Calcul^2

Programming Language Final Report

Junde Huang - jh3419
Kewei Ge - kg2481
Zhan Shu - zs2242
Jinxi Zhao - jz2540
Wenting Yin - wy2214

December 20, 2013

Content

1. Introduction

1.1 Background

Students may sometimes find it difficult to calculate integration, derivation and other math functions. In order to help students in calculus, we develop this CalCul^2 language. Users will find it very helpful when they are studying calculus. This language will give them easy access to programming integration, derivation and other math functions.

CalCul^2 is a programming language focusing on math calculation such as calculus. Like a powerful calculator, CalCul^2 can calculate values, deal with math functions, and even do calculus. Although the existing languages, such as C or Java, could also handle math computations, they often requires complicated programming, which is very tedious for users. Our CalCul^{2}will allow users to describe and manipulate calculus in a much simpler way.

1.2 Project overview

Our compiler does not actually compile the source code into binary code, it translates the CalCul^2 source code into correct C++ code, and we will get the executable file through the g++ compiler. The process translate CalCul^2 code to C++ code requires the collaboration of scanner, parser, ast, semantics, and code generation. We also build numerous tests to make it reliable. Details of these will be presented in the following parts.

1.3 Language features

CalCul^2 has the following features:

- dynamic and strong typed
- with powerful built-in function
- calculate derivation, integration, and other common math functions

2. Language Tutorial

2.1 Getting Started with the Compiler

Before all, our compiler recommends the system has installed the latest gcc compiler. Notice that the version of gcc should be at least 4.8. In addition, it is also required for the system to have ability to run makefile. At last, the system needs to support ocaml as well.

Then all you need to do is to uncompress the Calcul^2.tar.gz file, most likely you will see a lot of .ml/.mly/.mll files and also some .h/.sh files.

You are all set now. Please follow the instructions in following sections to run our compiler.

2.2 Installing and Compiling the Compiler

After uncompressing the Calcul^2.tar.gz file, we can run the following commands to compile our calcul^2 compiler.

$ make clean
$ make

The first command ensures there is no intermediate files in our working directory. The second command runs the makefile and you will see the output like this:

```
ocamlc -c ast.ml
ocamlyacc parser.mly
ocamlc -c parser.mli
ocamlc -c parser.ml
ocamllex scanner.mll
96 states, 3964 transitions, table size 16432 bytes
ocamlc -c scanner.ml
ocamlc -c semantics.ml
ocamlc -c codegen.ml
ocamlc -c compiler.ml
ocamlc -o calcul parser.cmo scanner.cmo semantics.cmo codegen.cmo compiler.cmo
```

To run your input file, please use the following command:

$ ./run.sh input.cul

Then you will see the output printed in the terminal, and there will be a output.cpp and output executable file generated in current directory.

But if you want to compile and run the input.cul step by step (without shell script), you can do it as following (assuming you have already run the make command):

$ ./calcul < input.cul
$ g++ -o output output.cpp
$ ./output

Hopefully you can get a output.cpp after the first command, and get the output execute file after the second command. You can run your code by ./output command.

2.3 A first example of Calcul[2] Program

This is a very simple and clear example to use the math function. The calcul^2 source code is as below:

```
main()
{
    f($x) = x ^ 3 + 3 * x;
    :f'(x);
}
```

We make it into the "input.cul" file. This sample declares a math function named f with respect to x, the formula is x ^ 3 + 3 * x. As we know, the derivation of this math function is 3 * x ^ 2 + 3.

To compile and run this input code, we run the following commands:

$ make clean
$ make
$ ./run.sh input.cul

You will see the output printed in the terminal as:

x^3*3/x+3

Also you will get a generated cpp file named output.cpp as:

```
#include <cstdio>
#include "calcul2.h"
using namespace std;
```

```
int main()
{
    double printer;

    vector<string> f_var;
    f_var.push_back("x");
    vector<double> f_begin, f_end, f_now;
    FTree f(f_var);

    f.AddNode(new FNode(T_OP,0,PLUS));
    f.AddNode(new FNode(T_OP,0,POWER));
    f.AddNode(new FNode(T_VAR,0,0));
    f.AddNode(new FNode(T_VAL,3.));
    f.AddNode(new FNode(T_OP,0,TIMES));
    f.AddNode(new FNode(T_VAL,3.));
    f.AddNode(new FNode(T_VAR,0,0));

    f_now.clear();
    f.Derive("x") -> Print();
    cout << "\n";

    return 0;
}
```

The output is x^3*3/x+3, which can be reduce to 3 * x ^ 2 + 3 that matches our expectation. The reduce step is really difficult to implement and we only support several basic reduce steps.

2.4 Additional example of Calcul^2 Program

In this section we will show a more complex example. The source code is as following:

```
main(){
      f($x)=x^x-1;
      g($x)=f'(x);
      :f;
      :g;
      :f'(x);
      :g@x(1,5);
      :f(5);
}
```

Then run the following command:

$ make clean
$ make
$ ./run.sh input.cul

Then you will get the following output cpp file:

```
#include <cstdio>
#include "calcul2.h"
using namespace std;

int main()
{
	double printer;

	vector<string> f_var;
	f_var.push_back("x");
	vector<double> f_begin, f_end, f_now;
	FTree f(f_var);

	f.AddNode(new FNode(T_OP,0,MINUS));
	f.AddNode(new FNode(T_OP,0,POWER));
	f.AddNode(new FNode(T_VAR,0,0));
	f.AddNode(new FNode(T_VAR,0,0));
	f.AddNode(new FNode(T_VAL,1.));

	vector<string> g_var;
	g_var.push_back("x");
	vector<double> g_begin, g_end, g_now;
	FTree g(g_var);

	g.AddNode(f.Derive("x") -> Copy());

	f.Print();
	cout << "\n";

	g.Print();
	cout << "\n";

	f_now.clear();
	f.Derive("x") -> Print();
	cout << "\n";
```

```
        g_begin.clear();
        g_end.clear();
        g_begin.push_back(1.);
        g_end.push_back(5.);
        cout << g.GetIntegal(g_begin, g_end);
        cout << "\n";

        f_now.clear();
        f_now.push_back(5.);
        printer = f.GetValue(f_now);
        printf("%lf\n",printer);
        cout << "\n";

        return 0;
}
```

The ourput printed in the terminal is as following:

```
x^x-1
x^x*(1+ln(x))
x^x*(1+ln(x))
3124
3124.000000
```

3. Language Manual

3.1 Introduction

3.1.1 Language Description

CalCul^2 is a simple yet powerful programming language that deals with limits and the differentiation and integration of functions of one or more variables. CalCul^2 supports basic mathematical as well as calculus operations, which involves ordinary differential equation and definite integral calculation. Instead of redundancy types, developers can solve problems with clear and simple codes in CalCul^2. E.g. To make codes concise, CalCul^2 does not require a type definition for each variable, even a function, instead it will automatically recognize the type and might adjust it with the processing of programs. Furthermore, CalCul^2 also provides ability to simple evaluation of functions. CalCul^2 is a concise language for calculus: just using less to accomplish more.

3.2. Program Definition

CalCul^2 is a programming language that deals with limits and the differentiation and integration of functions of one or more variables.

3.3. Lexical Conventions

3.3.1 Comments

In line comments are represented by ##, while block comments are delimited by #* and *#.

3.3.2 Identifiers

Identifiers are comprised of uppercase letters, lower letters, digits and underscore _. The identifiers are case sensitive. The first character cannot be a digit, nor can it be an underscore_.

3.3.3 Keywords

The following keywords are reserved:

float	sqrt
if	sin
else	cos
for	tan
while	asin
log	acos
ln	atan

3.3.4 Constants

3.3.4.1 *Float constants*

Floats must have exactly one decimal point, with digits on either side.
Only decimal representations are allowed. Example:

```
## Define floats
a=3.0;
b=0.;
c=.155;
d=5;
```

3.3.5 Operators

3.3.5.1 *Arithmetic Operators*

Operator	Definition
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Power
=	Assignment
’	Derivative
@	Integral

3.3.5.2 *Logic Operators*

<	less than
>	greater than
<=	equal to or less than
>=	equal to or greater than
==	equal to
!=	not equal to

&&	and
||	or
!	not

3.3.6 Punctuators

Symbol	Symbol Definition
;	Marks the end of a statement
( )	Used for grouping parameters in function call, or indicates priority in expression
{}	Marks the beginning and end of a group of statements
:	Used to get output
,	Separates parameters of a function

3.4. Data Types

Calculus Calculator Language don't need the user to designate data type for every variable. It will be determined by the initial definition. The build-in data types are as follows:

Type	Definition
float	The float type is used for all floating-point calculationls
function	A function is a mathematical function with several variables

3.5. Expression and Operations

3.5.1 Primary expressions

3.5.1.1 *identifier*

Its type can be a variable or a function without explicit declaration.

3.5.1.2 *constant*

Its type is int or float.

3.5.1.3 ***(expression)***
3.5.1.4 ***identifier[expression]***
It yields the value at the index of a vector.

3.5.2 Unary operators

3.5.2.1 ***!expression***
Performs negation on the expression, which must be boolean.

3.5.2.2 ***expression'***
Performs derivation of function with respect to a variable, where there must be only one variable in this function.

3.5.3 Basic Arithmetic Operators

As a calculating language we definitely need to bring users the basic arithmetic operators, including '+', '-', '*', '/', '^'(power). And also we provide logical operators like '<', '>', '<=', '>=', '==', '!=', '&&', '|||' and '!'.

3.5.3.1 Multiplicative operators

3.5.3.1.1 *expression* * *expression*
Multiplication operator * is valid between two floats.

3.5.3.1.2 expression / expression
Division operator / is valid between two floats.

3.5.3.1.3 *expression1* ^ *expression2*
Power operator ^ is used to calculate the value of base expression1 with power of expression2.

3.5.3.2 Additive operators

3.5.3.2.1 *expression* + *expression*
The + operator returns the sum of two expressions.

3.5.3.2.2 *expression* - *expression*
The - operator returns the difference between two expressions.

3.5.3.3 Comparison operators

3.5.3.3.1 *expression* == *expression*
Returns a boolean value whether these two expressions are equal.

3.5.3.3.2 *expression* > *expression (expression* >= *expression)*
Returns a boolean value whether the left expression is larger (larger or equal) than the right expression.

3.5.3.2.2 *expression* < *expression (expression* <= *expression)*

Returns a boolean value whether the left expression is smaller (smaller or equal) than the right expression.

3.5.3.2.2 *expression != expression*
Returns a boolean value whether these two expressions are not equal.

3.5.3.4 Logical operators

3.5.3.4.1 *expression && expression*
Logical AND between two boolean expressions.

