt_float.4, i32 0, i32 0), double %g_result7)
  %g_result9 = call double @g(double 5.430000e+02)
  %printf10 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
([4 x i8], [4 x i8]* @fmt_float.5, i32 0, i32 0), double %g_result9)
  %g_result11 = call double @g(double 1.700000e+01)
  %printf12 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
([4 x i8], [4 x i8]* @fmt_float.6, i32 0, i32 0), double %g_result11)
  %g_result13 = call double @g(double 7.000000e+00)
  %printf14 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
([4 x i8], [4 x i8]* @fmt_float.7, i32 0, i32 0), double %g_result13)
  %g_result15 = call double @g(double 5.500000e+01)
  %printf16 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
([4 x i8], [4 x i8]* @fmt_float.8, i32 0, i32 0), double %g_result15)
  ret i32 0
}
```

test_collatz.s

```
.section    __TEXT,__text,regular,pure_instructions
      .macosx_version_min 10, 17
      .section    __TEXT,__literal8,8byte_literals
      .p2align    3               ## -- Begin function f
```

```
LCPI0_0:
      .quad 4613937818241073152      ## double 3
LCPI0_1:
      .quad 4607182418800017408      ## double 1
LCPI0_2:
      .quad 4611686018427387904      ## double 2
      .section   __TEXT,__text,regular,pure_instructions
      .globl     _f
      .p2align   4, 0x90
_f:                                   ## @f
      .cfi_startproc
## %bb.0:                             ## %entry
      movsd %xmm0, -16(%rsp)
      cvttsd2si  %xmm0, %rax
      testb $1, %al
      je    LBB0_1
## %bb.2:                             ## %else
      cvttsd2si  -16(%rsp), %rax
      testb $1, %al
      je    LBB0_5
## %bb.3:                             ## %then_tail
      movsd -16(%rsp), %xmm0         ## xmm0 = mem[0],zero
      mulsd LCPI0_0(%rip), %xmm0
      addsd LCPI0_1(%rip), %xmm0
      jmp   LBB0_4
LBB0_1:                               ## %then
      movsd -16(%rsp), %xmm0         ## xmm0 = mem[0],zero
      divsd LCPI0_2(%rip), %xmm0
LBB0_4:                               ## %merge
      movsd %xmm0, -8(%rsp)
LBB0_5:                               ## %merge
      movsd -8(%rsp), %xmm0          ## xmm0 = mem[0],zero
      retq
      .cfi_endproc
                                      ## -- End function
      .section   __TEXT,__literal8,8byte_literals
      .p2align   3                ## -- Begin function g
LCPI1_0:
      .quad 4607182418800017408      ## double 1
```

```
	.section	__TEXT,__text,regular,pure_instructions
	.globl	_g
	.p2align	4, 0x90
_g:                                     ## @g
	.cfi_startproc
## %bb.0:                               ## %entry
	subq	$24, %rsp
	.cfi_def_cfa_offset 32
	movsd	%xmm0, 16(%rsp)
	callq	_f
	ucomisd	LCPI1_0(%rip), %xmm0
	jne	LBB1_2
	jp	LBB1_2
## %bb.1:                               ## %then
	movabsq	$4607182418800017408, %rax ## imm =
0x3FF0000000000000
	movq	%rax, 8(%rsp)
	jmp	LBB1_4
LBB1_2:                                 ## %else
	movsd	16(%rsp), %xmm0         ## xmm0 = mem[0],zero
	callq	_f
	ucomisd	LCPI1_0(%rip), %xmm0
	je	LBB1_4
## %bb.3:                               ## %then_tail
	movsd	16(%rsp), %xmm0         ## xmm0 = mem[0],zero
	callq	_f
	callq	_g
	movsd	%xmm0, 8(%rsp)
LBB1_4:                                 ## %merge
	movsd	8(%rsp), %xmm0          ## xmm0 = mem[0],zero
	addq	$24, %rsp
	retq
	.cfi_endproc
                                        ## -- End function
	.section	__TEXT,__literal8,8byte_literals
	.p2align	3               ## -- Begin function main
LCPI2_0:
	.quad	4624633867356078080     ## double 15
LCPI2_1:
```

```
	.quad	4624070917402656768	## double 14
LCPI2_2:
	.quad	4637722453773123584	## double 114
LCPI2_3:
	.quad	4647987494330040320	## double 543
LCPI2_4:
	.quad	4625478292286210048	## double 17
LCPI2_5:
	.quad	4619567317775286272	## double 7
LCPI2_6:
	.quad	4632937379169042432	## double 55
	.section	__TEXT,__text,regular,pure_instructions
	.globl	_main
	.p2align	4, 0x90
_main:                                  ## @main
	.cfi_startproc
## %bb.0:                               ## %entry
	pushq	%rax
	.cfi_def_cfa_offset 16
	movsd	LCPI2_0(%rip), %xmm0    ## xmm0 = mem[0],zero
	callq	_f
	leaq	L_fmt_float(%rip), %rdi
	movb	$1, %al
	callq	_printf
	movsd	LCPI2_1(%rip), %xmm0    ## xmm0 = mem[0],zero
	callq	_f
	leaq	L_fmt_float.1(%rip), %rdi
	movb	$1, %al
	callq	_printf
	movsd	LCPI2_1(%rip), %xmm0    ## xmm0 = mem[0],zero
	callq	_f
	callq	_f
	leaq	L_fmt_float.2(%rip), %rdi
	movb	$1, %al
	callq	_printf
	movsd	LCPI2_1(%rip), %xmm0    ## xmm0 = mem[0],zero
	callq	_g
	leaq	L_fmt_float.3(%rip), %rdi
	movb	$1, %al
```

```
	callq	_printf
	movsd	LCPI2_2(%rip), %xmm0    ## xmm0 = mem[0],zero
	callq	_g
	leaq	L_fmt_float.4(%rip), %rdi
	movb	$1, %al
	callq	_printf
	movsd	LCPI2_3(%rip), %xmm0    ## xmm0 = mem[0],zero
	callq	_g
	leaq	L_fmt_float.5(%rip), %rdi
	movb	$1, %al
	callq	_printf
	movsd	LCPI2_4(%rip), %xmm0    ## xmm0 = mem[0],zero
	callq	_g
	leaq	L_fmt_float.6(%rip), %rdi
	movb	$1, %al
	callq	_printf
	movsd	LCPI2_5(%rip), %xmm0    ## xmm0 = mem[0],zero
	callq	_g
	leaq	L_fmt_float.7(%rip), %rdi
	movb	$1, %al
	callq	_printf
	movsd	LCPI2_6(%rip), %xmm0    ## xmm0 = mem[0],zero
	callq	_g
	leaq	L_fmt_float.8(%rip), %rdi
	movb	$1, %al
	callq	_printf
	xorl	%eax, %eax
	popq	%rcx
	retq
	.cfi_endproc
                                        ## -- End function
	.section	__TEXT,__cstring,cstring_literals
L_fmt_float:                            ## @fmt_float
	.asciz	"%g\n"

L_fmt_float.1:                          ## @fmt_float.1
	.asciz	"%g\n"

L_fmt_float.2:                          ## @fmt_float.2
```

```
      .asciz     "%g\n"

L_fmt_float.3:                          ## @fmt_float.3
      .asciz     "%g\n"

L_fmt_float.4:                          ## @fmt_float.4
      .asciz     "%g\n"

L_fmt_float.5:                          ## @fmt_float.5
      .asciz     "%g\n"

L_fmt_float.6:                          ## @fmt_float.6
      .asciz     "%g\n"

L_fmt_float.7:                          ## @fmt_float.7
      .asciz     "%g\n"

L_fmt_float.8:                          ## @fmt_float.8
      .asciz     "%g\n"


.subsections_via_symbols
```

Lessons Learned

Weicheng Zhao

The project let me learn a lot beyond PLT itself, because what this project impressed me most is how big the difference between the language that people used in normal life, or that is to say, natural language, and programming language is. At the very beginning when we made the proposal, I thought, “Emmm… Natural languages like English are ambiguous, as we all know, but what about Math language, which should be precise and logical. And we even have LaTeX to represent these symbols in mathematics and need not to design again.” However, it was proved that there are full of ambiguous things in the mathematical world, even after we tried to limit the usage of every symbol into one, for example, a vertical bar, could be used to represent absolute value, divisible, or used in some places like E(X | X > 0) to denote the fact that the material to the left of the vertical bar is being conditioned on what's to the right of the vertical bar.

I could make some examples about the ambiguity in maths, for example, what does “\sin 2x” mean? You may claim that it is for sure that it represents “\sin (2*x)``But why? Because the

multiplication has a higher priority than the sin function, but how about “\sin x\sin y”? Why does not it mean “(\sin x)*(\sin y)” instead of “\sin (x * \sin y)”? Or maybe it is just because most people would consider unreasonable to compute “\sin 2” or “\sin (x * \sin y)” in most cases, but there is no doubt that there is ambiguity. Another example is what does “| a | b | c |` mean? Is it “| a | * b * | c |” or “| a * | b | * c |”? I was about to make a decision between these two ways of explanation so that we could eliminate ambiguity in this part, however, even in our team we get a 2 vs 2 result, at the end we have no choice but to let the user decide manually about left and right. Even after we worked a lot on the AST, we even found that if there is a “f(a)”, should it be a “f*a”, or a function call or even a function definition? However, I did not notice these problems before this project and had no idea how much common sense helps in everyday life, and the reason why programming languages are different from natural languages is because the computer does not have common sense, for example, there is no difference between computing “\sin 2” and “\sin 2\pi” for it, while humans would surely think there might be something wrong to compute “\sin 2”. This could even apply to the natural language, since if two persons do not share the same common sense, they will for sure have a totally different understanding of the same words, and in most cases, it brings meaningless debates or even tussles.

Back to PLT stuff, the project and the course let me know functional programming and OCaml language, as well as the thinking in functional programming. It also brings me a further understanding of LR syntax, and LLVM IR. Especially because the documentation of the LLVM module in OCaml does not help a lot thus we have to dig into the source and refer to the documents of the LLVM IR itself to get a better understanding how it works and how to generate the correct LLVM IR in the codegen. I also improved my skills on devOps by developing the test suites and preparing the Continuous Integration.

Hu Zheng

It has long been my wish to make mathematical expressions written in LaTeX computable and even verifiable. And I’m so happy to be able to partially realize that in this project. Here are the lesions that I have learned:

1. You have to make tradeoffs in deciding which phase you want to reject unexpected inputs. It may be possible to do it earlier or later, but the cost varies. Initially I wanted to handle compound identifiers like `\mathrm{\alpha this is an identifier}` in the parser, but Weicheng insisted that we can do it in the scanner. It turned out that we made it. On the other hand, I handled identifiers with subscripts in the parser, which seemed more elegant than doing that in the scanner.
2. We might need more powerful parser generators. We suffered when trying to parse absolute value expressions, since the left and right delimiters are the same. This made it not feasible in LR(1) grammar. We ended up letting the user manually appoint whether the delimiter is left or right (with \left\|, for example), which is not natural at all.

However, ANTLR, which is an LL(*) parser generator, seems to be able to deal with the problem elegantly.

3. Refactoring is hard. When making conceptually simple modifications, like adding support for identifiers with subscripts, you have to make edits everywhere. It is even harder for more complex refactors, like changing the precedence of different operators. Maybe there is a more efficient way to do this, but we haven't found it.
4. It is interesting to look at some inherent ambiguities in mathematical expressions that are often overlooked. The meaning of expressions like `sin 2x` and `sin x cos x` are clear in conventions, but it is hard to express them rigorously.
5. It's exciting to see your concepts being implemented and eventually working. For example, when dealing with the ambiguity of expressions like f(x) (whether it's a function call or multiplication), I had the idea of simply treating it as a combination without semantics in the parser, and deciding what it actually is in the type checker. It's great that we manage to realize this idea.
6. It's a pity that we didn't go as far as symbolic calculations, let alone formal verifications, for the sake of approachability. It's really far more work than I imagined, as the numerical part already took us a semester. Nevertheless, I would not refrain from pursuing this ultimate goal in the future.

Advice: Testing is important. A robust test suite allowed us to focus on refactoring to approach our goal. I couldn't imagine how that could be possible without the tests.

Rachel Liu

While doing the project, I felt the difficulty in translating a mathematical expression into a programming language. Even though OCaml is a quite straightforward language, which is close to how we think, there are still many boundaries and ambiguities. As we added more functions, we faced more problems. We had to consider each math expression in all aspects, in order to reduce the ambiguity to the program. Also, every time we changed the parser, we had to change code generation, SAST, AST, semantics as well. I learned to think more thoroughly from the project.

In addition, through this group project, I developed more skills in project planning and management. At first, we added many functions without doing the test at the same time. In this case, when we tried to run the Hello World program, we faced many failure cases and shift-reduce errors. In this case, we deleted many functions, and started to test each function one by one. After one test was passed, we kept adding other functions.

Advice: Doing a PLT project requires an amount of time and effort. Frequent group meetings are super important. If the situation allows, I suggest to meet in person. When you start the project, I suggest not to add too many functions at once. Keep adding more once each function is passed.

Also, OCaml has some tricky parts, which needs to be paid attention to. For example, the type of token ID is different from the type of expression_calc. ID's type need to be changed to be applied with expression_calc.

Unal Yigit Ozulku

In addition to teaching me about programming in OCaml, this project taught me a lot about different aspects of testing. Since I didn't any prior software development experience in the industry, I didn't know about different kinds of testing (unit testing, integration testing, etc.) work, and I had no prior knowledge in test automation either, so the PLT project was a great opportunity for me to learn more about shell scripting and test automation.

Another interesting lesson I gained from this class is that being a good tester is very closely tied to how well you communicate and work with other team members: for example, changing code in the parser requires you to also edit most other components within the system architecture (AST, SAST, code generator etc.), so thinking about how to create and then resolve test cases that will find errors in the compiler requires understanding of all these other components as well. In other words, it is very hard to work on a specific part of the system architecture in a vacuum, and it is crucial to understand what features the others are working on and ask questions when you don't. After all, what appears to make sense in a unit test might fail during the integration test suite based on the cross–dependencies between different code components, and picking up those possibilities allows one to create tests targeting those dependencies instead of trying random cases and hoping some of them will fail.

This brings us to my final takeaway from this project, which is to build up the codebase slowly from the ground and being very quick in finding errors. It works much better to have a working scanner and parser early and implementing functions one by one than waiting longer to merge thousands of lines of code and not understanding why errors occur. Shorter code is easier to maintain, so my advice would be to test functions one at a time and only move to other functions after resolving the shift-reduce errors.

Appendix

Git Logs

```
* commit 0dd9b3a37c6752254c73f58114c32e39b5057b5b (HEAD -> master,
origin/master, origin/HEAD)
| Author: Hu Zheng <hz2709@columbia.edu>
| Date:   Wed Dec 22 22:34:19 2021 -0500
```

```
|
|     Add test for Ackermann function. (#27)
|
| * commit b60e070ff0322435b27ccc7f5c7c527c26e639ed
(origin/zhenghu/add_ackermann_test)
|/  Author: Hu Zheng <hz2709@columbia.edu>
|   Date:   Wed Dec 22 22:01:55 2021 -0500
|
|       Add test for Ackermann function.
|
* commit 76bcfb6b55c1dcf710e9b097ebd2503fc8187308
| Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
| Date:   Tue Dec 21 11:20:07 2021 -0500
|
|     Fix failed xyz test (#26)
|
|     Co-authored-by: weichenzhao <weichenzhao@tencent.com>
|
| * commit a074d8cc0f4a94702b6a044dd97c9ac992db0613
|/  Author: weichenzhao <weichenzhao@tencent.com>
|   Date:   Tue Dec 21 11:05:24 2021 -0500
|
|       Fix failed xyz test
|
* commit c6defd99a15fbffe0f161cd034af4cf4487a0e6e
| Author: weichenzhao <weichenzhao@tencent.com>
| Date:   Tue Dec 21 02:25:37 2021 -0500
|
|     Fix make tool ld problem
|
* commit 3a59d360aa53799ef0b4026980fd9e7a73aee255
| Author: Hu Zheng <hz2709@columbia.edu>
| Date:   Tue Dec 21 02:13:52 2021 -0500
|
|     Support identifiers with subscripts. (#25)
|
|     * Add more test.
|
|     * Add more test.
```

```
|
|     * Fix tests.
|
|     * Add fibonacci test.
|
|     * Fix asm name.
|
|     * Add test for backslash in id.
|
|     * Fix escape.
|
|     * Support subscripted identifier.
|
|     * Support subscripted identifier.
|
|     * Adapt tests.
|
|     * Add more tests.
|
|     * Fix bracket.
|
|     * Fix tex issue
|
|     * Remove reduntant open
|
|     * disable large op
|
|     Co-authored-by: Axure <freetiger18@gmail.com>
|     Co-authored-by: weichenzhao <weichenzhao@tencent.com>
|
* commit f698d121d7f621bc597fbac4912bbc0b51158c00
| Author: Hu Zheng <hz2709@columbia.edu>
| Date:   Tue Dec 21 01:09:55 2021 -0500
|
|     Add test for backslash in identifier. (#24)
|
| * commit f1e2e6331d1f7839acb0d256e602e2f6ec9618df
| | Author: weichenzhao <weichenzhao@tencent.com>
| | Date:   Tue Dec 21 01:55:00 2021 -0500
```

```
| |
| |     disable large op
| |
| * commit c6d260bfad099e4209fff6bcff3b7fc6e33bcb83
| | Author: weichenzhao <weichenzhao@tencent.com>
| | Date:   Tue Dec 21 01:52:50 2021 -0500
| |
| |     Remove reduntant open
| |
| * commit a153e91e57561f91575c88fa64ac82a10aa94386
| | Author: weichenzhao <weichenzhao@tencent.com>
| | Date:   Tue Dec 21 01:50:23 2021 -0500
| |
| |     Fix tex issue
| |
| *   commit 928436ae3bb0ac245534029ca81babaad8d90608
| |\  Merge: fa3919f 23f0a58
| | | Author: Hu Zheng <hz2709@columbia.edu>
| | | Date:   Tue Dec 21 01:02:30 2021 -0500
| | |
| | |     Merge branch 'zhenghu/furnish_tests' into
zhenghu/add_underscore_ident
| | |
| | * commit 23f0a5889a5f38fb25bb8217fa9951adb6ffd709
| | | Author: Hu Zheng <hz2709@columbia.edu>
| | | Date:   Tue Dec 21 01:02:18 2021 -0500
| | |
| | |     Fix bracket.
| | |
| * | commit fa3919f736c9b9263c8d0fc9063061b034ea93b4
| | | Author: Hu Zheng <hz2709@columbia.edu>
| | | Date:   Tue Dec 21 00:56:44 2021 -0500
| | |
| | |     Add more tests.
| | |
| * | commit ad55831767b0b768b7e2f9712accb2faaf15c987
| | | Author: Hu Zheng <hz2709@columbia.edu>
| | | Date:   Tue Dec 21 00:55:35 2021 -0500
| | |
```

```
| | |     Adapt tests.
| | |
| * | commit 4c06d688ff22f7c50aa60af6b54d17505d59a501
| | | Author: Hu Zheng <hz2709@columbia.edu>
| | | Date:   Tue Dec 21 00:55:20 2021 -0500
| | |
| | |     Support subscripted identifier.
| | |
| * | commit 4d8488315e981ee336279c23224e431360bfedf6
| |/  Author: Hu Zheng <hz2709@columbia.edu>
| |   Date:   Tue Dec 21 00:54:56 2021 -0500
| |
| |       Support subscripted identifier.
| |
| * commit 204356dfb5ca9c1e270f22119306f67df2283aba
| | Author: Hu Zheng <hz2709@columbia.edu>
| | Date:   Tue Dec 21 00:54:13 2021 -0500
| |
| |     Fix escape.
| |
| *   commit e1c0aa47a7d6c5c8b27cdd43de2ad724a4d7c3f8
| |\  Merge: 2b0e953 aaf839e
| |/  Author: Hu Zheng <hz2709@columbia.edu>
|/|   Date:   Tue Dec 21 00:45:50 2021 -0500
| |
| |       Merge manually.
| |
* | commit aaf839e284de3e1057b24bf4505eeac2f9dafd89
| | Author: Hu Zheng <hz2709@columbia.edu>
| | Date:   Mon Dec 20 17:36:57 2021 -0500
| |
| |     Add more test. (#22)
| |
| |     * Add more test.
| |
| |     * Add more test.
| |
| |     * Fix tests.
| |
```

```
| |     * Add fibonacci test.
| |
| |     * Fix asm name.
| |
| |     * Add ounit.
| |
| |     * Fix link
| |
| |     Co-authored-by: Axure <freetiger18@gmail.com>
| |     Co-authored-by: weichenzhao <weichenzhao@tencent.com>
| |
| * commit 2b0e9535ed0f6f39529aa1a4b1f946ca60a800f9
| | Author: Hu Zheng <hz2709@columbia.edu>
| | Date:   Tue Dec 21 00:44:23 2021 -0500
| |
| |     Add test for backslash in id.
| |
| | * commit 7744eb3910d9259a993afd92a8bfb5f651604e63
| | | Author: weichenzhao <weichenzhao@tencent.com>
| | | Date:   Mon Dec 20 17:29:58 2021 -0500
| | |
| | |     Fix link
| | |
| | * commit 377283fc4d2da95c4b7aae2b4c86f213a536221c
| |/  Author: Hu Zheng <hz2709@columbia.edu>
| |   Date:   Mon Dec 20 17:24:31 2021 -0500
| |
| |       Add ounit.
| |
| * commit 69c7d372e0a24f3c57ea7234a1e9b7d0ae83c101
| | Author: Axure <freetiger18@gmail.com>
| | Date:   Mon Dec 20 17:16:42 2021 -0500
| |
| |     Fix asm name.
| |
| *   commit 39f95c38fa8c033364a38f5250010baa5a0ad123
| |\  Merge: e7e5d91 fc9639d
| |/  Author: Axure <freetiger18@gmail.com>
|/|   Date:   Mon Dec 20 16:58:32 2021 -0500
```

```
| |
| |       Merge remote-tracking branch 'origin/master' into
zhenghu/add_more_tests
| |
* | commit fc9639d7cf522b03d27649885fe60d545a08d271
| | Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
| | Date:   Mon Dec 20 16:58:15 2021 -0500
| |
| |     Dev_fix_fgn (#23)
| |
| |     * Fix case type issue
| |
| |     * Fix fgn parse issue
| |
| |     * Add test for Collatz conjecture.
| |
| |     Co-authored-by: weichenzhao <weichenzhao@tencent.com>
| |     Co-authored-by: Axure <freetiger18@gmail.com>
| |
| *   commit e7e5d91cd579a79560c3b27174d259456712bd69
| |\  Merge: b7b06bf b93d662
| |/  Author: Axure <freetiger18@gmail.com>
|/|   Date:   Mon Dec 20 16:47:52 2021 -0500
| |
| |       Merge branch 'master' into zhenghu/add_more_tests
| |
| * commit b7b06bfa8961fc800183232c2d7309525a0cbcdb
| | Author: Axure <freetiger18@gmail.com>
| | Date:   Mon Dec 20 15:33:44 2021 -0500
| |
| |     Add fibonacci test.
| |
| * commit 71847768d49603c5cd61bae6d220eaf7f9c566e2
| | Author: Axure <freetiger18@gmail.com>
| | Date:   Mon Dec 20 10:03:23 2021 -0500
| |
| |     Fix tests.
| |
| * commit 3f869bf931d0071ba2c8a1acab7f180e69e5e3aa
```

```
| | Author: Axure <freetiger18@gmail.com>
| | Date:   Sun Dec 19 17:46:55 2021 -0500
| |
| |     Add more test.
| |
| * commit ad06f85eb2b496d81fe7a623c5623ccd725601d9
| | Author: Axure <freetiger18@gmail.com>
| | Date:   Sun Dec 19 17:30:10 2021 -0500
| |
| |     Add more test.
| |
| | * commit 07879644c09fdbedb29b7c164b92b30ba6006e49
| | | Author: Axure <freetiger18@gmail.com>
| | | Date:   Mon Dec 20 16:41:31 2021 -0500
| | |
| | |     Add test for Collatz conjecture.
| | |
| | * commit 510ca5ca624842dedc08b15664e1c405585fd0c3
| | | Author: weichenzhao <weichenzhao@tencent.com>
| | | Date:   Mon Dec 20 16:37:17 2021 -0500
| | |
| | |     Fix fgn parse issue
| | |
| | * commit 554b8be2293ff301e93ec1b701546b7dc1559be5
| |/  Author: weichenzhao <weichenzhao@tencent.com>
|/|   Date:   Mon Dec 20 16:23:07 2021 -0500
| |
| |       Fix case type issue
| |
* | commit b93d662753d0cf3fb92513ff8d7a34dbc2f45baf
|/  Author: weichenzhao <weichenzhao@tencent.com>
|   Date:   Sun Dec 19 17:26:47 2021 -0500
|
|       Update top level makefile
|
| *   commit 9a31174996f643921f1230e2d295d0f9504dd663
| |\  Merge: 6a9e347 050d38f
| |/  Author: Axure <freetiger18@gmail.com>
|/|   Date:   Sun Dec 19 17:21:39 2021 -0500
```

```
| |
| |       Manually merge with master.
| |
* | commit 050d38fd287c0befb51e2b4d07c10365d52cde71
| | Author: Hu Zheng <hz2709@columbia.edu>
| | Date:   Sun Dec 19 17:20:06 2021 -0500
| |
| |     Add unit test and replace backslash in scanner. (#21)
| |
| * commit 6a9e347a2e53d429e901b4a04b5c454bad3fdd0a
| | Author: weichenzhao <weichenzhao@tencent.com>
| | Date:   Sun Dec 19 16:36:44 2021 -0500
| |
| |     Format everything and prepare tools
| |
| * commit 00037d05cd6148197674b672ed0cb603af919e93
| | Author: weichenzhao <weichenzhao@tencent.com>
| | Date:   Sun Dec 19 16:36:15 2021 -0500
| |
| |     Add format
| |
| * commit aa0b5e27ba1a96435cf245edb341470920d6d97d
| | Author: weichenzhao <weichenzhao@tencent.com>
| | Date:   Sat Dec 18 22:45:41 2021 -0500
| |
| |     Fix mid
| |
| * commit 45689e72aa11776a2785ef2ad28d9ddf9ffc076c
| | Author: weichenzhao <weichenzhao@tencent.com>
| | Date:   Sat Dec 18 19:45:39 2021 -0500
| |
| |     fix return case and args
| |
| * commit 795fea1394eb192e260328534fde7e5b494e3ea5
| | Author: weichenzhao <weichenzhao@tencent.com>
| | Date:   Sat Dec 18 16:54:37 2021 -0500
| |
| |     Fix segfault
| |
```

```
| * commit 9f2e980b78c97fece52d904b27079fbb6f3886c9
| | Author: weichenzhao <weichenzhao@tencent.com>
| | Date:   Sat Dec 18 11:49:03 2021 -0500
| |
| |     Fix arg type
| |
| * commit c9201ba691c2f3f10bcde52a62ec999fad8ddb00
| | Author: weichenzhao <weichenzhao@tencent.com>
| | Date:   Sat Dec 18 11:39:25 2021 -0500
| |
| |     Tiny clear to scall
| |
| * commit 94d6f685c3d52bb3fda3cce9f01f15f247e74512
| | Author: weichenzhao <weichenzhao@tencent.com>
| | Date:   Sat Dec 18 11:03:26 2021 -0500
| |
| |     add fab test
| |
| * commit 118b45c4dac255b23c6ad7518ecafe8f47830a2f
| | Author: weichenzhao <weichenzhao@tencent.com>
| | Date:   Sat Dec 18 11:02:55 2021 -0500
| |
| |     fix codegen call
| |
| *   commit b0f0126c8fe269bd0762b9731eb1943b38fcd635
| |\  Merge: c0c4af2 f4f6874
| | | Author: Axure <freetiger18@gmail.com>
| | | Date:   Sat Dec 18 10:52:50 2021 -0500
| | |
| | |     Merge remote-tracking branch 'origin/master' into
dev_fix_fa
| | |
| * | commit c0c4af2f013d07c33c3f2c5ba150ff09851f7ebc
| | | Author: weichenzhao <weichenzhao@tencent.com>
| | | Date:   Sat Dec 18 02:57:49 2021 -0500
| | |
| | |     little fix
| | |
| * | commit 4abb27db38a3a48581e67513f609324b75c889e9
```

```
| | | Author: weichenzhao <weichenzhao@tencent.com>
| | | Date:   Sat Dec 18 02:37:06 2021 -0500
| | |
| | |     fix unable to find identifier in type checker
| | |
| * | commit 58ae9fe8517263dd893d99ee9d2ffb88a5673fff
| | | Author: Unal Yigit Ozulku <uyozulku@gmail.com>
| | | Date:   Fri Dec 17 19:15:40 2021 -0500
| | |
| | |     updated tests relating to functions definitions
| | |
| * | commit 7f3167074a00fbe0aea62b03b28cc75f31b23e88
| | | Author: Unal Yigit Ozulku <uyozulku@gmail.com>
| | | Date:   Fri Dec 17 19:03:00 2021 -0500
| | |
| | |     added new tests
| | |
| * | commit e0618c3d6eb91521766df8c2e6091ca589d8b4fe
| | | Author: Unal Yigit Ozulku <uyozulku@gmail.com>
| | | Date:   Fri Dec 17 18:53:52 2021 -0500
| | |
| | |     add tests
| | |
| * | commit 2221f5c03a2fe652de21ce10bf26aa7258df0fdb
| | | Author: weichenzhao <weichenzhao@tencent.com>
| | | Date:   Fri Dec 17 18:25:36 2021 -0500
| | |
| | |     Refine semant and codegen
| | |
| * | commit 3113bddfec77fb000f45b9bdf26ec07ee0c992ea
| | | Author: Hu Zheng <hz2709@columbia.edu>
| | | Date:   Fri Dec 17 18:03:01 2021 -0500
| | |
| | |     Add scanner test cases.
| | |
| * | commit 715008f2a35b6cfd70df0b9ef96d617f1c8c31de
| | | Author: Hu Zheng <hz2709@columbia.edu>
| | | Date:   Fri Dec 17 18:01:40 2021 -0500
| | |
```

```
| | |     Fix redundant DBS.
| | |
| * | commit beceeb200c526bdefc420c6f8cbc90413ec827a4
| | | Author: Hu Zheng <hz2709@columbia.edu>
| | | Date:   Fri Dec 17 18:00:24 2021 -0500
| | |
| | |     Add some test cases.
| | |
| | | * commit af8af227272aa94057f319646bfe508502702d42
| | | | Author: Axure <freetiger18@gmail.com>
| | | | Date:   Sun Dec 19 17:19:34 2021 -0500
| | | |
| | | |     Remove redundant debug.
| | | |
| | | * commit a852350179337e2742291e9d7e42dc9531e5fff5
| |_|/  Author: Axure <freetiger18@gmail.com>
|/| |   Date:   Sun Dec 19 17:18:57 2021 -0500
| | |
| | |       Add unit test and replace backslash in scanner.
| | |
* | | commit 537166474863a085ac7a0c75f2c951f56ad099df
| |/  Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
|/|   Date:   Sun Dec 19 16:44:24 2021 -0500
| |
| |       Dev_fix_fa (#20)
| |
| |       * refine parser
| |
| |       * refine again
| |
| |       * ast edit
| |
| |       * ast rework
| |
| |       * merge func in ast
| |
| |       * Add support of no args in func call and def
| |
| |       * sast codegen edit1
```

```
| |
| |       * Add pretty print for ast
| |       Fix conflict
| |
| |       * prepare for sast
| |
| |       * semant edit1
| |
| |       * Fix a scanner output.
| |
| |       * Fix call_expr.
| |
| |       * Add some test cases.
| |
| |       * Fix redundant DBS.
| |
| |       * Add scanner test cases.
| |
| |       * Refine semant and codegen
| |
| |       * add tests
| |
| |       * added new tests
| |
| |       * updated tests relating to functions definitions
| |
| |       * fix unable to find identifier in type checker
| |
| |       * little fix
| |
| |       * fix codegen call
| |
| |       * add fab test
| |
| |       * Tiny clear to scall
| |
| |       * Fix arg type
| |
| |       * Fix segfault
```

```
| |
| |       * fix return case and args
| |
| |       * Fix mid
| |
| |       * Add format
| |
| |       * Format everything and prepare tools
| |
| |       Co-authored-by: weichenzhao <weichenzhao@tencent.com>
| |       Co-authored-by: Rachel <wl2784@columbia.edu>
| |       Co-authored-by: Hu Zheng <hz2709@columbia.edu>
| |       Co-authored-by: Unal Yigit Ozulku <uyozulku@gmail.com>
| |       Co-authored-by: Axure <freetiger18@gmail.com>
| |
* | commit f4f6874571ea5f83f577cf6a81e0b140c0d8deaa
| | Author: Hu Zheng <hz2709@columbia.edu>
| | Date:   Sat Dec 18 10:50:49 2021 -0500
| |
| |     Fix value of digit. (#19)
| |
| |     Was ASCII.
| |
| | * commit 4632bbfef607daff055187bb4cd05f4843c64442
| |/  Author: Hu Zheng <hz2709@columbia.edu>
|/|   Date:   Sat Dec 18 10:50:41 2021 -0500
| |
| |       Fix value of digit.
| |
| |       Was ASCII.
| |
| | * commit 858b3dfd4bca3db91f6140fd07cc962b0e65f908
| | | Author: Unal Yigit Ozulku <uyozulku@gmail.com>
| | | Date:   Fri Dec 17 17:29:04 2021 -0500
| | |
| | |     testing function definitions
| | |
| | *   commit 4aac66795742e1a199611004f9e7ef9ca5748a10
| | |\  Merge: 1326d1f 79b0301
```