3.5.3.4.2 *expression* || *expression*
Logical OR between two boolean expressions.

3.5.4 Advanced Arithmetic Operators

We will provide advanced arithmetic operators as follows:

3.5.4.1 ***f(expression1, expression2, ...)***
Assume f is a function with respect to multiple variables or a single variable, thus f(expr1, expr2,..) operator is used to get the value of this function when these variables are specified a value. The return type is a value (integer or float). E.g. f($x, $y) = 3 * x ^ 2 + y /3; Hence we get f(2, 3) = 13.

3.5.4.2 ***f'(expression)***
Unlike the unary operator *f'* mentioned above, where the function *f* has only a single variable, here the derivation operator supports multiple variables. That is, it gives the derivation of function *f* with respect to some *expression*. The return type is a function. E.g. f($x, $y) = 3 * x ^ 2 + 3 * x * y; We have f'(x) = 6 * x + 3 * y.

3.5.4.3 ***f'(expression1, expression2, …)***
This operator gives the gradient of function *f* at *expression1, expression2* and possibly other variables, which will be returned in type of vector.

3.5.4.4 ***f@expression1(expression2, expression3)***
Definite integral operator @ is used for calculate the definite integral of function *f* with respect to *expression* in the range from *expression2* to *expression3*. The return type is a float value. E.g. f(x) = 3 * x ^ 2 + x; We have f@x(3, 10) = 1018.5.

All these Arithmetic Operators except *f(expr1, expr2,...)* are only allowed to be applied on pure single-variate mathematical functions(i.e. function not include recursions, loops or condition controls).

3.6. Statement

3.6.1 What is a statement

A statement is ended with semi-colon(';'). A statement is defined as a declaration and definition of variables or functions, an assignment, an execution of a function, or an input/ouput flow control.

3.6.2 Control flow

In CalCul[2], we also provide if-else condition controls and while and for loops. They are all very easy to use.

```
If-Else:
      if (bool_expr) {
          Statement 1;
      }
      else{
          Statement 2;
      }
```

For If-Else Statement, if the boolean expression *bool_expr* holds, then the program will run *Statement 1*. Otherwise the program will run *Statement 2*.

loops:

While Statement:

```
      while (bool_expr) {
          Statement;
      }
```

The While Statement will keep running *Statement* until the boolean expression *bool_expr* doesn't hold.

For Statement:

```
      for (stmt; stmt; stmt) {
          Statement;
      }
```

The For statement is exactly the same as one in C++.

3.7. Declaration

3.7.1 Variable

A variable need not to be declared before assigning it a value, instead, a variable will be declare internally by the system when you assign a value to it. And the type of the variable is determined by its first definition.

For example, x=3, y=2.1,z=(1,2), x, y, z will be declared as type int, float, vector respectively. After first assignment, the type of a variable cannot be changed any more. That is to say, if x is assigned any value of type other than integer, an exception will be raised.

A variable name can be a sequence of letters, or a sequence of letters followed by a digit.

Example of valid variable names: x, a1, abc, abc2

Example of invalid variable names: _x, 1y, a2b

3.7.2 Function

A function declaration is a function name followed by a pair of parenthesis, inside which there should be the argument list. Function name can only be a letter or a sequence of letters. Note that a function has to be declared and defined at the same time as well, doing just either one is not allowed. In addition, just like C, a main function is required. e.g. main(){...}

There are two way to declare and define a function:

1) A function declaration followed by '=' and expression, e.g. f($x)=x+1, f($x,$y)=x*y
   Note: "$" needs to be as a prefix of variables when declared in parameter list.
2) A function declaration followed by a pair of braces, inside which there should be statements, e.g. f(x1,x2){ statement1; statement2; }
   Note: "$" is not needed.

3.8. System Functions

3.8.1 Math Functions

The math functions include *'sqrt','sin', 'cos', 'tan', 'asin', 'acos', 'atan', 'log'* and *'ln'*.

sqrt(expr): return the square root of the parameter.
sin(expr): return the sine value of the parameter. The unit of the parameter is radian.
cos(expr): return the cosine value of the parameter.
tan(expr): return the tangent value of the parameter.
asin(expr): return the inverse sine value of the parameter. The unit of the return value is radian.
acos(expr): return the inverse cosine value of the parameter.
atan(expr): return the inverse tangent value of the parameter.
log(expr): return the logarithm value of the parameter based on 10.

ln(expr): return the logarithm value of the parameter based on *e*.

3.9 Example

3.9.1 Define a variable and a function

```
## Define a variable
x = 3;

## Define a function f(x) and function variable x
f($x) = 3*x;
```

3.9.2 Evaluate a function

```
## Evaluate f($x) = 3*x + 2 such that x = 4, output: f(x)=14
f($x) = 3*x + 2;
:f(4);
```

3.9.3 Get Derivative of function

```
## Get derivative of function f($x)=3*x^2 , using " ' "
f($x) = 3*x^2;
:f'(x);              ## Output will be f'(x) = 6*x

## Get the derivative of function f($x1, $x2)=2*x1+3*x2 with respect of x1
f($x1,$x2) = 2*x1+3*x2;
:f'(x1);             ## Output will be f'(x1) = 2

## Get the gradient of function f($x1, $x2)=2*x1+3*x2 when x1=2 and x2=3
f($x1,$x2) = 2*x1+3*x2;
:f'(2,3);            ## Output will be f'(2,3) = [2,3];
```

3.9.4 Get Definite Integral of function

```
## Compute definite integral of f(x) within interval of 3 and 10
f($x) = 3*x^2  + x;
:f@x(3,10);                  ## Output will be f@x(3,10)=1018.5
```

3.9.5 Recursion and Flow control

```
## using recursion and if-else control
f(x){
  if(x==0)
     f=0;
  else
     f=f(x-1)+x;
}
:f(3);    ## Output will be f(3)=6
```

```
## using loop control
f(x){
  f = 0;
  while(not x==0){
     f = f + x;
     x = x - 1;
   }
}
:f(3);    ## Output will be f(3)=6
:f'(3);   ## Error, the function f(x) is not a mathematical function
```

4. Project Plan

4.1 Overview

Our team consist of five people, a group meeting was held every week. We didn't specify responsibility for every teammate; instead, we tried to get everyone involved in every part of the project.

Github is used as a tool of code sharing and version control throughout the whole process, so that codes can be updated in a timely manner.

4.2 Planning

The whole project is split into seven parts: 1) proposal; 2) language manual; 3) ast and parser; 4) semantic check; 5) code generation; 6) C++ code; 7)final report.

Since we have to play with a brand new programming language, none of us are familiar with it. So after finalizing the proposal and language manual, we spent a few week learning and discussing potential coding problem we might have, then we started to get cracking on the "real" project.

During weekly meeting, for every part of the project, five of us was split into groups to tackle different problems, and eventually combined the work at the end.

4.3 Programing Style Guide

Even though we are writing our codes on different platform (Linux, Windows, Mac), we try to enforce a uniform coding style for everyone. That is to make sure that we comment on all functions that we defines, and on everything that can improve readability of codes; when editing, we do not delete codes arbitrarily, instead, make it as comments just in case it is needed somehow in some way. Another important rule is to make the codes neat by applying proper indentations.

4.4 Project Timeline

Time	Work
Sep 9 - Sep 16	Language defined

Sep 16 - Sep 25	Writing Proposal
Sep 25 - Oct 2	Proposal modified according to TA's feedback
Oct 9 - Oct 28	Define language detail Write LRM
Oct 28 - Nov 10	Study MicroC and learn Ocaml
Nov 10 - Nov 28	Write Scanner, Parser, and AST
Nov 28 - Dec 15	Semantics, Code generation, C++ code for calculate
Dec 15 - 20	Testing and writing final report

4.5 Roles and Responsibilities

As stated above, we try to get everyone involved in every part of the project, and everyone indeed made contribution to every part more or less. The following shows a more specific overview about who did MAJOR contribution to what part.

Proposal (proposal.pdf)	All of us
Language Manual (LRM.pdf)	All of us
Scanner (scanner.mll)	All of us
AST (ast,ml)	All of us
Parser (parser.mly)	All of us
Semantic Check (semantics.ml)	Wenting Yin, Zhan Shu
Code Generation (codegen.ml)	Kewei Ge, Jinxi Zhao
C++ Codes (calcul2.h)	Junde Huang
Testing (*.cul files)	All of us
Presentation (calcul.ppt)	All of us
Final report (calcul.pdf)	All of us

4.6 Development environment used (tools and languages)

Since our computers run in different operating systems: Mac OS X, Windows 7 and Linux(Ubuntu 12.04), we use different developing tools and environment for our task. We adapt ocamllex for lexical analysis and ocamlyacc for passing, vim and g++ compiler for c++ code part.

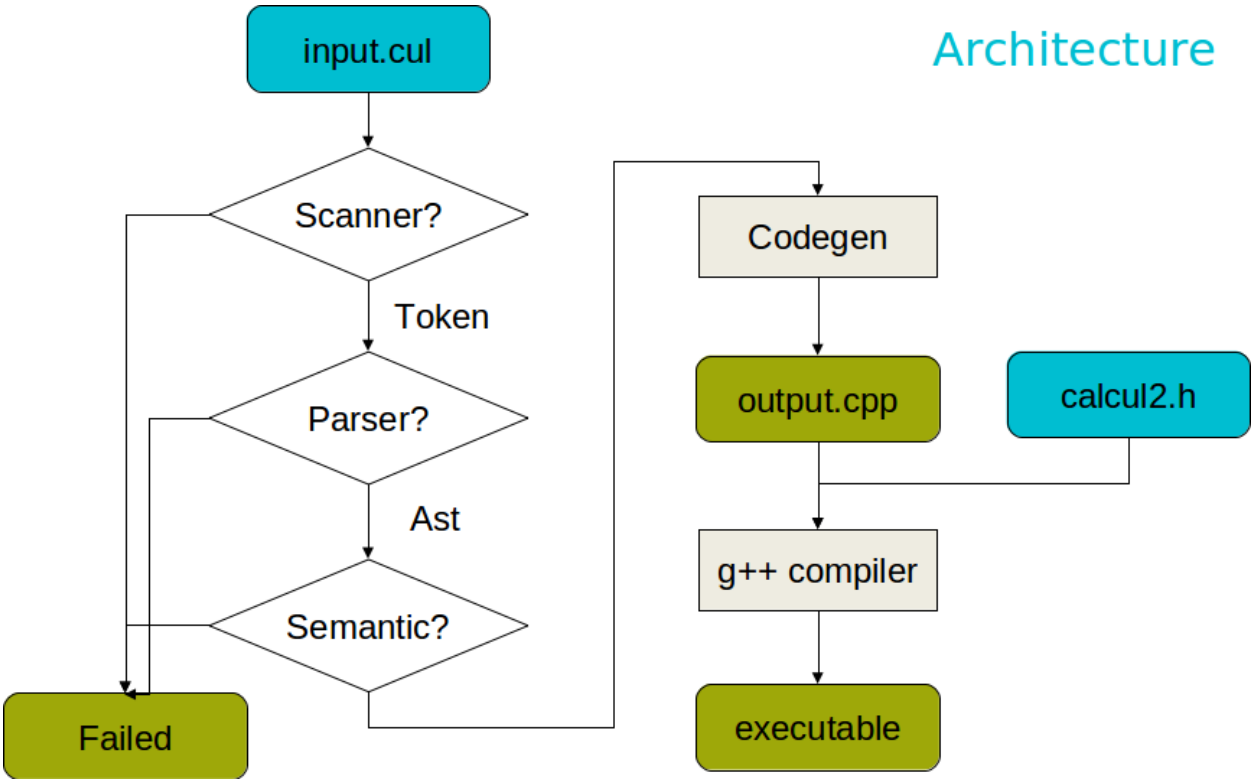
The compiler is written in O'Caml and we write the calcul2.h header file in C++ to implement the functions built in the CalCul^2 program such as the derivation function and integration functions.

In addition, we also utilize other tools and software to make our work organized. We take advantage of Github to share and control different version of codes and get everyone updated with the latest version. Also, we adapt google docs to share documents such as final reports and reading materials among group members. We even established a weixin group to discuss project remotely to help enhance our work efficiency.

4.7 Project Log

As mentioned above, we used Github for version control. We append the commit logs for our project is in appendix (8.13).

5. Architectural Design



Input

The program takes a file with extension .cul as input.

Scanner

The scanner scans the input file and transforms the source code into tokens. Comments are omitted and Illegal characters can be captured. Once an exception is raised, the program stops. This part is written in scanner.mll, and is compiled with ocamllex.
file.

Parser

The parser the tokens generated by scanner to an abstract syntax tree, according to the tree structure defined in ast.ml file. Illegal structure or type, which is not defined in AST, can be captured and parser throws an exception for that, resulting in program failure. This part is written in parser.mly file, and is compiled with ocamlyacc.

AST

Abstract Syntax Tree, it is a kind of hierarchy structure whose nodes are related to

productions. It is the interface between the scanner and parser, and between parser and semantic checker.

Semantic

With the AST, the semantic checker checks the semantic correctness of the program, which is parsed by the parser. It recursively walks over the AST and checks each node. For example, It checks whether a function name has been defined more then once, whether the both sides of a binary operator is legal, whether a variable is used without assigning any value to it, etc.

Code Generation

In code generation phase, it translates the source code that passes semantic check phase to target code, which is C++ code in our case. C++ code is output into file output.cpp.

G++ Compiler

G++ compiler is used to compile c++ codes and generate executable file.

6. Test Plan

Tests are really helpful to find and fix problems of our language. We covered all kinds of possible tests to improve our language. In this part, we will mainly describe three kinds of tests when we were working with our project. They are semantics test, flow control test, and derivation and integral test. Semantics tests are very important because it guarantees the code quality of our language. Flow control is also an important feature of a language so we need to check it as well. Derivation and integral functionality is the core feature of our language, we need to focus on the majority test plans on it in order to gain a robust and complete language.

6.1 Semantics Test

Semantics part is indispensable to a language as it ensures the syntax of the source code. We tested a large amount of semantic tests in our work. Not only we covered the normal semantical checks like cpp does, but also we did some for our calcul^2 style tests.

We checked basic semantics such as no main function declared, duplicate variables, no return values, duplication function names, nonexistent functions, nonexistent variables and many other cases. In addition, we also checked the semantics for math functions, including duplication math function name, nonexistent math function, math function with not matched arguments, derivation and integral not correctly used, and some other test cases.

Here are some semantics test samples:

```
Failure("Duplicate parameter in function add")
add(x,x){
      return x+y;
}

main(){
    x=3;
}


Failure("No Main Function exist!")
add(x,y){
      return x+y;
}


Failure("Function whose name is add has been defined more than once")
add(x,y){
      return x+y;
}
```

```
add(x){
      return x;
}

main(){
    x=3;
}


Failure("Undefined function or math function: add1 is used!")
add(x,y){
      return x+y;
}

main(){
    x=3;
    add1(x,y);
}


exception Failure("Math function has duplicate parameter!")
main(){
    x=3;
    f=1;
    h($y,$y)=y+1;
}



exception Failure("Using math function that parameter not match!")
main(){
    x=3;
    h($x)=x+1;
    g($y)=y+1;
    f($x)=h+g+1;
}


exception Failure("ID : t not valid!")
main(){
    x=3;
    y=t+1;
}
```

```
Failure("function parameter not match!")
add(x,y){
    return x+y;
}

main(){
    x=3;
    y=1+add(3,5,2);
}


Failure("Undefined ID : x!")
main(){
{x=3;}
y=x+1;
}


Failure("Derivation is not properly used!")
main(){
    f($x)=x*x+2;
    :f'(y);
}


Failure("Integral is not properly used!")
main(){
    f($x)=x*x+2;
    :f'(x);
    :f@y(1,3);
}


Failure("Integral is not properly used!")
main(){
    f($x)=x*x+2;
    :f'(x);
    :f@x(1,3,5);
}


Failure("Integral is not properly used!")
main(){
    f($x)=x*x+2;
    :f'(x);
    :h@x(1,3,5);
```

```
}


exception Failure("Using math function that parameter not match!")
main(){
    x=3;
    h($x)=x+1;
    g($x,$y)=x+y+1;
    f($x)=h+g;
}
```

6.2 Flow Control Test

Our Calcul^2 language supports the flow control of ifelse, while and for. They are similar to the flow controls in cpp style. For If-Else Statement, if the boolean expression *bool_expr* holds, then the program will run *Statement 1*. Otherwise the program will run *Statement 2*. The While Statement will keep running *Statement* until the boolean expression *bool_expr* doesn't hold. The For Statement will initially let *val* be *a* and keep running statement until *val* exceeds *b*.

As such, we take several flow control tests for our language. In this case we list the simple greatest common divisor example as below. It will translate the code into the corresponding cpp code.

Here is the gcd example:

```
Calcul^2 source code:
f(x, y) {
    while (x != y) {
        if (x > y) x = x - y;
        else y = y - x;
    }
    return x+y;
}

main()
{
      :f(15, 20);
}


Generated cpp code:
#include <cstdio>
#include "calcul2.h"
using namespace std;
```

```
double f(double x, double y)
{
	while (x != y) {
	  if (x > y)
	      x = x - y;
	  else
	      y = y - x;
	}

	return x + y;

}

int main()
{
	double printer;

	cout << f(15., 20.)<<endl;

	return 0;
}
```

6.3 Derivation and Integral Test

As the core features of our language, we definitely need to test our derivation and integral functionality. In order to do that we need to test different kind of math functions, also to see whether the derivation of them are correct. After that, we also test the integrals of the math functions.

We take the derivation of a math function, and then take the integral of it, so then we should get back the original math function. Notice that our language only supports finite integral so we need to give a range of i to j in get the integral value.

In this example, we test the math function x^x - 1. It is really a difficult function to get derivation even when we use handwriting. But our language can do that. In order to see this, we first output the derivation of it to compare it with the original function. Then we get the integral of the derivation with respect to 1 and 5. Then we compute the difference between the original function with respect to 1 and 5. As the integral of the derivation of a math function should also be itself, we would get the same value.

Here is the math function sample:

Calcul^2 source code:

```
main(){
	f($x)=x^x-1;
	g($x)=f'(x);
	:f;
	:g;
	:f'(x);
	:g@x(1,5);
	:f(5);
}
```

Generated cpp code:

```
#include <cstdio>
#include "calcul2.h"
using namespace std;

int main()
{
	double printer;

	vector<string> f_var;
	f_var.push_back("x");
	vector<double> f_begin, f_end, f_now;
	FTree f(f_var);

	f.AddNode(new FNode(T_OP,0,MINUS));
	f.AddNode(new FNode(T_OP,0,POWER));
	f.AddNode(new FNode(T_VAR,0,0));
	f.AddNode(new FNode(T_VAR,0,0));
	f.AddNode(new FNode(T_VAL,1.));

	vector<string> g_var;
	g_var.push_back("x");
	vector<double> g_begin, g_end, g_now;
	FTree g(g_var);

	g.AddNode(f.Derive("x") -> Copy());

	f.Print();
	cout << "\n";

	g.Print();
	cout << "\n";
```

```
        f_now.clear();
        f.Derive("x") -> Print();
        cout << "\n";

        g_begin.clear();
        g_end.clear();
        g_begin.push_back(1.);
        g_end.push_back(5.);
        cout << g.GetIntegal(g_begin, g_end);
        cout << "\n";

        f_now.clear();
        f_now.push_back(5.);
        printer = f.GetValue(f_now);
        printf("%lf\n",printer);
        cout << "\n";

        return 0;
}
```

7. Lessons Learned

7.1 Junde Huang

Project Lives on Teamwork

The most significant thing is we should work as a team. Everyone of us in the team plays an important role. We need to work together to fix all the problems we have and finish all the codes we need for the project. We can't finish the project with only one people because we also have other classes and projects to finish. But when someone is in a very heavy pressure, the other teammates can help him to reduce some at least in the project.

Plan Limited by Time

We have to say that we plan too much for the project at the very first beginning. We have imagined that we could finish a very big project with all kinds of functions and features that are very amazing, but things come out that we don't have that much time so we have to cut down some of the amazing functions that we hoped to finish. However, we have still finished all the basic project functions, which are also powerful and attractive enough, that we have designed at the beginning in time.

Problems Lessen by Testing

Some problems are very easy to be found at a glance, but others are not. The best way to prevent as most problems as we can is to try more tests on the programs and codes we have written. Only by this way can we find out where the problems are and fix them before it's too late. We have tried plenty of tests at the end of the semester, fixed piles, although not every, problems in the code to make it runs well and output the right thing.

Practical Lively Things

Thanks to Mr. Edward, we have learned so many practical things that we can use not only in the project, but also in the future work. We are so appreciated that all these knowledges are taught in a funny and lively way. It's a pretty cool class!

7.2 Kewei Ge

PLT is one of the most interesting classes I took in Columbia. This is the last semester for me, thanks to plt, never had I ever seen such a simple yet powerful language before. I'm happy to take this class and really learnt much more than a course.