```
| | |/  Author: Unal Yigit Ozulku <uyozulku@gmail.com>
| |/|   Date:   Fri Dec 17 17:25:10 2021 -0500
| | |
| | |       Merge remote-tracking branch 'origin/dev_fix_fa' into
unal-tests
| | |
| * | commit 79b0301075327fc98f999788c2031f5d201b09f8
| | | Author: Hu Zheng <hz2709@columbia.edu>
| | | Date:   Fri Dec 17 17:01:08 2021 -0500
| | |
| | |     Fix call_expr.
| | |
| * | commit 0587f4bcaaf127f6a4d1c4c572b6b3d38eaf8908
| | | Author: Hu Zheng <hz2709@columbia.edu>
| | | Date:   Fri Dec 17 17:00:43 2021 -0500
| | |
| | |     Fix a scanner output.
| | |
| | * commit 1326d1f1103452e371e39ef62d390c807cd7de67
| | | Author: Unal Yigit Ozulku <uyozulku@gmail.com>
| | | Date:   Fri Dec 17 16:55:43 2021 -0500
| | |
| | |     associativity tests added
| | |
| | *   commit 608537cdda093feedbf847c3ec35bf6ddee7048d
| | |\  Merge: 9646256 d066006
| | |/  Author: Unal Yigit Ozulku <uyozulku@gmail.com>
| |/|   Date:   Fri Dec 17 16:50:54 2021 -0500
| | |
| | |       Merge remote-tracking branch 'origin/dev_fix_fa' into
unal-tests
| | |
| * | commit d066006f94164352b8cc93776e287bda0848bd7a
| | | Author: Rachel <wl2784@columbia.edu>
| | | Date:   Fri Dec 17 15:15:57 2021 -0500
| | |
| | |     semant edit1
| | |
| * | commit 7057d149bcb1880dc529c4dad6eed756ef03f289
```

```
| | | Author: weichenzhao <weichenzhao@tencent.com>
| | | Date:   Fri Dec 17 14:22:30 2021 -0500
| | |
| | |     prepare for sast
| | |
| * | commit 312cf94a41bcdf0488399974e88ae161a7c3eb9e
| | | Author: weichenzhao <weichenzhao@tencent.com>
| | | Date:   Fri Dec 17 13:59:12 2021 -0500
| | |
| | |     Add pretty print for ast
| | |     Fix conflict
| | |
| * | commit 4bf9e5e65ebe60eb8fc8f96a77485ceca14d632b
| | | Author: Rachel <wl2784@columbia.edu>
| | | Date:   Fri Dec 17 13:51:56 2021 -0500
| | |
| | |     sast codegen edit1
| | |
| * | commit c37df658a69804382d629d16e48174cfdd6619bb
| | | Author: weichenzhao <weichenzhao@tencent.com>
| | | Date:   Fri Dec 17 11:39:47 2021 -0500
| | |
| | |     Add support of no args in func call and def
| | |
| * | commit cfc0e54d1df98766dbf50b2dd70f1d395d19b63c
| | | Author: weichenzhao <weichenzhao@tencent.com>
| | | Date:   Fri Dec 17 11:04:00 2021 -0500
| | |
| | |     merge func in ast
| | |
| * | commit cb75877290a0ac7248fa12450a1de0fe86645c40
| | | Author: weichenzhao <weichenzhao@tencent.com>
| | | Date:   Thu Dec 16 20:44:20 2021 -0500
| | |
| | |     ast rework
| | |
| * | commit 3be393ee126079a28eb96e8f99074d5ca3e00347
| | | Author: Rachel <wl2784@columbia.edu>
| | | Date:   Thu Dec 16 17:42:49 2021 -0500
```

```
| | |
| | |     ast edit
| | |
| * | commit 88a6f51c5391786fc7f45c9c086fa9515e16fde9
| | | Author: weichenzhao <weichenzhao@tencent.com>
| | | Date:   Thu Dec 16 14:41:08 2021 -0500
| | |
| | |     refine again
| | |
| * | commit fe721825c975dfdcc7eabc36b4e3e40b7a813b86
|/ /  Author: weichenzhao <weichenzhao@tencent.com>
| |   Date:   Thu Dec 16 14:07:06 2021 -0500
| |
| |       refine parser
| |
* | commit 15d834a21e3f9943451f5f5fcd9e8a7f54ffb309
| | Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
| | Date:   Thu Dec 16 13:08:58 2021 -0500
| |
| |     Rachel/fix grammar (#17)
| |
| |     * Fix grammar.
| |
| |     * ast scanner update
| |
| |     * grammar fixed
| |
| |     * refine
| |
| |     * refine
| |
| |     * refine codegen
| |
| |     Co-authored-by: Rachel <wl2784@columbia.edu>
| |     Co-authored-by: weichenzhao <weichenzhao@tencent.com>
| |
| * commit 9646256f95310ea496caac7a421d20e815940326
|/  Author: Unal Yigit Ozulku <uyozulku@gmail.com>
|   Date:   Fri Dec 17 16:20:17 2021 -0500
```

```
|
|       added new tests
|
| * commit 7b67bda9e2112542e92638d5dae7909934cfeed6
| | Author: weichenzhao <weichenzhao@tencent.com>
| | Date:   Thu Dec 16 12:59:15 2021 -0500
| |
| |     refine codegen
| |
| * commit 35ccebed382aed33436b27ea108ca52d1ef77269
| | Author: weichenzhao <weichenzhao@tencent.com>
| | Date:   Thu Dec 16 12:44:06 2021 -0500
| |
| |     refine
| |
| * commit b1a36f609d996f8baefa728a4125670c59afb7fa
| | Author: weichenzhao <weichenzhao@tencent.com>
| | Date:   Thu Dec 16 12:23:36 2021 -0500
| |
| |     refine
| |
| *   commit d332c88f73c6ffb6d5904f34218ea7e7ac1d206e
| |\  Merge: d088d9e d145906
| |/  Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
|/|   Date:   Thu Dec 16 12:12:33 2021 -0500
| |
| |       Merge branch 'master' into rachel/fix-grammar
| |
* | commit d145906bd6be1bdc18d42ddcd9b4a539ba8ee4b6
| | Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
| | Date:   Thu Dec 16 12:10:59 2021 -0500
| |
| |     Dev_wz_codegen_iter3 (#16)
| |
| |     * Add math funcs to codegen
| |
| |     * almost finish up codegen
| |
| |     * Fix the else builder
```

```
| |
| |     Co-authored-by: weichenzhao <weichenzhao@tencent.com>
| |
| * commit d088d9eacac3eb27debb8f50fb5a1ccfbbd72eeb
| | Author: Rachel <wl2784@columbia.edu>
| | Date:   Wed Dec 15 17:25:37 2021 -0500
| |
| |     grammar fixed
| |
| * commit d3f584b82320dfd8d31482d3d03168df00e8a06c
| | Author: Rachel <wl2784@columbia.edu>
| | Date:   Fri Dec 3 06:04:59 2021 -0500
| |
| |     ast scanner update
| |
| * commit 91e334fff086894bcb82988c454ad8d7624f7ba0
|/  Author: Rachel <wl2784@columbia.edu>
|   Date:   Tue Nov 30 05:22:42 2021 -0500
|
|       Fix grammar.
|
| * commit 312d5a2f49f62ac29662dca9ef58172b18da972c
| | Author: weichenzhao <weichenzhao@tencent.com>
| | Date:   Sat Dec 4 00:46:26 2021 -0500
| |
| |     Fix the else builder
| |
| * commit abf8d4566e7d795dd29f7cbc44c6b62df33ac712
| | Author: weichenzhao <weichenzhao@tencent.com>
| | Date:   Sat Dec 4 00:41:59 2021 -0500
| |
| |     almost finish up codegen
| |
| * commit 18640522f6896d694cd7b6842ef478eb3557ae37
|/  Author: weichenzhao <weichenzhao@tencent.com>
|   Date:   Tue Nov 23 20:32:59 2021 -0500
|
|       Add math funcs to codegen
|
```

```
* commit 03b488594fb37fa71493f6b9c4c0e83cc3505720
| Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
| Date:   Tue Nov 23 16:57:56 2021 -0500
|
|     Change the structure of sast (#15)
|
|     Co-authored-by: weichenzhao <weichenzhao@tencent.com>
|
* commit b31f233bc8e40640fd0d6dd05df11c8ed17c1e1c
| Author: weichenzhao <weichenzhao@tencent.com>
| Date:   Mon Nov 22 15:22:51 2021 -0500
|
|     Fix stupid SR conflict
|
* commit 89911920fbf2e165a3efdc34b42c37b7bd28a1dd
| Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
| Date:   Mon Nov 22 13:07:06 2021 -0500
|
|     Fix empty stmt list (#14)
|
|     Co-authored-by: weichenzhao <weichenzhao@tencent.com>
|
| * commit 17bb0b2a6af85df8aa1e31d15a20e63426353b51
| | Author: Hu Zheng <hz2709@columbia.edu>
| | Date:   Mon Nov 29 15:23:07 2021 -0500
| |
| |     Stash.
| |
| | * commit 3c3f163e3dddf4b2ae3a71da744ac80272acbf22
| |/  Author: weichenzhao <weichenzhao@tencent.com>
|/|   Date:   Mon Nov 22 12:49:31 2021 -0500
| |
| |       Fix empty stmt list
| |
* | commit fe6a0cb31db9dd0f70792690cc931e97d7a56431
| | Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
| | Date:   Mon Nov 22 11:04:49 2021 -0500
| |
| |     Fix priority of the implicit multiply (#12)
```

```
| |
| |     * Add parser test cases
| |
| |     * Add e2e cases
| |
| |     * Use modern math env
| |
| |     * make auto tools better
| |
| |     * Add 2 test cases
| |
| |     * Move the prio of the impl mult
| |
| |     * Fix path issue in sh
| |
| |     * try to fix actions
| |
| |     * Try to fix the action
| |
| |     * fix permission issue
| |
| |     Co-authored-by: weichenzhao <weichenzhao@tencent.com>
| |
* | commit f0e6ad37e395f1a5d6117c86853e1659f649e2b6
| | Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
| | Date:   Sun Nov 21 01:12:25 2021 -0500
| |
| |     Update makefile.yml
| |
| | * commit 4401bc4d05dc87813b041b45ef63b8e946f03f62
| | | Author: weichenzhao <weichenzhao@tencent.com>
| | | Date:   Mon Nov 22 10:57:16 2021 -0500
| | |
| | |     fix permission issue
| | |
| | * commit 67711d8332b4db4cc842353c77cff933472df8bf
| | | Author: weichenzhao <weichenzhao@tencent.com>
| | | Date:   Mon Nov 22 10:43:43 2021 -0500
| | |
```

```
| | |     Try to fix the action
| | |
| | * commit 9a51d069955305c95a8af35dad633ad98907382e
| | | Author: weichenzhao <weichenzhao@tencent.com>
| | | Date:   Sun Nov 21 10:05:15 2021 -0500
| | |
| | |     try to fix actions
| | |
| | * commit dfbe40420d89040fa9d91b7c71b52414243c32ab
| | | Author: weichenzhao <weichenzhao@tencent.com>
| | | Date:   Sun Nov 21 02:09:21 2021 -0500
| | |
| | |     Fix path issue in sh
| | |
| | * commit 7561f4626750da070bec1d8f46389348dbb6d63a
| | | Author: weichenzhao <weichenzhao@tencent.com>
| | | Date:   Sun Nov 21 01:57:02 2021 -0500
| | |
| | |     Move the prio of the impl mult
| | |
| | * commit 3dda6d0116f952f3ea2e7c089afdf97eb41a74f6
| | | Author: weichenzhao <weichenzhao@tencent.com>
| | | Date:   Sun Nov 21 01:56:47 2021 -0500
| | |
| | |     Add 2 test cases
| | |
| | * commit 1e7115ef4ba77c44e0b40cae9dae33c4eae9d623
| | | Author: weichenzhao <weichenzhao@tencent.com>
| | | Date:   Sun Nov 21 01:56:27 2021 -0500
| | |
| | |     make auto tools better
| | |
| | * commit 240892de506586312105aac143b7568078255fa2
| | | Author: weichenzhao <weichenzhao@tencent.com>
| | | Date:   Sun Nov 21 01:28:07 2021 -0500
| | |
| | |     Use modern math env
| | |
| | * commit 358dd5f23101381079c400c076a5cfe75e50e183
```

```
| | | Author: weichenzhao <weichenzhao@tencent.com>
| | | Date:   Sun Nov 21 01:27:54 2021 -0500
| | |
| | |     Add e2e cases
| | |
| | * commit 37670cd157702a9afc20382fc98e012fe41a1bb2
| |/  Author: weichenzhao <weichenzhao@tencent.com>
|/|   Date:   Sun Nov 21 01:27:14 2021 -0500
| |
| |       Add parser test cases
| |
* | commit 0b62890daf0c81f6c33557d88d8ac61eb464cb1a
| | Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
| | Date:   Sun Nov 21 00:58:24 2021 -0500
| |
| |     Update makefile.yml
| |
* | commit 5f906ebb950c18e6f696e9862d65753287595347
| | Author: Hu Zheng <hz2709@columbia.edu>
| | Date:   Fri Nov 19 16:29:44 2021 -0500
| |
| |     Fix scanner test. (#10)
| |
| |     * Fix scanner test.
| |
| |     * Fix scanner test.
| |
* | commit 17b21b5aa9100d985601cf757c503ffc1f058441
|/  Author: Hu Zheng <hz2709@columbia.edu>
|   Date:   Fri Nov 19 16:29:12 2021 -0500
|
|       Fix permission. (#9)
|
| * commit be3d4c96ccdc32991b7a19ebceac74ef1610c499
| | Author: Hu Zheng <hz2709@columbia.edu>
| | Date:   Fri Nov 19 16:28:54 2021 -0500
| |
| |     Fix scanner test.
| |
```

```
| * commit 7ebb8350b36f8992fb36f904f4ef4a89579ab390
| | Author: Hu Zheng <hz2709@columbia.edu>
| | Date:   Fri Nov 19 16:27:14 2021 -0500
| |
| |     Fix scanner test.
| |
| | * commit 4d95e2205599d77dd8c1f783d88a751bf06a6a7b
| |/  Author: Hu Zheng <hz2709@columbia.edu>
|/|   Date:   Fri Nov 19 16:28:22 2021 -0500
| |
| |       Fix permission.
| |
* | commit 88a233937356b69b8170716c510bbc94af735863
|/  Author: Hu Zheng <hz2709@columbia.edu>
|   Date:   Fri Nov 19 16:27:35 2021 -0500
|
|       Fix scanner test. (#8)
|
* commit 6aec44cc1ea6590ddf130d28418f9eb4aaa08c47
| Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
| Date:   Mon Nov 15 13:54:00 2021 -0500
|
|     Update makefile.yml
|
* commit dae01407aa8cccfddf7c1dba00f9bcc63d259bc3
| Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
| Date:   Sat Nov 13 19:21:36 2021 -0500
|
|     Update makefile.yml
|
* commit ba81b9f1b934a15a36ad6884c9f044acc7cd8a64
| Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
| Date:   Sat Nov 13 19:16:35 2021 -0500
|
|     Update makefile.yml
|
* commit 908c9e8f6c6d4abf3086fe62893ebc2fe557e25d
| Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
| Date:   Sat Nov 13 19:11:32 2021 -0500
```

```
|
|     Update makefile.yml
|
* commit 2c15d0ced9f019e54752a7b5e601cda75249b65e
| Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
| Date:   Sat Nov 13 19:03:22 2021 -0500
|
|     Create makefile.yml
|
* commit 8b2fcc7dcbfcd0e0327c10c0e30fee638f2c98b8
| Author: weichenzhao <weichenzhao@tencent.com>
| Date:   Sat Nov 13 17:49:38 2021 -0500
|
|     Add e2e test sutie and top-level makefile
|
* commit e7905d4025217298e69ac5864f1e18b473686ecd
| Author: weichenzhao <weichenzhao@tencent.com>
| Date:   Sat Nov 13 16:24:07 2021 -0500
|
|     Fix scanner and parser test suite
|
*   commit af1eaf77a455ebc3995f76ca9ae6bc1198713e1b
|\  Merge: 8317d94 a57de55
| | Author: raccchel <wl2784@columbia.edu>
| | Date:   Sat Nov 13 03:37:54 2021 -0500
| |
| |     checkMerge branch 'master' of
https://github.com/cs4115-CTeX/CTeX
| |
| * commit a57de559ca51fde42894ac13e3e504f47fd4eccc
| | Author: Axure <freetiger18@gmail.com>
| | Date:   Sat Nov 13 01:40:18 2021 -0500
| |
| |     Comment scanner.
| |
* | commit 8317d944709f0c1559126fa4693f6a76fcc1bce7
|/  Author: raccchel <wl2784@columbia.edu>
|   Date:   Sat Nov 13 03:37:11 2021 -0500
|
```

```
|       parser new edit
|
* commit 50977e51b1c5f5ccbc296af858fee6be73ee9725
| Author: raccchel <wl2784@columbia.edu>
| Date:   Sat Nov 13 01:36:28 2021 -0500
|
|     final edit parser
|
* commit 01f114eb41e1da43ee35343ba5fc0e26233d1ed5
| Author: Axure <freetiger18@gmail.com>
| Date:   Sat Nov 13 01:11:45 2021 -0500
|
|     Add digit.
|
* commit 905c94bc72e05dc9d41333084194baf6dfa4d00c
| Author: raccchel <wl2784@columbia.edu>
| Date:   Sat Nov 13 01:07:19 2021 -0500
|
|     add parser
|
| *   commit 3300c8a192af4a79966c744d2fba64a2ef668107
| |\  Merge: 235774a 4770e83
| |/  Author: weichenzhao <weichenzhao@tencent.com>
|/|   Date:   Sat Nov 13 01:04:26 2021 -0500
| |
| |       Merge branch 'master' into dev_wz
| |
* |   commit 4770e83a981be17fd226f98f663cb8697e0305fd
|\ \  Merge: 0180f4d a79ae7d
| | | Author: raccchel <wl2784@columbia.edu>
| | | Date:   Sat Nov 13 00:57:08 2021 -0500
| | |
| | |     merge
| | |
* | | commit 0180f4d161dd53e0ecc0f835d4ae38d8880de01b
| | | Author: raccchel <wl2784@columbia.edu>
| | | Date:   Sat Nov 13 00:53:29 2021 -0500
| | |
| | |     add scanner
```

```
| | |
* | | commit a744626fd5145168d2242648e263412aff70d585
| | | Author: Wanchen Liu <wanchenliu@wanchendembp.myfiosgateway.com>
| | | Date:   Sat Nov 13 00:50:08 2021 -0500
| | |
| | |      x
| | |
| | *   commit 235774ac3bd6ca52358e6d9f8fc7677d335675f3
| | |\  Merge: 7eb7a16 11f74dd
| | | | Author: weichenzhao <weichenzhao@tencent.com>
| | | | Date:   Sat Nov 13 00:49:40 2021 -0500
| | | |
| | | |     Merge branch 'dev_wz' of github.com:cs4115-CTeX/CTeX into
dev_wz
| | | |
| | | * commit 11f74dd8c76e6fa9ed645a1c2c176df78cd7a5e4
| | | | Author: raccchel <66702203+raccchel@users.noreply.github.com>
| | | | Date:   Fri Nov 12 23:49:46 2021 -0500
| | | |
| | | |     Update parser.mly
| | | |
| | * | commit 7eb7a16fdadab6058682d77604b3421917d9366a
| | |/  Author: weichenzhao <weichenzhao@tencent.com>
| | |   Date:   Sat Nov 13 00:47:50 2021 -0500
| | |
| | |       Add open to ctex.ml
| | |
| | * commit 3e40b68aedc85ad88fe99021e4f3a08743ece451
| |/  Author: weichenzhao <weichenzhao@tencent.com>
| |   Date:   Fri Nov 12 23:43:28 2021 -0500
| |
| |       fix parser
| |
| * commit a79ae7d2ad0d18f3530447dfdc194da5ae3ce910
| | Author: raccchel <66702203+raccchel@users.noreply.github.com>
| | Date:   Fri Nov 12 18:32:38 2021 -0500
| |
| |     Add files via upload
| |
```

```
| * commit 593d67c9df74bc0a047f27811a9be34dbd9334e0
| | Author: Hu Zheng <hz2709@columbia.edu>
| | Date:   Fri Nov 12 18:28:47 2021 -0500
| |
| |     Add EOF for print enclosure. (#7)
| |
| * commit 26059240a9957639b75f50fa74e3baf21c331f56
| | Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
| | Date:   Fri Nov 12 18:28:31 2021 -0500
| |
| |     Dev wz (#6)
| |
| |     * Test: add test tools and basic test cases for the scanner
| |
| |     * Fix: fix wrong git ignore setting.
| |
| |     * Update .gitignore
| |
| |     * Fix: modify run_tests.sh to support test cases that does
not provide a reference file
| |
| |     * Chore: add new line after the files
| |
| |     * Add test for basic identifier
| |
| |     * Fix a case that should be rejected
| |
| |     * Tests: add multiple tests for the scanner
| |
| |     * Add parser test
| |
| |     * Split common paren and paren like op
| |
| |     * sast first version
| |
| |     * Convert to LF
| |
| |     * fix name
| |
```

```
| |     * merge rebase
| |
| |     * Add test for basic identifier
| |
| |     * Fix a case that should be rejected
| |
| |     * Tests: add multiple tests for the scanner
| |
| |     * merge rebase
| |
| |     * merge rebase
| |
| |     * fix name
| |
| |     * Add type checker
| |
| |     * Add entry for semant
| |
| |     * first version of code gen
| |
| |     * Draft ver of codegen
| |
| |     Co-authored-by: weichenzhao <weichenzhao@tencent.com>
| |     Co-authored-by: Hu Zheng <hz2709@columbia.edu>
| |
| | * commit b4edc03102b09858d272e518dce4024f720f60a9
| |/  Author: Hu Zheng <hz2709@columbia.edu>
| |   Date:   Fri Nov 12 18:24:47 2021 -0500
| |
| |       Add EOF for print enclosure.
| |
| * commit 05df622b82042a8682f2b786eebb9ff9de525ee8
|/  Author: Hu Zheng <hz2709@columbia.edu>
|   Date:   Fri Nov 12 16:26:31 2021 -0500
|
|       Add EOF to the print list.
|
* commit 05b7a241846c83e41c88e8f81e234538cc9fadf0
| Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
```

```
| Date:   Fri Nov 12 16:14:52 2021 -0500
|
|     Dev wz (#5)
|
|     * Test: add test tools and basic test cases for the scanner
|
|     * Fix: fix wrong git ignore setting.
|
|     * Update .gitignore
|
|     * Fix: modify run_tests.sh to support test cases that does not
provide a reference file
|
|     * Chore: add new line after the files
|
|     * Add test for basic identifier
|
|     * Fix a case that should be rejected
|
|     * Tests: add multiple tests for the scanner
|
|     * Add parser test
|
|     * Split common paren and paren like op
|
|     * sast first version
|
|     * Convert to LF
|
|     * fix name
|
|     * merge rebase
|
|     * Add test for basic identifier
|
|     * Fix a case that should be rejected
|
|     * Tests: add multiple tests for the scanner
|
```

```
|     * merge rebase
|
|     * merge rebase
|
|     * fix name
|
|     * Add type checker
|
|     * Add entry for semant
|
|     Co-authored-by: weichenzhao <weichenzhao@tencent.com>
|     Co-authored-by: Hu Zheng <hz2709@columbia.edu>
|
* commit cdf9494dc8726eb24513cbd282cd7f9a7c29c8bd
| Author: Hu Zheng <hz2709@columbia.edu>
| Date:   Thu Nov 11 19:49:45 2021 -0500
|
|     Add scanner. (#3)
|
|     * Add scanner.
|
|     * Add to Makefile.
|
|     Co-authored-by: Axure <freetiger18@gmail.com>
|     Co-authored-by: 朝尘
<35715683+ZhaoWeicheng98@users.noreply.github.com>
|
| *   commit fa318b1cceacf95561a71c398019a8460abd5420
| |\  Merge: ae9f88d 40938c5
| |/  Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
|/|   Date:   Thu Nov 11 19:49:40 2021 -0500
| |
| |       Merge branch 'master' into zhenghu_scanner
| |
* | commit 40938c5d576b12e30baa4eca62abbc18091c67f4
| | Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
| | Date:   Thu Nov 11 19:40:14 2021 -0500
| |
| |     Dev wz (#4)
```

```
| |
| |     * Test: add test tools and basic test cases for the scanner
| |
| |     * Fix: fix wrong git ignore setting.
| |
| |     * Update .gitignore
| |
| |     * Fix: modify run_tests.sh to support test cases that does
not provide a reference file
| |
| |     * Chore: add new line after the files
| |
| |     * Add test for basic identifier
| |
| |     * Fix a case that should be rejected
| |
| |     * Tests: add multiple tests for the scanner
| |
| |     * Add parser test
| |
| |     * Split common paren and paren like op
| |
| |     * sast first version
| |
| |     * Convert to LF
| |
| |     * fix name
| |
| |     Co-authored-by: weichenzhao <weichenzhao@tencent.com>
| |     Co-authored-by: Hu Zheng <hz2709@columbia.edu>
| |
* | commit ae5ce955bd4adc73cb2520bf9d33fa0c95804ddc
| | Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
| | Date:   Thu Nov 11 16:42:18 2021 -0500
| |
| |     Dev_wz (#2)
| |
| |     * Test: add test tools and basic test cases for the scanner
| |
```

```
| |     * Fix: fix wrong git ignore setting.
| |
| |     * Update .gitignore
| |
| |     * Fix: modify run_tests.sh to support test cases that does
not provide a reference file
| |
| |     * Chore: add new line after the files
| |
| |     * Add test for basic identifier
| |
| |     * Fix a case that should be rejected
| |
| |     * Tests: add multiple tests for the scanner
| |
| |     * Add parser test
| |
| |     Co-authored-by: weichenzhao <weichenzhao@tencent.com>
| |     Co-authored-by: Hu Zheng <hz2709@columbia.edu>
| |
| * commit ae9f88d44023f7c76217833e0b76bb0eb5d2b54e
| | Author: Axure <freetiger18@gmail.com>
| | Date:   Thu Nov 11 19:16:52 2021 -0500
| |
| |     Add to Makefile.
| |
| * commit a4b8a928b8c29739897d8a591a5130cb9d21d3af
|/  Author: Axure <freetiger18@gmail.com>
|   Date:   Thu Nov 11 19:15:05 2021 -0500
|
|       Add scanner.
|
* commit 820f778066cfdcc5074f5d3a0dd8fa70ecb8589d
| Author: 朝尘 <35715683+ZhaoWeicheng98@users.noreply.github.com>
| Date:   Wed Oct 27 14:47:57 2021 -0400
|
|     Test: add test tools and basic test cases for the scanner (#1)
|
|     * Test: add test tools and basic test cases for the scanner
```

```
|
|     * Fix: fix wrong git ignore setting.
|
|     * Update .gitignore
|
|     * Fix: modify run_tests.sh to support test cases that does not
provide a reference file
|
|     * Chore: add new line after the files
|
|     Co-authored-by: weichenzhao <weichenzhao@tencent.com>
|     Co-authored-by: Hu Zheng <hz2709@columbia.edu>
|
* commit 735c419b8cd2aaaf2fd31ab9bc202eb63432ed07
  Author: Axure <freetiger18@gmail.com>
  Date:   Tue Oct 26 20:43:53 2021 -0400

      Setup skeleton.
```

Ctex2tex.sh (Weicheng Zhao)

```
#!/bin/sh

echo '
\\documentclass{article}
\\usepackage{amsmath,amssymb,amsthm}
\\begin{document}
\\begin{align}'

sed -r '/^\s*$/d' "$@"

echo '\\end{align}
\\end{document}
'
```

Run_ctex.sh (Weicheng Zhao)

```
#!/bin/sh
```

```
# Regression testing script for CTeX
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"

# Path to the ctex compiler.  Usually "../src/ctex.native"
CTEX="./src/ctex.native"

LIBPATH="./src/binom.o ./src/gcd.o"

LATEX_COV="/bin/sh ./scripts/ctex2tex.sh"

PDFLATEX="pdflatex"

# Set time limit for all operations
ulimit -t 30

globallog=run_ctex.log
rm -f $globallog

error=0
globalerror=0

keep=0
pdf=1
build=1
run=1
```

```
Usage() {
   echo "Usage: run_ctex.sh [options] [.ctex files]"
   echo "-b    Only build executable file from .ctex file"
   echo "-r    Only build and run executable file from .ctex file"
   echo "-k    Keep intermediate files"
   echo "-p    Only generate pdf file from .ctex file"
   echo "-h    Print this help"
   exit 1
}

SignalError() {
   if [ $error -eq 0 ] ; then
   echo "FAILED"
   error=1
   fi
   echo "  $1"
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
   echo $* 1>&2
   eval $* || {
   SignalError "$1 failed on $*"
   return 1
   }
}

Check() {
   error=0
   basename=`echo $1 | sed 's/.*\\///
                            s/.ctex//'`

   echo "###### Working on $basename"

   echo 1>&2
   echo "###### Working on $basename" 1>&2

   generatedfiles=""
```

```
   if [ $pdf -eq 1 ] ; then
       generatedfiles="$generatedfiles ${basename}.tex
${basename}.aux ${basename}.out ${basename}.log
${basename}.synctex.gz" &&
       Run "$LATEX_COV" "$1" > "${basename}.tex" &&
       Run "$PDFLATEX" "-output-directory=./" "${basename}.tex"
   fi

   if [ $build -eq 1 ] ; then
       generatedfiles="$generatedfiles ${basename}.ll ${basename}.s"
&&
       Run "$CTEX" < "$1" ">" "${basename}.ll" &&
       Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">"
"${basename}.s" &&
       Run "$CC" "-o" "${basename}.exe" "${basename}.s" "${LIBPATH}"
"-lm"
   fi

   if [ $run -eq 1 ] ; then
       generatedfiles="$generatedfiles ${basename}.exe" &&
       Run "./${basename}.exe"
   fi

   # Report the status and clean up the generated files

   if [ $error -eq 0 ] ; then
   if [ $keep -eq 0 ] ; then
       rm -f $generatedfiles
   fi
   echo "###### FINISHED SUCCESSFULLY"
   echo "###### FINISHED SUCCESSFULLY" 1>&2
   else
   echo "###### FAILED"
   echo "###### FAILED" 1>&2
   globalerror=$error
   fi
}
```

```
while getopts kbrph c; do
   case $c in
   k) # Keep intermediate files
       keep=1
       ;;
   b)
       run=0
       pdf=0
       ;;
   r)
       pdf=0
       ;;
   p)
       build=0
       run=0
       ;;
   h) # Help
       Usage
       ;;
   esac
done

shift `expr $OPTIND - 1`

LLIFail() {
 echo "Could not find the LLVM interpreter \"$LLI\"."
 echo "Check your LLVM installation and/or modify the LLI variable in
testall.sh"
 exit 1
}

which "$LLI" >> $globallog || LLIFail

for file in $@
do
   Check $file 2>> $globallog
done

exit $globalerror
```

Scanner (Hu Zheng)

```
{
 open Printf
 open Parser
 module StringSet = Set.Make(String)
 exception UnexpectedToken of string
 type ident_buffer = { mutable name : string; }
 let buffer : ident_buffer = { name = "" }
 let temp : ident_buffer = { name = "" }
 let fmt : ident_buffer = { name = "" }

 type flags_type = { mutable in_print : bool; }
 let flags = { in_print = false; }

 let standardize name = String.map (fun x -> if x = '\\' then '_'
else x) name
(*  type token =*)
(*    (* whitespaces  *)*)
(*      NL*)
(*    | TAB*)
(*    | SPACE*)
(*    (* identifier *)*)
(*    | ID of string*)
(*    (* literals *)*)
(*    | LIT_FL of float*)
(*    | LIT_INT of int*)
(*    | DIGIT of int*)
(*    (* separators *)*)
(*    | AMP*)
(*    | DBS*)
(*    | PCT*)
(*    | COMMA*)
(*    (* basics *)*)
(*    | VAR of char*)
(*    (* formatting *)*)
```

```
(*    | SUP*)
(*    | SUB*)
(*    (* enclosures *)*)
(*    | LPRN*)
(*    | RPRN*)
(*    | ABS*)
(*    | LBRC*)
(*    | RBRC*)
(*    | LFLOOR*)
(*    | RFLOOR*)
(*    | LCEIL*)
(*    | RCEIL*)
(*    (* envs *)*)
(*    | LCASE*)
(*    | RCASE*)
(*    (* operators *)*)
(*    | ADD*)
(*    | MINUS*)
(*    | MUL*)
(*    | DIV*)
(*    | EQ*)
(*    | LT*)
(*    | GT*)
(*    (* operators from command *)*)
(*    | LEQ*)
(*    | GEQ*)
(*    | NEQ*)
(*    | MID*)
(*    | NMID*)
(*    (*  logic  *)*)
(*    | NEG*)
(*    | WEDGE*)
(*    | VEE*)
(*    (* special functions *)*)
(*    | SUM*)
(*    | PROD*)
(*    | FRAC*)
(*    | BINOM*)
(*    (* binary functions *)*)
```

```
(*    | MAX*)
(*    | MIN*)
(*    | GCD*)
(*    (* basic functions *)*)
(*    | SQRT*)
(*    | EXP*)
(*    (* trigonometric functions *)*)
(*    | SIN*)
(*    | COS*)
(*    | TAN*)
(*    | COT*)
(*    | CSC*)
(*    | SEC*)
(*    | SINH*)
(*    | TANH*)
(*    | COTH*)
(*    | COSH*)
(*    | ARCSIN*)
(*    | ARCCOS*)
(*    | ARCTAN*)
(*    (* arithmetic functions *)*)
(*    | MOD*)
(*    (* log functions *)*)
(*    | LG*)
(*    | LN*)
(*    | LOG*)
(*    (*  bad  *)*)
(*    | BAD_CMD of string*)
(*    (*  misc  *)*)
(*    | CHAR of char*)
(*    | EOF*)
(*    | EOL*)
 let print_token t h =
   let _ = match t with
(*        NL -> if h then Printf.printf "\n" else Printf.printf
"EOL"*)
(*      | TAB -> if h then Printf.printf "\t"  *)
(*      | SPACE -> if h then Printf.printf " "*)
     (* identifier *)
```

```
      | ID(s) -> if h then Printf.printf "ID(%s)" s else Printf.printf
"ID"
      (* literals *)
      | LIT_FL(f) -> if h then Printf.printf "FL(%f)" f else
Printf.printf "LIT_FL"
      | LIT_INT(i) -> if h then Printf.printf "INT(%d)" i else
Printf.printf "LIT_INT"
      | DIGIT(i) -> if h then Printf.printf "DIGIT(%d)" i else
Printf.printf "DIGIT"
      (* separators *)
      | AMP -> if h then Printf.printf "&" else Printf.printf "AMP"
      | DBS -> if h then Printf.printf "\n" else Printf.printf "DBS"
      | PCT -> if h then Printf.printf "%%" else Printf.printf "PCT"
      | COMMA -> if h then Printf.printf "," else Printf.printf
"COMMA"
      (* basics *)
(*      | VAR(v) -> if h then Printf.printf "VAR(%c)" v*)
      (* formatting *)
      | SUP -> if h then Printf.printf "^" else Printf.printf "SUP"
      | SUB -> if h then Printf.printf "_" else Printf.printf "SUB"
      (* enclosures *)
      | LPRN -> if h then Printf.printf "(" else Printf.printf "LPRN"
      | RPRN -> if h then Printf.printf ")" else Printf.printf "RPRN"
      | LABS -> if h then Printf.printf "|" else Printf.printf "LABS"
      | RABS -> if h then Printf.printf "|" else Printf.printf "RABS"
      | LBRC -> if h then Printf.printf "{" else Printf.printf "LBRC"
      | RBRC -> if h then Printf.printf "}" else Printf.printf "RBRC"
      | LFLOOR -> Printf.printf "LFLOOR"
      | RFLOOR -> Printf.printf "RFLOOR"
      | LCEIL -> Printf.printf "LCEIL"
      | RCEIL -> Printf.printf "RCEIL"
      (* envs *)
      | LCASE -> if h then Printf.printf "\\begin{case}" else
Printf.printf "LCASE"
      | RCASE -> if h then Printf.printf "\\end{case}" else
Printf.printf "RCASE"
      | LSPLIT -> if h then Printf.printf "\\begin{split}" else
Printf.printf "LSPLIT"
      | RSPLIT -> if h then Printf.printf "\\end{split}" else
```

```
Printf.printf "RSPLIT"
     (* operators *)
     | ADD -> if h then Printf.printf "+" else Printf.printf "ADD"
     | MINUS -> if h then Printf.printf "-" else Printf.printf
"MINUS"
     | MUL -> if h then Printf.printf "*" else Printf.printf "MUL"
     | DIV -> if h then Printf.printf "/" else Printf.printf "DIV"
     | EQ -> if h then Printf.printf "=" else Printf.printf "EQ"
     | LT -> if h then Printf.printf "<" else Printf.printf "LT"
     | GT -> if h then Printf.printf ">" else Printf.printf "GT"
     | LEQ -> if h then Printf.printf "≤" else Printf.printf "LEQ"
     | GEQ -> if h then Printf.printf "≥" else Printf.printf "GEQ"
     | NEQ -> if h then Printf.printf "≠" else Printf.printf "NEQ"
     | MID -> if h then Printf.printf "∣" else Printf.printf "MID"
     | NMID -> if h then Printf.printf "∤" else Printf.printf "NMID"
     (*  logic  *)
     | NEG -> if h then Printf.printf "¬" else Printf.printf "NEG"
     | WEDGE -> if h then Printf.printf "∧" else Printf.printf
"WEDGE"
     | VEE -> if h then Printf.printf "∨" else Printf.printf "VEE"
     (* special functions *)
     | SUM -> if h then Printf.printf "∑" else Printf.printf "SUM"
     | PROD -> if h then Printf.printf "∏" else Printf.printf "PROD"
     | FRAC -> Printf.printf "FRAC"
     | BINOM -> Printf.printf "BINOM"
     (* binary functions *)
     | MAX ->  Printf.printf "MAX"
     | MIN -> Printf.printf "MIN"
     | GCD -> Printf.printf "GCD"
     (* basic functions *)
     | SQRT -> if h then Printf.printf "√" else Printf.printf "SQRT"
(*      | EXP -> Printf.printf "EXP"*)
     (* trigonometric functions *)
     | SIN -> Printf.printf "SIN"
     | COS -> Printf.printf "COS"
     | TAN -> Printf.printf "TAN"
     | COT -> Printf.printf "COT"
     | CSC -> Printf.printf "CSC"
     | SEC -> Printf.printf "SEC"
```

```
      | SINH -> Printf.printf "SINH"
      | TANH -> Printf.printf "TANH"
      | COTH -> Printf.printf "COTH"
      | COSH -> Printf.printf "COSH"
      | ARCSIN -> Printf.printf "ARCSIN"
      | ARCCOS -> Printf.printf "ARCCOS"
      | ARCTAN -> Printf.printf "ARCTAN"
      (* arithmetic functions *)
      | MOD -> if h then Printf.printf "(mod)" else Printf.printf
"MOD"
      (* log functions *)
      | LG -> Printf.printf "LG"
      | LN -> Printf.printf "LN"
      | LOG -> Printf.printf "LOG"
      (*  bad  *)
(*      | BAD_CMD(s) -> if h then Printf.printf "BAD_CMD(%s)" s*)
      (*  misc  *)
(*      | CHAR(c) -> if h then Printf.printf "CHAR(%c)" c*)
      | EOL -> Printf.printf "EOL"
      | EOF -> Printf.printf "EOF"

      | _ -> Printf.printf "Unimplemented"
    in let () = Printf.printf " "
    in ()

 exception InvalidCommand of string
 let valid_greek_letters = StringSet.of_list [
   "alpha" ;
   "beta" ;
   "Gamma" ;
   "gamma" ;
   "Delta" ;
   "delta" ;
   "epsilon" ;
   "varepsilon" ;
   "zeta" ;
   "eta" ;
   "Theta" ;
   "theta" ;
```

```
   "vartheta" ;
   "iota" ;
   "kappa" ;
   "varkappa" ;
   "Lambda";
   "lambda" ;
   "mu" ;
   "nu" ;
   "Xi" ;
   "xi" ;
   "Pi" ;
   "pi" ;
   "varpi" ;
   "rho" ;
   "varrho" ;
   "Sigma" ;
   "sigma" ;
   "varsigma" ;
   "tau" ;
   "Upsilon" ;
   "upsilon" ;
   "Phi" ;
   "phi" ;
   "varphi" ;
   "chi" ;
   "Psi" ;
   "psi" ;
   "Omega" ;
   "omega" ;
 ]

 let parse_greek raw = let letter = String.sub raw 1 ((String.length
raw) - 1)
   in match StringSet.find_opt letter valid_greek_letters with
     | Some(l) -> Some(letter)
     | None -> None

 let parse_greek_in_ident raw = match parse_greek raw with
   | Some(l) -> buffer.name <- buffer.name ^ "__" ^ l
```

```
   | None -> raise (InvalidCommand raw)

 let parse_command raw = let command = String.sub raw 1
((String.length raw) - 1)
   in match command with
     | "times" -> MUL
     | "cdot" -> MUL
     | "div" -> DIV
     | "leq" -> LEQ
     | "geq" -> GEQ
     | "neq" -> NEQ
     | "mid" -> MID
     | "nmid" -> NMID
     (*  logic  *)
     | "neg" -> NEG
     | "wedge" -> WEDGE
     | "vee" -> VEE
     (* special functions *)
     | "sum" -> SUM
     | "prod" -> PROD
     | "frac" -> FRAC
     | "binom" -> BINOM
     (* binary functions *)
     | "max" -> MAX
     | "min" -> MIN
     | "gcd" -> GCD
     (* basic functions *)
     | "sqrt" -> SQRT
(*      | "exp" -> EXP*)
     (* trigonometric functions *)
     | "sin" -> SIN
     | "cos" -> COS
     | "tan" -> TAN
     | "cot" -> COT
     | "sec" -> SEC
     | "csc" -> CSC
     | "sinh" -> SINH
     | "cosh" -> COSH
     | "tanh" -> TANH
```

```
      | "csc" -> CSC
      | "coth" -> COTH
      | "arcsin" -> ARCSIN
      | "arccos" -> ARCCOS
      | "arctan" -> ARCTAN
      (* arithmetic functions *)
      | "mod" -> MOD
      (* log functions *)
      | "lg" -> LG
      | "ln" -> LN
      | "log" -> LOG
      | _ -> match parse_greek raw with
        | Some(l) -> ID("__" ^ l)
        | None -> raise (UnexpectedToken raw)
}

let cmd = '\\'['a'-'z' 'A'-'Z']+
let int_r = ['0'-'9'] ['0'-'9']+
let int_part = ['0'-'9']+
let float_r = (int_part? '.' int_part) | (int_part '.')
let formatter =
   "mathrm"
 | "mathit"
 | "mathbf"
 | "mathsf"
 | "mathtt"
 | "mathfrak"
 | "mathcal"
 | "mathbb"
 | "mathscr"

rule ctex_read_token = parse
 (* whitespaces  *)
 | [' ' '\t' '\r'] { ctex_read_token lexbuf }
 | ['\n'] { if flags.in_print then let () = flags.in_print <- false
in EOL else ctex_read_token lexbuf }
 (* separators *)
 | "\\\\" { DBS }
 | '&' { AMP }
```

```
 | "%%" { ctex_read_comment lexbuf; if flags.in_print then let () =
flags.in_print <- false in EOL else ctex_read_token lexbuf }
 | '%' { let () = flags.in_print <- true in PCT }
 | ',' { COMMA }
 (* envs *)
 | "\\begin{cases}" { LCASE }
 | "\\end{cases}" { RCASE }
 | "\\begin{split}" { LSPLIT }
 | "\\end{split}" { RSPLIT }
 (* enclosures *)
 | "(" { LPRN }
 | ")" { RPRN }
 | "\\left\\|" { LABS }
 | "\\right\\|" { RABS }
 | "{" { LBRC }
 | "}" { RBRC }
 | "\\lfloor" { LFLOOR }
 | "\\rfloor" { RFLOOR }
 | "\\lceil" { LCEIL }
 | "\\rceil" { RCEIL }
 (* formatting *)
 | '^' { SUP }
 | '_' { SUB }
 (* operatings *)
 | '+' { ADD }
 | '-' { MINUS }
 | '*' { MUL }
 | '/' { DIV }
 | '=' { EQ }
 | '<' { LT }
 | '>' { GT }
 (* literals *)
 | float_r as float_s { LIT_FL(float_of_string float_s) }
 | int_r as int_s { LIT_INT(int_of_string int_s) }
 | ['0'-'9'] as d_s { DIGIT(int_of_string (String.make 1 d_s)) }
 (* complex identifier *)
 | '\\' formatter as f '{' { fmt.name <- f ; ctex_read_ident lexbuf;
ID(standardize(fmt.name ^ "___"^ String.trim temp.name ^ "___")) }
 (* commands, including functions and part of enclosures, IDs,
```

```
operators *)
 | cmd as s { parse_command s }
 (* identifier *)
 | ['a'-'z' 'A'-'Z'] as v { ID(String.make 1 v) }
 (* misc *)
 | _ as c { raise (UnexpectedToken (String.make 1 c)) }
 | eof { EOF }
and ctex_read_comment = parse
 | '\n' | eof { () }
 | _ { ctex_read_comment lexbuf }
and ctex_read_ident = parse
 | [' ' '\t' '\r' '\n'] { ctex_read_ident lexbuf }
 | '}' { temp.name <- buffer.name ; buffer.name <- "" ; () }
 | cmd as s { parse_greek_in_ident s ; ctex_read_ident lexbuf }
 | ['a'-'z' 'A'-'Z'] as c { buffer.name <- buffer.name ^ String.make
1 c ; ctex_read_ident lexbuf }
 | _ as c { raise (UnexpectedToken (String.make 1 c)) }
```

Parser (Rachel Liu, Weicheng Zhao, Hu Zheng)

```
%{ open Ast %}

%token COMMA DBS LPRN RPRN LABS RABS LBRC RBRC LFLOOR RFLOOR LCEIL
RCEIL AMP LCASE RCASE PCT LSPLIT RSPLIT
%token SUP SUB
%token ADD MINUS MUL DIV
%token LEQ GEQ NEQ MID NMID EQ LT GT
%token NEG WEDGE VEE
%token SUM FRAC PROD BINOM
%token MAX MIN GCD
%token SQRT
%token SIN COS TAN COT CSC SEC SINH TANH COTH COSH ARCSIN ARCCOS
ARCTAN
%token MOD
%token LG LN LOG
%token <int> LIT_INT DIGIT
%token <float> LIT_FL
%token <string> ID FID
%token EOF EOL
```

```
%start program
%type <Ast.program> program

%%

program:
 stmt_list EOF { List.rev $1 }

ident:
| ID { SId($1) }
| ID SUB DIGIT { SSId($1, $3) }

stmt:
 expr_calc DBS { Expr($1) }
| ident EQ expr_calc DBS { Assign($1, $3) }
| PCT expr_calc EOL { Print($2) }
| ident LPRN RPRN EQ stmt { FuncDef($1, [], $5)}
| fun_def RPRN EQ stmt { let (id, args) = $1 in FuncDef(id, List.rev
args, $4) }
| expr_spec EQ stmt { let (id, arg) = $1 in FuncDef(id, [arg], $3) }
| fun_def RPRN DBS         { let (id, args) = $1 in
Expr(ExprNoId(Call(FId id, List.map (fun a->Id a) (List.rev args))))
}
| ident EQ fun_def RPRN DBS { let (id, args) = $3 in Assign($1,
ExprNoId(Call(FId id, List.map (fun a->Id a) (List.rev args)))) }
| PCT fun_def RPRN EOL { let (id, args) = $2 in
Print(ExprNoId(Call(FId id, List.map (fun a->Id a) (List.rev args))))
}
| LCASE sutie_list RCASE { Case(List.rev $2) }
| LSPLIT stmt_list RSPLIT { StmtClosure(List.rev $2) }

stmt_list:
   stmt { [$1] }
 | stmt_list stmt { $2 :: $1 }

short_atom:
 DIGIT { ShortLiteral($1) }
```

```
long_atom:
 LIT_INT { IntLiteral($1) }
| LIT_FL { FloatLiteral($1) }
| LABS expr_calc RABS { ParenthLike(Abs, $2) }
| LFLOOR expr_calc RFLOOR { ParenthLike(Floor, $2) }
| LCEIL expr_calc RCEIL { ParenthLike(Ceil, $2) }

paren_no_id:
| LPRN expr_no_id RPRN { ExprNoId($2) }
| LPRN expr_spec RPRN { SpecImplId($2) }

paren_id:
| LPRN ident RPRN { $2 }

atom:
 short_atom { ShortAtom($1) }
| long_atom { LongAtom($1) }

atom_closure:
LBRC expr_calc RBRC { $2 }

expr_pow:
 atom { Atom($1) }
| atom SUP atom_closure { PowerFuncAtom($1, $3) }
| ident SUP atom_closure { PowerFuncId(Id($1), $3) }

expr_log:
 expr_pow { ExprPow($1) }
| LOG SUB atom_closure expr_log { Log($3, $4) }
| LG expr_log { LogLike(Lg, $2) }
| LN expr_log { LogLike(Ln, $2) }
| SQRT expr_log { LogLike(Sqrt, $2) }
| SIN expr_log { LogLike(Sin, $2) }
| COS expr_log { LogLike(Cos, $2) }
| TAN expr_log { LogLike(Tan, $2) }
| ARCSIN expr_log { LogLike(Arcsin, $2) }
| ARCCOS expr_log { LogLike(Arccos, $2) }
| ARCTAN  expr_log { LogLike(Arctan, $2) }
| SINH  expr_log { LogLike(Sinh, $2) }
```

```
| TANH  expr_log { LogLike(Tanh, $2) }
| COSH  expr_log { LogLike(Cosh, $2) }
| CSC  expr_log { LogLike(Csc, $2) }
| SEC  expr_log { LogLike(Sec, $2) }
| COT  expr_log { LogLike(Cot, $2) }
| COTH  expr_log { LogLike(Coth, $2) }
| LOG SUB atom_closure ident { LogC($3, Id($4)) }
| LG ident { LogLikeC(Lg, Id($2)) }
| LN ident { LogLikeC(Ln, Id($2)) }
| SQRT ident { LogLikeC(Sqrt, Id($2)) }
| SIN ident { LogLikeC(Sin, Id($2)) }
| COS ident { LogLikeC(Cos, Id($2)) }
| TAN ident { LogLikeC(Tan, Id($2)) }
| ARCSIN ident { LogLikeC(Arcsin, Id($2)) }
| ARCCOS ident { LogLikeC(Arccos, Id($2)) }
| ARCTAN  ident { LogLikeC(Arctan, Id($2)) }
| SINH  ident { LogLikeC(Sinh, Id($2)) }
| TANH  ident { LogLikeC(Tanh, Id($2)) }
| COSH  ident { LogLikeC(Cosh, Id($2)) }
| CSC  ident { LogLikeC(Csc, Id($2)) }
| SEC  ident { LogLikeC(Sec, Id($2)) }
| COT  ident { LogLikeC(Cot, Id($2)) }
| COTH  ident { LogLikeC(Coth, Id($2)) }
| LOG SUB atom_closure paren_no_id { LogC($3, $4) }
| LG paren_no_id { LogLikeC(Lg, $2 ) }
| LN paren_no_id { LogLikeC(Ln, $2 ) }
| SQRT paren_no_id { LogLikeC(Sqrt, $2 ) }
| SIN paren_no_id { LogLikeC(Sin, $2 ) }
| COS paren_no_id { LogLikeC(Cos, $2 ) }
| TAN paren_no_id { LogLikeC(Tan, $2 ) }
| ARCSIN paren_no_id { LogLikeC(Arcsin, $2 ) }
| ARCCOS paren_no_id { LogLikeC(Arccos, $2 ) }
| ARCTAN  paren_no_id { LogLikeC(Arctan, $2 ) }
| SINH  paren_no_id { LogLikeC(Sinh, $2 ) }
| TANH  paren_no_id { LogLikeC(Tanh, $2 ) }
| COSH  paren_no_id { LogLikeC(Cosh, $2 ) }
| CSC  paren_no_id { LogLikeC(Csc, $2 ) }
| SEC  paren_no_id { LogLikeC(Sec, $2 ) }
| COT  paren_no_id { LogLikeC(Cot, $2 ) }
```

```
| COTH  paren_no_id { LogLikeC(Coth, $2 ) }
| LOG SUB atom_closure paren_id { LogC($3, Id($4)) }
| LG paren_id { LogLikeC(Lg, Id($2) ) }
| LN paren_id { LogLikeC(Ln, Id($2) ) }
| SQRT paren_id { LogLikeC(Sqrt, Id($2) ) }
| SIN paren_id { LogLikeC(Sin, Id($2) ) }
| COS paren_id { LogLikeC(Cos, Id($2) ) }
| TAN paren_id { LogLikeC(Tan, Id($2) ) }
| ARCSIN paren_id { LogLikeC(Arcsin, Id($2) ) }
| ARCCOS paren_id { LogLikeC(Arccos, Id($2) ) }
| ARCTAN  paren_id { LogLikeC(Arctan, Id($2) ) }
| SINH  paren_id { LogLikeC(Sinh, Id($2) ) }
| TANH  paren_id { LogLikeC(Tanh, Id($2) ) }
| COSH  paren_id { LogLikeC(Cosh, Id($2) ) }
| CSC  paren_id { LogLikeC(Csc, Id($2) ) }
| SEC  paren_id { LogLikeC(Sec, Id($2) ) }
| COT  paren_id { LogLikeC(Cot, Id($2) ) }
| COTH  paren_id { LogLikeC(Coth, Id($2) ) }

expr_impl_mult:
 expr_log { ExprLog($1) }
| expr_impl_mult expr_log { ImplMult($1, $2) }
| expr_impl_mult ident { ImplMultMC($1, Id($2)) }
| expr_impl_mult paren_no_id { ImplMultMC($1, $2) }
| expr_impl_mult paren_id { ImplMultMC($1, Id($2)) }
| ident expr_log { ImplMultC(Id($1), $2) }
| ident ident {ImplMultCMC(Id($1), Id($2))}
| ident paren_no_id { SpecImplNoId($1, $2) }
| paren_no_id expr_log { ImplMultC($1, $2) }
| paren_no_id ident { ImplMultCMC($1, Id($2)) }
| paren_no_id paren_no_id { ImplMultCMC($1, $2) }
| paren_no_id paren_id { ImplMultCMC($1, Id($2)) }
| paren_id expr_log { ImplMultC(Id($1), $2) }
| paren_id ident { ImplMultCMC(Id($1), Id($2)) }
| paren_id paren_no_id { ImplMultCMC(Id($1), $2) }
| paren_id paren_id { ImplMultCMC(Id($1), Id($2)) }

expr_spec:
| ident paren_id { ($1, $2) }
```

```
expr_unary:
 expr_impl_mult { ExprImplMult($1) }
| paren_id { Id($1) }
| MINUS expr_unary { Unop(Uminus, $2) }
| ADD expr_unary{ Unop(Uplus, $2) }
| MINUS expr_spec { UnopC(Uminus, SpecImplId($2)) }
| ADD expr_spec { UnopC(Uplus, SpecImplId($2)) }
| MINUS ident { UnopC(Uminus, Id($2)) }
| ADD ident { UnopC(Uplus, Id($2)) }

expr_mult:
 expr_unary { ExprUnary($1) }
| expr_mult MUL expr_unary { MultLike($1, Mult, $3) }
| expr_mult DIV expr_unary { MultLike($1, Div, $3) }
| expr_mult MOD expr_unary { MultLike($1, Mod, $3) }
| expr_mult MUL ident { MultLikeMC($1, Mult, Id($3)) }
| expr_mult DIV ident { MultLikeMC($1, Div, Id($3)) }
| expr_mult MOD ident { MultLikeMC($1, Mod, Id($3)) }
| expr_mult MUL expr_spec { MultLikeMC($1, Mult, SpecImplId($3)) }
| expr_mult DIV expr_spec { MultLikeMC($1, Div, SpecImplId($3)) }
| expr_mult MOD expr_spec { MultLikeMC($1, Mod, SpecImplId($3)) }
| ident MUL expr_unary { MultLikeC(Id($1), Mult, $3) }
| ident DIV expr_unary { MultLikeC(Id($1), Div, $3) }
| ident MOD expr_unary { MultLikeC(Id($1), Mod, $3) }
| ident MUL ident { MultLikeCMC(Id($1), Mult, Id($3)) }
| ident DIV ident { MultLikeCMC(Id($1), Div, Id($3)) }
| ident MOD ident { MultLikeCMC(Id($1), Mod, Id($3)) }
| ident MUL expr_spec { MultLikeCMC(Id($1), Mult, SpecImplId($3)) }
| ident DIV expr_spec { MultLikeCMC(Id($1), Div, SpecImplId($3)) }
| ident MOD expr_spec { MultLikeCMC(Id($1), Mod, SpecImplId($3)) }
| expr_spec MUL expr_unary { MultLikeC(SpecImplId($1), Mult, $3) }
| expr_spec DIV expr_unary { MultLikeC(SpecImplId($1), Div, $3) }
| expr_spec MOD expr_unary { MultLikeC(SpecImplId($1), Mod, $3) }
| expr_spec MUL expr_spec { MultLikeCMC(SpecImplId($1), Mult,
SpecImplId($3)) }
| expr_spec DIV expr_spec { MultLikeCMC(SpecImplId($1), Div,
SpecImplId($3)) }
| expr_spec MOD expr_spec { MultLikeCMC(SpecImplId($1), Mod,
```

```
SpecImplId($3)) }
| expr_spec MUL ident { MultLikeCMC(SpecImplId($1), Mult, Id($3)) }
| expr_spec DIV ident { MultLikeCMC(SpecImplId($1), Div, Id($3)) }
| expr_spec MOD ident { MultLikeCMC(SpecImplId($1), Mod, Id($3)) }

expr_add:
 expr_mult { ExprMult($1) }
| expr_add ADD expr_mult {AddLike($1, Plus, $3) }
| expr_add MINUS expr_mult {AddLike($1, Minus, $3) }
| expr_add ADD ident {AddLikeMC($1, Plus, Id($3)) }
| expr_add MINUS ident {AddLikeMC($1, Minus, Id($3)) }
| expr_add ADD expr_spec {AddLikeMC($1, Plus, SpecImplId($3)) }
| expr_add MINUS expr_spec {AddLikeMC($1, Minus, SpecImplId($3)) }
| ident ADD expr_mult {AddLikeC(Id($1), Plus, $3) }
| ident MINUS expr_mult {AddLikeC(Id($1), Minus, $3) }
| ident ADD ident {AddLikeCMC(Id($1), Plus, Id($3)) }
| ident MINUS ident {AddLikeCMC(Id($1), Minus, Id($3)) }
| ident ADD expr_spec {AddLikeCMC(Id($1), Plus, SpecImplId($3)) }
| ident MINUS expr_spec {AddLikeCMC(Id($1), Minus, SpecImplId($3)) }
| expr_spec ADD expr_mult {AddLikeC(SpecImplId($1), Plus, $3) }
| expr_spec MINUS expr_mult {AddLikeC(SpecImplId($1), Minus, $3) }
| expr_spec ADD expr_spec {AddLikeCMC(SpecImplId($1), Plus,
SpecImplId($3)) }
| expr_spec MINUS expr_spec {AddLikeCMC(SpecImplId($1), Minus,
SpecImplId($3)) }
| expr_spec ADD ident {AddLikeCMC(SpecImplId($1), Plus, Id($3)) }
| expr_spec MINUS ident {AddLikeCMC(SpecImplId($1), Minus, Id($3)) }

expr_no_id:
 expr_add { ExprAdd($1) }
| ident LPRN RPRN {Call(FId($1), [])}
| call_expr RPRN { let (id, args) = $1 in Call(id, List.rev args) }
| GCD LPRN arg_list RPRN { Call(Gcd, $3) }
| MAX LPRN arg_list RPRN { Call(Max, $3) }
| MIN LPRN arg_list RPRN { Call(Min, $3) }
| FRAC atom_closure atom_closure { FracLike(Frac, $2, $3) }
| BINOM atom_closure atom_closure { FracLike(Binom, $2, $3) }

fun_def:
```

```
| ident LPRN ident COMMA ident       { ($1, [$5; $3]) }
| fun_def COMMA ident     { let (id, args) = $1 in (id, $3 :: args) }

call_expr:
| ident LPRN ident COMMA expr_no_id { (FId($1), [ExprNoId($5);
Id($3)])}
| ident LPRN expr_no_id COMMA expr_no_id { (FId($1), [ExprNoId($5);
ExprNoId($3)])}
| ident LPRN expr_no_id COMMA ident { (FId($1), [Id($5);
ExprNoId($3)])}
| ident LPRN expr_spec COMMA expr_no_id { (FId($1), [ExprNoId($5);
SpecImplId($3)])}
| ident LPRN expr_spec COMMA expr_spec { (FId($1), [SpecImplId($5);
SpecImplId($3)])}
| ident LPRN expr_no_id COMMA expr_spec { (FId($1), [SpecImplId($5);
ExprNoId($3)])}
| ident LPRN expr_spec COMMA ident { (FId($1), [Id($5);
SpecImplId($3)])}
| ident LPRN ident COMMA expr_spec { (FId($1), [SpecImplId($5);
Id($3)])}
| fun_def COMMA expr_no_id { let (id, args) = $1 in (FId id,
ExprNoId($3) :: List.map (fun a->Id a) args) }
| call_expr COMMA expr_calc { let (id, args) = $1 in (id, $3 :: args)
}

expr_calc:
 ident         { Id($1) }
| expr_spec { SpecImplId($1) }
| expr_no_id { ExprNoId($1) }

arg_list:
 expr_calc { [$1] }
| arg_list COMMA expr_calc { $3 :: $1 }

expr_comp:
 expr_calc LT expr_calc { Comp($1, Lt, $3) }
| expr_calc GT expr_calc { Comp($1, Gt, $3) }
| expr_calc LEQ expr_calc { Comp($1, Leq, $3) }
| expr_calc GEQ expr_calc { Comp($1, Geq, $3) }
```

```
| expr_calc EQ expr_calc { Comp($1, Eq, $3) }
| expr_calc NEQ expr_calc { Comp($1, Neq, $3) }
| expr_calc NMID expr_calc { Comp($1, Nmid, $3) }
| expr_calc MID expr_calc { Comp($1, Mid, $3) }

expr_logic_atom:
 expr_comp { ExprComp($1) }
| LPRN expr_logic RPRN { ParenLogic($2) }
| NEG expr_logic_atom { Not($2)}

expr_logic:
 expr_logic_atom { LogicAtom($1) }
| expr_logic WEDGE expr_logic_atom { Logic($1, And, $3) }
| expr_logic VEE expr_logic_atom { Logic($1, Or, $3) }

sutie:
 stmt AMP expr_logic DBS { CaseSutie($1, $3) }

sutie_list:
   sutie { [$1] }
 | sutie_list sutie { $2 :: $1 }
```

AST (Rachel Liu, Weicheng Zhao)

```
type paren_op = Abs | Floor | Ceil

type log_like_op =
 | Lg
 | Ln
 | Sqrt
 | Sin
 | Cos
 | Tan
 | Arcsin
 | Arccos
 | Arctan
 | Sinh
 | Cosh
```

```
  | Tanh
  | Cot
  | Sec
  | Csc
  | Coth

type unop = Uplus | Uminus

type mult_like_op = Mult | Div | Mod

type add_like_op = Plus | Minus

type func = FId of string | Gcd | Min | Max

type frac_like_op = Frac | Binom

type large_op = Sum | Prod

type comp_op = Lt | Gt | Leq | Geq | Eq | Neq | Nmid | Mid

type logic_binop = And | Or

type short_atom = ShortLiteral of int

and long_atom =
  | IntLiteral of int
  | FloatLiteral of float
  | ParenthLike of paren_op * expr_calc

and atom = ShortAtom of short_atom | LongAtom of long_atom

and expr_pow =
  | Atom of atom
  | PowerFuncAtom of atom * expr_calc
  | PowerFuncId of expr_calc * expr_calc

and expr_log =
  | ExprPow of expr_pow
  | LogLike of log_like_op * expr_log
```

```
  | LogLikeC of log_like_op * expr_calc
  | Log of expr_calc * expr_log
  | LogC of expr_calc * expr_calc

and expr_impl_mult =
  | ExprLog of expr_log
  | ImplMult of expr_impl_mult * expr_log
  | ImplMultC of expr_calc * expr_log
  | ImplMultMC of expr_impl_mult * expr_calc
  | ImplMultCMC of expr_calc * expr_calc
  | SpecImplNoId of string * expr_calc

and expr_spec = string * string

and expr_unary =
  | ExprImplMult of expr_impl_mult
  | Id of string
  | ExprNoId of expr_no_id
  | Unop of unop * expr_unary
  | UnopC of unop * expr_calc

and expr_mult =
  | ExprUnary of expr_unary
  | MultLike of expr_mult * mult_like_op * expr_unary
  | MultLikeMC of expr_mult * mult_like_op * expr_calc
  | MultLikeC of expr_calc * mult_like_op * expr_unary
  | MultLikeCMC of expr_calc * mult_like_op * expr_calc

and expr_add =
  | ExprMult of expr_mult
  | AddLike of expr_add * add_like_op * expr_mult
  | AddLikeMC of expr_add * add_like_op * expr_calc
  | AddLikeC of expr_calc * add_like_op * expr_mult
  | AddLikeCMC of expr_calc * add_like_op * expr_calc

and expr_no_id =
  | ExprAdd of expr_add
  | Call of func * expr_calc list
  | FracLike of frac_like_op * expr_calc * expr_calc
```

```
  | LargeOp of large_op * string * expr_calc * expr_calc * expr_calc

and expr_calc =
  | Id of string
  | SpecImplId of expr_spec
  | ExprNoId of expr_no_id

and expr_comp = Comp of expr_calc * comp_op * expr_calc

and expr_logic_atom =
  | ExprComp of expr_comp
  | ParenLogic of expr_logic
  | Not of expr_logic_atom

and expr_logic =
  | LogicAtom of expr_logic_atom
  | Logic of expr_logic * logic_binop * expr_logic_atom

type sutie = CaseSutie of stmt * expr_logic

and stmt =
  | Expr of expr_calc
  | Assign of string * expr_calc
  | Print of expr_calc
  | FuncDef of string * string list * stmt
  | Case of sutie list
  | StmtClosure of stmt list

type program = stmt list

(* Pretty-printing functions *)

let string_of_log_like_op = function
  | Lg -> "\\lg "
  | Ln -> "\\ln "
  | Sqrt -> "\\sqrt "
  | Sin -> "\\sin "
  | Cos -> "\\cos "
  | Tan -> "\\tan "
```