It's a little difficult for students to understand and use ocaml when they are new to it. At first glance, I was full of recursions in my mind, and might get stuck in 10 minutes just staying there, trying to think how the code runs, where it should go, and what would it do just for a few lines code. It seems hard to imagine such a thing could happen to a CS student who is very familiar with C and Java, ha.. However, just at some point, I had a flash of intuition and found I got the soul of ocaml. It made me smarter and the world became much brighter. I guess that's the part of ocaml's charm.

Another important thing I learnt is that always do things before the deadline, the earlier the better. There are only three months in a semester, and we all have finals and projects in the end of the semester. Things would come to mess up if we make everything very late. A team meeting every one week is much helpful to develop our project. A reasonable plan could make a complexity project simpler and easier than we thought.

Meanwhile, I have to say thanks to my team. We are such a fanatic team that I believe our power can make everything. You made me understand that a team of five is much more than five individuals. A problem even it's extremely difficult could be reduced to a simple problem because we are a team. You can hardly imagine what would generate when lots of different ideas come up together. We can even change the world.

Taking this class at the last semester is really lucky to me. Thanks to Prof. Edwards, you made me understand how a compiler works; gave me an opportunity to create a new language; and made me meet with our team. I will always remember this course and the time our team spent together.

7.3 Wenting Yin

I think it is really cool to get a chance to create a new language. During this interesting project, I learned a lot.

Firstly, this project gives me a deep understanding of general programming language implementation system. As a student who does not major in Computer Science during undergraduate study, I find it a bit hard to get the whole picture of how to do our project at first. After taking more courses, talking to my teammates, and as the project goes on, I gradually know what we need to do and why we need do each step: first scan the language, parse it to a tree according to ast, check the semantic error of our language, then generate code into another language to achieve the goal. This project solids my foundation of CS.

Ocaml is an interesting language. Grasping the basic grammar is not easy, since it seems very different from C++ or JAVA that I am familiar with. But after using it for some time, I find it is

easy to write and could do many things with a few codes. The *let in* and *pattern matching* are very powerful when writing compiler.

A challenge that I met during programming is that our language keep the local variable and math function dynamic, that means to let variable visible in right scope and a variable can only be used after definition. MicroC defines all the variable in the head of the function body so the parser could output a local variable list. But our parser do not handle this. To keep the variable list and math function list dynamic, I create a recursive function with the variable list and math function list as the parameter to keep track of the variable list that is now visible. This handle the problem properly.

Another important thing that I learn is teamwork. CS students are very busy and not everyone have the same schedule, we use Github to control version and set proper time to hold the group meeting, these work out well. I feel the power of group work. A problem maybe very hard and costs a lot of time when doing it alone, but after discussing, it could be solved more quickly and better since we collect more ideas. It is very happy to work with my teammates, thanks to them a lot.

7.4 Jinxi Zhao

For me, this course is more than Programming Language and Translators, I also have ocaml to play with. As a student with a bachelor degree of non-CS major, I have to admit that this class might be a little bit too much for me. Though I have some programming experience, I have never touched the field of compilers. Thanks to the amazing lectures by Prof. Edward, I can manage to catch with the course stuffs. However, ocaml is the real pain for me. Even for the homework, I have spent a few hours reading tutorials and materials to do the “small-program” coding assignment. And I also think this is the most difficult part of our project - that was why we spent a few weeks to tackle ocaml before getting on our project. But once I got involved, I found the learning curve for ocaml is pretty steep. Just as Kewei said, he can now do coding with ocaml as he is coding with C++. Ocaml seems to be an efficient tool for developing compiler.

I have to say, I have learned so much from this course and project, seriously. I don’t know what a PLT course supposed to be taught, but Prof. Edwin, you made this course interesting. From scanner to parser, from NFA to SLR, from name scope to types control, flow to code generation to optimization, everything is new to me; but now I am confident with them. Your homework assignment really helped me understand the course materials.

The project gave me a chance to apply everything in class to the real world and helped me understand by exposing me to a deeper level of compiler. When it comes to the project, from my perspective, the reduce/reduce or reduce/shift error is the most challenging part. After we writing

the parser, it just scared me off when I saw tons of reduce/reduce or shift/reduce errors. Sometimes, when you fixes one, more errors raises. For some error, we even have to compromise and revise our language definition to get it over. I can still remember the night when I kept awake until 6am just to debug and fix some “simple and stupid” errors. For the code generation, since our language has two kinds of functions - function and math function. I had a hard time figuring out an efficient way to recognize them and generate proper codes for them. The way I dealt with it is to maintain a global string list to hold math function name, and check the list every time it encounters a function. It is kind of like the semantic checker’s job, but I have to do it again in code generation phase.

Honestly, this is the programming project of my life. And thanks to my phenomenal teammates, they made this memorable experiences. If it weren’t for their hard work and teamwork, we couldn’t have made this; If it weren’t for them, I could have also made it through this project somehow, but I’m sure it wouldn’t be of such fun. It is amazing that you pay for one course, but you get three: PLT, ocaml, and friendship. If there is any advice I can give, that would be getting good teammates, which is the most important thing you need to do at the very beginning of the semester.

7.5 Zhan Shu

This project is about O’Caml which is really a fantastic functional language. Before the PLT course, I have very limited knowledge on functional language. Thus at first glance, it is terrible to look at the syntax of O’Caml, but it is really powerful to let us complete our project with little lines of codes. O’Caml can make us think in a very different way as compared to Java or c. I suppose this kind of thought puts programming in a simple and easy way.

Through the final project, I have gotten a deeper understanding of the general programming language implementation system. I learned a lot about the working principles of scanner, parser, semantic analyzer and code generator. During the course, I just get to know the concept of the basic components of compiler. But with the project, I know, in practice, how the scanner transforms program elements into a stream of tokens, and how the parser will transform the tokens into AST, and how the semantic analyzer checks the validity of the AST, as well as how the Code Generator transforms the AST into executable codes. In a nutshell, I learned what is the fundamental process to develop a new language.

Finally, working with great team members is the most passionate and fortunate thing in the world. I’m so excited to work with my talented teammates. We actively set up meetings to discuss our project, we efficiently fixed the problems in front of us together. We learned to use github to share and communicate code. We not only developed the project together, but also had fun together.

8. Appendix

8.1 scanner.mll

```
{ open Parser }

let digit = ['0'-'9']

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "#*"     { comment lexbuf }           (* Comments *)
| "##"     { comline lexbuf }           (* Comments in Line*)
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '^'      { POWER }

(*
| "//"     { INTDIVIDE }
| '%'      { MOD }
*)

| '='      { ASSIGN }
| '\''      { DERIV }
| '@'      { INTEG }
| "sqrt"   { SQRT }
| "sin"    { SIN }
| "cos"    { COS }
| "tan"    { TAN }
| "asin"   { ASIN }
| "acos"   { ACOS }
| "atan"   { ATAN }
| "log"    { LOG }
| "ln"     { LN }

| "=="     { EQ }
| "!="     { NEQ }
```

```
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "&&"     { AND }
| "||"     { OR }
| '!'      { NOT}

| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
(*
| '['      { LVEC }
| ']'      { RVEC }
*)
| ';'      { SEMI }
| ':'      { OUTPUT }
| ','      { COMMA }
| '$'      { DOLLAR }

| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }


| digit+ as lxm { REAL(float_of_string lxm) }


| (digit+'.'digit*)('e'['+''-']?digit+)? | digit+'e'['+''-']?digit+ as lxm
{ REAL(float_of_string lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }
```

```
and comment = parse
  "*#" { token lexbuf }
| _    { comment lexbuf }

and comline = parse
  '\n' { token lexbuf }
| _    { comline lexbuf }
```

8.2 ast.ml

```
(* binary operators *)
type op = Add | Sub | Mult | Div | Pow | Deriv | Integ | Eq | Neq | Less | Leq |
Greater | Geq | And | Or

(* unary operators *)
type preop = Sqrt | Sin | Cos | Tan | ASin | ACos | ATan | Log | Ln | Not

(* expressions *)
type expr =
    Real of float
  | Id of string
  | Binop of expr * op * expr
  | PreUnaop of preop * expr
  | Assign of string * expr
  | Call of string * expr list
  | Noexpr

(* statements *)
type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt
  | Output of expr
```

```
  | Math_func of string * string list * expr

(* declaration for functions *)
type func_decl = {
    fname : string;
    formals : string list;
    body : stmt list;
}

(* definition for type of program *)
type program = func_decl list
```