```
  | Arcsin -> "\\arcsin "
  | Arccos -> "\\arccos "
  | Arctan -> "\\arctan "
  | Sinh -> "\\sinh "
  | Cosh -> "\\cosh "
  | Tanh -> "\\tanh "
  | Cot -> "\\cot "
  | Sec -> "\\sec "
  | Csc -> "\\csc "
  | Coth -> "\\coth "

let string_of_uop = function Uplus -> "+" | Uminus -> "-"

let string_of_multi_like_op = function
  | Mult -> "*"
  | Div -> "/"
  | Mod -> "\\mod "

let string_of_add_like_op = function Plus -> "+" | Minus -> "-"

let string_of_frac_like_op = function
  | Frac -> "\\frac "
  | Binom -> "\\binom "

let string_of_func = function
  | FId s -> s
  | Gcd -> "\\gcd "
  | Min -> "\\min "
  | Max -> "\\max "

let string_of_comp_op = function
  | Lt -> "< "
  | Gt -> "> "
  | Leq -> "\\leq "
  | Geq -> "\\geq "
  | Eq -> "= "
  | Neq -> "\\neq "
  | Nmid -> "\\nmid "
  | Mid -> "\\mid "
```

```
let string_of_logic_op = function And -> "\\wedge " | Or -> "\\vee "

let rec string_of_short_atom = function ShortLiteral l ->
string_of_int l

and string_of_long_atom = function
 | IntLiteral l -> string_of_int l
 | FloatLiteral l -> string_of_float l
 | ParenthLike (op, e) -> (
     match op with
     | Abs -> "|" ^ string_of_expr_calc e ^ "|"
     | Floor -> "\\lfloor " ^ string_of_expr_calc e ^ " \\rfloor"
     | Ceil -> "\\lceil " ^ string_of_expr_calc e ^ " \\rceil")

and string_of_atom = function
 | ShortAtom a -> string_of_short_atom a
 | LongAtom a -> string_of_long_atom a

and string_of_expr_pow = function
 | Atom a -> string_of_atom a
 | PowerFuncAtom (a, e) -> string_of_atom a ^ "^" ^
string_of_expr_calc e
 | PowerFuncId (a, e) -> string_of_expr_calc a ^ "^" ^
string_of_expr_calc e

and string_of_expr_log = function
 | ExprPow e -> string_of_expr_pow e
 | LogLike (op, e) ->
     string_of_log_like_op op ^ "(" ^ string_of_expr_log e ^ ")"
 | LogLikeC (op, e) ->
     string_of_log_like_op op ^ "(" ^ string_of_expr_calc e ^ ")"
 | Log (a, e) ->
     "\\log_" ^ string_of_expr_calc a ^ " " ^ string_of_expr_log e
 | LogC (a, e) ->
     "\\log_" ^ string_of_expr_calc a ^ " " ^ string_of_expr_calc e

and string_of_expr_impl_mult = function
 | ExprLog e -> string_of_expr_log e
```

```
 | ImplMult (e1, e2) ->
     string_of_expr_impl_mult e1 ^ "•" ^ string_of_expr_log e2
 | ImplMultC (e1, e2) ->
     string_of_expr_calc e1 ^ "•" ^ string_of_expr_log e2
 | ImplMultMC (e1, e2) ->
     string_of_expr_impl_mult e1 ^ "•" ^ string_of_expr_calc e2
 | ImplMultCMC (e1, e2) ->
     string_of_expr_calc e1 ^ "•" ^ string_of_expr_calc e2
 | SpecImplNoId (s, e) -> s ^ "(" ^ string_of_expr_calc e ^ ")"

and string_of_expr_spec (s1, s2) = s1 ^ "(" ^ s2 ^ ")"

and string_of_expr_unary = function
 | ExprImplMult e -> string_of_expr_impl_mult e
 | Id s -> s
 | ExprNoId e -> string_of_expr_no_id e
 | Unop (op, e) -> string_of_uop op ^ string_of_expr_unary e
 | UnopC (op, e) -> string_of_uop op ^ string_of_expr_calc e

and string_of_expr_mult = function
 | ExprUnary e -> string_of_expr_unary e
 | MultLike (e1, op, e2) ->
     string_of_expr_mult e1
     ^ string_of_multi_like_op op
     ^ string_of_expr_unary e2
 | MultLikeMC (e1, op, e2) ->
     string_of_expr_mult e1
     ^ string_of_multi_like_op op
     ^ string_of_expr_calc e2
 | MultLikeC (e1, op, e2) ->
     string_of_expr_calc e1
     ^ string_of_multi_like_op op
     ^ string_of_expr_unary e2
 | MultLikeCMC (e1, op, e2) ->
     string_of_expr_calc e1
     ^ string_of_multi_like_op op
     ^ string_of_expr_calc e2

and string_of_expr_add = function
```

```
 | ExprMult e -> string_of_expr_mult e
 | AddLike (e1, op, e2) ->
     string_of_expr_add e1 ^ string_of_add_like_op op
     ^ string_of_expr_mult e2
 | AddLikeMC (e1, op, e2) ->
     string_of_expr_add e1 ^ string_of_add_like_op op
     ^ string_of_expr_calc e2
 | AddLikeC (e1, op, e2) ->
     string_of_expr_calc e1 ^ string_of_add_like_op op
     ^ string_of_expr_mult e2
 | AddLikeCMC (e1, op, e2) ->
     string_of_expr_calc e1 ^ string_of_add_like_op op
     ^ string_of_expr_calc e2

and string_of_expr_no_id = function
 | ExprAdd e -> string_of_expr_add e
 | Call (f, el) ->
     string_of_func f ^ "("
     ^ String.concat ", " (List.map string_of_expr_calc el)
     ^ ")"
 | FracLike (op, e1, e2) ->
     string_of_frac_like_op op ^ "{" ^ string_of_expr_calc e1 ^ "} {"
     ^ string_of_expr_calc e2 ^ "}"
 | LargeOp (op, s, es, ee, e) -> "Unimpl"

and string_of_expr_calc = function
 | Id s -> s
 | SpecImplId s -> string_of_expr_spec s
 | ExprNoId e -> string_of_expr_no_id e

and string_of_expr_comp = function
 | Comp (e1, op, e2) ->
     string_of_expr_calc e1 ^ string_of_comp_op op ^
string_of_expr_calc e2

and string_of_expr_logic_atom = function
 | ExprComp e -> string_of_expr_comp e
 | ParenLogic e -> "(" ^ string_of_expr_logic e ^ ")"
 | Not e -> "\\neg " ^ string_of_expr_logic_atom e
```

```
and string_of_expr_logic = function
 | LogicAtom e -> string_of_expr_logic_atom e
 | Logic (e1, op, e2) ->
     string_of_expr_logic e1 ^ string_of_logic_op op
     ^ string_of_expr_logic_atom e2

let rec string_of_suite = function
 | CaseSutie (s, e) ->
     string_of_stmt s ^ "&" ^ string_of_expr_logic e ^ "\\\\"

and string_of_stmt = function
 | Expr e -> string_of_expr_calc e ^ "\\\\"
 | Assign (s, e) -> s ^ "=" ^ string_of_expr_calc e ^ "\\\\"
 | Print e -> "print " ^ string_of_expr_calc e ^ "\n"
 | FuncDef (s, fl, ss) ->
     s ^ "(" ^ String.concat ", " fl ^ ") = " ^ string_of_stmt ss
 | Case sl ->
     "\\begin{cases}\n"
     ^ String.concat "\n" (List.map string_of_suite sl)
     ^ "\n\\end{cases}\n"
 | StmtClosure sl ->
     "\\begin{split}\n"
     ^ String.concat "\n" (List.map string_of_stmt sl)
     ^ "\n\\end{split}\n"

let string_of_program sl = String.concat "\n" (List.map
string_of_stmt sl)
Semantics (Weicheng Zhao, Rachel Liu)

open Ast
open Sast
module StringMap = Map.Make (String)

let check_program prog =
 let type_of_identifier lb s =
   try StringMap.find s lb
   with Not_found -> raise (Failure ("undeclared identifier " ^ s))
 in
```

```
 let find_func lf s =
   try StringMap.find s l
```

Semantics (Weicheng Zhao, Rachel Liu)

```
open Ast
open Sast
module StringMap = Map.Make (String)

let check_program prog =
 let type_of_identifier lb s =
   try StringMap.find s lb
   with Not_found -> raise (Failure ("undeclared identifier " ^ s))
 in

 let find_func lf s =
   try StringMap.find s lf
   with Not_found -> raise (Failure ("unrecognized function " ^ s))
 in

 let atom_of_expr_calc e : s_atom = (fst e, SParen e)
 and power_of_atom e = (fst e, SAtom e)
 and log_of_power e = (fst e, SExprPow e)
 and imp_of_log e = (fst e, SExprLog e)
 and unary_of_imp e = (fst e, SExprImplMult e)
 and mult_of_unary e = (fst e, SExprUnary e)
 and add_of_mult e = (fst e, SExprMult e)
 and calc_of_add e = (fst e, SExprAdd e) in
 let impl t1 t2 e1' e2' : s_expr_impl_mult =
   match (t1, t2) with
   | Int, Int -> (Int, SImplMult (e1', e2'))
   | Float, Float | Int, Float | Float, Int -> (Float, SImplMult
(e1', e2'))
   | _ -> raise (Failure "Invalid bool type appears")
 in
 let mult op t1 t2 e1' e2' =
   match op with
```

```
   | Mod -> (Int, SMultLike (e1', Mod, e2'))
   | (Mult | Div) when t1 = t2 && t1 = Int -> (Int, SMultLike (e1',
op, e2'))
   | (Mult | Div) when t1 != Bool && t2 != Bool ->
       (Float, SMultLike (e1', op, e2'))
   | _ -> raise (Failure "Invalid bool type appears")
 in
 let add op t1 t2 e1' e2' =
   match op with
   | (Plus | Minus) when t1 = t2 && t1 = Int ->
       (Int, SAddLike (e1', op, e2'))
   | (Plus | Minus) when t1 != Bool && t2 != Bool ->
       (Float, SAddLike (e1', op, e2'))
   | _ -> raise (Failure "Invalid bool type appears")
 in
 let rec check_short_atom lb lf e =
   match e with ShortLiteral l -> (Int, SIntLiteral l)
 and check_long_atom lb lf e =
   match e with
   | IntLiteral l -> (Int, SIntLiteral l)
   | FloatLiteral fl -> (Float, SFloatLiteral fl)
   | ParenthLike (op, e) -> (
       match op with
       | Abs ->
           let ((t, se) as e') = check_expr_calc lb lf e in
           (t, SParenthLike (Abs, e'))
       | Floor | Ceil ->
           let ((t, se) as e') = check_expr_calc lb lf e in
           (Int, SParenthLike (op, e')))
 and check_atom lb lf e =
   match e with
   | ShortAtom a -> check_short_atom lb lf a
   | LongAtom a -> check_long_atom lb lf a
 and check_expr_pow lb lf e =
   match e with
   | Atom a ->
       let ((t, sa) as a') = check_atom lb lf a in
       (t, SAtom a')
   | PowerFuncAtom (a1, a2) ->
```

```
      let ((t1, sa1) as a1') = check_atom lb lf a1
      and ((t2, sa2) as a2') = check_expr_calc lb lf a2 in
      (Float, SPowerFunc (a1', a2'))
  | PowerFuncId (a1, a2) ->
      let ((t1, sa1) as a1') = check_expr_calc lb lf a1
      and ((t2, sa2) as a2') = check_expr_calc lb lf a2 in
      (Float, SPowerFunc ((t1, SParen a1'), a2'))
and check_expr_log lb lf e =
  match e with
  | ExprPow e ->
      let ((t, se) as e') = check_expr_pow lb lf e in
      (t, SExprPow e')
  | LogLike (op, e) -> (Float, SLogLike (op, check_expr_log lb lf
e))
  | LogLikeC (op, e) ->
      ( Float,
        SLogLike
          ( op,
            log_of_power
              (power_of_atom (atom_of_expr_calc (check_expr_calc lb
lf e)))
          ) )
  | Log (ec, e) ->
      (Float, SLog (check_expr_calc lb lf ec, check_expr_log lb lf
e))
  | LogC (ec, e) ->
      ( Float,
        SLog
          ( check_expr_calc lb lf ec,
            log_of_power
              (power_of_atom (atom_of_expr_calc (check_expr_calc lb
lf e)))
          ) )
and check_expr_impl_mult lb lf e =
  match e with
  | ExprLog e ->
      let ((t, se) as e') = check_expr_log lb lf e in
      (t, SExprLog e')
  | ImplMult (e1, e2) ->
```

```
      let ((t1, se1) as e1') = check_expr_impl_mult lb lf e1
      and ((t2, se2) as e2') = check_expr_log lb lf e2 in
      impl t1 t2 e1' e2'
  | ImplMultC (e1, e2) ->
      let ((t1, se1) as e1') = check_expr_calc lb lf e1
      and ((t2, se2) as e2') = check_expr_log lb lf e2 in
      let e1' =
        imp_of_log (log_of_power (power_of_atom (atom_of_expr_calc
e1')))
      in
      impl t1 t2 e1' e2'
  | ImplMultMC (e1, e2) ->
      let ((t1, se1) as e1') = check_expr_impl_mult lb lf e1
      and ((t2, se2) as e2') = check_expr_calc lb lf e2 in
      let e2' = log_of_power (power_of_atom (atom_of_expr_calc e2'))
in
      impl t1 t2 e1' e2'
  | ImplMultCMC (e1, e2) ->
      let ((t1, se1) as e1') = check_expr_calc lb lf e1
      and ((t2, se2) as e2') = check_expr_calc lb lf e2 in
      let e1' =
        imp_of_log (log_of_power (power_of_atom (atom_of_expr_calc
e1')))
      and e2' = log_of_power (power_of_atom (atom_of_expr_calc e2'))
in
      impl t1 t2 e1' e2'
  | SpecImplNoId (s, e) -> (
      match StringMap.find_opt (string_of_ident s) lb with
      | Some t ->
          check_expr_impl_mult lb lf (ImplMultCMC (Id s, e))
      | None -> (
          match StringMap.find_opt (string_of_ident s) lf with
          | Some fd ->
              let param_length = List.length fd.sformals in
              if 1 != param_length then
                raise
                  (Failure
                     ("expecting "
                     ^ string_of_int param_length
```

```
                    ^ " arguments in "))
              else (fd.styp, SCall (FId s, [ check_expr_calc lb lf e
]))
          | None -> raise (Failure ("Unknown bind:" ^
(string_of_ident s)))))
 and check_expr_unary lb lf e =
   match e with
   | ExprImplMult e ->
       let ((t, se) as e') = check_expr_impl_mult lb lf e in
       (t, SExprImplMult e')
   | Id s ->
       let t = type_of_identifier lb (string_of_ident s) in
       (t, SId (string_of_ident s))
   | ExprNoId e ->
       let ((t, se) as e') = check_expr_no_id lb lf e in
       (t, SParen e')
   | Unop (op, e) ->
       let ((t, se) as e') = check_expr_unary lb lf e in
       (t, SUnop (op, e'))
   | UnopC (op, e) ->
       let ((t, se) as e') = check_expr_calc lb lf e in
       ( t,
         SUnop
           ( op,
             unary_of_imp
               (imp_of_log
                  (log_of_power (power_of_atom (atom_of_expr_calc
e')))) )
       )
 and check_expr_mult lb lf e =
   match e with
   | ExprUnary e ->
       let ((t, se) as e') = check_expr_unary lb lf e in
       (t, SExprUnary e')
   | MultLike (e1, op, e2) ->
       let ((t1, se1) as e1') = check_expr_mult lb lf e1
       and ((t2, se2) as e2') = check_expr_unary lb lf e2 in
       mult op t1 t2 e1' e2'
   | MultLikeMC (e1, op, e2) ->
```

```
       let ((t1, se1) as e1') = check_expr_mult lb lf e1
       and ((t2, se2) as e2') = check_expr_calc lb lf e2 in
       let e2' =
         unary_of_imp
           (imp_of_log
              (log_of_power (power_of_atom (atom_of_expr_calc e2'))))
       in
       mult op t1 t2 e1' e2'
   | MultLikeC (e1, op, e2) ->
       let ((t1, se1) as e1') = check_expr_calc lb lf e1
       and ((t2, se2) as e2') = check_expr_unary lb lf e2 in
       let e1' =
         mult_of_unary
           (unary_of_imp
              (imp_of_log
                 (log_of_power (power_of_atom (atom_of_expr_calc
e1')))))
       in
       mult op t1 t2 e1' e2'
   | MultLikeCMC (e1, op, e2) ->
       let ((t1, se1) as e1') = check_expr_calc lb lf e1
       and ((t2, se2) as e2') = check_expr_calc lb lf e2 in
       let e1' =
         mult_of_unary
           (unary_of_imp
              (imp_of_log
                 (log_of_power (power_of_atom (atom_of_expr_calc
e1')))))
       and e2' =
         unary_of_imp
           (imp_of_log
              (log_of_power (power_of_atom (atom_of_expr_calc e2'))))
       in
       mult op t1 t2 e1' e2'
 and check_expr_add lb lf e =
   match e with
   | ExprMult e ->
       let ((t, se) as e') = check_expr_mult lb lf e in
       (t, SExprMult e')
```

```
   | AddLike (e1, op, e2) ->
       let ((t1, se1) as e1') = check_expr_add lb lf e1
       and ((t2, se2) as e2') = check_expr_mult lb lf e2 in
       add op t1 t2 e1' e2'
   | AddLikeMC (e1, op, e2) ->
       let ((t1, se1) as e1') = check_expr_add lb lf e1
       and ((t2, se2) as e2') = check_expr_calc lb lf e2 in
       let e2' =
         mult_of_unary
           (unary_of_imp
              (imp_of_log
                 (log_of_power (power_of_atom (atom_of_expr_calc
e2')))))
       in
       add op t1 t2 e1' e2'
   | AddLikeC (e1, op, e2) ->
       let ((t1, se1) as e1') = check_expr_calc lb lf e1
       and ((t2, se2) as e2') = check_expr_mult lb lf e2 in
       let e1' =
         add_of_mult
           (mult_of_unary
              (unary_of_imp
                 (imp_of_log
                    (log_of_power (power_of_atom (atom_of_expr_calc
e1'))))))
       in
       add op t1 t2 e1' e2'
   | AddLikeCMC (e1, op, e2) ->
       let ((t1, se1) as e1') = check_expr_calc lb lf e1
       and ((t2, se2) as e2') = check_expr_calc lb lf e2 in
       let e1' =
         add_of_mult
           (mult_of_unary
              (unary_of_imp
                 (imp_of_log
                    (log_of_power (power_of_atom (atom_of_expr_calc
e1'))))))
       and e2' =
         mult_of_unary
```

```
           (unary_of_imp
              (imp_of_log
                 (log_of_power (power_of_atom (atom_of_expr_calc
e2')))))
       in
       add op t1 t2 e1' e2'
 and check_expr_no_id lb lf e =
   match e with
   | ExprAdd e ->
       let ((t, se) as e') = check_expr_add lb lf e in
       (t, SExprAdd e')
   | Call (fname, args) -> (
       match fname with
       | FId (s : ident) ->
           let fd = find_func lf (string_of_ident s) in
           let param_length = List.length fd.sformals in
           if List.length args != param_length then
             raise
               (Failure
                  ("expecting "
                  ^ string_of_int param_length
                  ^ " arguments in "))
           else
             (fd.styp, SCall (FId s, List.map (check_expr_calc lb lf)
args))
       | Gcd ->
           if List.length args != 2 then
             raise (Failure "expecting 2 arguments for gcd")
           else (Int, SCall (Gcd, List.map (check_expr_calc lb lf)
args))
       | Min | Max ->
           if List.length args < 2 then
             raise (Failure "expecting >2 arguments for min/max")
           else if
             List.exists
               (fun e -> fst (check_expr_calc lb lf e) = Float)
               args
           then (Float, SCall (fname, List.map (check_expr_calc lb
lf) args))
```

```
          else (Int, SCall (fname, List.map (check_expr_calc lb lf)
args)))
   | FracLike (op, ac1, ac2) -> (
       let t1, sa1 = check_expr_calc lb lf ac1
       and t2, sa2 = check_expr_calc lb lf ac2 in
       match op with
       | Frac -> (Float, SFracLike (Frac, (t1, sa1), (t2, sa2)))
       | Binom -> (Int, SFracLike (Binom, (t1, sa1), (t2, sa2))))
(*    | LargeOp (op, s, se, ee, ce) ->*)
(*          (Int, SIntLiteral 123)*)
(*        (Int, SCallLargeOp ":random")*)
(*        ( Float,*)
(*          SLargeOp*)
(*            ( op,*)
(*              (Int, s),*)
(*              check_expr_calc lb lf se,*)
(*              check_expr_calc lb lf ee,*)
(*              check_expr_calc lb lf ce ) )*)
 and check_expr_calc lb lf e =
   match e with
   | Id s ->
       let t = type_of_identifier lb (string_of_ident s) in
       let e = (t, SId (string_of_ident s)) in
       calc_of_add (add_of_mult (mult_of_unary e))
   | SpecImplId e -> (
       let s1, s2 = e in
       match StringMap.find_opt (string_of_ident s1) lb with
       | Some t ->
           calc_of_add
             (add_of_mult
                (mult_of_unary
                   (unary_of_imp
                      (check_expr_impl_mult lb lf
                         (ImplMultCMC (Id s1, Id s2))))))
       | None -> (
           match StringMap.find_opt (string_of_ident s1) lf with
           | Some fd ->
               let param_length = List.length fd.sformals in
               if 1 != param_length then
```

```
                  raise
                    (Failure
                       ("expecting "
                       ^ string_of_int param_length
                       ^ " arguments in "))
                else
                  let t = type_of_identifier lb (string_of_ident s2)
in
                  let e = (t, SId (string_of_ident s2)) in
                  ( fd.styp,
                    SCall
                      ( FId s1,
                        [ calc_of_add (add_of_mult (mult_of_unary e))
] ) )
            | None -> raise (Failure ("Unknown bind:" ^
(string_of_ident s1)))))
   | ExprNoId e -> check_expr_no_id lb lf e
 and check_expr_comp lb lf e =
   match e with
   | Comp (e1, op, e2) ->
       (Bool, SComp (check_expr_calc lb lf e1, op, check_expr_calc lb
lf e2))
 and check_expr_logic_atom lb lf e =
   match e with
   | ExprComp e ->
       let ((t, se) as e') = check_expr_comp lb lf e in
       (t, SExprComp e')
   | ParenLogic e ->
       let ((t, se) as e') = check_expr_logic lb lf e in
       (t, SParenLogic e')
   | Not e -> (Bool, SNot (check_expr_logic_atom lb lf e))
 and check_expr_logic lb lf e =
   match e with
   | LogicAtom e ->
       let ((t, se) as e') = check_expr_logic_atom lb lf e in
       (t, SLogicAtom e')
   | Logic (e1, op, e2) ->
       ( Bool,
         SLogic
```

```
          (check_expr_logic lb lf e1, op, check_expr_logic_atom lb
lf e2)
       )
 in

 let add_bind s t lb lf =
   if StringMap.find_opt s lf != None then
     raise (Failure (s ^ "has been bound to a function previously"))
   else
     match StringMap.find_opt s lb with
     | Some nt ->
         if nt != t then
           raise
             (Failure (s ^ "has been bound to a different type
previously"))
         else lb
     | None -> StringMap.add s t lb
 in
 let rec check_sutie (lb : typ StringMap.t) (lf : sfunc_decl
StringMap.t)
     (ss : sutie) =
   match ss with
   | CaseSutie (s, e) ->
       let ((et, se) as e') = check_expr_logic lb lf e in
       if et != Bool then
         raise (Failure "the case is not a bool expression")
       else
         let (((lb, lf), (t, ss)) as s') = check_stmt lb lf s in
         (lb, (t, SCaseSutie (Some (t, ss), e')))
 and check_stmt (lb : typ StringMap.t) (lf : sfunc_decl StringMap.t)
     (ss : stmt) =
   match ss with
   | Expr e ->
       let ((t, se) as e') = check_expr_calc lb lf e in
       ((lb, lf), (t, SExpr e'))
   | Assign (s, e) ->
       let ((t, se) as e') = check_expr_calc lb lf e in
       ((add_bind (string_of_ident s) t lb lf, lf), (t, SAssign
((string_of_ident s), e')))
```

```
   | Print e ->
       let ((t, se) as e') = check_expr_calc lb lf e in
       ((lb, lf), (t, SPrint e'))
   | FuncDef (fname, formals, body) ->
       if StringMap.find_opt (string_of_ident fname) lb != None then
         raise (Failure ((string_of_ident fname) ^ "has been bound to
a variable previously"))
       else if StringMap.find_opt (string_of_ident fname) lf != None
then
         raise (Failure ((string_of_ident fname) ^ "has been bound to
a function previously"))
       else
         let sd =
           {
             styp = Float;
             sfname = string_of_ident fname;
             sformals = List.map (fun a -> (Float, string_of_ident
a)) formals;
             sframe = StringMap.fold (fun n t tl -> (t, n) :: tl) lb
[];
             slocals = [];
             sbody = None;
           }
         in
         let llb =
           List.fold_left (fun bs a -> add_bind (string_of_ident a)
Float bs lf) lb formals
         and llf = StringMap.add (string_of_ident fname) sd lf in
         let (slb, slf), (t, ssb) = check_stmt llb llf body in
         let not_exist k m = not (StringMap.mem k m) in
         let sl = StringMap.filter (fun k v -> not_exist k llb) slb
in
         let sd =
           {
             styp = t;
             sfname = string_of_ident fname;
             sformals = List.map (fun a -> (Float, string_of_ident
a)) formals;
             sframe = StringMap.fold (fun n t tl -> (t, n) :: tl) lb
```

```
[];
              slocals = StringMap.fold (fun n t tl -> (t, n) :: tl) sl
[];
              sbody = Some (t, ssb);
            }
          in
          ((lb, StringMap.add (string_of_ident fname) sd slf), (t,
SFuncDef sd))
    | Case suites ->
        if
          List.exists
            (fun s -> fst (snd (check_sutie lb lf s)) = Float)
            suites
        then
          ( (lb, lf),
            ( Float,
              SCase (List.map (fun s -> snd (check_sutie lb lf s))
suites) )
          )
        else
          ( (lb, lf),
            ( Int,
              SCase (List.map (fun s -> snd (check_sutie lb lf s))
suites) )
          )
    | StmtClosure ss ->
        let (slb, slf), nss =
          List.fold_left_map (fun (b, f) s -> check_stmt b f s) (lb,
lf) ss
        in
        ( (slb, slf),
          ( fst (List.hd (List.rev nss)),
            SStmtClosure (List.map (fun (t, s) -> Some (t, s)) nss) )
)
 in
 let (binds, func_decls), stmts =
   List.fold_left_map
     (fun (b, f) s -> check_stmt b f s)
     (StringMap.empty, StringMap.empty)
```

```
    prog
 in
 let maind =
   {
     styp = Int;
     sfname = "main";
     sformals = [];
     sframe = [];
     slocals = StringMap.fold (fun n t tl -> (t, n) :: tl) binds [];
     sbody =
       Some
         ( fst (List.hd (List.rev stmts)),
           SStmtClosure (List.map (fun (t, s) -> Some (t, s)) stmts)
);
   }
 in
 let fds = StringMap.add "main" maind func_decls in
 List.rev (StringMap.fold (fun fn fd tl -> fd :: tl) fds [])
```

SAST (Weicheng Zhao, Rachel Liu)

```
open Ast

type typ = Int | Bool | Float

type bind = typ * string

type s_atom_x =
 | SId of string
 | SIntLiteral of int
 | SFloatLiteral of float
 | SParen of s_expr_calc
 | SParenthLike of paren_op * s_expr_calc

and s_atom = typ * s_atom_x

and s_expr_pow_x = SAtom of s_atom | SPowerFunc of s_atom *
s_expr_calc
```

```
and s_expr_pow = typ * s_expr_pow_x

and s_expr_log_x =
 | SExprPow of s_expr_pow
 | SLogLike of log_like_op * s_expr_log
 | SLog of s_expr_calc * s_expr_log

and s_expr_log = typ * s_expr_log_x

and s_expr_impl_mult_x =
 | SExprLog of s_expr_log
 | SImplMult of s_expr_impl_mult * s_expr_log
 | SCall of func * s_expr_calc list

and s_expr_impl_mult = typ * s_expr_impl_mult_x

and s_expr_unary_x =
 | SExprImplMult of s_expr_impl_mult
 | SUnop of unop * s_expr_unary
 | SId of string
 | SParen of s_expr_calc

and s_expr_unary = typ * s_expr_unary_x

and s_expr_mult_x =
 | SExprUnary of s_expr_unary
 | SMultLike of s_expr_mult * mult_like_op * s_expr_unary

and s_expr_mult = typ * s_expr_mult_x

and s_expr_add_x =
 | SExprMult of s_expr_mult
 | SAddLike of s_expr_add * add_like_op * s_expr_mult

and s_expr_add = typ * s_expr_add_x

and s_expr_calc_x =
 | SExprAdd of s_expr_add
```

```
 | SCall of func * s_expr_calc list
 | SCallLargeOp of string
 | SFracLike of frac_like_op * s_expr_calc * s_expr_calc
(*  | SLargeOp of large_op * bind * s_expr_calc * s_expr_calc *
s_expr_calc*)

and s_expr_calc = typ * s_expr_calc_x

and s_expr_comp_x = SComp of s_expr_calc * comp_op * s_expr_calc

and s_expr_comp = typ * s_expr_comp_x

and s_expr_logic_atom_x =
 | SExprComp of s_expr_comp
 | SParenLogic of s_expr_logic
 | SNot of s_expr_logic_atom

and s_expr_logic_atom = typ * s_expr_logic_atom_x

and s_expr_logic_x =
 | SLogicAtom of s_expr_logic_atom
 | SLogic of s_expr_logic * logic_binop * s_expr_logic_atom

and s_expr_logic = typ * s_expr_logic_x

type sfunc_decl = {
 styp : typ;
 sfname : string;
 sformals : bind list;
 sframe : bind list;
 slocals : bind list;
 sbody : sstmt;
}

and ssutie_x = SCaseSutie of sstmt * s_expr_logic

and ssutie = typ * ssutie_x

and sstmt_x =
```

```
 | SExpr of s_expr_calc
 | SAssign of string * s_expr_calc
 | SPrint of s_expr_calc
 | SFuncDef of sfunc_decl
 | SCase of ssutie list
 | SStmtClosure of sstmt list

and sstmt = Some of typ * sstmt_x | None

type sprogram = sfunc_decl list

let string_of_typ = function
 | Int -> "int"
 | Bool -> "bool"
 | Float -> "float"

let string_of_bind (t, s) = string_of_typ t ^ ":" ^ s

let rec string_of_s_atom (t, e) =
 "(" ^ string_of_typ t ^ " : "
 ^ (match e with
   | SId s -> s
   | SIntLiteral l -> string_of_int l
   | SFloatLiteral l -> string_of_float l
   | SParen e -> "(" ^ string_of_s_expr_calc e ^ ")"
   | SParenthLike (op, e) -> (
       match op with
       | Abs -> "|" ^ string_of_s_expr_calc e ^ "|"
       | Floor -> "\\lfloor " ^ string_of_s_expr_calc e ^ " \\rfloor"
       | Ceil -> "\\lceil " ^ string_of_s_expr_calc e ^ " \\rceil"))
 ^ ")"

and string_of_s_expr_pow (t, e) =
 "(" ^ string_of_typ t ^ " : "
 ^ (match e with
   | SAtom a -> string_of_s_atom a
   | SPowerFunc (a, e) -> string_of_s_atom a ^ "^" ^
string_of_s_expr_calc e)
 ^ ")"
```

```
and string_of_s_expr_log (t, e) =
 "(" ^ string_of_typ t ^ " : "
 ^ (match e with
   | SExprPow e -> string_of_s_expr_pow e
   | SLogLike (op, e) ->
       string_of_log_like_op op ^ "(" ^ string_of_s_expr_log e ^ ")"
   | SLog (a, e) ->
       "\\log_" ^ string_of_s_expr_calc a ^ " " ^
string_of_s_expr_log e)
 ^ ")"

and string_of_s_expr_impl_mult (t, e) =
 "(" ^ string_of_typ t ^ " : "
 ^ (match e with
   | SExprLog e -> string_of_s_expr_log e
   | SImplMult (e1, e2) ->
       string_of_s_expr_impl_mult e1 ^ "•" ^ string_of_s_expr_log e2
   | SCall (f, el) ->
       string_of_func f ^ "("
       ^ String.concat ", " (List.map string_of_s_expr_calc el)
       ^ ")")
 ^ ")"

and string_of_s_expr_unary (t, e) =
 "(" ^ string_of_typ t ^ " : "
 ^ (match e with
   | SExprImplMult e -> string_of_s_expr_impl_mult e
   | SUnop (op, e) -> string_of_uop op ^ string_of_s_expr_unary e
   | SId s -> s
   | SParen e -> "(" ^ string_of_s_expr_calc e ^ ")")
 ^ ")"

and string_of_s_expr_mult (t, e) =
 "(" ^ string_of_typ t ^ " : "
 ^ (match e with
   | SExprUnary e -> string_of_s_expr_unary e
   | SMultLike (e1, op, e2) ->
       string_of_s_expr_mult e1
```

```
       ^ string_of_multi_like_op op
       ^ string_of_s_expr_unary e2)
 ^ ")"

and string_of_s_expr_add (t, e) =
 "(" ^ string_of_typ t ^ " : "
 ^ (match e with
   | SExprMult e -> string_of_s_expr_mult e
   | SAddLike (e1, op, e2) ->
       string_of_s_expr_add e1 ^ string_of_add_like_op op
       ^ string_of_s_expr_mult e2)
 ^ ")"

and string_of_s_expr_calc (t, e) =
 "(" ^ string_of_typ t ^ " : "
 ^ (match e with
   | SExprAdd e -> string_of_s_expr_add e
   | SCall (f, el) ->
       string_of_func f ^ "("
       ^ String.concat ", " (List.map string_of_s_expr_calc el)
       ^ ")"
   | SFracLike (op, e1, e2) ->
       string_of_frac_like_op op ^ "{" ^ string_of_s_expr_calc e1 ^
"} {"
       ^ string_of_s_expr_calc e2 ^ "}"
   )
 ^ ")"

and string_of_s_expr_comp (t, e) =
 "(" ^ string_of_typ t ^ " : "
 ^ (match e with
   | SComp (e1, op, e2) ->
       string_of_s_expr_calc e1 ^ string_of_comp_op op
       ^ string_of_s_expr_calc e2)
 ^ ")"

and string_of_s_expr_logic_atom (t, e) =
 "(" ^ string_of_typ t ^ " : "
 ^ (match e with
```

```
    | SExprComp e -> string_of_s_expr_comp e
    | SParenLogic e -> "(" ^ string_of_s_expr_logic e ^ ")"
    | SNot e -> "\\neg " ^ string_of_s_expr_logic_atom e)
  ^ ")"

and string_of_s_expr_logic (t, e) =
  "(" ^ string_of_typ t ^ " : "
  ^ (match e with
    | SLogicAtom e -> string_of_s_expr_logic_atom e
    | SLogic (e1, op, e2) ->
        string_of_s_expr_logic e1 ^ string_of_logic_op op
        ^ string_of_s_expr_logic_atom e2)
  ^ ")"

let rec string_of_sfunc_decl fdecl =
  string_of_typ fdecl.styp ^ " " ^ fdecl.sfname ^ "("
  ^ String.concat ", " (List.map snd fdecl.sformals)
  ^ ")\n<Frame: "
  ^ String.concat "; " (List.map string_of_bind fdecl.sframe)
  ^ ">\n[Locals:"
  ^ String.concat "; " (List.map string_of_bind fdecl.slocals)
  ^ "]\n{\n"
  ^ string_of_sstmt fdecl.sbody
  ^ "}\n"

and string_of_ssutie (t, s) =
  "(" ^ string_of_typ t ^ " : "
  ^ (match s with
    | SCaseSutie (s, e) ->
        string_of_sstmt s ^ " when " ^ string_of_s_expr_logic e)
  ^ ")"

and string_of_sstmt = function
  | Some (t, s) ->
      "(" ^ string_of_typ t ^ " : "
      ^ (match s with
        | SExpr e -> string_of_s_expr_calc e ^ "\\\\"
        | SAssign (s, e) -> s ^ "=" ^ string_of_s_expr_calc e ^ "\\\\"
        | SPrint e -> "print " ^ string_of_s_expr_calc e ^ "\n"
```

```
      | SFuncDef fd -> "define " ^ string_of_sfunc_decl fd ^ "\n"
      | SCase sl ->
          "\\begin{cases}\n"
          ^ String.concat "\n" (List.map string_of_ssutie sl)
          ^ "\n\\end{cases}\n"
      | SStmtClosure sl ->
          "\\begin{split}\n"
          ^ String.concat "\n" (List.map string_of_sstmt sl)
          ^ "\n\\end{split}\n")
    ^ ")"
 | None -> raise (Failure "Empty body found in function")

let string_of_sprogram sl =
 String.concat "\n" (List.map string_of_sfunc_decl sl)
```

Code Generation (Weicheng Zhao)

```
module L = Llvm
module A = Ast
open Sast
module StringMap = Map.Make (String)

let translate_program fdecl_list =
 let context = L.global_context () in

 (* Create the LLVM compilation module into which we will generate
code *)
 let the_module = L.create_module context "CTeX" in

 (* Get types from the context *)
 let i32_t = L.i32_type context
 and i8_t = L.i8_type context
 and i1_t = L.i1_type context
 and float_t = L.double_type context in

 (* Return the LLVM type for CTeX type *)
 let ltype_of_typ = function
   | Int -> i32_t
```

```
  | Bool -> i1_t
  | Float -> float_t
in

(* GNU C BUILTIN FUNCTIONS *)
let printf_t : L.lltype =
  L.var_arg_function_type i32_t [| L.pointer_type i8_t |]
in
let printf_func : L.llvalue =
  L.declare_function "printf" printf_t the_module
in
let abs_t = L.function_type i32_t [| i32_t |] in
let abs_func = L.declare_function "abs" abs_t the_module in
let fabs_t = L.function_type float_t [| float_t |] in
let fabs_func = L.declare_function "fabs" fabs_t the_module in
let floor_t = L.function_type i32_t [| float_t |] in
let floor_func = L.declare_function "abs" floor_t the_module in
let ceil_t = L.function_type i32_t [| float_t |] in
let ceil_func = L.declare_function "abs" ceil_t the_module in
let pow_t = L.function_type float_t [| float_t; float_t |] in
let pow_func = L.declare_function "pow" pow_t the_module in
let lg_t = L.function_type float_t [| float_t |] in
let lg_func = L.declare_function "log10" lg_t the_module in
let ln_t = L.function_type float_t [| float_t |] in
let ln_func = L.declare_function "log" ln_t the_module in
let sqrt_t = L.function_type float_t [| float_t |] in
let sqrt_func = L.declare_function "sqrt" sqrt_t the_module in
let sin_t = L.function_type float_t [| float_t |] in
let sin_func = L.declare_function "sin" sin_t the_module in
let cos_t = L.function_type float_t [| float_t |] in
let cos_func = L.declare_function "cos" cos_t the_module in
let tan_t = L.function_type float_t [| float_t |] in
let tan_func = L.declare_function "tan" tan_t the_module in
let arcsin_t = L.function_type float_t [| float_t |] in
let arcsin_func = L.declare_function "asin" arcsin_t the_module in
let arccos_t = L.function_type float_t [| float_t |] in
let arccos_func = L.declare_function "acos" arccos_t the_module in
let arctan_t = L.function_type float_t [| float_t |] in
let arctan_func = L.declare_function "atan" arctan_t the_module in
```

```
 let sinh_t = L.function_type float_t [| float_t |] in
 let sinh_func = L.declare_function "sinh" sinh_t the_module in
 let cosh_t = L.function_type float_t [| float_t |] in
 let cosh_func = L.declare_function "cosh" cosh_t the_module in
 let tanh_t = L.function_type float_t [| float_t |] in
 let tanh_func = L.declare_function "tanh" tanh_t the_module in

 (* EXTENDED BUILTIN FUNCTION *)
 let binom_t = L.function_type i32_t [| i32_t; i32_t |] in
 let binom_func = L.declare_function "binom" binom_t the_module in
 let gcd_t = L.function_type i32_t [| i32_t; i32_t |] in
 let gcd_func = L.declare_function "gcd" gcd_t the_module in

 let function_decls : (L.llvalue * L.lltype * sfunc_decl) StringMap.t
=
   let function_decl m fdecl =
     let name = fdecl.sfname
     and formal_types =
       Array.of_list
         (if List.length fdecl.sframe == 0 then
          List.map (fun (t, _) -> ltype_of_typ t) fdecl.sformals
         else
           let func_frame_type =
             L.named_struct_type context (fdecl.sfname ^ "_frame")
           in
           L.struct_set_body func_frame_type
             (Array.of_list
                (List.map (fun (t, _) -> ltype_of_typ t)
fdecl.sframe))
             false;
           func_frame_type
           :: List.map (fun (t, _) -> ltype_of_typ t) fdecl.sformals)
     in
     let ftype = L.function_type (ltype_of_typ fdecl.styp)
formal_types in
     StringMap.add name
       (L.define_function name ftype the_module, ftype, fdecl)
       m
   in
```

```
   List.fold_left function_decl StringMap.empty fdecl_list
 in

 (* MAIN ENTRY BUILDER *)
 let build_func fdecl =
   let the_function, the_types, _ =
     StringMap.find fdecl.sfname function_decls
   in
   let builder = L.builder_at_end context (L.entry_block
the_function) in
   (* Construct the function's "locals": formal arguments and locally
      declared variables. Allocate each on the stack, initialize
their
      value, if appropriate, and remember their values in the
"locals"
      map *)
   let local_vars =
     let add_formal m (t, n) p =
       L.set_value_name n p;
       let local = L.build_alloca (ltype_of_typ t) n builder in
       ignore (L.build_store p local builder);
       StringMap.add n local m
     and add_frame_element m (t, n) p =
       let frame_element_ptr =
         L.build_struct_gep p (StringMap.cardinal m)
           (fdecl.sfname ^ "_framearg_"
           ^ string_of_int (StringMap.cardinal m))
           builder
       in
       let frame_element_p = L.build_alloca (ltype_of_typ t) n
builder in
       let frame_element_v = L.build_load frame_element_ptr n builder
in
       ignore (L.build_store frame_element_v frame_element_p
builder);
       StringMap.add n frame_element_p m
     (* Allocate space for any locally declared variables and add the
      * resulting registers to our map *)
     and add_local m (t, n) =
```

```
      let local_var = L.build_alloca (ltype_of_typ t) n builder in
      StringMap.add n local_var m
    in
    if List.length fdecl.sframe == 0 then
      let formals =
        List.fold_left2 add_formal StringMap.empty fdecl.sformals
          (Array.to_list (L.params the_function))
      in
      List.fold_left add_local formals fdecl.slocals
    else
      let frame_arg = List.hd (Array.to_list (L.params
the_function)) in
      let frame_type = List.hd (Array.to_list (L.param_types
the_types)) in
      let frame =
        L.build_alloca frame_type (fdecl.sfname ^ "_frame_ptr")
builder
      in
      ignore (L.build_store frame_arg frame builder);
      let frame_elements =
        List.fold_left
          (fun m (t, n) -> add_frame_element m (t, n) frame)
          StringMap.empty fdecl.sframe
      in
      let formals =
        List.fold_left2 add_formal frame_elements fdecl.sformals
          (List.tl (Array.to_list (L.params the_function)))
      in
      List.fold_left add_local formals fdecl.slocals
  in

  (* Return the value for a variable or formal argument. Check local
names
     first, then global names *)
  let lookup lb n =
    try StringMap.find n lb
    with Not_found -> raise (Failure ("Unbound Variable " ^ n))
  in
  let add_terminal builder instr =
```

```
    match L.block_terminator (L.insertion_block builder) with
    | Some _ -> ()
    | None -> ignore (instr builder)
  in
  let rec trans_atom builder lb ((t, se) : s_atom) =
    match se with
    | SId s -> L.build_load (lookup lb s) s builder
    | SIntLiteral l -> L.const_int i32_t l
    | SFloatLiteral l -> L.const_float float_t l
    | SParen e -> trans_expr_calc builder lb e
    | SParenthLike (op, (et, e)) -> (
        let e' = trans_expr_calc builder lb (et, e) in
        match op with
        | A.Abs -> (
            match et with
            | Int -> L.build_call abs_func [| e' |] "abs" builder
            | Float -> L.build_call fabs_func [| e' |] "fabs"
builder
            | Bool ->
                raise
                  (Failure "Compile-time Error: Bool type found in
abs"))
        | A.Floor -> (
            match et with
            | Int -> e'
            | Float -> L.build_call floor_func [| e' |] "floor"
builder
            | Bool ->
                raise
                  (Failure "Compile-time Error: Bool type found in
floor"))
        | A.Ceil -> (
            match et with
            | Int -> e'
            | Float -> L.build_call ceil_func [| e' |] "ceil"
builder
            | Bool ->
                raise
                  (Failure "Compile-time Error: Bool type found in
```

```
ceil")))
   and trans_expr_pow builder lb ((t, se) : s_expr_pow) =
     match se with
     | SAtom a -> trans_atom builder lb a
     | SPowerFunc ((t1, a1), (t2, a2)) -> (
         let a1' = trans_atom builder lb (t1, a1)
         and a2' = trans_expr_calc builder lb (t2, a2) in
         match (t1, t2) with
         | Int, Int ->
             L.build_call pow_func
               [|
                 L.build_sitofp a1' float_t "pow_tmp_a1" builder;
                 L.build_sitofp a2' float_t "pow_tmp_a2" builder;
               |]
               "pow" builder
         | Int, Float ->
             L.build_call pow_func
               [| L.build_sitofp a1' float_t "pow_tmp_a1" builder;
a2' |]
               "pow" builder
         | Float, Int ->
             L.build_call pow_func
               [| a1'; L.build_sitofp a2' float_t "pow_tmp_a2"
builder |]
               "pow" builder
         | Float, Float ->
             L.build_call pow_func [| a1'; a2' |] "pow" builder
         | _ -> raise (Failure "Compile-time Error: Bool type found
in pow")
         )
   and trans_expr_log builder lb ((t, se) : s_expr_log) =
     match se with
     | SExprPow e -> trans_expr_pow builder lb e
     | SLogLike (op, (et, e)) -> (
         let e' = trans_expr_log builder lb (et, e) in
         let e' =
           match et with
           | Int -> L.build_sitofp e' float_t "log_like_tmp" builder
           | Float -> e'
```

```
            | Bool ->
                raise
                  (Failure
                     "Compile-time Error: Bool type found in log like
\
                      function")
          in
          match op with
          | A.Lg -> L.build_call lg_func [| e' |] "log10" builder
          | A.Ln -> L.build_call ln_func [| e' |] "log" builder
          | A.Sqrt -> L.build_call sqrt_func [| e' |] "sqrt" builder
          | A.Sin -> L.build_call sin_func [| e' |] "sin" builder
          | A.Cos -> L.build_call cos_func [| e' |] "cos" builder
          | A.Tan -> L.build_call tan_func [| e' |] "tan" builder
          | A.Arcsin -> L.build_call arcsin_func [| e' |] "asin"
builder
          | A.Arccos -> L.build_call arccos_func [| e' |] "acos"
builder
          | A.Arctan -> L.build_call arctan_func [| e' |] "atan"
builder
          | A.Sinh -> L.build_call sinh_func [| e' |] "sinh" builder
          | A.Cosh -> L.build_call cosh_func [| e' |] "cosh" builder
          | A.Tanh -> L.build_call tanh_func [| e' |] "tanh" builder
          | A.Csc ->
              L.build_fdiv
                (L.const_float float_t 1.0)
                (L.build_call sin_func [| e' |] "sin" builder)
                "csc" builder
          | A.Sec ->
              L.build_fdiv
                (L.const_float float_t 1.0)
                (L.build_call cos_func [| e' |] "cos" builder)
                "sec" builder
          | A.Cot ->
              L.build_fdiv
                (L.const_float float_t 1.0)
                (L.build_call tan_func [| e' |] "tan" builder)
                "cot" builder
          | A.Coth ->
```

```
            L.build_fdiv
              (L.const_float float_t 1.0)
              (L.build_call tanh_func [| e' |] "tanh" builder)
              "coth" builder)
    | SLog ((t1, a), (t2, e)) ->
        let a' = trans_expr_calc builder lb (t1, a)
        and e' = trans_expr_log builder lb (t2, e) in
        let a' =
          match t1 with
          | Int -> L.build_sitofp a' float_t "log_base_tmp" builder
          | Float -> a'
          | Bool ->
              raise
                (Failure
                   "Compile-time Error: Bool type found in log
function")
        and e' =
          match t2 with
          | Int -> L.build_sitofp e' float_t "log_value_tmp" builder
          | Float -> e'
          | Bool ->
              raise
                (Failure
                   "Compile-time Error: Bool type found in log
function")
        in
        L.build_fdiv
          (L.build_call ln_func [| e' |] "log_value" builder)
          (L.build_call ln_func [| a' |] "log_base" builder)
          "log" builder
  and trans_expr_impl_mult builder lb ((t, se) : s_expr_impl_mult) =
    match se with
    | SExprLog e -> trans_expr_log builder lb e
    | SImplMult ((t1, e1), (t2, e2)) -> (
        let e1' = trans_expr_impl_mult builder lb (t1, e1)
        and e2' = trans_expr_log builder lb (t2, e2) in
        match (t1, t2) with
        | Int, Int -> L.build_mul e1' e2' "impl_tmp" builder
        | Float, Int ->
```

```
            L.build_fmul e1'
              (L.build_sitofp e2' float_t "impl_tmp_e2" builder)
              "impl_tmp" builder
        | Int, Float ->
            L.build_fmul
              (L.build_sitofp e1' float_t "impl_tmp_e1" builder)
              e2' "impl_tmp" builder
        | Float, Float -> L.build_fmul e1' e2' "impl_tmp" builder
        | _ ->
            raise
              (Failure
                 "Compile-time Error: Bool type found in implicit
multiply")
        )
    | SCall (fname, args) -> (
        match fname with
        | A.FId s ->
            let fdef, ft, fdecl = StringMap.find s function_decls in
            let trans_arg (t, e) =
              let e' = trans_expr_calc builder lb (t, e) in
              match t with
              | Int -> L.build_sitofp e' float_t "arg_value_tmp"
builder
              | Float -> e'
              | Bool ->
                  raise
                    (Failure
                       "Compile-time Error: Bool type found in
function \
                        args")
            in
            let llargs =
              if List.length fdecl.sframe != 0 then
                let frame =
                  L.build_load
                    (lookup lb (s ^ "_frame_ptr"))
                    (s ^ "_frame") builder
                in
                frame :: List.rev (List.map trans_arg (List.rev
```

```
args))
              else List.rev (List.map trans_arg (List.rev args))
            in
            L.build_call fdef (Array.of_list llargs) (s ^ "_result")
              builder
        | A.Gcd ->
            let t1, e1 = List.hd args
            and t2, e2 = List.hd (List.tl (List.rev args)) in
            let e1' = trans_expr_calc builder lb (t1, e1)
            and e2' = trans_expr_calc builder lb (t2, e2) in
            let e1' =
              match t1 with
              | Int -> e1'
              | Float -> L.build_fptoui e1' i32_t "gcd_tmp_e1"
builder
              | Bool ->
                  raise
                    (Failure "Compile-time Error: Bool type found in
gcd")
            and e2' =
              match t2 with
              | Int -> e2'
              | Float -> L.build_fptoui e2' i32_t "gcd_tmp_e2"
builder
              | Bool ->
                  raise
                    (Failure "Compile-time Error: Bool type found in
gcd")
            in
            L.build_call gcd_func [| e1'; e2' |] "gcd" builder
        | A.Min | A.Max ->
            let rec build_minmax args =
              match args with
              | [ (t, e) ] -> (t, trans_expr_calc builder lb (t, e))
              | (t, e) :: tl ->
                  let e = trans_expr_calc builder lb (t, e)
                  and rt, rv = build_minmax tl in
                  let cond =
                    match (t, rt, fname) with
```

```
                    | Int, Int, A.Min ->
                        L.build_icmp L.Icmp.Sle e rv "leq_tmp"
builder
                    | Int, Int, A.Max ->
                        L.build_icmp L.Icmp.Sge e rv "geq_tmp"
builder
                    | Float, Int, A.Min ->
                        L.build_fcmp L.Fcmp.Ole e
                          (L.build_sitofp rv float_t "leq_tmp_e2"
builder)
                          "leq_tmp" builder
                    | Float, Int, A.Max ->
                        L.build_fcmp L.Fcmp.Oge e
                          (L.build_sitofp rv float_t "geq_tmp_e2"
builder)
                          "geq_tmp" builder
                    | Int, Float, A.Min ->
                        L.build_fcmp L.Fcmp.Ole
                          (L.build_sitofp e float_t "leq_tmp_e1"
builder)
                          rv "leq_tmp" builder
                    | Int, Float, A.Max ->
                        L.build_fcmp L.Fcmp.Oge
                          (L.build_sitofp e float_t "geq_tmp_e1"
builder)
                          rv "geq_tmp" builder
                    | Float, Float, A.Min ->
                        L.build_fcmp L.Fcmp.Ole e rv "leq_tmp"
builder
                    | Float, Float, A.Max ->
                        L.build_fcmp L.Fcmp.Oge e rv "geq_tmp"
builder
                    | _ ->
                        raise
                          (Failure
                             "Compile-time Error: Bool type found in
comp")
                in
                let typ = if t == Int && rt == Int then Int else
```

```
Float in
                    (typ, L.build_select cond e rv "minmax_tmp"
builder)
                | [] -> failwith "no element"
              in
              snd (build_minmax args))
   and trans_expr_unary builder lb ((t, se) : s_expr_unary) =
     match se with
     | SExprImplMult e -> trans_expr_impl_mult builder lb e
     | SUnop (op, (et, e)) -> (
         let ee = trans_expr_unary builder lb (et, e) in
         match op with
         | A.Uplus -> ee
         | A.Uminus -> (
             match et with
             | Int -> L.build_neg ee "unary_tmp" builder
             | Float -> L.build_fneg ee "unary_tmp" builder
             | Bool ->
                 raise
                   (Failure "Compile-time Error: Bool type found in
neg")))
     | SId s -> L.build_load (lookup lb s) s builder
     | SParen e -> trans_expr_calc builder lb e
   and trans_expr_mult builder lb ((t, se) : s_expr_mult) =
     match se with
     | SExprUnary e -> trans_expr_unary builder lb e
     | SMultLike ((t1, e1), op, (t2, e2)) -> (
         let e1' = trans_expr_mult builder lb (t1, e1)
         and e2' = trans_expr_unary builder lb (t2, e2) in
         match op with
         | A.Mult -> (
             match (t1, t2) with
             | Int, Int -> L.build_mul e1' e2' "mul_tmp" builder
             | Float, Int ->
                 L.build_fmul e1'
                   (L.build_sitofp e2' float_t "mul_tmp_e2" builder)
                   "mul_tmp" builder
             | Int, Float ->
                 L.build_fmul
```

```
                  (L.build_sitofp e1' float_t "mul_tmp_e1" builder)
                  e2' "mul_tmp" builder
            | Float, Float -> L.build_fmul e1' e2' "mul_tmp" builder
            | _ ->
                raise
                  (Failure
                     "Compile-time Error: Bool type found in
multiply"))
         | A.Div ->
            let e1' =
              match t1 with
              | Int -> L.build_sitofp e1' float_t "div_tmp_e1"
builder
              | Float -> e1'
              | Bool ->
                  raise
                    (Failure "Compile-time Error: Bool type found in
div")
            and e2' =
              match t2 with
              | Int -> L.build_sitofp e2' float_t "div_tmp_e2"
builder
              | Float -> e2'
              | Bool ->
                  raise
                    (Failure "Compile-time Error: Bool type found in
div")
            in
            L.build_fdiv e1' e2' "div_tmp" builder
         | A.Mod ->
            let e1' =
              match t1 with
              | Int -> e1'
              | Float -> L.build_fptoui e1' i32_t "mod_tmp_e1"
builder
              | Bool ->
                  raise
                    (Failure "Compile-time Error: Bool type found in
mod")
```

```
            and e2' =
              match t2 with
              | Int -> e2'
              | Float -> L.build_fptoui e2' i32_t "mod_tmp_e2"
builder
              | Bool ->
                  raise
                    (Failure "Compile-time Error: Bool type found in
mod")
            in
            L.build_urem e1' e2' "mod_tmp" builder)
   and trans_expr_add builder lb ((t, se) : s_expr_add) =
     match se with
     | SExprMult e -> trans_expr_mult builder lb e
     | SAddLike ((t1, e1), op, (t2, e2)) -> (
         let e1' = trans_expr_add builder lb (t1, e1)
         and e2' = trans_expr_mult builder lb (t2, e2) in
         match op with
         | A.Plus -> (
             match (t1, t2) with
             | Int, Int -> L.build_add e1' e2' "add_tmp" builder
             | Float, Int ->
                 L.build_fadd e1'
                   (L.build_sitofp e2' float_t "add_tmp_e2" builder)
                   "add_tmp" builder
             | Int, Float ->
                 L.build_fadd
                   (L.build_sitofp e1' float_t "add_tmp_e1" builder)
                   e2' "add_tmp" builder
             | Float, Float -> L.build_fadd e1' e2' "add_tmp" builder
             | _ ->
                 raise
                   (Failure "Compile-time Error: Bool type found in
add"))
         | A.Minus -> (
             match (t1, t2) with
             | Int, Int -> L.build_sub e1' e2' "sub_tmp" builder
             | Float, Int ->
                 L.build_fsub e1'
```

```
                  (L.build_sitofp e2' float_t "sub_tmp_e2" builder)
                  "sub_tmp" builder
              | Int, Float ->
                  L.build_fsub
                  (L.build_sitofp e1' float_t "sub_tmp_e1" builder)
                  e2' "sub_tmp" builder
              | Float, Float -> L.build_fsub e1' e2' "sub_tmp" builder
              | _ ->
                  raise
                  (Failure "Compile-time Error: Bool type found in
add")))
   and trans_expr_calc builder lb ((t, se) : s_expr_calc) =
     match se with
     | SExprAdd e -> trans_expr_add builder lb e
     | SCall (fname, args) -> (
         match fname with
         | A.FId s ->
             let fdef, ft, fdecl = StringMap.find s function_decls in
             let trans_arg (t, e) =
               let e' = trans_expr_calc builder lb (t, e) in
               match t with
               | Int -> L.build_sitofp e' float_t "arg_value_tmp"
builder
               | Float -> e'
               | Bool ->
                   raise
                     (Failure
                        "Compile-time Error: Bool type found in
function \
                         args")
             in
             let llargs =
               if List.length fdecl.sframe != 0 then
                 let frame =
                   L.build_load
                     (lookup lb (s ^ "_frame_ptr"))
                     (s ^ "_frame") builder
                 in
                 frame :: List.rev (List.map trans_arg (List.rev
```

```
args))
              else List.rev (List.map trans_arg (List.rev args))
            in
            L.build_call fdef (Array.of_list llargs) (s ^ "_result")
              builder
        | A.Gcd ->
            let t1, e1 = List.hd args
            and t2, e2 = List.hd (List.tl (List.rev args)) in
            let e1' = trans_expr_calc builder lb (t1, e1)
            and e2' = trans_expr_calc builder lb (t2, e2) in
            let e1' =
              match t1 with
              | Int -> e1'
              | Float -> L.build_fptoui e1' i32_t "gcd_tmp_e1"
builder
              | Bool ->
                  raise
                    (Failure "Compile-time Error: Bool type found in
gcd")
            and e2' =
              match t2 with
              | Int -> e2'
              | Float -> L.build_fptoui e2' i32_t "gcd_tmp_e2"
builder
              | Bool ->
                  raise
                    (Failure "Compile-time Error: Bool type found in
gcd")
            in
            L.build_call gcd_func [| e1'; e2' |] "gcd" builder
        | A.Min | A.Max ->
            let rec build_minmax args =
              match args with
              | [ (t, e) ] -> (t, trans_expr_calc builder lb (t, e))
              | (t, e) :: tl ->
                  let e = trans_expr_calc builder lb (t, e)
                  and rt, rv = build_minmax tl in
                  let cond =
                    match (t, rt, fname) with
```

```
                    | Int, Int, A.Min ->
                        L.build_icmp L.Icmp.Sle e rv "leq_tmp"
builder
                    | Int, Int, A.Max ->
                        L.build_icmp L.Icmp.Sge e rv "geq_tmp"
builder
                    | Float, Int, A.Min ->
                        L.build_fcmp L.Fcmp.Ole e
                          (L.build_sitofp rv float_t "leq_tmp_e2"
builder)
                          "leq_tmp" builder
                    | Float, Int, A.Max ->
                        L.build_fcmp L.Fcmp.Oge e
                          (L.build_sitofp rv float_t "geq_tmp_e2"
builder)
                          "geq_tmp" builder
                    | Int, Float, A.Min ->
                        L.build_fcmp L.Fcmp.Ole
                          (L.build_sitofp e float_t "leq_tmp_e1"
builder)
                          rv "leq_tmp" builder
                    | Int, Float, A.Max ->
                        L.build_fcmp L.Fcmp.Oge
                          (L.build_sitofp e float_t "geq_tmp_e1"
builder)
                          rv "geq_tmp" builder
                    | Float, Float, A.Min ->
                        L.build_fcmp L.Fcmp.Ole e rv "leq_tmp"
builder
                    | Float, Float, A.Max ->
                        L.build_fcmp L.Fcmp.Oge e rv "geq_tmp"
builder
                    | _ ->
                        raise
                          (Failure
                             "Compile-time Error: Bool type found in
comp")
                in
                let typ = if t == Int && rt == Int then Int else
```

```
Float in
                    (typ, L.build_select cond e rv "minmax_tmp"
builder)
                | [] -> failwith "no element"
              in
              snd (build_minmax args))
      | SFracLike (op, (t1, a1), (t2, a2)) -> (
          let e1' = trans_expr_calc builder lb (t1, a1)
          and e2' = trans_expr_calc builder lb (t2, a2) in
          match op with
          | A.Frac ->
              let e1' =
                match t1 with
                | Int -> L.build_sitofp e1' float_t "frac_tmp_e1"
builder
                | Float -> e1'
                | Bool ->
                    raise
                      (Failure "Compile-time Error: Bool type found in
frac")
              and e2' =
                match t2 with
                | Int -> L.build_sitofp e2' float_t "frac_tmp_e2"
builder
                | Float -> e2'
                | Bool ->
                    raise
                      (Failure "Compile-time Error: Bool type found in
frac")
              in
              L.build_fdiv e1' e2' "frac_tmp" builder
          | A.Binom ->
              let e1' =
                match t1 with
                | Int -> e1'
                | Float -> L.build_fptoui e1' i32_t "div_tmp_e1"
builder
                | Bool ->
                    raise
```

```
                    (Failure "Compile-time Error: Bool type found in
mod")
              and e2' =
                match t2 with
                | Int -> e2'
                | Float -> L.build_fptoui e2' i32_t "div_tmp_e2"
builder
                | Bool ->
                    raise
                      (Failure "Compile-time Error: Bool type found in
mod")
              in
              L.build_call binom_func [| e1'; e2' |] "binom" builder)
     | SLargeOp (op, (ti, i), (tstart, estart), (tend, aend), (texp,
exp))
       ->
         raise (Failure "Unimplmented")
   (* let acc_p = L.build_alloca float_t "acc" builder in let acc =
      L.build_store (L.const_float float_t (if op == A.Sum then 0.0
else
      1.0)) acc_p builder in let piv_p = L.build_alloca i32_t "piv"
builder
      in let slb = StringMap.add i piv_p lb in let esv =
trans_expr_calc
      builder slb (tstart,estart) in let piv = L.build_store esv
piv_p
      builder in let pred_bb = L.append_block context "while"
the_function
      in ignore(L.build_br pred_bb builder); let body_bb =
L.append_block
      context "while_body" the_function in let body_builder =
      L.builder_at_end context body_bb in let ev = trans_expr_calc
      body_builder slb (texp,exp) in let ev' = if texp == Float then
ev else
      L.build_sitofp ev float_t "large_op_tmp_ev" body_builder in let
acc =
      if op == A.Sum then L.build_fadd acc ev' "new_acc_tmp"
body_builder
      else L.build_fmul acc ev' "new_acc_tmp" body_builder in
```

```
      ignore(L.build_store acc acc_p body_builder); let piv =
L.build_store
      (L.build_add piv (L.const_int i32_t 1) "new_piv_tmp"
body_builder)
      piv_p body_builder in ignore(L.build_br pred_bb body_builder);
let
      pred_builder = L.builder_at_end context pred_bb in let aev =
      trans_expr_calc pred_builder slb (tend,aend) in let bool_val =
      L.build_icmp L.Icmp.Sle piv aev "leq_tmp" pred_builder in let
merge_bb
      = L.append_block context "merge" the_function in
      ignore(L.build_cond_br bool_val body_bb merge_bb pred_builder);
      L.builder_at_end context merge_bb *)
   and trans_expr_comp builder lb ((t, se) : s_expr_comp) =
     match se with
     | SComp ((t1, e1), op, (t2, e2)) -> (
         let e1' = trans_expr_calc builder lb (t1, e1)
         and e2' = trans_expr_calc builder lb (t2, e2) in
         match (t1, t2) with
         | Int, Int -> (
             match op with
             | A.Lt -> L.build_icmp L.Icmp.Slt e1' e2' "lt_tmp"
builder
             | A.Gt -> L.build_icmp L.Icmp.Sgt e1' e2' "gt_tmp"
builder
             | A.Leq -> L.build_icmp L.Icmp.Sle e1' e2' "leq_tmp"
builder
             | A.Geq -> L.build_icmp L.Icmp.Sge e1' e2' "geq_tmp"
builder
             | A.Eq -> L.build_icmp L.Icmp.Eq e1' e2' "eq_tmp"
builder
             | A.Neq -> L.build_icmp L.Icmp.Ne e1' e2' "neq_tmp"
builder
             | A.Nmid ->
                 L.build_icmp L.Icmp.Ne
                   (L.build_srem e2' e1' "nmid_mod_tmp" builder)
                   (L.const_int i32_t 0) "nmid_tmp" builder
             | A.Mid ->
                 L.build_icmp L.Icmp.Eq
```