8.3 parser.mly

```
%{
open Printf
open Ast
%}

%token PLUS MINUS TIMES DIVIDE POWER
%token ASSIGN DERIV INTEG SQRT SIN COS TAN ASIN ACOS ATAN LOG LN
%token EQ NEQ LT LEQ GT GEQ AND OR NOT
%token LPAREN RPAREN LBRACE RBRACE SEMI OUTPUT COMMA
%token IF ELSE FOR WHILE
%token RETURN OUTPUT

%token DOLLAR
%token <float> REAL
%token <string> ID
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right OUTPUT
%right ASSIGN
%left EQ NEQ
%left LT LEQ GT GEQ AND OR
```

```
%left PLUS MINUS
%left TIMES DIVIDE
%right POWER DERIV INTEG SQRT SIN COS TAN ASIN ACOS ATAN LOG LN NOT DOLLAR

%start program
%type <Ast.program> program

%%

program:
                { [] }
 | fdecl program { $1 :: $2 }

/*           { [], [] }
 | program vdecl { ($2 :: fst $1), snd $1 }
 | program math_fdecl { ($2 :: fst $1), snd $1 }
*/

/* Function declaration:
foo(x, y) {z=3;...}
or
foo(x, y)=x+y;
*/

fdecl:
   ID LPAREN formals_opt RPAREN LBRACE stmt_list RBRACE
     { { fname = $1;
       formals = $3;
       body = List.rev $6 } }

formals_opt:
    /* nothing */  { [] }
  | formal_list   { List.rev $1 }


formal_list:
```

```
    ID                  { [$1] }
  | formal_list COMMA ID { $3 :: $1 }


actuals_opt:
        /* nothing */        { [] }
  | actuals_list { List.rev $1 }

actuals_list:
     expr                                { [$1] }
  |actuals_list COMMA expr { $3 :: $1 }


stmt_list:
    /* nothing */  { [] }
  | stmt_list stmt { $2 :: $1 }

stmt:
    expr SEMI { Expr($1) }
  | RETURN expr SEMI { Return($2) }
  | LBRACE stmt_list RBRACE { Block(List.rev $2) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7) }
  | FOR LPAREN expr_opt SEMI expr_opt SEMI expr_opt RPAREN stmt
     { For($3, $5, $7, $9) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5) }
  | OUTPUT expr SEMI{ Output($2) }
  | ID LPAREN var_list RPAREN ASSIGN expr SEMI { Math_func($1, List.rev $3, $6) }


var_list:
        DOLLAR ID                       { [$2] }
  | var_list COMMA DOLLAR ID                { $4 :: $1 }


expr_opt:
```

```
    /* nothing */ { Noexpr }
  | expr          { $1 }

expr:
    expr PLUS   expr { Binop($1, Add,   $3) }
  | expr MINUS  expr { Binop($1, Sub,   $3) }
  | expr TIMES  expr { Binop($1, Mult,  $3) }
  | expr DIVIDE expr { Binop($1, Div,   $3) }
  | expr POWER  expr { Binop($1, Pow,   $3) }
  | expr EQ     expr { Binop($1, Eq, $3) }
  | expr NEQ    expr { Binop($1, Neq,   $3) }
  | expr LT     expr { Binop($1, Less,  $3) }
  | expr LEQ    expr { Binop($1, Leq,   $3) }
  | expr GT     expr { Binop($1, Greater,  $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3) }
  | expr AND    expr { Binop($1, And,   $3) }
  | expr OR     expr { Binop($1, Or,    $3) }

  | expr DERIV  expr { Binop($1, Deriv,   $3) }
  | expr INTEG  expr { Binop($1, Integ,   $3) }

  | SQRT expr        {PreUnaop(Sqrt,$2)}
  | SIN expr         {PreUnaop(Sin,$2)}
  | COS expr         {PreUnaop(Cos,$2)}
  | TAN expr         {PreUnaop(Tan,$2)}
  | ASIN expr        {PreUnaop(ASin,$2)}
  | ACOS expr        {PreUnaop(ACos,$2)}
  | ATAN expr        {PreUnaop(ATan,$2)}
  | LOG expr         {PreUnaop(Log,$2)}
  | LN expr          {PreUnaop(Ln,$2)}
  | NOT expr         {PreUnaop(Not,$2)}

  | REAL             {Real($1)}
  | ID               {Id($1)}

  | ID ASSIGN expr   {Assign($1,$3)}
```

```
  | ID LPAREN actuals_opt RPAREN {Call($1,$3)}
  | LPAREN expr RPAREN {$2}
```

8.4 semantics.ml

```
open Ast

(* is to be modified according to the AST*)

type env = {
        mutable functions : func_decl list;
}

(* a function to test whether a function's name is equal to a string 'name'*)
let func_equal_name name = function
        | func -> func.fname = name

(* a function to check whether a function's name has been defined more than once*)
let fun_exist func env =
        let name = func.fname in
         try
                              let _ = List.find (func_equal_name name) env.functions
in
                                      let e = "Function whose name "^ name ^" has been
defined more than once" in
                                                raise (Failure e)
                         with Not_found -> false

(*a function to check whether a function's name exist in the env*)
let exist_func_name name env = List.exists (func_equal_name name) env.functions

(*a function to return the function object if you give its name*)
let get_func_by_name name env =
        try
               let result = List.find (func_equal_name name) env.functions in
                                result
  with Not_found -> raise(Failure("Function "^ name ^ " has not been declared!"))
```

```
(*a function to check whether a function has a parameter appears more than once*)
let count_fpara func = function a
  -> let f count b =
              if b=a then count+1
                        else count
                        in
                           let count = List.fold_left f 0 func.formals in
                              if count > 1
                                        then raise(Failure("Duplicate parameter in
function " ^ func.fname))
                                        else
                                                count

let check_fpara_duplicate func =
        List.map (count_fpara func) func.formals


(* a function to check whether there is a main function*)
let exists_main env =
        if exist_func_name "main" env
         then true else raise(Failure("No Main Function exist!"))

(*a function to check whether a id is in a list*)
let exist_id id id_list= List.exists (function x -> x = id) id_list

(*a function to check whether a fname is in a list*)
let exist_mathf fname mathf_list = List.exists (function ( a, _) -> a = fname)
mathf_list


(*a function to get math function given the math function name*)
let get_math_fun_by_name fname mathf_list =
        try
               let result = List.find (function ( a, _) -> a = fname) mathf_list  in
```

```
                                result
        with Not_found -> raise(Failure("Math function "^ fname ^ " has not been
declared!"))

(*a function to check whether a math function name is valid*)
let check_math_func_name_valid mfname mathf_list id_list funcformal env=
        if (exist_mathf mfname mathf_list)||(exist_id mfname
id_list)||(exist_func_name mfname env)||(exist_id mfname funcformal)
        then raise(Failure("Math function name: "^ mfname ^ " has been used!"))
        else true

(*a function to check whether a math function's parameter are valid*)
let rec check_math_func_para_valid mfname mathf_list id_list  funcformal env =
function
        [] -> true
        |hd::tl ->
           if (exist_mathf hd mathf_list) then raise(Failure("Math function parameter
not valid!"))
           else if (exist_id hd id_list) then raise(Failure("Math function parameter
not valid!"))
           else if (exist_func_name hd env) then raise(Failure("Math function
parameter not valid!"))
            else if (exist_id hd funcformal) then raise(Failure("Math function
parameter not valid!"))

           else check_math_func_para_valid mfname mathf_list id_list funcformal env
tl

(*a function to check whether a math function has a parameter appears more than
once*)
let count_para paralist = function
  b -> let f count c = if c=b then count+1 else count
        in
           let count = List.fold_left f 0 paralist in
                           if count > 1
```

```
                                        then raise(Failure("Math function has
Duplicate parameter!"))
                                        else
                                                count

let check_math_para_duplicate paralist =
        List.map (count_para paralist) paralist

(*a function to check whether a id name is valid when defined*)
let check_id_valid id mathf_list funcformal env=
        if (exist_mathf id mathf_list)||(exist_func_name id env)||(exist_id id
funcformal) then raise(Failure("Math function name: "^ id ^ " has been used!"))
        else true


(*a function to check whether a expr is valid: uses local variables and functions and
math functions that has been defined, functions and math functions have parameter
that matches*)
let rec valid_expr funcformal expr id_list mathf_list env=
         match expr with
           |Assign(id, e1) -> if exist_id id id_list
                              then let _ =  valid_expr funcformal e1 id_list
mathf_list env in id_list
                                else if (check_id_valid id mathf_list funcformal env)
then let _ =  valid_expr funcformal e1 id_list mathf_list env in let id_list= id ::
id_list in id_list
                                else raise (Failure("id has been used!"))
             |Call(fname,exprlist) -> if exist_func_name fname env

                                      then let f1 = get_func_by_name fname env in
                                              if (List.length f1.formals ==
List.length exprlist)
                                              then let _ = List.map (fun e ->
valid_expr funcformal e id_list mathf_list env) exprlist in
                                              id_list
```

```
                                        else raise(Failure("function parameter
not match!"))
                                   else if exist_mathf fname mathf_list
                                        then let (m1,p1)=get_math_fun_by_name
fname mathf_list    in
                                         if (List.length p1 == List.length
exprlist)
                                             then  let _ = List.map (fun e ->
valid_expr funcformal e id_list mathf_list env) exprlist in

                                             id_list
                                             else  raise(Failure("math function
parameter not match!"))

                                        else raise(Failure("Undefined function or
math function: "^ fname ^ " is used!"))
            |Id(id)  -> if exist_id id id_list
                        then id_list
                          else raise(Failure("ID : "^ id ^ " not valid!"))

            |Binop (e1,o,e2)->
                        ( match o with
                        Deriv -> ( match e1 with
                                   Id(id1)  ->  if exist_mathf id1 mathf_list
then
                                                              ( match e2 with
                                                                        Id(id2)
->  let (m1,p1)= get_math_fun_by_name id1 mathf_list in

if exist_id id2 p1 then id_list

else raise(Failure("Derivation is not properly used!"))

                                                                      | _ ->
raise(Failure("Derivation is not properly used!"))
                                                              )
```

```
                                                    else  raise(Failure("Derivation
is not properly used!"))