```
                    (L.build_srem e2' e1' "mid_mod_tmp" builder)
                    (L.const_int i32_t 0) "mid_tmp" builder)
          | Float, Int -> (
              match op with
              | A.Lt ->
                  L.build_fcmp L.Fcmp.Olt e1'
                    (L.build_sitofp e2' float_t "lt_tmp_e2" builder)
                    "lt_tmp" builder
              | A.Gt ->
                  L.build_fcmp L.Fcmp.Ogt e1'
                    (L.build_sitofp e2' float_t "gt_tmp_e2" builder)
                    "gt_tmp" builder
              | A.Leq ->
                  L.build_fcmp L.Fcmp.Ole e1'
                    (L.build_sitofp e2' float_t "leq_tmp_e2" builder)
                    "leq_tmp" builder
              | A.Geq ->
                  L.build_fcmp L.Fcmp.Oge e1'
                    (L.build_sitofp e2' float_t "ged_tmp_e2" builder)
                    "geq_tmp" builder
              | A.Eq ->
                  L.build_fcmp L.Fcmp.Oeq e1'
                    (L.build_sitofp e2' float_t "eq_tmp_e2" builder)
                    "eq_tmp" builder
              | A.Neq ->
                  L.build_fcmp L.Fcmp.One e1'
                    (L.build_sitofp e2' float_t "neq_tmp_e2" builder)
                    "neq_tmp" builder
              | A.Nmid ->
                  L.build_icmp L.Icmp.Ne
                    (L.build_srem
                       (L.build_fptoui e2' i32_t "nmid_mod_tmp_e1"
builder)
                       e1' "nmid_mod_tmp" builder)
                    (L.const_int i32_t 0) "nmid_tmp" builder
              | A.Mid ->
                  L.build_icmp L.Icmp.Eq
                    (L.build_srem
                       (L.build_fptoui e2' i32_t "nmid_mod_tmp_e1"
```

```
builder)
                    e1' "mid_mod_tmp" builder)
                (L.const_int i32_t 0) "nmid_tmp" builder)
        | Int, Float -> (
            match op with
            | A.Lt ->
                L.build_fcmp L.Fcmp.Olt
                  (L.build_sitofp e1' float_t "lt_tmp_e1" builder)
                  e2' "lt_tmp" builder
            | A.Gt ->
                L.build_fcmp L.Fcmp.Ogt
                  (L.build_sitofp e1' float_t "gt_tmp_e1" builder)
                  e2' "gt_tmp" builder
            | A.Leq ->
                L.build_fcmp L.Fcmp.Ole
                  (L.build_sitofp e1' float_t "leq_tmp_e1" builder)
                  e2' "leq_tmp" builder
            | A.Geq ->
                L.build_fcmp L.Fcmp.Oge
                  (L.build_sitofp e1' float_t "ged_tmp_e1" builder)
                  e2' "geq_tmp" builder
            | A.Eq ->
                L.build_fcmp L.Fcmp.Oeq
                  (L.build_sitofp e1' float_t "eq_tmp_e1" builder)
                  e2' "eq_tmp" builder
            | A.Neq ->
                L.build_fcmp L.Fcmp.One
                  (L.build_sitofp e1' float_t "neq_tmp_e1" builder)
                  e2' "neq_tmp" builder
            | A.Nmid ->
                L.build_icmp L.Icmp.Ne
                  (L.build_srem e2'
                     (L.build_fptoui e1' i32_t "nmid_mod_tmp_e2"
builder)
                     "nmid_mod_tmp" builder)
                  (L.const_int i32_t 0) "nmid_tmp" builder
            | A.Mid ->
                L.build_icmp L.Icmp.Eq
                  (L.build_srem e2'
```

```
                    (L.build_fptoui e1' i32_t "nmid_mod_tmp_e2"
builder)
                    "mid_mod_tmp" builder)
                (L.const_int i32_t 0) "mid_tmp" builder)
        | Float, Float -> (
            match op with
            | A.Lt -> L.build_fcmp L.Fcmp.Olt e1' e2' "lt_tmp"
builder
            | A.Gt -> L.build_fcmp L.Fcmp.Ogt e1' e2' "gt_tmp"
builder
            | A.Leq -> L.build_fcmp L.Fcmp.Ole e1' e2' "leq_tmp"
builder
            | A.Geq -> L.build_fcmp L.Fcmp.Oge e1' e2' "geq_tmp"
builder
            | A.Eq -> L.build_fcmp L.Fcmp.Oeq e1' e2' "eq_tmp"
builder
            | A.Neq -> L.build_fcmp L.Fcmp.One e1' e2' "neq_tmp"
builder
            | A.Nmid ->
                L.build_icmp L.Icmp.Ne
                  (L.build_srem
                     (L.build_fptoui e2' i32_t "nmid_mod_tmp_e1"
builder)
                     (L.build_fptoui e1' i32_t "nmid_mod_tmp_e2"
builder)
                     "nmid_mod_tmp" builder)
                  (L.const_int i32_t 0) "nmid_tmp" builder
            | A.Mid ->
                L.build_icmp L.Icmp.Eq
                  (L.build_srem
                     (L.build_fptoui e2' i32_t "mid_mod_tmp_e1"
builder)
                     (L.build_fptoui e1' i32_t "nmid_mod_tmp_e2"
builder)
                     "mid_mod_tmp" builder)
                  (L.const_int i32_t 0) "mid_tmp" builder)
        | _ ->
            raise (Failure "Compile-time Error: Bool type found in
comp"))
```

```
   and trans_expr_logic_atom builder lb ((t, se) : s_expr_logic_atom)
=
     match se with
     | SExprComp e -> trans_expr_comp builder lb e
     | SParenLogic e -> trans_expr_logic builder lb e
     | SNot (t, e) ->
         let e' = trans_expr_logic_atom builder lb (t, e) in
         L.build_not e' "not_tmp" builder
   and trans_expr_logic builder lb ((t, se) : s_expr_logic) =
     match se with
     | SLogicAtom e -> trans_expr_logic_atom builder lb e
     | SLogic ((t1, e1), op, (t2, e2)) ->
         let e1' = trans_expr_logic builder lb (t1, e1)
         and e2' = trans_expr_logic_atom builder lb (t2, e2) in
         (match op with A.And -> L.build_and | A.Or -> L.build_or)
           e1' e2' "logic_tmp" builder
   in
   let rec trans_stmt (builder : L.llbuilder) (lb : L.llvalue
StringMap.t)
       (oss : sstmt) =
     match oss with
     | Some (t, ss) -> (
         match ss with
         | SExpr e -> ((builder, lb), trans_expr_calc builder lb e)
         | SAssign (s, e) ->
             let e' = trans_expr_calc builder lb e in
             ignore (L.build_store e' (lookup lb s) builder);
             ((builder, lb), e')
         | SPrint e ->
             let e' = trans_expr_calc builder lb e in
             let fmt =
               match t with
               | Int | Bool ->
                   L.build_global_stringptr "%d\n" "fmt_int" builder
               | Float ->
                   L.build_global_stringptr "%g\n" "fmt_float"
builder
             in
             ( (builder, lb),
```

```
              L.build_call printf_func [| fmt; e' |] "printf"
builder )
          | SFuncDef sfd -> (
              if List.length sfd.sframe != 0 then (
                let _, ft, _ = StringMap.find sfd.sfname
function_decls in
                let frame_type =
                  List.hd (Array.to_list (L.param_types ft))
                in
                let frame =
                  L.build_alloca frame_type
                    (sfd.sfname ^ "_frame_ptr")
                    builder
                in
                let store_local index p =
                  let ele_ptr =
                    L.build_struct_gep frame index
                      (sfd.sfname ^ "_framearg_" ^ string_of_int
index)
                      builder
                  in
                  let pv =
                    L.build_load p (sfd.sfname ^ "_frameargv") builder
                  in
                  ignore (L.build_store pv ele_ptr builder)
                in
                ignore
                  (List.iteri
                     (fun i (t, e) ->
                       let p = lookup lb e in
                       store_local i p)
                     sfd.sframe);
                let slb =
                  StringMap.add (sfd.sfname ^ "_frame_ptr") frame lb
                in
                ((builder, slb), frame))
              else
                ( (builder, lb),
                  match sfd.styp with
```

```
                | Int | Bool -> L.const_int i32_t 0
                | Float -> L.const_float_of_string float_t "0.0" ))
        | SCase suites ->
            let result_p =
              match t with
              | Int -> L.build_alloca i32_t "case_result" builder
              | Float -> L.build_alloca float_t "case_result"
builder
              | Bool -> raise (Failure "Unsupported type in case")
            in
            let rec build_suties b suites =
              match suites with
              | [ (st, ssuit) ] -> (
                  match ssuit with
                  | SCaseSutie (ss, pred) ->
                      let bool_val = (trans_expr_logic b lb) pred in
                      let merge_bb =
                        L.append_block context "merge_tail"
the_function
                      in
                      let merge_b = L.builder_at_end context
merge_bb in
                      let then_bb =
                        L.append_block context "then_tail"
the_function
                      in
                      let (then_b, lb), then_rll =
                        trans_stmt (L.builder_at_end context
then_bb) lb ss
                      in
                      let then_rll =
                          (match (st, t) with
                          | Int, Int | Float, Float -> then_rll
                          | Int, Float ->
                              L.build_sitofp then_rll float_t
                                "cond_ret_tmp" then_b
                          | Float, Int ->
                              L.build_fptosi then_rll float_t
                                "cond_ret_tmp" then_b
```

```
                            | Bool, _ | _, Bool ->
                                raise (Failure "Unable to return
bool"))
                      in

                      ignore (L.build_store then_rll result_p
then_b);
                      ignore (L.build_br merge_bb then_b);
                      let else_bb =
                        L.append_block context "else_tail"
the_function
                      in
                      ignore
                        (L.build_br merge_bb
                           (L.builder_at_end context else_bb));
                      ignore (L.build_cond_br bool_val then_bb
else_bb b);
                      (merge_b, merge_bb))
               | (st, ssuit) :: tl -> (
                   match ssuit with
                   | SCaseSutie (ss, pred) ->
                       let bool_val = (trans_expr_logic b lb) pred in
                       let merge_bb =
                         L.append_block context "merge" the_function
                       in
                       let merge_b = L.builder_at_end context
merge_bb in
                       let then_bb =
                         L.append_block context "then" the_function
                       in
                       let (then_b, lb), then_rll =
                         trans_stmt (L.builder_at_end context
then_bb) lb ss
                       in
                       let then_rll =
                         (match (st, t) with
                         | Int, Int | Float, Float -> then_rll
                         | Int, Float ->
                             L.build_sitofp then_rll float_t
```

```
                                "cond_ret_tmp" then_b
                          | Float, Int ->
                              L.build_fptosi then_rll float_t
                                "cond_ret_tmp" then_b
                          | Bool, _ | _, Bool ->
                              raise (Failure "Unable to return bool"))
                      in
                      ignore (L.build_store then_rll result_p
then_b);
                      ignore (L.build_br merge_bb then_b);

                      let else_bb =
                        L.append_block context "else" the_function
                      in
                      let else_b, else_ibb =
                        build_suties (L.builder_at_end context
else_bb) tl
                      in
                      ignore (L.build_br merge_bb else_b);
                      ignore (L.build_cond_br bool_val then_bb
else_bb b);
                      (merge_b, merge_bb))
              | [] -> failwith "no element"
            in
            let b, bb = build_suties builder suites in
            let b = L.builder_at_end context bb in
            ((b, lb), L.build_load result_p "cond_result" b)
        | SStmtClosure scss ->
            let (b, lb), ll =
              List.fold_left_map
                (fun (b, lb) s -> trans_stmt b lb s)
                (builder, lb) scss
            in
            ((b, lb), List.hd (List.rev ll)))
    | None -> raise (Failure "Empty body found in function")
  in

  (* print_string(fdecl.sfname ^ " : " ^ string_of_int
(StringMap.cardinal
```

```
      local_vars) ^ " " ^ string_of_int (List.length fdecl.sformals)^
" " ^
      string_of_int (List.length fdecl.slocals)); *)
   let (b, lb), rv = trans_stmt builder local_vars fdecl.sbody in
   if compare fdecl.sfname "main" = 0 then
     ignore (L.build_ret (L.const_int i32_t 0) b)
   else
     let t =
       match fdecl.sbody with
       | Some (t, ss) -> t
       | None -> raise (Failure "Empty body found in function")
     in
     add_terminal b
       (match fdecl.styp with
       | Float -> (
           match t with
           | Int -> L.build_ret (L.build_sitofp rv float_t "ret_tmp"
b)
           | Float -> L.build_ret rv
           | Bool -> raise (Failure "Unable to return bool"))
       | Int -> (
           match t with
           | Int -> L.build_ret rv
           | Float -> L.build_ret (L.build_fptosi rv i32_t "ret_tmp"
b)
           | Bool -> raise (Failure "Unable to return bool"))
       | Bool -> raise (Failure "Unable to return bool"))
 in
 List.iter build_func fdecl_list;
 the_module
Test (Unal, Hu Zheng)
open OUnit2
open Scanner
open Parser

let token_list str =
let lexbuf = Lexing.from_string str in
let rec next l =
  match ctex_read_
```

Unit Test (Hu Zheng)

```
open OUnit2
open Scanner
open Parser

let token_list str =
let lexbuf = Lexing.from_string str in
let rec next l =
  match ctex_read_token lexbuf with
      EOF -> List.rev (EOF :: l)
    | s -> next (s :: l)
in next [];;

let assert_and_print_on_failure l1 l2 =
 if List.equal (fun a b -> a = b) l1 l2 then assert_equal 0 0 else
   let () = Printf.printf("\n") in
   let () = List.iter (fun t -> print_token t true) l1 in
   let () = Printf.printf("\n") in
   let () = List.iter (fun t -> print_token t true) l2 in
   assert_equal 0 1

let scanner_test_list = [
 ("Empty", "", [ EOF ]) ;
 ("Digit", "1", [ DIGIT 1 ; EOF ]) ;
 ("Backslash should be replaced in ID", "\\mathrm{id}", [ ID
"_mathrm___id___" ; EOF ]) ;
 ("Backslash should be replaced in ID", "\\alpha", [ ID "__alpha" ;
EOF ]) ;
 ("Complex ID", "\\mathrm{id\\alpha\\beta}", [ ID
"_mathrm___id__alpha__beta___" ; EOF ]) ;
 ("ID with underscore", "a_3", [ ID "a" ; SUB ; DIGIT 3 ; EOF ]) ;
]

let scanner_tests = "Test suite for Scanner" >::: List.map (fun
(desc, input, result) ->
 desc >:: (fun _ -> assert_and_print_on_failure (token_list input)
result)
) scanner_test_list
```

```
let _ = run_test_tt_main scanner_tests
```

Ctex.ml (Weicheng Zhao, Hu Zheng)

```
(* Top-level of the CTex compiler: scan & parse the input, check the
   resulting AST and generate an SAST from it, generate LLVM IR, and
dump the
   module *)

type action = Ast | Sast | LLVM_IR | Compile

let () =
 let action = ref Compile in
 let set_action a () = action := a in
 let speclist =
   [
     ("-a", Arg.Unit (set_action Ast), "Print the AST");
     ("-s", Arg.Unit (set_action Sast), "Print the SAST");
     ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM
IR");
     ( "-c",
       Arg.Unit (set_action Compile),
       "Check and print the generated LLVM IR (default)" );
   ]
 in
 let usage_msg = "usage: ./ctex.native [-a|-s|-l|-c] [file.ctex]" in
 let channel = ref stdin in
 Arg.parse speclist (fun filename -> channel := open_in filename)
usage_msg;

 let lexbuf = Lexing.from_channel !channel in
 let ast = Parser.program Scanner.ctex_read_token lexbuf in
 match !action with
 | Ast -> print_string (Ast.string_of_program ast)
 | _ -> (
     let sast = Semant.check_program ast in
     match !action with
```

```
    | Ast -> ()
    | Sast -> print_string (Sast.string_of_sprogram sast)
    | LLVM_IR ->
        print_string
          (Llvm.string_of_llmodule (Codegen.translate_program sast))
    | Compile ->
        let m = Codegen.translate_program sast in
        Llvm_analysis.assert_valid_module m;
        print_string (Llvm.string_of_llmodule m))
```

Binom.c (Weicheng Zhao)

```
int binom(int n, int k)
{
   if (k > n)
       return 0;
   if (k == 0 || k == n)
       return 1;
    return binom(n - 1, k - 1)
          + binom(n - 1, k);
}

#ifdef BUILD_TEST
#include <stdio.h>
int main()
{
   int n = 5, k = 2;
   printf("Value of C(%d, %d) is %d ", n, k,
          binom(n, k));
   return 0;
}
#endif
```

Gcd.c (Weicheng Zhao)

```
int gcd(int n1, int n2)
{
   n1 = ( n1 > 0) ? n1 : -n1;
```

```
   n2 = ( n2 > 0) ? n2 : -n2;

   while(n1!=n2)
   {
       if(n1 > n2)
           n1 -= n2;
       else
           n2 -= n1;
   }
   return n1;
}


#ifdef BUILD_TEST
#include <stdio.h>
int main()
{
   int n = 81, k = -153;
   printf("Value of GCDC(%d, %d) is %d ", n, k,
          gcd(n, k));
   return 0;
}
#endif
```

Ctex.tb (Hu Zheng)

```
s = \sum_{i_1}^{100} i^3
```

Makefile (Weicheng Zhao)

```
# "make all" builds the executable as well as the "printbig" library
designed
# to test linking external code

.PHONY : all
all : ctex.native binom.o gcd.o unit

debug : ctex.d.byte binom.o gcd.o
```

```
# "make ctex.native" compiles the compiler
#
# The _tags file controls the operation of ocamlbuild, e.g., by
including
# packages, enabling warnings
#
# See
https://github.com/ocaml/ocamlbuild/blob/master/manual/manual.adoc

ctex.native :
   opam config exec -- \
   ocamlbuild -use-ocamlfind ctex.native

ctex.d.byte :
   opam config exec -- \
   ocamlbuild -use-ocamlfind -tag 'debug' ctex.d.byte

unit :
   opam config exec -- \
   ocamlbuild -package oUnit test.byte && \
 ./test.byte


# "make clean" removes all generated files

.PHONY : clean
clean :
   ocamlbuild -clean
```

```
all :
   $(MAKE) -C ./src $(MAKECMDGOALS) && $(MAKE) -C ./tests
$(MAKECMDGOALS)

test :
   $(MAKE) -C ./tests $(MAKECMDGOALS)

run : src/ctex.native
   /bin/sh ./scripts/run_ctex.sh -r $(ctex)
```

```
build : src/ctex.native
   /bin/sh ./scripts/run_ctex.sh -b $(ctex)

pdf : src/ctex.native
   /bin/sh ./scripts/run_ctex.sh -p $(ctex)

TOPTARGETS := clean

SUBDIRS := ./src ./tests

$(TOPTARGETS) : $(SUBDIRS)
$(SUBDIRS) :
   $(MAKE) -C $@ $(MAKECMDGOALS)

.PHONY: $(TOPTARGETS) $(SUBDIRS)
```

Tests (Unal, Hu Zheng, Weicheng Zhao)

Makefile

```
TOPTARGETS := all clean

SUBDIRS := ./test_scanner ./test_parser ./test_e2e

$(TOPTARGETS) : $(SUBDIRS)
$(SUBDIRS) :
   $(MAKE) -C $@ $(MAKECMDGOALS)

.PHONY: $(TOPTARGETS) $(SUBDIRS)
```

lexm.sh

```
#!/bin/sh
cat "$@"

echo '
```

```
{   let () =
     let human = ref false in
     let set_human a () = human := a in
     let speclist = [
       ("-h", Arg.Unit (set_human true), "Enable human readable printing");
     ] in
     let usage_msg = "usage: ./scanner [-h] [file.ctex]" in
     let channel = ref stdin in
     Arg.parse speclist (fun filename -> channel := open_in filename)
usage_msg;
     let lexbuf = Lexing.from_channel stdin in
           let token_list =
             let rec next l =
               match ctex_read_token lexbuf with
                   EOF -> EOF :: l
                 | x -> next (x :: l)
             in List.rev (next [])
           in let () = List.iter (fun t -> print_token t !human) token_list
           in Printf.printf "\\n"
}
'
```

Test_e2e

Test_e2e.sh

```
#!/bin/sh

# Regression testing script for CTeX
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"
```

```
# Path to the ctex compiler.  Usually "../src/ctex.native"
CTEX="../../src/ctex.native"

LIBPATH="../../src/binom.o ../../src/gcd.o"

LATEX_COV="/bin/sh ../../scripts/ctex2tex.sh"

PDFLATEX="pdflatex"

# Set time limit for all operations
ulimit -t 30

globallog=test_e2e.log
rm -f $globallog

tmp_output_dir=test_tmp
rm -rf $tmp_output_dir
mkdir $tmp_output_dir

pdfs_dir=test_pdfs
rm -rf $pdfs_dir
mkdir $pdfs_dir

error=0
globalerror=0

keep=0

Usage() {
   echo "Usage: test_e2e.sh [options] [.ctex files]"
   echo "-k    Keep intermediate files"
   echo "-h    Print this help"
   exit 1
}

SignalError() {
   if [ $error -eq 0 ] ; then
   echo "FAILED"
```

```
   error=1
   fi
   echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to
difffile
Compare() {
   generatedfiles="$generatedfiles $3"
   echo diff -b $1 $2 ">" $3 1>&2
   diff -b "$1" "$2" > "$3" 2>&1 || {
   SignalError "$1 differs"
   echo "FAILED $1 differs from $2" 1>&2
   }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
   echo $* 1>&2
   eval $* || {
   SignalError "$1 failed on $*"
   return 1
   }
}

# RunFail <args>
# Report the command, run it, and expect an error
RunFail() {
   echo $* 1>&2
   eval $* && {
   SignalError "failed: $* did not report an error"
   return 1
   }
   return 0
}

Check() {
```

```
   error=0
   basename=`echo $1 | sed 's/.*\\///
                           s/.ctex//'`
   reffile=`echo $1 | sed 's/.ctex$//'`
   basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

   echo -n "$basename..."

   echo 1>&2
   echo "###### Testing $basename" 1>&2

   generatedfiles=""

   generatedfiles="$generatedfiles ${tmp_output_dir}/${basename}.ll
${tmp_output_dir}/${basename}.s ${tmp_output_dir}/${basename}.exe
${tmp_output_dir}/${basename}.out ${pdfs_dir}/${basename}.tex
${pdfs_dir}/${basename}.aux ${pdfs_dir}/${basename}.out
${pdfs_dir}/${basename}.log ${pdfs_dir}/${basename}.synctex.gz" &&
   Run "$LATEX_COV" "$1" > "${pdfs_dir}/${basename}.tex" &&
   Run "$PDFLATEX" "-output-directory=${pdfs_dir}/"
"${pdfs_dir}/${basename}.tex" &&
   Run "$CTEX" < "$1" ">" "${tmp_output_dir}/${basename}.ll" &&
   Run "$LLC" "-relocation-model=pic"
"${tmp_output_dir}/${basename}.ll" ">"
"${tmp_output_dir}/${basename}.s" &&
   Run "$CC" "-o" "${tmp_output_dir}/${basename}.exe"
"${tmp_output_dir}/${basename}.s" "${LIBPATH}" "-lm" &&
   Run "${tmp_output_dir}/${basename}.exe" >
"${tmp_output_dir}/${basename}.out" &&
   Compare ${tmp_output_dir}/${basename}.out ${reffile}.reference
${tmp_output_dir}/${basename}.diff

   # Report the status and clean up the generated files

   if [ $error -eq 0 ] ; then
   if [ $keep -eq 0 ] ; then
       rm -f $generatedfiles
   fi
   echo "OK"
```

```
   echo "###### SUCCESS" 1>&2
   else
   echo "###### FAILED" 1>&2
   globalerror=$error
   fi
}

CheckFail() {
   error=0
   basename=`echo $1 | sed 's/.*\\///
                            s/.ctex//'`
   reffile=`echo $1 | sed 's/.ctex$//'`
   basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

   echo -n "$basename..."

   echo 1>&2
   echo "###### Testing $basename" 1>&2

   generatedfiles=""

   generatedfiles="$generatedfiles ${tmp_output_dir}/${basename}.err
${tmp_output_dir}/${basename}.diff" &&
   RunFail "$CTEX" "<" $1 "2>" "${tmp_output_dir}/${basename}.err"
">>" $globallog &&
   Compare ${tmp_output_dir}/${basename}.err ${reffile}.reference
${tmp_output_dir}/${basename}.diff

   # Report the status and clean up the generated files

   if [ $error -eq 0 ] ; then
   if [ $keep -eq 0 ] ; then
       rm -f $generatedfiles
   fi
   echo "OK"
   echo "###### SUCCESS" 1>&2
   else
   echo "###### FAILED" 1>&2
   globalerror=$error
```

```
   fi
}

while getopts kdpsh c; do
   case $c in
   k) # Keep intermediate files
       keep=1
       ;;
   h) # Help
       Usage
       ;;
   esac
done

shift `expr $OPTIND - 1`

LLIFail() {
 echo "Could not find the LLVM interpreter \"$LLI\"."
 echo "Check your LLVM installation and/or modify the LLI variable in
testall.sh"
 exit 1
}

which "$LLI" >> $globallog || LLIFail

# if [ ! -f printbig.o ]
# then
#     echo "Could not find printbig.o"
#     echo "Try \"make printbig.o\""
#     exit 1
# fi

if [ $# -ge 1 ]
then
   files=$@
else
   files="test_cases/test_*.ctex test_cases/bad_*.ctex"
fi
```

```
for file in $files
do
   case $file in
   *bad_*)
       CheckFail $file 2>> $globallog
       ;;
   *test_*)
       Check $file 2>> $globallog
       ;;
   *)
       echo "unknown file type $file"
       globalerror=1
       ;;
   esac
done

exit $globalerror
```

Test_e2e.log

```
/opt/homebrew/opt/llvm@11/bin/lli

###### Testing test_42
./test_e2e.sh: line 106: /bin/sh ./ctex2tex.sh: No such file or
directory
###### SUCCESS

###### Testing test_42_a
./test_e2e.sh: line 106: /bin/sh ./ctex2tex.sh: No such file or
directory
###### SUCCESS

###### Testing test_42_af
./test_e2e.sh: line 106: /bin/sh ./ctex2tex.sh: No such file or
directory
###### SUCCESS

###### Testing test_42_f
./test_e2e.sh: line 106: /bin/sh ./ctex2tex.sh: No such file or
```

```
directory
###### SUCCESS

###### Testing test_add
./test_e2e.sh: line 106: /bin/sh ./ctex2tex.sh: No such file or
directory
###### SUCCESS

###### Testing test_add_and_lg
./test_e2e.sh: line 106: /bin/sh ./ctex2tex.sh: No such file or
directory
###### SUCCESS

###### Testing bad_plus
../../src/ctex.native < test_cases/bad_plus.ctex 2>
test_tmp/bad_plus.err >> test_e2e.log
diff -b test_tmp/bad_plus.err test_cases/bad_plus.reference >
test_tmp/bad_plus.diff
###### SUCCESS
```

Makefile

```
.PHONY : all
all : ../../src/ctex.native
   /bin/sh ./test_e2e.sh

../../src/ctex.native :
   $(MAKE) -C ../../src all

.PHONY : clean
clean :
   $(MAKE) -C ../../src clean
```

Test cases:

Bad_plus.ctex

```
1+
```

Bad_plus.reference

```
Fatal error: exception Stdlib.Parsing.Parse_error
```

Test_ackerman.ctex

```
A(m, n) =
\begin{cases}
  n + 1 \\ & m = 0 \\
  A(m - 1, 1) \\ & n=0 \\
  A(m - 1, A(m, n -1)) \\ & m \neq 0 \wedge n \neq 0 \\
\end{cases}

% A(3,4) %% evaluates to 125
% A(4,1) %% evaluates to 65533
```

Test_ackerman.reference

```
125
65533
```

Test_42_a.ctex

```
a = 42 \\
% a
```

Test_42_a.reference

```
42
```

Test_42_af.ctex

```
a = 10 \\
f(b) = a + 21 + b \\
% f(11)
```

Test_42_af.reference

```
42
```

Test_42_f.ctex

```
f() = 42 \\
% f()
```

Test_42_f.reference

```
42
```

Test_42_fab.ctex

```
f(a,b) = a + b \\
% f(20, 22)
```

Test_42_fab.reference

```
42
```

Test_42_fab2.ctex

```
f(a,b) = 2*a + b \\
% f(10, 22)
```

Test_42_fab2.reference

```
42
```

Test_42_fabxy.ctex

```
f(a,b) = a + b \\
x = 20 \\
y = 22 \\
% f(x, y)
```

Test_42_fabxy.reference

```
42
```

Test_42.ctex

```
% 42
```

Test_42.reference

```
42
```

Test_add_and_lg.ctex

```
% 12 + 18\lg 100
```

Test_add_and_lg.reference

```
48
```

Test_add.ctex

```
% 12 + 34
```

Test_add.reference

```
46
```

Test_cases.ctex

```
f(n) =
\begin{cases}
  1 \\ & n = 1 \\
  1 \\ & n \neq 1 \\
\end{cases}
```

Test_cases.reference

```
```

Test_chained_functions.ctex

```
% \lg \sqrt 36
```

Test_chained_functions.reference

```
0.778151
```

Test_collatz.ctex

```
f(n) =
\begin{cases}
  n / 2 \\ & n \mod 2 = 0 \\
  3 n + 1 \\ & n \mod 2 = 1 \\
\end{cases}

g(n) =
\begin{cases}
  1 \\ & f(n) = 1 \\
  g(f(n)) \\ & f(n) \neq 1 \\
\end{cases}

% f(15) %% evaluates to 46
% f(14) %% evaluates to 7
% f(f(14)) %% evaluates to 22

%% Collatz conjecture:

% g(14) %% evaluates to 1
% g(114) %% evaluates to 1
% g(543) %% evaluates to 1
% g(17) %% evaluates to 1
% g(7) %% evaluates to 1
% g(55) %% evaluates to 1
```

Test_collatz.reference

```
46
7
22
1
```

```
1
1
1
1
1
```

Test_complex_id.ctex

```
x_2=3 \\
f_1(x)=xxx \\

% f_1(x_2)
```

Test_complex_id.reference

```
27
```

Test_fg.ctex

```
f(n) =
\begin{cases}
  n / 2 \\ & n \mod 2 = 0 \\
  3 n + 1 \\ & n \mod 2 = 1 \\
\end{cases}

g(n) =
\begin{cases}
  1 \\ & f(n) = 1 \\
  f(n) \\ & f(n) \neq 1 \\
\end{cases}
```

Test_fg.reference

```

```

Test_fib.ctex

```
\mathrm{fib}(n) =
\begin{cases}
  n * \mathrm{fib}(n - 1) \\ &n > 0 \\
  1 \\ & n=0 \\
\end{cases}
% \mathrm{fib}(6) %% evaluates to 720
```

Test_fib.reference

```
720
```

Test_gcd.ctex

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

% \mathrm{gcd}(105,63) %% evaluates to 21
```

Test_gcd.reference

```
21
```

Test_gfn.ctex

```
f(n) =
\begin{cases}
  n / 2 \\ & n \mod 2 = 0 \\
  3 n + 1 \\ & n \mod 2 = 1 \\
\end{cases}

g(n) =
\begin{cases}
  1 \\ & f(n) = 1 \\
  g(f(n)) \\ & f(n) \neq 1 \\
\end{cases}
```

Test_gfn.reference

```

```

Test_gfn2.ctex

```
f(n) =
\begin{cases}
  n / 2 \\ & n \mod 2 = 0 \\
  3 n + 1 \\ & n \mod 2 = 1 \\
\end{cases}

g(n) = 1 \\
% f(15) %% evaluates to 46
% f(14) %% evaluates to 7
% f(f(14)) %% evaluates to 22
```

Test_gfn2.reference

```
46
7
22
```

Test_mid.ctex

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

% f(6, 3) %% evaluates to 3
% f(6, 5) %% evaluates to 1
```

Test_mid.reference

```
3
1
```

Test_split.ctex

```
f(x) = \begin{split}
  y = 1 \\
  y + x \\
\end{split}

% f(5) %% evaluates to 6
```

Test_split.reference

```
6
```

Test_xyz.ctex

```
y = 1 \\

f(x) = \begin{split}
  z = y + 1 \\
  g(w) = w + z \\
  g(x) \\
\end{split}

% f(5) %% evaluates to 7
```

Test_xyz.reference

```
7
```

Test_parser

Test_parser.sh

```
#!/bin/sh
SCANNER="./scanner"
MENHIR="menhir --interpret --interpret-show-cst ../../src/parser.mly"
REJECT_REF="test_cases/reject.reference"
# Set time limit for all operations
ulimit -t 30

globallog=test_parser.log
rm -f $globallog

tmp_output_dir=test_tmp
rm -rf $tmp_output_dir
mkdir $tmp_output_dir
error=0
globalerror=0

keep=0

Usage() {
   echo "Usage: test_parser.sh [options] [.ctex files]"
   echo "-k    Keep intermediate files"
   echo "-h    Print this help"
   exit 1
}

SignalError() {
   if [ $error -eq 0 ] ; then
   echo "FAILED"
   error=1
   fi
   echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
```

```
# Compares the outfile with reffile.  Differences, if any, written to
difffile
Compare() {
   generatedfiles="$generatedfiles $3"
   echo diff -b $1 $2 ">" $3 1>&2
   diff -b "$1" "$2" > "$3" 2>&1 || {
   SignalError "$1 differs"
   echo "FAILED $1 differs from $2" 1>&2
   }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
   echo $* 1>&2
   eval $* || {
   SignalError "$1 failed on $*"
   return 1
   }
}