                                          | _ -> raise(Failure("Derivation is not
properly used!"))
                                         )

                              |Integ -> ( match e1 with
                                          Id(id1)  ->  if exist_mathf id1 mathf_list
then
                                                                          ( match e2 with

Call(a,elist)  ->  let (m1,p1)= get_math_fun_by_name id1 mathf_list in

if  exist_id a p1 &&  List.length elist=2 then id_list

else raise(Failure("Derivation is not properly used!"))

                                                                                | _ ->
raise(Failure("Derivation is not properly used!"))

                                                                                )

                                                    else  raise(Failure("Derivation
is not properly used!"))

                                          | _ -> raise(Failure("Derivation is not
properly used!")) )


                              |_ -> let _ =  valid_expr funcformal e1 id_list
mathf_list env in
                                      let _ =valid_expr funcformal e2 id_list
mathf_list env in
                                         id_list
```

```
                            )
            | PreUnaop(o,e2)-> let _ =  valid_expr funcformal e2 id_list mathf_list
env in id_list

            |_ -> id_list


(*a function to check whether the body of math funtion is valid*)
let rec check_math_func_body_valid mfname paralist id_list mathf_list env expr=
            match expr with
                |Id(id)  -> if (exist_id id id_list || exist_id id paralist)
                            then true
                               else if exist_mathf id mathf_list
                                    then let (m1,p1)= get_math_fun_by_name id
mathf_list in
                                            if (p1 = paralist)
                                            then true
                                            else raise(Failure("Using math function
that parameter not match!"))

                                    else raise(Failure("Undefined ID : "^ id ^ "is
used!"))

                |Call(fname,exprlist) -> if exist_func_name fname env

                                     then let f1 = get_func_by_name fname env in
                                             if (List.length f1.formals ==
List.length exprlist)
                                             then let _ = List.map (fun e ->
check_math_func_body_valid mfname paralist id_list mathf_list env e) exprlist in
                                             true
                                             else raise(Failure("function parameter
not match!"))
                                     else if exist_mathf fname mathf_list
                                          then let (m1,p1)= get_math_fun_by_name
fname mathf_list in
```

```
                                                    if (List.length p1 == List.length
exprlist)
                                                    then  let _ = List.map (fun e -
>check_math_func_body_valid mfname paralist id_list mathf_list env e) exprlist in

                                                    true
                                                    else  raise(Failure("math function
parameter not match!"))

                                            else raise(Failure("Undefined function or
math function: "^ fname ^ "is used!"))

                | Binop (e1,_,e2)-> let _ = check_math_func_body_valid mfname
paralist id_list mathf_list env e1 in
                                        let _ = check_math_func_body_valid mfname
paralist id_list mathf_list env e2 in true
                | PreUnaop(e1,e2)-> true
                |_ -> true


(*a function to check whether the expr in output is valid*)
let rec check_output_valid paralist id_list mathf_list env expr=
             match expr with
                |Id(id)  -> if (exist_id id id_list || exist_id id paralist ||
exist_mathf id mathf_list)

                          then true
                            else raise(Failure("Invalid ID : "^ id ^ " in return!"))

                |Call(fname,exprlist) -> if exist_func_name fname env

                                        then let f1 = get_func_by_name fname env in
                                                if (List.length f1.formals ==
List.length exprlist)
                                                then let _ = List.map (fun e ->
check_output_valid paralist id_list mathf_list env e) exprlist in
```

```
                                        true
                                        else raise(Failure("function parameter
not match!"))
                                  else if exist_mathf fname mathf_list
                                       then let (m1,p1)= get_math_fun_by_name
fname mathf_list in
                                            if (List.length p1 == List.length
exprlist)
                                            then  let _ = List.map (fun e -
>check_output_valid paralist id_list mathf_list env e) exprlist in

                                            true
                                            else  raise(Failure("math function
parameter not match!"))

                                       else raise(Failure("Undefined function or
math function: "^ fname ^ "is used!"))

                | Binop (e1,o, e2)->
                        ( match o with
                           Deriv -> ( match e1 with
                                      Id(id1)  ->  if exist_mathf id1 mathf_list
then
                                                              ( match e2 with
                                                                        Id(id2)
->  let (m1,p1)= get_math_fun_by_name id1 mathf_list in

if exist_id id2 p1 then true
else raise(Failure("Derivation is not properly used!"))

                                                                      | _ ->
raise(Failure("Derivation is not properly used!"))
                                                              )

                                            else  raise(Failure("Derivation
is not properly used!"))
```

```
                                        | _ -> raise(Failure("Derivation is not
properly used!"))
                                      )

                              |Integ -> ( match e1 with
                                         Id(id1)  ->  if exist_mathf id1 mathf_list
then
                                                                    ( match e2 with

Call(a,elist)  ->  let (m1,p1)= get_math_fun_by_name id1 mathf_list in

if  exist_id a p1 &&  List.length elist=2 then true

else raise(Failure("Integral is not properly used!"))

                                                                                | _ ->
raise(Failure("Integral is not properly used!"))

                                                                                )

                                                        else  raise(Failure("Integral is
not properly used!"))

                                        | _ -> raise(Failure("Integral is not properly
used!")) )


                              |_ -> let _ = check_output_valid paralist id_list
mathf_list env e1 in
                                     let _ = check_output_valid paralist id_list
mathf_list env e2 in true

                              )
```

```
                | PreUnaop(e1,e2)-> true
                |_ -> true



(*a function to check whether the body of a func is valid*)
let check_func_body_valid func env=
     let stmt_lt = func.body in
       let funcformal= func.formals in
        let rec f id_list mathf_list stmt_list =
                if (List.length stmt_list == 0) then true
                else let stmt_tl= List.tl stmt_list in
                        match List.hd stmt_list  with
                     |Expr(expr) -> let idl=valid_expr funcformal expr id_list
mathf_list env in f idl mathf_list stmt_tl

                         |Math_func(mfname, paralist ,expr) ->let _=
check_math_func_name_valid mfname mathf_list id_list funcformal env in
                                                                (* let _=
check_math_func_para_valid mfname  mathf_list id_list funcformal env paralist in*)
                                                                  let
_=check_math_para_duplicate paralist in
                                                                    let _=
check_math_func_body_valid mfname paralist id_list mathf_list env expr in
                                                                       let
mathf_list=(mfname, paralist)::mathf_list
                                                                           in f
id_list mathf_list stmt_tl
                         |Return(expr)  -> let idl=valid_expr funcformal expr
id_list mathf_list env in f idl mathf_list stmt_tl

                         |If(expr,stmt1,stmt2)  -> let idl=valid_expr funcformal
expr id_list mathf_list env in
                                                        (f idl mathf_list [stmt1;stmt1])
&& (f idl mathf_list [stmt2;stmt2]) && (f id_list mathf_list stmt_tl)
```

```
                        |For(expr1,expr2,expr3,stmt) -> let idl1=valid_expr
funcformal expr1 id_list mathf_list env in
                                                        let idl2=valid_expr
funcformal expr2 idl1 mathf_list env in
                                                          let idl3=valid_expr
funcformal expr3 idl2 mathf_list env in
                                                                    (f idl3
mathf_list [stmt;stmt]) && (f id_list mathf_list stmt_tl)

                        |While(expr,stmt)  -> let idl = valid_expr funcformal expr
id_list mathf_list env in (f idl mathf_list [stmt;stmt]) && (f id_list mathf_list
stmt_tl)

                        |Output(expr)  -> let _=check_output_valid funcformal
id_list mathf_list env expr in f id_list mathf_list stmt_tl

                        |Block(stmtlist) -> (f id_list mathf_list stmtlist) && (f
id_list mathf_list stmt_tl)

         in f func.formals [] stmt_lt




(*a function to check each function's validity*)
let check_func f env =
                let _dup_name = fun_exist f env in
                   let _ = env.functions <- (f) ::env.functions in
                   let _dup_formals = check_fpara_duplicate f in
                                  let _vbody = check_func_body_valid f env in
                                       true


(*The final function to check the program*)
let check_program fun_list =
        let env = {functions = [];} in
```

```
        let _dovalidation = List.map (fun f -> check_func f env) fun_list in
            let  _mainexist = exists_main env in
                let _ = print_endline "\nThe semantic check has been finished!\n"
in
                        true
```