# RunFail <args>
# Report the command, run it, and expect an error
RunFail() {
   echo $* 1>&2
   eval $* && {
   SignalError "failed: $* did not report an error"
   return 1
   }
   return 0
}

Check() {
   error=0
   basename=`echo $1 | sed 's/.*\\///
                            s/.ctex//'`
   reffile=`echo $1 | sed 's/.ctex$//'`
   basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."
```

```
   echo -n "$basename..."

   echo 1>&2
   echo "###### Testing $basename" 1>&2

   generatedfiles=""

   generatedfiles="$generatedfiles ${tmp_output_dir}/${basename}.out
${tmp_output_dir}/${basename}.diff" &&
   Run "$SCANNER" "<" "$1" "|" "$MENHIR" ">"
"${tmp_output_dir}/${basename}.out" "2>>" $globallog &&
   Compare ${tmp_output_dir}/${basename}.out ${reffile}.reference
${tmp_output_dir}/${basename}.diff

   # Report the status and clean up the generated files

   if [ $error -eq 0 ] ; then
   if [ $keep -eq 0 ] ; then
       rm -f $generatedfiles
   fi
   echo "OK"
   echo "###### SUCCESS" 1>&2
   else
   echo "###### FAILED" 1>&2
   globalerror=$error
   fi
}

CheckFail() {
   error=0
   basename=`echo $1 | sed 's/.*\\///
                             s/.ctex//'`
   reffile=`echo $1 | sed 's/.ctex$//'`
   basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

   echo -n "$basename..."

   echo 1>&2
   echo "###### Testing $basename" 1>&2
```

```
   generatedfiles=""

   generatedfiles="$generatedfiles ${tmp_output_dir}/${basename}.err
${tmp_output_dir}/${basename}.diff" &&
   Run "$SCANNER" "<" "$1" "|" "$MENHIR" ">"
"${tmp_output_dir}/${basename}.err" "2>>" $globallog &&
   Compare ${tmp_output_dir}/${basename}.err ${REJECT_REF}
${tmp_output_dir}/${basename}.diff

   # Report the status and clean up the generated files

   if [ $error -eq 0 ] ; then
   if [ $keep -eq 0 ] ; then
       rm -f $generatedfiles
   fi
   echo "OK"
   echo "###### SUCCESS" 1>&2
   else
   echo "###### FAILED" 1>&2
   globalerror=$error
   fi
}

while getopts kdpsh c; do
   case $c in
   k) # Keep intermediate files
       keep=1
       ;;
   h) # Help
       Usage
       ;;
   esac
done


if [ $# -ge 1 ]
then
   files=$@
```

```
else
   files="test_cases/test_*.ctex test_cases/bad_*.ctex"
fi

for file in $files
do
   case $file in
   *bad_*)
       CheckFail $file 2>> $globallog
       ;;
   *test_*)
       Check $file 2>> $globallog
       ;;
   *)
       echo "unknown file type $file"
       globalerror=1
       ;;
   esac
done

exit $globalerror
```

Test_parser.log

```
###### Testing test_42
./scanner < test_cases/test_42.ctex | menhir --interpret
--interpret-show-cst ../../src/parser.mly > test_tmp/test_42.out 2>>
test_parser.log
File "../../src/parser.mly", line 14, characters 21-26:
Warning: the token DIGIT is unused.
Warning: one state has shift/reduce conflicts.
Warning: one shift/reduce conflict was arbitrarily resolved.
diff -b test_tmp/test_42.out test_cases/test_42.reference >
test_tmp/test_42.diff
###### SUCCESS

###### Testing test_add
./scanner < test_cases/test_add.ctex | menhir --interpret
--interpret-show-cst ../../src/parser.mly > test_tmp/test_add.out 2>>
```

```
test_parser.log
File "../../src/parser.mly", line 14, characters 21-26:
Warning: the token DIGIT is unused.
Warning: one state has shift/reduce conflicts.
Warning: one shift/reduce conflict was arbitrarily resolved.
diff -b test_tmp/test_add.out test_cases/test_add.reference >
test_tmp/test_add.diff
###### SUCCESS

###### Testing bad_plus
./scanner < test_cases/bad_plus.ctex | menhir --interpret
--interpret-show-cst ../../src/parser.mly > test_tmp/bad_plus.err 2>>
test_parser.log
File "../../src/parser.mly", line 14, characters 21-26:
Warning: the token DIGIT is unused.
Warning: one state has shift/reduce conflicts.
Warning: one shift/reduce conflict was arbitrarily resolved.
diff -b test_tmp/bad_plus.err test_cases/reject.reference >
test_tmp/bad_plus.diff
###### SUCCESS
```

Scanner.mll

```
{
 open Printf
 open Parser
 module StringSet = Set.Make(String)
 exception UnexpectedToken of string
 type ident_buffer = { mutable name : string; }
 let buffer : ident_buffer = { name = "" }
 let temp : ident_buffer = { name = "" }
 let fmt : ident_buffer = { name = "" }

 type flags_type = { mutable in_print : bool; }
 let flags = { in_print = false; }

(*  type token =*)
(*    (* whitespaces  *)*)
(*      NL*)
(*    | TAB*)
(*    | SPACE*)
(*    (* identifier *)*)
```

```
(*    | ID of string*)
(*    (* literals *)*)
(*    | LIT_FL of float*)
(*    | LIT_INT of int*)
(*    | DIGIT of int*)
(*    (* separators *)*)
(*    | AMP*)
(*    | DBS*)
(*    | PCT*)
(*    | COMMA*)
(*    (* basics *)*)
(*    | VAR of char*)
(*    (* formatting *)*)
(*    | SUP*)
(*    | SUB*)
(*    (* enclosures *)*)
(*    | LPRN*)
(*    | RPRN*)
(*    | ABS*)
(*    | LBRC*)
(*    | RBRC*)
(*    | LFLOOR*)
(*    | RFLOOR*)
(*    | LCEIL*)
(*    | RCEIL*)
(*    (* envs *)*)
(*    | LCASE*)
(*    | RCASE*)
(*    (* operators *)*)
(*    | ADD*)
(*    | MINUS*)
(*    | MUL*)
(*    | DIV*)
(*    | EQ*)
(*    | LT*)
(*    | GT*)
(*    (* operators from command *)*)
(*    | LEQ*)
(*    | GEQ*)
(*    | NEQ*)
(*    | MID*)
(*    | NMID*)
(*    (*  logic  *)*)
```

```
(*    | NEG*)
(*    | WEDGE*)
(*    | VEE*)
(*    (* special functions *)*)
(*    | SUM*)
(*    | PROD*)
(*    | FRAC*)
(*    | BINOM*)
(*    (* binary functions *)*)
(*    | MAX*)
(*    | MIN*)
(*    | GCD*)
(*    (* basic functions *)*)
(*    | SQRT*)
(*    | EXP*)
(*    (* trigonometric functions *)*)
(*    | SIN*)
(*    | COS*)
(*    | TAN*)
(*    | COT*)
(*    | CSC*)
(*    | SEC*)
(*    | SINH*)
(*    | TANH*)
(*    | COTH*)
(*    | COSH*)
(*    | ARCSIN*)
(*    | ARCCOS*)
(*    | ARCTAN*)
(*    (* arithmetic functions *)*)
(*    | PMOD*)
(*    (* log functions *)*)
(*    | LG*)
(*    | LN*)
(*    | LOG*)
(*    (*  bad  *)*)
(*    | BAD_CMD of string*)
(*    (*  misc  *)*)
(*    | CHAR of char*)
(*    | EOF*)
(*    | EOL*)
 let print_token t h =
   let _ = match t with
```

```
(*        NL -> if h then Printf.printf "\n" else Printf.printf "EOL"*)
(*      | TAB -> if h then Printf.printf "\t"  *)
(*      | SPACE -> if h then Printf.printf " "*)
     (* identifier *)
     | ID(s) -> if h then Printf.printf "ID(%s)" s else Printf.printf "ID"
     (* literals *)
     | LIT_FL(f) -> if h then Printf.printf "FL(%f)" f else Printf.printf
"LIT_FL"
     | LIT_INT(i) -> if h then Printf.printf "INT(%d)" i else Printf.printf
"LIT_INT"
     | DIGIT(i) -> if h then Printf.printf "DIGIT(%d)" i else Printf.printf
"DIGIT"
     (* separators *)
     | AMP -> if h then Printf.printf "&" else Printf.printf "AMP"
     | DBS -> if h then Printf.printf "\n" else Printf.printf "DBS"
     | PCT -> if h then Printf.printf "%%" else Printf.printf "PCT"
     | COMMA -> if h then Printf.printf "," else Printf.printf "COMMA"
     (* basics *)
(*      | VAR(v) -> if h then Printf.printf "VAR(%c)" v*)
     (* formatting *)
     | SUP -> if h then Printf.printf "^" else Printf.printf "SUP"
     | SUB -> if h then Printf.printf "_" else Printf.printf "SUB"
     (* enclosures *)
     | LPRN -> if h then Printf.printf "(" else Printf.printf "LPRN"
     | RPRN -> if h then Printf.printf ")" else Printf.printf "RPRN"
     | LABS -> if h then Printf.printf "|" else Printf.printf "LABS"
     | RABS -> if h then Printf.printf "|" else Printf.printf "RABS"
     | LBRC -> if h then Printf.printf "{" else Printf.printf "LBRC"
     | RBRC -> if h then Printf.printf "}" else Printf.printf "RBRC"
     | LFLOOR -> Printf.printf "LFLOOR"
     | RFLOOR -> Printf.printf "RFLOOR"
     | LCEIL -> Printf.printf "LCEIL"
     | RCEIL -> Printf.printf "RCEIL"
     (* envs *)
     | LCASE -> if h then Printf.printf "\\begin{case}" else Printf.printf
"LCASE"
     | RCASE -> if h then Printf.printf "\\end{case}" else Printf.printf
"RCASE"
     | LSPLIT -> if h then Printf.printf "\\begin{split}" else
Printf.printf "LSPLIT"
     | RSPLIT -> if h then Printf.printf "\\end{split}" else Printf.printf
"RSPLIT"
     (* operators *)
```

```
    | ADD -> if h then Printf.printf "+" else Printf.printf "ADD"
    | MINUS -> if h then Printf.printf "-" else Printf.printf "MINUS"
    | MUL -> if h then Printf.printf "*" else Printf.printf "MUL"
    | DIV -> if h then Printf.printf "/" else Printf.printf "DIV"
    | EQ -> if h then Printf.printf "=" else Printf.printf "EQ"
    | LT -> if h then Printf.printf "<" else Printf.printf "LT"
    | GT -> if h then Printf.printf ">" else Printf.printf "GT"
    | LEQ -> if h then Printf.printf "≤" else Printf.printf "LEQ"
    | GEQ -> if h then Printf.printf "≥" else Printf.printf "GEQ"
    | NEQ -> if h then Printf.printf "≠" else Printf.printf "NEQ"
    | MID -> if h then Printf.printf "|" else Printf.printf "MID"
    | NMID -> if h then Printf.printf "∤" else Printf.printf "NMID"
    (*  logic  *)
    | NEG -> if h then Printf.printf "¬" else Printf.printf "NEG"
    | WEDGE -> if h then Printf.printf "∧" else Printf.printf "WEDGE"
    | VEE -> if h then Printf.printf "∨" else Printf.printf "VEE"
    (* special functions *)
    | SUM -> if h then Printf.printf "∑" else Printf.printf "SUM"
    | PROD -> if h then Printf.printf "∏" else Printf.printf "PROD"
    | FRAC -> Printf.printf "FRAC"
    | BINOM -> Printf.printf "BINOM"
    (* binary functions *)
    | MAX ->  Printf.printf "MAX"
    | MIN -> Printf.printf "MIN"
    | GCD -> Printf.printf "GCD"
    (* basic functions *)
    | SQRT -> if h then Printf.printf "√" else Printf.printf "SQRT"
(*      | EXP -> Printf.printf "EXP"*)
    (* trigonometric functions *)
    | SIN -> Printf.printf "SIN"
    | COS -> Printf.printf "COS"
    | TAN -> Printf.printf "TAN"
    | COT -> Printf.printf "COT"
    | CSC -> Printf.printf "CSC"
    | SEC -> Printf.printf "SEC"
    | SINH -> Printf.printf "SINH"
    | TANH -> Printf.printf "TANH"
    | COTH -> Printf.printf "COTH"
    | COSH -> Printf.printf "COSH"
    | ARCSIN -> Printf.printf "ARCSIN"
    | ARCCOS -> Printf.printf "ARCCOS"
    | ARCTAN -> Printf.printf "ARCTAN"
    (* arithmetic functions *)
```

```
    | PMOD -> if h then Printf.printf "(mod)" else Printf.printf "MOD"
    (* log functions *)
    | LG -> Printf.printf "LG"
    | LN -> Printf.printf "LN"
    | LOG -> Printf.printf "LOG"
    (*  bad  *)
(*      | BAD_CMD(s) -> if h then Printf.printf "BAD_CMD(%s)" s*)
    (*  misc  *)
(*      | CHAR(c) -> if h then Printf.printf "CHAR(%c)" c*)
    | EOL -> Printf.printf "EOL"
    | EOF -> Printf.printf "EOF"

    | _ -> Printf.printf "Unimplemented"
  in let () = Printf.printf " "
  in ()

exception InvalidCommand of string
let valid_greek_letters = StringSet.of_list [
  "alpha" ;
  "beta" ;
  "Gamma" ;
  "gamma" ;
  "Delta" ;
  "delta" ;
  "epsilon" ;
  "varepsilon" ;
  "zeta" ;
  "eta" ;
  "Theta" ;
  "theta" ;
  "vartheta" ;
  "iota" ;
  "kappa" ;
  "varkappa" ;
  "Lambda";
  "lambda" ;
  "mu" ;
  "nu" ;
  "Xi" ;
  "xi" ;
  "Pi" ;
  "pi" ;
  "varpi" ;
```

```
    "rho" ;
    "varrho" ;
    "Sigma" ;
    "sigma" ;
    "varsigma" ;
    "tau" ;
    "Upsilon" ;
    "upsilon" ;
    "Phi" ;
    "phi" ;
    "varphi" ;
    "chi" ;
    "Psi" ;
    "psi" ;
    "Omega" ;
    "omega" ;
  ]

  let parse_greek raw = let letter = String.sub raw 1 ((String.length raw) -
1)
    in match StringSet.find_opt letter valid_greek_letters with
      | Some(l) -> Some(letter)
      | None -> None

  let parse_greek_in_ident raw = match parse_greek raw with
    | Some(l) -> buffer.name <- buffer.name ^ "\\" ^ l
    | None -> raise (InvalidCommand raw)

  let parse_command raw = let command = String.sub raw 1 ((String.length
raw) - 1)
    in match command with
      | "times" -> MUL
      | "cdot" -> MUL
      | "div" -> DIV
      | "leq" -> LEQ
      | "geq" -> GEQ
      | "neq" -> NEQ
      | "mnid" -> MID
      | "nmid" -> NMID
      (*  logic  *)
      | "neg" -> NEG
      | "wedge" -> WEDGE
      | "vee" -> VEE
```

```
    (* special functions *)
    | "sum" -> SUM
    | "prod" -> PROD
    | "frac" -> FRAC
    | "binom" -> BINOM
    (* binary functions *)
    | "max" -> MAX
    | "min" -> MIN
    | "gcd" -> GCD
    (* basic functions *)
    | "sqrt" -> SQRT
(*      | "exp" -> EXP*)
    (* trigonometric functions *)
    | "sin" -> SIN
    | "cos" -> COS
    | "tan" -> TAN
    | "cot" -> COT
    | "sec" -> SEC
    | "csc" -> CSC
    | "sinh" -> SINH
    | "cosh" -> COSH
    | "tanh" -> TANH
    | "csc" -> CSC
    | "coth" -> COTH
    | "arcsin" -> ARCSIN
    | "arccos" -> ARCCOS
    | "arctan" -> ARCTAN
    (* arithmetic functions *)
    | "pmod" -> PMOD
    (* log functions *)
    | "lg" -> LG
    | "ln" -> LN
    | "log" -> LOG
    | _ -> match parse_greek raw with
      | Some(l) -> ID("\\" ^ l)
      | None -> raise (UnexpectedToken raw)
}

let cmd = '\\'['a'-'z' 'A'-'Z']+
let int_r = ['0'-'9'] ['0'-'9']+
let int_part = ['0'-'9']+
let float_r = (int_part? '.' int_part) | (int_part '.')
let formatter =
```

```
    "mathrm"
  | "mathit"
  | "mathbf"
  | "mathsf"
  | "mathtt"
  | "mathfrak"
  | "mathcal"
  | "mathbb"
  | "mathscr"

rule ctex_read_token = parse
  (* whitespaces  *)
  | [' ' '\t' '\r'] { ctex_read_token lexbuf }
  | ['\n'] { if flags.in_print then let () = flags.in_print <- false in EOL
else ctex_read_token lexbuf }
  (* separators *)
  | "\\\\" { DBS }
  | '&' { AMP }
  | "%%" { ctex_read_comment lexbuf; if flags.in_print then let () =
flags.in_print <- false in EOL else ctex_read_token lexbuf }
  | '%' { let () = flags.in_print <- true in PCT }
  | ',' { COMMA }
  (* envs *)
  | "\\begin{cases}" { LCASE }
  | "\\end{cases}" { RCASE }
  | "\\begin{split}" { LSPLIT }
  | "\\end{split}" { RSPLIT }
  (* enclosures *)
  | "(" { LPRN }
  | ")" { RPRN }
  | "\\left\\|" { LABS }
  | "\\right\\|" { RABS }
  | "{" { LBRC }
  | "}" { RBRC }
  | "\\lfloor" { LFLOOR }
  | "\\rfloor" { RFLOOR }
  | "\\lceil" { LCEIL }
  | "\\rceil" { RCEIL }
  (* formatting *)
  | '^' { SUP }
  | '_' { SUB }
  (* operatings *)
  | '+' { ADD }
```

```
 | '-' { MINUS }
 | '*' { MUL }
 | '/' { DIV }
 | '=' { EQ }
 | '<' { LT }
 | '>' { GT }
 (* literals *)
 | float_r as float_s { LIT_FL(float_of_string float_s) }
 | int_r as int_s { LIT_INT(int_of_string int_s) }
 | ['0'-'9'] as d_s { DIGIT(int_of_char d_s) }
 (* complex identifier *)
 | '\\' formatter as f '{' { fmt.name <- f ; ctex_read_ident lexbuf;
ID(fmt.name ^ "{"^ String.trim temp.name ^ "}") }
 (* commands, including functions and part of enclosures, IDs, operators *)
 | cmd as s { parse_command s }
 (* identifier *)
 | ['a'-'z' 'A'-'Z'] as v { ID(String.make 1 v) }
 (* misc *)
 | _ as c { raise (UnexpectedToken (String.make 1 c)) }
 | eof { EOF }
and ctex_read_comment = parse
 | '\n' | eof { () }
 | _ { ctex_read_comment lexbuf }
and ctex_read_ident = parse
 | [' ' '\t' '\r' '\n'] { ctex_read_ident lexbuf }
 | '}' { temp.name <- buffer.name ; buffer.name <- "" ; () }
 | cmd as s { parse_greek_in_ident s ; ctex_read_ident lexbuf }
 | ['a'-'z' 'A'-'Z'] as c { buffer.name <- buffer.name ^ String.make 1 c ;
ctex_read_ident lexbuf }
 | _ as c { raise (UnexpectedToken (String.make 1 c)) }

{   let () =
     let human = ref false in
     let set_human a () = human := a in
     let speclist = [
       ("-h", Arg.Unit (set_human true), "Enable human readable printing");
     ] in
     let usage_msg = "usage: ./scanner [-h] [file.ctex]" in
     let channel = ref stdin in
     Arg.parse speclist (fun filename -> channel := open_in filename)
usage_msg;
     let lexbuf = Lexing.from_channel stdin in
           let token_list =
```

```
            let rec next l =
              match ctex_read_token lexbuf with
                  EOF -> EOF :: l
                | x -> next (x :: l)
            in List.rev (next [])
          in let () = List.iter (fun t -> print_token t !human) token_list
          in Printf.printf "\n"
}
```

Parser.mly

```
%{ open Ast %}

%token COMMA DBS LPRN RPRN LABS RABS LBRC RBRC LFLOOR RFLOOR LCEIL
RCEIL AMP LCASE RCASE PCT LSPLIT RSPLIT
%token SUP SUB
%token ADD MINUS MUL DIV
%token LEQ GEQ NEQ MID NMID EQ LT GT
%token NEG WEDGE VEE
%token SUM FRAC PROD BINOM
%token MAX MIN GCD
%token SQRT
%token SIN COS TAN COT CSC SEC SINH TANH COTH COSH ARCSIN ARCCOS
ARCTAN
%token PMOD
%token LG LN LOG
%token <int> LIT_INT DIGIT
%token <float> LIT_FL
%token <string> ID FID
%token EOF EOL

%start program
%type <Ast.program> program

%%

program:
 stmt_list EOF { $1 }

stmt:
```

```
 expr_calc DBS { Expr($1) }
| ID EQ expr_calc DBS { Assign(Id($1), $3) }
| PCT expr_calc EOL { Print($2) }
| fun_def_x EQ stmt DBS { let (id, args) = $1 in FuncDef(id, args,
$3) }
| expr_spec EQ stmt DBS { let (id, arg) = $1 in FuncDef(id, [arg],
$3) }
| fun_def_x DBS              { let (id, args) =  $1 in Call(id, args)
}   /* parses as a definition, but is a call */
| LCASE sutie_list RCASE { Case($2) }
| LSPLIT stmt_list RSPLIT { StmtClosure($2) }

stmt_list:
   stmt { [$1] }
 | stmt_list stmt { $2 :: $1 }

short_atom:
 DIGIT { Digit($1) }

long_atom:
 LIT_INT { IntLiteral($1) }
| LIT_FL { FloatLiteral($1) }
| LABS expr_calc RABS { ParenthLike(Abs, $2) }
| LFLOOR expr_calc RFLOOR { ParenthLike(Floor, $2) }
| LCEIL expr_calc RCEIL { ParenthLike(Ceil, $2) }

paren_no_id:
| LPRN expr_no_id RPRN { Paren($2) }

paren_id:
| LPRN ID RPRN { Paren(Id($2)) }

atom:
 short_atom { $1 }
| long_atom { $1 }

atom_closure:
LBRC expr_calc RBRC { ParenWrapper($2) }
```

```
expr_pow:
 atom { $1 }
| atom SUP atom_closure { PowerFunc($1, $3) }
| ID SUP atom_closure { PowerFunc(Id($1), $3) }

expr_log:
 expr_pow { $1 }
| LOG SUB atom_closure expr_log { Log($3, $4) }
| LG expr_log { LogLike(Lg, $2) }
| LN expr_log { LogLike(Ln, $2) }
| SQRT expr_log { LogLike(Sqrt, $2) }
| SIN expr_log { LogLike(Sin, $2) }
| COS expr_log { LogLike(Cos, $2) }
| TAN expr_log { LogLike(Tan, $2) }
| ARCSIN expr_log { LogLike(Arcsin, $2) }
| ARCCOS expr_log { LogLike(Arccos, $2) }
| ARCTAN  expr_log { LogLike(Arctan, $2) }
| SINH  expr_log { LogLike(Sinh, $2) }
| TANH  expr_log { LogLike(Tanh, $2) }
| COSH  expr_log { LogLike(Cosh, $2) }
| CSC  expr_log { LogLike(Csc, $2) }
| SEC  expr_log { LogLike(Sec, $2) }
| COT  expr_log { LogLike(Cot, $2) }
| COTH  expr_log { LogLike(Coth, $2) }
| LOG SUB atom_closure ID { Log($3, Id($4)) }
| LG ID { LogLike(Lg, Id($2)) }
| LN ID { LogLike(Ln, Id($2)) }
| SQRT ID { LogLike(Sqrt, Id($2)) }
| SIN ID { LogLike(Sin, Id($2)) }
| COS ID { LogLike(Cos, Id($2)) }
| TAN ID { LogLike(Tan, Id($2)) }
| ARCSIN ID { LogLike(Arcsin, Id($2)) }
| ARCCOS ID { LogLike(Arccos, Id($2)) }
| ARCTAN  ID { LogLike(Arctan, Id($2)) }
| SINH  ID { LogLike(Sinh, Id($2)) }
| TANH  ID { LogLike(Tanh, Id($2)) }
| COSH  ID { LogLike(Cosh, Id($2)) }
| CSC  ID { LogLike(Csc, Id($2)) }
| SEC  ID { LogLike(Sec, Id($2)) }
```

```
| COT  ID { LogLike(Cot, Id($2)) }
| COTH  ID { LogLike(Coth, Id($2)) }
| LOG SUB atom_closure paren_no_id { Log($3, $4) }
| LG paren_no_id { LogLike(Lg, $2 ) }
| LN paren_no_id { LogLike(Ln, $2 ) }
| SQRT paren_no_id { LogLike(Sqrt, $2 ) }
| SIN paren_no_id { LogLike(Sin, $2 ) }
| COS paren_no_id { LogLike(Cos, $2 ) }
| TAN paren_no_id { LogLike(Tan, $2 ) }
| ARCSIN paren_no_id { LogLike(Arcsin, $2 ) }
| ARCCOS paren_no_id { LogLike(Arccos, $2 ) }
| ARCTAN  paren_no_id { LogLike(Arctan, $2 ) }
| SINH  paren_no_id { LogLike(Sinh, $2 ) }
| TANH  paren_no_id { LogLike(Tanh, $2 ) }
| COSH  paren_no_id { LogLike(Cosh, $2 ) }
| CSC  paren_no_id { LogLike(Csc, $2 ) }
| SEC  paren_no_id { LogLike(Sec, $2 ) }
| COT  paren_no_id { LogLike(Cot, $2 ) }
| COTH  paren_no_id { LogLike(Coth, $2 ) }
| LOG SUB atom_closure paren_id { Log($3, $4) }
| LG paren_id { LogLike(Lg, $2 ) }
| LN paren_id { LogLike(Ln, $2 ) }
| SQRT paren_id { LogLike(Sqrt, $2 ) }
| SIN paren_id { LogLike(Sin, $2 ) }
| COS paren_id { LogLike(Cos, $2 ) }
| TAN paren_id { LogLike(Tan, $2 ) }
| ARCSIN paren_id { LogLike(Arcsin, $2 ) }
| ARCCOS paren_id { LogLike(Arccos, $2 ) }
| ARCTAN  paren_id { LogLike(Arctan, $2 ) }
| SINH  paren_id { LogLike(Sinh, $2 ) }
| TANH  paren_id { LogLike(Tanh, $2 ) }
| COSH  paren_id { LogLike(Cosh, $2 ) }
| CSC  paren_id { LogLike(Csc, $2 ) }
| SEC  paren_id { LogLike(Sec, $2 ) }
| COT  paren_id { LogLike(Cot, $2 ) }
| COTH  paren_id { LogLike(Coth, $2 ) }

expr_impl_mult:
 expr_log { $1 }
```

```
| expr_impl_mult expr_log { ImplMult($1, $2) }
| expr_impl_mult ID { ImplMult($1, Id($2)) }
| expr_impl_mult paren_no_id { ImplMult($1, $2) }
| expr_impl_mult paren_id { ImplMult($1, $2) }
| ID expr_log { ImplMultId(Id($1), $2) }
| ID ID {ImplMultId(Id($1), Id($2))}
| ID paren_no_id { SpecImpl(Id($1), $2) }
| paren_no_id expr_log { ImplMult($1, $2) }
| paren_no_id ID { ImplMult($1, Id($2)) }
| paren_no_id paren_no_id { ImplMult($1, $2) }
| paren_no_id paren_id { ImplMult($1, $2) }
| paren_id expr_log { ImplMult($1, $2) }
| paren_id ID { ImplMult($1, Id($2)) }
| paren_id paren_no_id { ImplMult($1, $2) }
| paren_id paren_id { ImplMult($1, $2) }


expr_spec:
| ID paren_id { (Id($1), $2) }

expr_unary:
 expr_impl_mult { $1 }
| paren_id { $1 }
| paren_no_id { $1 }
| MINUS expr_unary { Unop(Uminus, $2) }
| ADD expr_unary{ Unop(Uplus, $2) }
| MINUS expr_spec { Unop(Uminus, $2) }
| ADD expr_spec { Unop(Uplus, $2) }
| MINUS ID { Unop(Uminus, Id($2)) }
| ADD ID { Unop(Uplus, Id($2)) }

expr_mult:
 expr_unary { $1 }
| expr_mult MUL expr_unary { MultLike($1, Mult, $3) }
| expr_mult DIV expr_unary { MultLike($1, Div, $3) }
| expr_mult PMOD expr_unary { MultLike($1, Mod, $3) }
| expr_mult MUL ID { MultLike($1, Mult, Id($3)) }
| expr_mult DIV ID { MultLike($1, Div, Id($3)) }
| expr_mult PMOD ID { MultLike($1, Mod, Id($3)) }
```

```
| expr_mult MUL expr_spec { MultLike($1, Mult, $3) }
| expr_mult DIV expr_spec { MultLike($1, Div, $3) }
| expr_mult PMOD expr_spec { MultLike($1, Mod, $3) }
| ID MUL expr_unary { MultLike(Id($1), Mult, $3) }
| ID DIV expr_unary { MultLike(Id($1), Div, $3) }
| ID PMOD expr_unary { MultLike(Id($1), Mod, $3) }
| ID MUL ID { MultLike(Id($1), Mult, Id($3)) }
| ID DIV ID { MultLike(Id($1), Div, Id($3)) }
| ID PMOD ID { MultLike(Id($1), Mod, Id($3)) }
| ID MUL expr_spec { MultLike(Id($1), Mult, $3) }
| ID DIV expr_spec { MultLike(Id($1), Div, $3) }
| ID PMOD expr_spec { MultLike(Id($1), Mod, $3) }
| expr_spec MUL expr_unary { MultLike($1, Mult, $3) }
| expr_spec DIV expr_unary { MultLike($1, Div, $3) }
| expr_spec PMOD expr_unary { MultLike($1, Mod, $3) }
| expr_spec MUL expr_spec { MultLike($1, Mult, $3) }
| expr_spec DIV expr_spec { MultLike($1, Div, $3) }
| expr_spec PMOD expr_spec { MultLike($1, Mod, $3) }
| expr_spec MUL ID { MultLike($1, Mult, Id($3)) }
| expr_spec DIV ID { MultLike($1, Div, Id($3)) }
| expr_spec PMOD ID { MultLike($1, Mod, Id($3)) }

expr_add:
 expr_mult { $1 }
| expr_add ADD expr_mult {AddLike($1, Plus, $3) }
| expr_add MINUS expr_mult {AddLike($1, Minus, $3) }
| expr_add ADD ID {AddLike($1, Plus, Id($3)) }
| expr_add MINUS ID {AddLike($1, Minus, Id($3)) }
| expr_add ADD expr_spec {AddLike($1, Plus, $3) }
| expr_add MINUS expr_spec {AddLike($1, Minus, $3) }
| ID ADD expr_mult {AddLike(Id($1), Plus, $3) }
| ID MINUS expr_mult {AddLike(Id($1), Minus, $3) }
| ID ADD ID {AddLike(Id($1), Plus, Id($3)) }
| ID MINUS ID {AddLike(Id($1), Minus, Id($3)) }
| ID ADD expr_spec {AddLike(Id($1), Plus, $3) }
| ID MINUS expr_spec {AddLike(Id($1), Minus, $3) }
| expr_spec ADD expr_mult {AddLike($1, Plus, $3) }
| expr_spec MINUS expr_mult {AddLike($1, Minus, $3) }
| expr_spec ADD expr_spec {AddLike($1, Plus, $3) }
```

```
| expr_spec MINUS expr_spec {AddLike($1, Minus, $3) }
| expr_spec ADD ID {AddLike($1, Plus, Id($3)) }
| expr_spec MINUS ID {AddLike($1, Minus, Id($3)) }

expr_no_id:
 expr_add { $1 }
| call_expr RPRN { $1 }
| GCD LPRN arg_list RPRN { Call(Gcd, $3) }
| MAX LPRN arg_list RPRN { Call(Max, $3) }
| MIN LPRN arg_list RPRN { Call(Min, $3) }
| FRAC atom_closure atom_closure { FracLike(Frac, $2, $3) }
| BINOM atom_closure atom_closure { FracLike(Binom, $2, $3) }
| SUM SUB LBRC ID EQ expr_calc RBRC SUP atom_closure expr_calc {
LargeOp(Sum, $4, $6, $9, $10) }
| PROD SUB LBRC ID EQ expr_calc RBRC SUP atom_closure expr_calc {
LargeOp(Prod, $4, $6, $9, $10) }

fun_def:
| ID LPRN ID COMMA ID       { (Id($1), [$3; $5]) }
| fun_def COMMA ID     { let (id, args) = $1 in (id, $3 : args) }

call_expr:
| fun_def COMMA expr_no_id { let (id, args) = $1 in Call(id, $3 :
args) }
| call_expr COMMA expr_calc     { let Call(id, args) = $1 in Call(id,
$3 : args) }

fun_def_x:
| ID LPRN RPRN { (Id($1),[]) }
| fun_def RPRN { $1 }

expr_calc:
 ID         { Id($1) }
| expr_spec { SpecImpl($1) }
| expr_no_id { $1 }

arg_list:
 expr_calc { [$1] }
| arg_list COMMA expr_calc { $3 :: $1 }
```

```
expr_comp:
 expr_calc LT expr_calc { Comp($1, Lt, $3) }
| expr_calc GT expr_calc { Comp($1, Gt, $3) }
| expr_calc LEQ expr_calc { Comp($1, Leq, $3) }
| expr_calc GEQ expr_calc { Comp($1, Geq, $3) }
| expr_calc EQ expr_calc { Comp($1, Eq, $3) }
| expr_calc NEQ expr_calc { Comp($1, Neq, $3) }
| expr_calc NMID expr_calc { Comp($1, Nmid, $3) }
| expr_calc MID expr_calc { Comp($1, Mid, $3) }

expr_logic_atom:
 expr_comp { $1 }
| LPRN expr_logic RPRN { Paren($2) }
| expr_neg { $1 }

expr_neg:
 NEG expr_logic_atom { Not($2)}

expr_logic:
 expr_logic_atom { $1 }
| expr_logic WEDGE expr_comp { Logic($1, And, $3) }
| expr_logic VEE expr_comp { Logic($1, Or, $3) }

sutie:
 stmt AMP expr_logic DBS { CaseSutie($1, $3) }

sutie_list:
   sutie { [$1] }
 | sutie_list sutie { $2 :: $1 }
```

Parser.mli

```
type token =
 | COMMA
 | DBS
 | LPRN
 | RPRN
 | LABS
```

```
| RABS
| LBRC
| RBRC
| LFLOOR
| RFLOOR
| LCEIL
| RCEIL
| AMP
| LCASE
| RCASE
| PCT
| LSPLIT
| RSPLIT
| SUP
| SUB
| ADD
| MINUS
| MUL
| DIV
| LEQ
| GEQ
| NEQ
| MID
| NMID
| EQ
| LT
| GT
| NEG
| WEDGE
| VEE
| SUM
| FRAC
| PROD
| BINOM
| MAX
| MIN
| GCD
| SQRT
| SIN
```

```
  | COS
  | TAN
  | COT
  | CSC
  | SEC
  | SINH
  | TANH
  | COTH
  | COSH
  | ARCSIN
  | ARCCOS
  | ARCTAN
  | PMOD
  | LG
  | LN
  | LOG
  | LIT_INT of (int)
  | DIGIT of (int)
  | LIT_FL of (float)
  | ID of (string)
  | FID of (string)
  | EOF
  | EOL

val program :
 (Lexing.lexbuf  -> token) -> Lexing.lexbuf -> Ast.program
```

Ast.ml

```
type paren_op = Abs | Floor | Ceil
type log_like_op = Lg | Ln | Sqrt | Sin | Cos | Tan | Arcsin | Arccos
| Arctan | Sinh | Cosh | Tanh | Cot | Sec | Csc | Coth
type unop = Uplus | Uminus
type mult_like_op = Mult | Div | Mod
type add_like_op = Plus | Minus
type func = FId of string | Gcd | Min | Max
type id = Id of string
type frac_like_op = Frac | Binom
type large_op = Sum | Prod
```

```
type comp_op = Lt | Gt | Leq | Geq | Eq | Neq | Nmid | Mid
type logic_binop = And | Or

type
short_atom = ShortLiteral of int
and
long_atom = IntLiteral of int | FloatLiteral of float | Paren of
expr_calc | ParenthLike of paren_op * expr_calc
and
paren_no_id = Paren of expr_no_id
and
paren_id = Paren of id
and
atom = ShortAtom of short_atom | LongAtom of long_atom
and
atom_closure = ParenWrapper of expr_calc
and
expr_pow = Atom of atom | PowerFunc of [`atom | `id] * atom_closure
and
expr_log = ExprPow of expr_pow | LogLike of log_like_op * [`expr_log
| `id | `paren_id | `paren_no_id] | Log of atom_closure * [`expr_log
| `id | `paren_no_id | `paren_id]
and
expr_impl_mult = ExprLog of expr_log | ImplMult of [`expr_impl_mult |
`paren_id | `paren_no_id] * [`expr_log | `id | `paren_id |
`paren_no_id]
 | ImplMultId of id * [`expr_log | `id] | SpecImpl of id *
paren_no_id
and
expr_spec = ExprSpec of id * paren_id
and
expr_unary = ExprImplMult of expr_impl_mult | ParenId of paren_id |
ParenNoId of paren_no_id | Unop of unop * [`expr_unary | `expr_spec |
`id]
and
expr_mult = ExprUnary of expr_unary | MultLike of [`expr_mult | `id |
`expr_spec] * mult_like_op * [`expr_unary | `expr_spec | `id]
and
```

```
expr_add = ExprMult of expr_mult | AddLike of [`expr_add | `id |
`expr_spec] * add_like_op * [`expr_mult | `id | `expr_spec]
and
expr_no_id = ExprAdd of expr_add  | Call of func * expr_calc list |
FracLike of frac_like_op * atom_closure * atom_closure | LargeOp of
large_op * id * expr_calc * atom_closure * expr_calc
and
expr_calc = [`id | `SpecImpl of expr_spec | `ExprNoId of expr_no_id]
and
expr_comp = Comp of expr_calc * comp_op * expr_calc
and
expr_logic_atom = ExprComp of expr_comp | Paren of expr_logic |
ExprNeg of expr_neg
and
expr_neg = Not of expr_logic_atom
and
expr_logic = LogicAtom of expr_logic_atom | Logic of expr_logic *
logic_binop * expr_comp


type func_decl = {
   fname : string;
   formals : string list;
   body : stmt;
 }
and
sutie = CaseSutie of stmt * expr_logic
and
stmt = Expr of expr_calc | Assign of id * expr_calc | Print of
expr_calc | FuncDef of id * id list * stmt | Call of id * expr_calc
list | Case of sutie list | StmtClosure of stmt list

type program = stmt list
```

Makefile

```
all: scanner ../../src/parser.mly
   /bin/sh ./test_parser.sh
```

```
scanner : scanner.ml parser.ml parser.mli ast.ml
   ocamlopt -o scanner $^

%.cmo : %.ml
   ocamlc -w A -c $<

%.cmi : %.mli
   ocamlc -w A -c $<

scanner.mll :  ../../src/scanner.mll ../lexm.sh
   /bin/sh ../lexm.sh ../../src/scanner.mll > scanner.mll

parser.mly : ../../src/parser.mly
   /bin/cp -f ../../src/parser.mly parser.mly

ast.ml : ../../src/ast.ml
   /bin/cp -f ../../src/ast.ml ast.ml

parser.ml parser.mli : parser.mly ast.cmo
   ocamlyacc parser.mly && ocamlc -w A -c parser.mli ast.ml

scanner.ml : scanner.mll
   ocamllex $^

test_scanner.out :scanner test_scanner.tb
   ./scanner < test_scanner.tb > test_scanner.out


##############################

.PHONY : clean
clean :
   rm -rf *.cmi *.cmo *.cmx *.o *.mli *.ml *.mll *.mly
test_scanner.out scanner
```

Test cases:
Bad_mismatched_parens.ctex

```
(  }
```

Bad_multiple_signs.ctex

```
2+*4
```

Bad_plus.ctex

```
1+
```

Reject.reference

```
Ready!
REJECT
```

Test_42_fab.ctex

```
% f(x, y)
```

Test_42_fab.reference

```
Ready!
ACCEPT
[program:
  [stmt_list:
    [stmt:
      PCT
      [fun_def: [ident: ID] LPRN [ident: ID] COMMA [ident: ID]]
      RPRN
      EOL
    ]
  ]
  EOF
]
```

Test_42.ctex

```
% 42
```

```
Test_42.reference
Ready!
ACCEPT
[program:
  [stmt_list:
    [stmt:
      PCT
      [expr_calc:
        [expr_no_id:
          [expr_add:
```

```
            [expr_mult:
              [expr_unary:
                [expr_impl_mult:
                  [expr_log: [expr_pow: [atom: [long_atom:
LIT_INT]]]]
                ]
              ]
            ]
          ]
        ]
      ]
      EOL
    ]
  ]
  EOF
]
```

Test_add.ctex

```
10 + 20 \\
```

Test_add.reference

```
Ready!
ACCEPT
[program:
  [stmt_list:
    [stmt:
      [expr_calc:
        [expr_no_id:
          [expr_add:
            [expr_add:
              [expr_mult:
                [expr_unary:
                  [expr_impl_mult:
                    [expr_log: [expr_pow: [atom: [long_atom:
LIT_INT]]]]
                  ]
                ]
              ]
            ]
            ADD
```

```
            [expr_mult:
              [expr_unary:
                [expr_impl_mult:
                  [expr_log: [expr_pow: [atom: [long_atom:
LIT_INT]]]]
                ]
              ]
            ]
          ]
        ]
      ]
      DBS
    ]
  ]
  EOF
]
```

Test_associativity.ctex

```
a = 20 \\
\lg \sin a + 10\\
```

Test_associativity.reference

```
Ready!
ACCEPT
[program:
  [stmt_list:
    [stmt_list:
      [stmt:
        [ident: ID]
        EQ
        [expr_calc:
          [expr_no_id:
            [expr_add:
              [expr_mult:
                [expr_unary:
                  [expr_impl_mult:
                    [expr_log: [expr_pow: [atom: [long_atom:
LIT_INT]]]]
                  ]
                ]
```

```
              ]
            ]
          ]
        ]
        DBS
      ]
    ]
    [stmt:
      [expr_calc:
        [expr_no_id:
          [expr_add:
            [expr_add:
              [expr_mult:
                [expr_unary:
                  [expr_impl_mult: [expr_log: LG [expr_log: SIN
[ident: ID]]]]
                ]
              ]
            ]
            ADD
            [expr_mult:
              [expr_unary:
                [expr_impl_mult:
                  [expr_log: [expr_pow: [atom: [long_atom:
LIT_INT]]]]
                ]
              ]
            ]
          ]
        ]
      ]
      DBS
    ]
  ]
  EOF
]
```

Test_chained_funcs.ctex

```
% \lg \sin \sqrt 36
```

Test_chained_funcs.reference

```
Ready!
ACCEPT
[program:
  [stmt_list:
    [stmt:
      PCT
      [expr_calc:
        [expr_no_id:
          [expr_add:
            [expr_mult:
              [expr_unary:
                [expr_impl_mult:
                  [expr_log:
                    LG
                    [expr_log:
                      SIN
                      [expr_log:
                        SQRT
                        [expr_log: [expr_pow: [atom: [long_atom:
LIT_INT]]]]
                      ]
                    ]
                  ]
                ]
              ]
            ]
          ]
        ]
      ]
      EOL
    ]
  ]
  EOF
]
```

Test_complex_call.ctex

```
f(x,1,g(y)) \\
```

Test_complex_call.reference

```
Ready!
ACCEPT
[program:
  [stmt_list:
    [stmt:
      [expr_calc:
        [expr_no_id:
          [call_expr:
            [call_expr:
              [ident: ID]
              LPRN
              [ident: ID]
              COMMA
              [expr_no_id:
                [expr_add:
                  [expr_mult:
                    [expr_unary:
                      [expr_impl_mult:
                        [expr_log: [expr_pow: [atom: [short_atom:
DIGIT]]]]
                      ]
                    ]
                  ]
                ]
              ]
              COMMA
              [expr_calc:
                [expr_spec: [ident: ID] [paren_id: LPRN [ident: ID]
RPRN]]
              ]
            ]
            RPRN
        ]
      ]
      DBS
    ]
  ]
  EOF
```

```
]
```

Test_complex_ident.ctex

```
x_4 = 3 \\
% x_4
```

Test_complex_ident.reference

```
Ready!
ACCEPT
[program:
  [stmt_list:
    [stmt_list:
      [stmt:
        [ident: ID SUB DIGIT]
        EQ
        [expr_calc:
          [expr_no_id:
            [expr_add:
              [expr_mult:
                [expr_unary:
                  [expr_impl_mult:
                    [expr_log: [expr_pow: [atom: [short_atom:
DIGIT]]]]
                  ]
                ]
              ]
            ]
          ]
        ]
        DBS
      ]
    ]
    [stmt: PCT [expr_calc: [ident: ID SUB DIGIT]] EOL]
  ]
  EOF
]
```

Test_func_def.ctex

```
f(x) = x + 1 \\
f(x, y) = x + 1 \\
```

```
f(x, y, z) = x + y + z \\
f(a,b) = a + b \\
```

Test_func_def.reference

```
Ready!
ACCEPT
[program:
  [stmt_list:
    [stmt_list:
      [stmt_list:
        [stmt_list:
          [stmt:
            [expr_spec: [ident: ID] [paren_id: LPRN [ident: ID]
RPRN]]
            EQ
            [stmt:
              [expr_calc:
                [expr_no_id:
                  [expr_add:
                    [ident: ID]
                    ADD
                    [expr_mult:
                      [expr_unary:
                        [expr_impl_mult:
                          [expr_log: [expr_pow: [atom: [short_atom:
DIGIT]]]]
                        ]
                      ]
                    ]
                  ]
                ]
              ]
              DBS
            ]
          ]
        ]
        [stmt:
          [fun_def: [ident: ID] LPRN [ident: ID] COMMA [ident: ID]]
          RPRN
```

```
          EQ
          [stmt:
            [expr_calc:
              [expr_no_id:
                [expr_add:
                  [ident: ID]
                  ADD
                  [expr_mult:
                    [expr_unary:
                      [expr_impl_mult:
                        [expr_log: [expr_pow: [atom: [short_atom:
DIGIT]]]]
                      ]
                    ]
                  ]
                ]
              ]
            ]
            DBS
          ]
        ]
      ]
      [stmt:
        [fun_def:
          [fun_def: [ident: ID] LPRN [ident: ID] COMMA [ident: ID]]
          COMMA
          [ident: ID]
        ]
        RPRN
        EQ
        [stmt:
          [expr_calc:
            [expr_no_id:
              [expr_add:
                [expr_add: [ident: ID] ADD [ident: ID]]
                ADD
                [ident: ID]
              ]
            ]
```

```
          ]
          DBS
        ]
      ]
    ]
    [stmt:
      [fun_def: [ident: ID] LPRN [ident: ID] COMMA [ident: ID]]
      RPRN
      EQ
      [stmt:
        [expr_calc:
          [expr_no_id: [expr_add: [ident: ID] ADD [ident: ID]]]
        ]
        DBS
      ]
    ]
  ]
  EOF
]
```

Test_function_with_geek_var.ctex

```
f(\alpha) = 10 + \alpha \\
```

Test_function_with_geek_var.reference

```
Ready!
ACCEPT
[program:
  [stmt_list:
    [stmt:
      [expr_spec: [ident: ID] [paren_id: LPRN [ident: ID] RPRN]]
      EQ
      [stmt:
        [expr_calc:
          [expr_no_id:
            [expr_add:
              [expr_add:
                [expr_mult:
                  [expr_unary:
                    [expr_impl_mult:
                      [expr_log: [expr_pow: [atom: [long_atom:
```

```
LIT_INT]]]]
                    ]
                  ]
                ]
              ]
              ADD
              [ident: ID]
            ]
          ]
        ]
        DBS
      ]
    ]
  ]
  EOF
]
```

Test_gcd.ctex

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

% \mathrm{gcd}(105,63) %% evaluates to 21
```

Test_gcd.reference

```
Ready!
ACCEPT
[program:
  [stmt_list:
    [stmt_list:
      [stmt:
        [fun_def: [ident: ID] LPRN [ident: ID] COMMA [ident: ID]]
        RPRN
        EQ
        [stmt:
          LCASE
```

```
          [sutie_list:
            [sutie_list:
              [sutie:
                [stmt:
                  [expr_calc:
                    [expr_no_id:
                      [call_expr:
                        [ident: ID]
                        LPRN
                        [ident: ID]
                        COMMA
                        [expr_no_id:
                          [expr_add: [expr_mult: [ident: ID] MOD
[ident: ID]]]
                        ]
                      ]
                      RPRN
                    ]
                  ]
                  DBS
                ]
                AMP
                [expr_logic:
                  [expr_logic_atom:
                    [expr_comp:
                      [expr_calc: [ident: ID]]
                      NEQ
                      [expr_calc:
                        [expr_no_id:
                          [expr_add:
                            [expr_mult:
                              [expr_unary:
                                [expr_impl_mult:
                                  [expr_log:
                                    [expr_pow: [atom: [short_atom:
DIGIT]]]
                                  ]
                                ]
                              ]
```

```
                          ]
                        ]
                      ]
                    ]
                  ]
                ]
              ]
              DBS
            ]
          ]
          [sutie:
            [stmt: [expr_calc: [ident: ID]] DBS]
            AMP
            [expr_logic:
              [expr_logic_atom:
                [expr_comp:
                  [expr_calc: [ident: ID]]
                  EQ
                  [expr_calc:
                    [expr_no_id:
                      [expr_add:
                        [expr_mult:
                          [expr_unary:
                            [expr_impl_mult:
                              [expr_log:
                                [expr_pow: [atom: [short_atom:
DIGIT]]]
                              ]
                            ]
                          ]
                        ]
                      ]
                    ]
                  ]
                ]
              ]
            ]
            DBS
          ]
```

```
          ]
          RCASE
        ]
      ]
    ]
    [stmt:
      PCT
      [expr_calc:
        [expr_no_id:
          [call_expr:
            [ident: ID]
            LPRN
            [expr_no_id:
              [expr_add:
                [expr_mult:
                  [expr_unary:
                    [expr_impl_mult:
                      [expr_log: [expr_pow: [atom: [long_atom:
LIT_INT]]]]
                    ]
                  ]
                ]
              ]
            ]
            COMMA
            [expr_no_id:
              [expr_add:
                [expr_mult:
                  [expr_unary:
                    [expr_impl_mult:
                      [expr_log: [expr_pow: [atom: [long_atom:
LIT_INT]]]]
                    ]
                  ]
                ]
              ]
            ]
          ]
          RPRN
```

```
        ]
      ]
      EOL
    ]
  ]
  EOF
]
```

Test_gfn.ctex

```
f(g(n)) \\
```

Test_gfn.reference

```
Ready!
ACCEPT
[program:
  [stmt_list:
    [stmt:
      [expr_calc:
        [expr_no_id:
          [expr_add:
            [expr_mult:
              [expr_unary:
                [expr_impl_mult:
                  [ident: ID]
                  [paren_no_id:
                    LPRN
                    [expr_spec: [ident: ID] [paren_id: LPRN [ident:
ID] RPRN]]
                    RPRN
                  ]
                ]
              ]
            ]
          ]
        ]
      ]
      DBS
    ]
  ]
  EOF
```

```
]
```

Test_gfn2.ctex

```
f(n) =
\begin{cases}
  n / 2 \\ & n \mod 2 = 0 \\
  3 n + 1 \\ & n \mod 2 = 1 \\
\end{cases}

g(n) = 1 \\
% f(15) %% evaluates to 46
% f(14) %% evaluates to 7
% f(f(14)) %% evaluates to 22
```

Test_gfn2.reference

```
Ready!
ACCEPT
[program:
  [stmt_list:
    [stmt_list:
      [stmt_list:
        [stmt_list:
          [stmt_list:
            [stmt:
              [expr_spec: [ident: ID] [paren_id: LPRN [ident: ID]
RPRN]]
              EQ
              [stmt:
                LCASE
                [sutie_list:
                  [sutie_list:
                    [sutie:
                      [stmt:
                        [expr_calc:
                          [expr_no_id:
                            [expr_add:
                              [expr_mult:
                                [ident: ID]
                                DIV
                                [expr_unary:
```

```
                                    [expr_impl_mult:
                                      [expr_log:
                                        [expr_pow: [atom: [short_atom:
DIGIT]]]
                                      ]
                                    ]
                                  ]
                                ]
                              ]
                            ]
                          ]
                          DBS
                        ]
                        AMP
                        [expr_logic:
                          [expr_logic_atom:
                            [expr_comp:
                              [expr_calc:
                                [expr_no_id:
                                  [expr_add:
                                    [expr_mult:
                                      [ident: ID]
                                      MOD
                                      [expr_unary:
                                        [expr_impl_mult:
                                          [expr_log:
                                            [expr_pow: [atom:
[short_atom: DIGIT]]]
                                          ]
                                        ]
                                      ]
                                    ]
                                  ]
                                ]
                              ]
                              EQ
                              [expr_calc:
                                [expr_no_id:
                                  [expr_add:
```

```
                                    [expr_mult:
                                      [expr_unary:
                                        [expr_impl_mult:
                                          [expr_log:
                                            [expr_pow: [atom:
[short_atom: DIGIT]]]
                                          ]
                                        ]
                                      ]
                                    ]
                                  ]
                                ]
                              ]
                            ]
                          ]
                          DBS
                        ]
                      ]
                      [sutie:
                        [stmt:
                          [expr_calc:
                            [expr_no_id:
                              [expr_add:
                                [expr_add:
                                  [expr_mult:
                                    [expr_unary:
                                      [expr_impl_mult:
                                        [expr_impl_mult:
                                          [expr_log:
                                            [expr_pow: [atom:
[short_atom: DIGIT]]]
                                          ]
                                          [ident: ID]
                                        ]
                                      ]
                                    ]
                                  ]
                                ]
```

```
                            ADD
                            [expr_mult:
                              [expr_unary:
                                [expr_impl_mult:
                                  [expr_log:
                                    [expr_pow: [atom: [short_atom:
DIGIT]]]
                                  ]
                                ]
                              ]
                            ]
                          ]
                        ]
                        DBS
                      ]
                      AMP
                      [expr_logic:
                        [expr_logic_atom:
                          [expr_comp:
                            [expr_calc:
                              [expr_no_id:
                                [expr_add:
                                  [expr_mult:
                                    [ident: ID]
                                    MOD
                                    [expr_unary:
                                      [expr_impl_mult:
                                        [expr_log:
                                          [expr_pow: [atom:
[short_atom: DIGIT]]]
                                        ]
                                      ]
                                    ]
                                  ]
                                ]
                              ]
                            ]
                          ]
                        ]
                        EQ
```

```
                          [expr_calc:
                            [expr_no_id:
                              [expr_add:
                                [expr_mult:
                                  [expr_unary:
                                    [expr_impl_mult:
                                      [expr_log:
                                        [expr_pow: [atom:
[short_atom: DIGIT]]]
                                      ]
                                    ]
                                  ]
                                ]
                              ]
                            ]
                          ]
                        ]
                      ]
                    ]
                  ]
                  DBS
                ]
              ]
              RCASE
            ]
          ]
        ]
        [stmt:
          [expr_spec: [ident: ID] [paren_id: LPRN [ident: ID]
RPRN]]
          EQ
          [stmt:
            [expr_calc:
              [expr_no_id:
                [expr_add:
                  [expr_mult:
                    [expr_unary:
                      [expr_impl_mult:
                        [expr_log: [expr_pow: [atom: [short_atom:
DIGIT]]]]]
```

```
                        ]
                      ]
                    ]
                  ]
                ]
              ]
              DBS
            ]
          ]
        ]
        [stmt:
          PCT
          [expr_calc:
            [expr_no_id:
              [expr_add:
                [expr_mult:
                  [expr_unary:
                    [expr_impl_mult:
                      [ident: ID]
                      [paren_no_id:
                        LPRN
                        [expr_no_id:
                          [expr_add:
                            [expr_mult:
                              [expr_unary:
                                [expr_impl_mult:
                                  [expr_log:
                                    [expr_pow: [atom: [long_atom:
LIT_INT]]]
                                  ]
                                ]
                              ]
                            ]
                          ]
                        ]
                        RPRN
                      ]
                    ]
                  ]
```

```
                ]
              ]
            ]
          ]
          EOL
        ]
      ]
      [stmt:
        PCT
        [expr_calc:
          [expr_no_id:
            [expr_add:
              [expr_mult:
                [expr_unary:
                  [expr_impl_mult:
                    [ident: ID]
                    [paren_no_id:
                      LPRN
                      [expr_no_id:
                        [expr_add:
                          [expr_mult:
                            [expr_unary:
                              [expr_impl_mult:
                                [expr_log:
                                  [expr_pow: [atom: [long_atom:
LIT_INT]]]
                                ]
                              ]
                            ]
                          ]
                        ]
                      ]
                      RPRN
                    ]
                  ]
                ]
              ]
            ]
          ]
```

```
        ]
        EOL
      ]
    ]
    [stmt:
      PCT
      [expr_calc:
        [expr_no_id:
          [expr_add:
            [expr_mult:
              [expr_unary:
                [expr_impl_mult:
                  [ident: ID]
                  [paren_no_id:
                    LPRN
                    [expr_no_id:
                      [expr_add:
                        [expr_mult:
                          [expr_unary:
                            [expr_impl_mult:
                              [ident: ID]
                              [paren_no_id:
                                LPRN
                                [expr_no_id:
                                  [expr_add:
                                    [expr_mult:
                                      [expr_unary:
                                        [expr_impl_mult:
                                          [expr_log:
                                            [expr_pow:
                                              [atom: [long_atom:
LIT_INT]]
                                              ]
                                            ]
                                          ]
                                        ]
                                      ]
                                    ]
                                  ]
                                ]
                              ]
                            ]
                          ]
                        ]
                      ]
                    ]
                  ]
                ]
              ]
            ]
          ]
        ]
      ]
    ]
```

```
                                RPRN
                              ]
                            ]
                          ]
                        ]
                      ]
                    ]
                    RPRN
                  ]
                ]
              ]
            ]
          ]
        ]
        EOL
      ]
    ]
    EOF
]
```

Test_simple_function.ctex

```
f(a) = 10 + a \\
```

Test_simple_function.reference

```
Ready!
ACCEPT
[program:
  [stmt_list:
    [stmt:
      [expr_spec: [ident: ID] [paren_id: LPRN [ident: ID] RPRN]]
      EQ
      [stmt:
        [expr_calc:
          [expr_no_id:
            [expr_add:
              [expr_add:
                [expr_mult:
                  [expr_unary:
                    [expr_impl_mult:
```

```
                      [expr_log: [expr_pow: [atom: [long_atom:
LIT_INT]]]]
                    ]
                  ]
                ]
              ]
              ADD
              [ident: ID]
            ]
          ]
        ]
        DBS
      ]
    ]
  ]
  EOF
]
```

Test_single_function_call2.ctex

```
f(x) \\
```

Test_single_function_call2.reference

```
Ready!
ACCEPT
[program:
  [stmt_list:
    [stmt:
      [expr_calc:
        [expr_spec: [ident: ID] [paren_id: LPRN [ident: ID] RPRN]]
      ]
      DBS
    ]
  ]
  EOF
]
```

Test_single_function_call.ctex

```
f(105) \\
```

Test_single_function_call.reference

```
Ready!
ACCEPT
[program:
  [stmt_list:
    [stmt:
      [expr_calc:
        [expr_no_id:
          [expr_add:
            [expr_mult:
              [expr_unary:
                [expr_impl_mult:
                  [ident: ID]
                  [paren_no_id:
                    LPRN
                    [expr_no_id:
                      [expr_add:
                        [expr_mult:
                          [expr_unary:
                            [expr_impl_mult:
                              [expr_log:
                                [expr_pow: [atom: [long_atom:
LIT_INT]]]
                              ]
                            ]
                          ]
                        ]
                      ]
                    ]
                    RPRN
                  ]
                ]
              ]
            ]
          ]
        ]
      ]
      DBS
    ]
  ]
```

```
  EOF
]
```

Test_scanner

Bad_constant_too_large_int.ctex

```
99999999999999999999999999999999999999999999999999
```

Bad_constant_too_large_int.reference

```
Fatal error: exception Failure("int_of_string")
```

bad_identifier_invalid_style.ctex

```
\mathrf{hello}
```

Bad_identifier_invalid_style.reference

```
Fatal error: exception Scanner.UnexpectedToken("\mathrf")
```

Bad_identifier_mixture_1.ctex

```
\mathbb{520l}
```

Bad_identifier_mixture_1.reference

```
Fatal error: exception Scanner.UnexpectedToken("5")
```

Bad_identifier_no_space_1.ctex

```
\thetaA
```

Bad_identifier_no_space_2.ctex

```
\mathbb{\thetaa}
```

Bad_identifier_no_space_2.reference

```
Fatal error: exception Scanner.InvalidCommand("\thetaa")
```

Bad_identifier_non_greek.ctex

```
\Alpha
```

Bad_identifier_non_greek.reference

```
Fatal error: exception Scanner.UnexpectedToken("\Alpha")
```

Bad_newline.ctex

```
\
\
```

Bad_newline.reference

```
Fatal error: exception Scanner.UnexpectedToken("\")
```

Bad_newline_fake.ctex

```
\|\\|
```

Bad_newline_fake.reference

```
Fatal error: exception Scanner.UnexpectedToken("\")
```

Test_basic_case_statements.ctex

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

Test_basic_case_statements.reference

```
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 DBS EOF
```

Test_basic_comments.ctex

```
%% comment % %
% 10+8
%% comment %% 1919 %% new-line
% comment %%
% 810 %% this % is % comment %%
```

Test_basic_comments.reference

```
PCT LIT_INT ADD DIGIT EOL PCT ID ID ID ID ID ID ID EOL PCT LIT_INT EOL EOF
```

Test_basic_constant.ctex

```
2
233
2147483647
+999
-999
1.
0.1
.1
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384460955058223172535940812848111745028410270193852110555964462294895493038196442881097566593344612847564823378678316527120190914564856692346034861045432664821339360726024914127372458700660631558817488152092096282925409171536436789259036001133053054882046652138414695194151160943305727036575959195309218611738193261179310511854807446237996274956735188575272489122793818301194912983367336244065664308602139494639522473719070217986094370277053921717629317675238467481846766940513200056812714526356082778577134275778960917363717872146844090122495343014654958537105079227968925892359235
+9.87654
-1.2345
```

```
a2a
a + 1.2
b - 3.4
56
1.000a000
```

Test_basic_constant.reference

```
DIGIT LIT_INT LIT_INT ADD LIT_INT MINUS LIT_INT LIT_FL LIT_FL LIT_FL LIT_FL ADD
LIT_FL MINUS LIT_FL ID DIGIT ID ID ADD LIT_FL ID MINUS LIT_FL LIT_INT LIT_FL ID
LIT_INT EOF
```

Test_basic_functions.ctex

```
\mathbb{foo}(a,b,\alpha,5,-1.02,\mathrm{bar}(\mathbb{ctex},))=b(
   h,
   a
   ,
   2p,
   \sin y
)
```

Test_basic_functions.reference

```
ID LPRN ID COMMA ID COMMA ID COMMA DIGIT COMMA MINUS LIT_FL COMMA ID LPRN ID COMMA
RPRN RPRN EQ ID LPRN ID COMMA ID COMMA DIGIT ID COMMA SIN ID RPRN EOF
```

Test_basic_identifier.ctex

```
a
aB
\Theta\alpha
a\theta
\theta a
\mathbb{ybb}
\mathbb{a\theta}
\mathbb{\theta a}
a_1
a_100
1a_123
a_aa
a_{100}
a_a
a_B
\theta_\alpha
\alpha_\Theta
\Theta_{999}
\mathbb{aaa}_9
\mathbb{aaa}_{ef\alpha a}
\mathbb{a\theta a}_B
```

```
\mathbb{thisisasupersupersupersupersupersupersupersupersupersupersupersupersup
ersupersupersupersupersupersupersupersupersupersupersupersupersupersupersupers
upersupersupersupersupersupersupersupersupersupersupersupersupersupersupersupe
rsupersupersupersupersupersupersupersupersupersupersupersupersupersupersupersu
persupersupersupersupersupersupersupersupersupersupersupersupersupersupersuper
supersupersupersupersupersupersupersupersupersupersupersupersupersupersupersup
ersupersupersupersupersupersupersupersupersupersupersupersupersupersupersupers
upersupersupersupersupersupersupersupersupersupersupersupersupersupersupersupe
rsupersupersupersupersupersupersupersupersupersupersupersupersupersupersupersu
persupersupersupersupersupersupersupersupersupersupersupersupersupersupersuper
supersupersupersupersupersupersupersupersupersupersupersupersupersupersupersup
ersupersupersupersupersupersupersupersupersupersupersupersupersupersupersupers
upersupersupersupersupersupersuperlongidentifier}
```

Test_basic_identifier.reference

```
ID ID ID ID ID ID ID ID ID ID ID ID SUB DIGIT ID SUB LIT_INT DIGIT ID SUB
LIT_INT ID SUB ID ID ID SUB LBRC LIT_INT RBRC ID SUB ID ID SUB ID ID SUB ID ID SUB
ID ID SUB LBRC LIT_INT RBRC ID SUB DIGIT ID SUB LBRC ID ID ID ID RBRC ID SUB ID ID
EOF
```

Test_basic_tokens.ctex

```
10 - 289
/114 * 51
```

Test_basic_tokens.reference

```
LIT_INT MINUS LIT_INT DIV LIT_INT MUL LIT_INT EOF
```

Test_boolean_exps.ctex

```
a_1 \wedge a_2
```

Test_boolean_exps.reference

```
ID SUB DIGIT WEDGE ID SUB DIGIT EOF
```

Test_gcd.ctex

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

% \mathrm{gcd}(105,63) %% evaluates to 21
```

Test_gcd.reference

```
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
```

Test_identifier_with_newline.ctex

```
\mathbb{this
is
an
identifier
with
newline
space
   and
       tab}
```

Test_identifier_with_newline.reference

```
ID EOF
```

Test_mismatched_parans.reference

```
LPRN RBRC EOF
```

Test_strange_comments.ctex

```
%% hello my %
% friend %%% a % % \alpha %%%
% tell
me what has happened %%%%
```

Test_strange_comments.reference

```
PCT ID ID ID ID ID ID EOL PCT ID ID ID ID EOL ID ID ID ID ID ID ID ID ID ID ID
ID ID ID ID ID EOF
```