8.5 codegen.ml

```
open Ast
open Printf

(* generate the return type of a function *)
let gen_type = function
    "main" -> "int"
  | _ -> "double"

(* variable list inside a function *)
let var_list = ref []

(* math function list inside a function *)
let mathf_list = ref []

let exist_id id id_list= List.exists (function x -> x = id) !id_list

let gen_var var =
        match (exist_id var var_list) with
        true -> var
  | false -> ignore(var_list := var :: !var_list); "double "^var

(* generate the cpp code for expression *)
let rec gen_expr = function
    Real(l) -> string_of_float l
  | Id(s) -> gen_var s
  | PreUnaop(preop, e) ->
          "" ^
          (match preop with
```

```
          Sqrt -> "sqrt" | Sin -> "sin" | Cos -> "cos" | Tan -> "tan" | ASin ->
"asin"
          | ACos -> "acos" | ATan -> "atan" | Log -> "log" | Ln -> "ln" | Not -> "!")
          ^ "(" ^ gen_expr e ^ ")"
  | Binop(e1, o, e2) ->
      (match o with
        Pow -> "pow(" ^ gen_expr e1 ^ ", " ^ gen_expr e2 ^ ")"
      | op -> gen_expr e1 ^ " " ^
              (match op with
            Add -> "+" | Sub -> "-" | Mult -> "*" | Div -> "/"
          | And -> "&&" | Or -> "||" | Eq -> "==" | Neq -> "!="
              | Less -> "<" | Leq -> "<=" | Greater -> ">" | Geq -> ">="
          | _ -> ""
          (*| Pow -> "pow" | Deriv -> "'" | Integ -> "@" *)   (*Pow and Deriv and
Integ to be determined*)

          ) ^ " " ^ gen_expr e2
           )
  | Assign(v, e) -> gen_var v^ " = " ^ gen_expr e
  | Call(fname, el) -> fname ^ "(" ^ String.concat ", " (List.map gen_expr el) ^ ")"
  | Noexpr -> ""

(* help to generate f.push_back("unknowns") *)
let rec pad (unks, fname) =
  match unks with
    [] -> []
  | u :: tl -> (fname^"_var.push_back(\""^u^"\");") :: (pad (tl, fname))

(* get the index of a given unknown *)
let rec get_index unk = function
    [] -> 0
  | h :: t -> if h = unk then 0 else 1 + get_index unk t

(* pad the new node code for a given function with declarations *)
let new_node fname decls =
  fname ^ ".AddNode(new FNode(" ^ decls ^ "));\n\t"
```

```
(* pad the add node code for a given function with declarations *)
let add_node fname decls =
  fname ^ ".AddNode(" ^ decls ^ ");\n\t"

(* generate the math arguments *)
let gen_math_args = function
    Id(s) -> s
  | Real(l) -> string_of_float l
  | _ -> ""

(* recursion function to generate the tree operator structures *)
let rec gen_tree_op (fname, unknowns) = function
    Real(l) -> new_node fname ("T_VAL,"^(string_of_float l))
  | Id(s) ->
    ( match (exist_id s mathf_list) with
        true -> add_node fname (s^".Copy()")
      | false -> new_node fname ("T_VAR,0,"^(string_of_int (get_index s unknowns)))
    )
  | Binop(e1, o, e2) ->
        ( match o with
            Add -> new_node fname "T_OP,0,PLUS"
          | Sub -> new_node fname "T_OP,0,MINUS"
          | Mult -> new_node fname "T_OP,0,TIMES"
          | Div -> new_node fname "T_OP,0,DIVIDE"
          | And -> new_node fname "T_OP,0,AND"
          | Or -> new_node fname "T_OP,0,OR"
          | Eq -> new_node fname "T_OP,0,EQUAL"
          | Neq -> new_node fname "T_OP,0,NOTEQUAL"
          | Less -> new_node fname "T_OP,0,LESS"
          | Leq -> new_node fname "T_OP,0,LESSEQUAL"
          | Greater -> new_node fname "T_OP,0,GREATER"
          | Geq -> new_node fname "T_OP,0,GREATEREQUAL"
          | Pow -> new_node fname "T_OP,0,POWER"
          | Deriv -> add_node fname (gen_math_args e1 ^ "." ^ "Derive(\""
                     ^ gen_math_args e2 ^ "\") -> Copy()")
```

```
          | _ -> ""
          (*| Integ -> new_node fname "T_OP,0,INTEG"*)
        )
        ^
        ( match o with
            Deriv -> ""
          | _ -> gen_tree_op (fname, unknowns) e1
            ^ gen_tree_op (fname, unknowns) e2
        )
  | PreUnaop(preop, e) ->
        ( match preop with
            Sqrt -> new_node fname "T_OP,0,SQRT"
          | Sin -> new_node fname "T_OP,0,SIN"
          | Cos -> new_node fname "T_OP,0,COS"
          | Tan -> new_node fname "T_OP,0,TAN"
          | ASin -> new_node fname "T_OP,0,ASIN"
          | ACos -> new_node fname "T_OP,0,ACOS"
          | ATan -> new_node fname "T_OP,0,ATAN"
          | Log -> new_node fname "T_OP,0,LOG"
          | Ln -> new_node fname "T_OP,0,LN"
          | Not -> new_node fname "T_OP,0,NOT"
        )
        ^ gen_tree_op (fname, unknowns) e
  | Noexpr -> ""
  | _ -> ""

(* construct the tree structure *)
let rec construct_tree (fname, unknowns, formula) =
  gen_tree_op (fname, unknowns) formula

(* add the function name and tree declarations *)
let add_fname (fname, unknowns) =
  let _ = match (exist_id fname mathf_list) with
      false -> ignore(mathf_list := fname :: !mathf_list); ""
    | _ -> "" in
  let dcl_fname = "vector<string> "^fname^"_var;\n\t" in
  let pbk_fname = String.concat "\n\t" (pad (unknowns, fname)) in
```

```
  let dcl_fval = "vector<double> "^fname^"_begin, "^fname^"_end, "^fname^"_now;\n\t"
in
  let dcl_tree = "FTree "^fname^"("^fname^"_var);\n\t" in
    dcl_fname ^ pbk_fname ^ "\n\t" ^ dcl_fval ^ dcl_tree ^ "\n\t"

(* construct the tree *)
let add_tree (fname, unknowns, formula) =
  let cst_tree = construct_tree (fname, unknowns, formula) in
    cst_tree

(* pad the cpp code for push_back *)
let rec pad_call (el, fname) =
  match el with
    [] -> []
  | u :: tl -> (fname^"_now.push_back(\""^u^"\");") :: (pad_call (tl, fname))

(* generate the cpp code for getting values of a function *)
let rec gen_value fname = function
    s -> fname^"_now.push_back("^gen_expr s^");\n\t"
  | _ -> ""
(*
   Real(l) -> fname^"_now.push_back("^(string_of_float l)^");\n\t"
 | Id(s) -> fname^"_now.push_back("^s^");\n\t"
 | _ -> ""
*)
(* generate the cpp code for setting the integal arguments *)
let rec gen_begin_end (fname, el) =
    fname ^ "_begin.clear();\n\t" ^ fname ^ "_end.clear();\n\t" ^
    (match el with
      a :: b :: _ -> fname ^ "_begin.push_back(" ^ gen_math_args a ^ ");\n\t" ^
                     fname ^ "_end.push_back(" ^ gen_math_args b ^ ");\n\t"
    | _ -> "")

(* generate the cpp code for setting the integal arguments *)
let gen_integ fname = function
      Call(unk, el) -> gen_begin_end (fname, el)
```

```
    | _ -> ""

(* generate the cpp code for getting the integal value *)
let get_integ fname =
    fname ^ ".GetIntegal(" ^ fname ^ "_begin, " ^ fname ^ "_end);\n\t"

(* handles a call expression *)
let rec gen_call_func = function
    Call(fname, el) as _func -> if exist_id fname mathf_list then (
        fname ^ "_now.clear();\n\t" ^ String.concat "" (List.map (gen_value fname)
el)
      ^ "printer = " ^ fname ^ ".GetValue(" ^ fname ^ "_now);\n\t"
      ^ "printf(\"%lf\\n\",printer);\n\t" ^ "cout << \"\\n\";\n\t")
                                else "cout << " ^ gen_expr _func ^ "<<endl;\n\t"
  | Binop(e1, o, e2) as _expr ->
        (  match o with
            Deriv -> gen_math_args e1 ^ "_now.clear();\n\t" ^ gen_math_args e1 ^ "."
^
                     "Derive(\"" ^ gen_math_args e2 ^ "\") -> Print();\n\t"
          | Integ -> gen_integ (gen_math_args e1) e2 ^ "cout << " ^ get_integ
(gen_math_args e1)
          | _ -> "cout << " ^ gen_expr _expr ^ ";\n\t"
        ) ^ "cout << \"\\n\";\n\t"
  | Id(s) ->
    (  match (exist_id s mathf_list) with
        true -> s ^ ".Print();\n\t"
      | false -> "cout << " ^ s ^ ";\n\t"
    ) ^ "cout << \"\\n\";\n\t"
  | _ -> ""


(* generate the math declaration and create tree structure *)
let gen_math (fname, unknowns, formula)=
  let addfname = add_fname (fname, unknowns) in
  let addtree = add_tree (fname, unknowns, formula) in
    addfname ^ addtree
```

```
        (*fname ^ "(" ^ String.concat ", " unknowns ^ "){\n\t\t" ^ (gen_expr formula)
^ "\n\t}\n"*)

(* generate the statement *)
let rec gen_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map gen_stmt stmts) ^ "}\n"
  | Expr(expr) -> gen_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ gen_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ gen_expr e ^ ")\n" ^ gen_stmt s
  | If(e, s1, s2) ->  "if (" ^ gen_expr e ^ ")\n" ^
      gen_stmt s1 ^ "else\n" ^ gen_stmt s2
  | For(e1, e2, e3, s) ->
      "for (" ^ gen_expr e1  ^ " ; " ^ gen_expr e2 ^ " ; " ^
      gen_expr e3  ^ ") " ^ gen_stmt s
  | While(e, s) -> "while (" ^ gen_expr e ^ ") " ^ gen_stmt s

  | Output(e) -> gen_call_func e
  | Math_func(s, sl, e) -> gen_math (s, sl, e)

(* generate the cpp code for formals, if it apears the first time, then give the type
definition *)
let gen_formals formals = ignore(List.map (fun(x)->var_list := x :: !var_list)
formals); List.map (fun(l) -> "double "^l) formals

(* generate the cpp code for function declaration *)
let gen_fdecl fdecl =
  let ftype = gen_type (fdecl.fname) in
  let fname = fdecl.fname in
  let formal_list = String.concat ", " (gen_formals fdecl.formals) in
  let body = String.concat "\n\t" (List.map gen_stmt fdecl.body) in
    ignore(var_list := []);ignore(mathf_list := []);
    match fname with
        "main" -> ftype^" "^fname^"("^formal_list^")\n{\n\tdouble
printer;\n\n\t"^body^"\n\treturn 0;\n}\n"
      | _ -> ftype^" "^fname^"("^formal_list^")\n{\n\t"^body^"\n}\n"
```

```
(* entrence of codegen, generate the cpp code for our calcul2 code *)
let gen_program prog =
  let header = "#include <cstdio>\n#include \"calcul2.h\"\nusing namespace std;\n" in
    let fdecls = String.concat "\n" (List.map gen_fdecl prog) in
      let _ = print_endline "Codegen completed.\nCompiling..\n" in
        header^"\n"^fdecls
```

8.6 compiler.ml

```
open Printf
        let _ =
        let lexbuf = Lexing.from_channel stdin in
        let parse_prog = Parser.program Scanner.token lexbuf in
        let _ = Semantics.check_program parse_prog in
        let prog = Codegen.gen_program parse_prog in
        let output_cpp = open_out "output.cpp" in
        output_string output_cpp prog
```

8.7 calcul2.h

```
#include<cmath>
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<iostream>
#include<vector>
#include<string>
using namespace std;

#define T_VAL        0
#define T_VAR        1
#define T_OP        2

#define PLUS        1
#define MINUS        2
#define TIMES        3
#define DIVIDE        4
```

```
#define POWER        5
#define SQRT        8
#define SIN        9
#define COS        10
#define TAN        11
#define ASIN        12
#define ACOS        13
#define ATAN        14
#define LOG        15
#define LN        16

#define TOT        1e5

int sgn(double v){
        if(fabs(v)<1e-8)return 0;
        return v>0?1:-1;
}

void PrintOp(int op){
        switch(op){
                case PLUS:
                        printf("+");break;
                case MINUS:
                        printf("-");break;
                case TIMES:
                        printf("*");break;
                case DIVIDE:
                        printf("/");break;
                case POWER:
                        printf("^");break;
                case SQRT:
                        printf("sqrt");break;
                case SIN:
                        printf("sin");break;
                case COS:
                        printf("cos");break;
```

```
                case TAN:
                        printf("tan");break;
                case ASIN:
                        printf("asin");break;
                case ACOS:
                        printf("acos");break;
                case ATAN:
                        printf("atan");break;
                case LOG:
                        printf("log");break;
                case LN:
                        printf("ln");break;
                default:
                        break;
        }
}

class FNode{
private:
        int type;
        double value;
        int op;
        FNode *left, *right;
public:
        FNode(int _type, double _value, int _op, FNode *_left, FNode *_right);
        FNode *Copy();
        bool IsSame(FNode *tre);
        FNode* GetLeft();
        FNode* GetRight();
        void Destroy();
        void SetLeft(FNode*_left);
        void SetRight(FNode*_right);
        void Print(int LastOp, vector<string>variable);
        double GetValue(vector<double>_var);
        void Reduce();
        FNode*Derive(int _var);
        bool AddNode(FNode*node);
```

```
};

FNode::FNode(int _type=0, double _value=0, int _op=0, FNode *_left=NULL,
FNode*_right=NULL):type(_type),
        value(_value),op(_op),left(_left),right(_right){}

FNode *FNode::Copy(){
        FNode *tmp=new FNode(type,value,op);
        if(left!=NULL)tmp->left=left->Copy();
        if(right!=NULL)tmp->right=right->Copy();
        return tmp;
}

bool FNode::IsSame(FNode *tre){
        if((left==NULL) ^ (tre->left==NULL))return false;
        if((right==NULL)^(tre->right==NULL))return false;
        if(type!=tre->type)return false;
        if(type==T_VAL)return(sgn(value-tre->value)==0);
        if(type==T_VAR)return(op==tre->op);
        if(op!=tre->op)return false;
        if(left!=NULL && !left->IsSame(tre->left))return false;
        if(right!=NULL && !right->IsSame(tre->right))return false;
        return true;
}

FNode* FNode::GetLeft(){
        return left;
}

FNode* FNode::GetRight(){
        return right;
}

void FNode::Destroy(){
        if(left!=NULL)left->Destroy();
        if(right!=NULL)right->Destroy();
```

```
        delete this;
}

void FNode::SetLeft(FNode* _left){
        if(left!=NULL)left->Destroy();
        left=_left;
}

void FNode::SetRight(FNode* _right){
        if(right!=NULL)right->Destroy();
        right=_right;
}

void FNode::Print(int LastOp, vector<string>variable){
        if(type==T_VAL){
                if(sgn(value-int(value+1e-8))==0 || sgn(value-int(value-1e-8))==0)
                        printf("%.0lf",value);
                else printf("%.3lf", value);
        }else if(type==T_VAR){
                printf("%s", variable[op].c_str());
        }else{
                if(op>POWER){
                        PrintOp(op);
                        printf("(");
                        if(right!=NULL)right->Print(op,variable);
                        printf(")");
                }else if(op>=PLUS && op<=POWER && LastOp>=PLUS && LastOp<=POWER &&
(LastOp>op || LastOp==POWER) && (LastOp+1)/2>=(op+1)/2){
                        printf("(");
                        if(op>=PLUS && op<=POWER && left!=NULL)left-
>Print(op,variable);
                        PrintOp(op);
                        if(right!=NULL)right->Print(op,variable);
                        printf(")");
                }
                else{
```

```
                        if(left!=NULL)left->Print(op,variable);
                        PrintOp(op);
                        if(right!=NULL)right->Print(op,variable);
                }
        }
}

double FNode::GetValue(vector<double>_var){
        if(type==T_VAL){
                return value;
        }else if(type==T_VAR){
                return _var[op];
        }else{
                double v_left=0,v_right=0;
                if(op>=PLUS && op<=POWER)v_left=left->GetValue(_var);
                v_right=right->GetValue(_var);
                if(isnan(v_left) || isnan(v_right))return NAN;
                switch(op){
                        case PLUS:
                                return v_left+v_right;
                        case MINUS:
                                return v_left-v_right;
                        case TIMES:
                                return v_left*v_right;
                        case DIVIDE:
                                if(sgn(v_right)==0){
                                        printf("Math Error: Divide By ZERO.\n");
                                        return NAN;
                                }
                                return v_left/v_right;
                        case POWER:
                                if(sgn(v_right)==0)v_right=0;
                                if(sgn(v_right)<0){
                                        printf("Math Error: Power of negative.\n");
                                        return NAN;
                                }
                                return pow(v_left,v_right);
```

```
                        case SQRT:
                                if(sgn(v_right)==0)v_right=0;
                                if(sgn(v_right)<0){
                                        printf("Math Error: Power of negative.\n");
                                        return NAN;
                                }
                                return sqrt(v_right);
                        case SIN:
                                return sin(v_right);
                        case COS:
                                return cos(v_right);
                        case TAN:
                                return tan(v_right);
                        case ASIN:
                                if(sgn(v_right-1)==0)v_right=1;
                                if(sgn(v_right+1)==0)v_right=-1;
                                if(fabs(v_right)>1){
                                        printf("Math Error: ASIN with number out of
[-1,1].\n");
                                        return NAN;
                                }
                                return asin(v_right);
                        case ACOS:
                                if(sgn(v_right-1)==0)v_right=1;
                                if(sgn(v_right+1)==0)v_right=-1;
                                if(fabs(v_right)>1){
                                        printf("Math Error: ACOS with number out of
[-1,1].\n");
                                        return NAN;
                                }
                                return acos(v_right);
                        case ATAN:
                                return atan(v_right);
                        case LOG:
                                if(sgn(v_right)==0)v_right=0;
                                if(sgn(v_right)<0){
```

```
                                        printf("Math Error: LOG with negative
number.\n");
                                        return NAN;
                                }
                                return log(v_right)/log(2);
                        case LN:
                                if(sgn(v_right)==0)v_right=0;
                                if(sgn(v_right)<0){
                                        printf("Math Error: LN with negative
number.\n");
                                        return NAN;
                                }
                                return log(v_right);
                }
        }
}

void FNode::Reduce(){
        if(type==T_VAL){
                return;
        }else if(type==T_VAR){
                return;
        }else{
                if(op>=PLUS && op<=POWER){
                        left->Reduce();
                }
                right->Reduce();

                if(op==PLUS && left->IsSame(right)){
                        op=TIMES;
                        left->type=T_VAL;
                        left->value=2;
                        if(left->left!=NULL)left->left->Destroy();
                        if(left->right!=NULL)left->right->Destroy();
                        left->left=left->right=NULL;
                }
                if(op==PLUS && right->type==T_VAL && sgn(right->value)==0){
```

```
                right->Destroy();
                FNode*tmp=left;
                (*this)=(*tmp);
                delete(tmp);
                return;
        }
        if(op==PLUS && left->type==T_VAL && sgn(left->value)==0){
                left->Destroy();
                FNode*tmp=right;
                (*this)=(*tmp);
                delete(tmp);
                return;
        }
        if(op==MINUS && left->IsSame(right)){
                type=T_VAL;
                value=0;
                left->Destroy();
                right->Destroy();
                left=right=NULL;
                return;
        }
        if(op==MINUS && right->type==T_VAL && sgn(right->value)==0){
                right->Destroy();
                FNode*tmp=left;
                (*this)=(*tmp);
                delete(tmp);
                return;
        }
        if(op==TIMES && (left->type==T_VAL && sgn(left->value)==0 || right-
>type==T_VAL && sgn(right->value)==0)){
                type=T_VAL;
                value=0;
                left->Destroy();
                right->Destroy();
                left=right=NULL;
                return;
        }
```

```
if(op==TIMES && right->type==T_VAL && sgn(right->value-1)==0){
        right->Destroy();
        FNode*tmp=left;
        (*this)=(*tmp);
        delete(tmp);
        return;
}
if(op==TIMES && left->type==T_VAL && sgn(left->value-1)==0){
        left->Destroy();
        FNode*tmp=right;
        (*this)=(*tmp);
        delete(tmp);
        return;
}
if(op==TIMES && left->IsSame(right)){
        op=POWER;
        right->type=T_VAL;
        right->value=2;
        if(right->left!=NULL)right->left->Destroy();
        if(right->right!=NULL)right->right->Destroy();
        right->left=right->right=NULL;
        return;
}
if(op==DIVIDE && right->type==T_VAL && sgn(right->value-1)==0){
        right->Destroy();
        FNode*tmp=left;
        (*this)=(*tmp);
        delete(tmp);
        return;
}
if(op==DIVIDE && left->type==T_VAL && sgn(left->value)==0){
        right->Destroy();
        FNode*tmp=left;
        (*this)=(*tmp);
        delete(tmp);
        return;
}
```

```
                if(op==DIVIDE && left->IsSame(right)){
                        type=T_VAL;
                        value=1;
                        left->Destroy();
                        right->Destroy();
                        left=right=NULL;
                        return;
                }
                if(op==POWER && right->type==T_VAL && sgn(right->value-1)==0){
                        right->Destroy();
                        FNode*tmp=left;
                        (*this)=(*tmp);
                        delete(tmp);
                        return;
                }
                if(op==POWER && right->type==T_VAL && sgn(right->value)==0){
                        type=T_VAL;
                        value=1;
                        left->Destroy();
                        right->Destroy();
                        left=right=NULL;
                        return;
                }

                if((left==NULL || left->type==T_VAL) && right->type==T_VAL){
                        vector<double> empty;
                        value=GetValue(empty);
                        type=T_VAL;
                        left->Destroy();
                        right->Destroy();
                        left=right=NULL;
                        return;
                }
        }
}

FNode* FNode::Derive(int _var){
```

```
FNode *tmp=new FNode;
if(type==T_VAL){
        tmp->type=T_VAL;
        tmp->value=0;
}else if(type==T_VAR){
        tmp->type=T_VAL;
        tmp->value=(_var==op?1:0);
}else{
        tmp->type=T_OP;
        switch(op){
                case PLUS:
                        tmp->op=PLUS;
                        tmp->left=left->Derive(_var);
                        tmp->right=right->Derive(_var);
                        break;
                case MINUS:
                        tmp->op=MINUS;
                        tmp->left=left->Derive(_var);
                        tmp->right=right->Derive(_var);
                        break;
                case TIMES:
                        tmp->op=PLUS;
                        tmp->left=new FNode(T_OP,0,TIMES);
                        tmp->left->left=left->Derive(_var);
                        tmp->left->right=right->Copy();
                        tmp->right=new FNode(T_OP,0,TIMES);
                        tmp->right->left=left->Copy();
                        tmp->right->right=right->Derive(_var);
                        break;
                case DIVIDE:
                        tmp->op=DIVIDE;
                        tmp->left=new FNode(T_OP,0,MINUS);
                        tmp->left->left=new FNode(T_OP,0,TIMES);
                        tmp->left->left->left=left->Derive(_var);
                        tmp->left->left->right=right->Copy();
                        tmp->left->right=new FNode(T_OP,0,TIMES);
                        tmp->left->right->left=left->Copy();
```

```
            tmp->left->right->right=right->Derive(_var);
            tmp->right=new FNode(T_OP,0,POWER);
            tmp->right->left=right->Copy();
            tmp->right->right=new FNode(T_VAL,2);
            break;
        case POWER:
            tmp->op=TIMES;
            tmp->left=new FNode(T_OP,0,POWER);
            tmp->left->left=left->Copy();
            tmp->left->right=right->Copy();
            tmp->right=new FNode(T_OP,0,PLUS);
            tmp->right->left=new FNode(T_OP,0,DIVIDE);
            tmp->right->left->left=new FNode(T_OP,0,TIMES);
            tmp->right->left->left->left=right->Copy();
            tmp->right->left->left->right=left->Derive(_var);
            tmp->right->left->right=left->Copy();
            tmp->right->right=new FNode(T_OP,0,TIMES);
            tmp->right->right->left=new FNode(T_OP,0,LN);
            tmp->right->right->left->right=left->Copy();
            tmp->right->right->right=right->Derive(_var);
            break;
        case SQRT:
            tmp->op=DIVIDE;
            tmp->left=right->Derive(_var);
            tmp->right=new FNode(T_OP,0,TIMES);
            tmp->right->left=new FNode(T_VAL,2);
            tmp->right->right=new FNode(T_OP,0,SQRT);
            tmp->right->right->right=right->Copy();
            break;
        case SIN:
            tmp->op=TIMES;
            tmp->left=new FNode(T_OP,0,COS);
            tmp->left->right=right->Copy();
            tmp->right=right->Derive(_var);
            break;
        case COS:
            tmp->op=TIMES;
```

```
            tmp->left=new FNode(T_OP,0,TIMES);
            tmp->left->left=new FNode(T_VAL,-1);
            tmp->left->right=new FNode(T_OP,0,SIN);
            tmp->left->right->right=right->Copy();
            tmp->right=right->Derive(_var);
            break;
        case TAN:
            tmp->op=DIVIDE;
            tmp->left=right->Derive(_var);
            tmp->right=new FNode(T_OP,0,POWER);
            tmp->right->left=new FNode(T_OP,0,COS);
            tmp->right->left->right=right->Copy();
            tmp->right->right=new FNode(T_VAL,2);
            break;
        case ASIN:
            tmp->op=DIVIDE;
            tmp->left=right->Derive(_var);
            tmp->right=new FNode(T_OP,0,SQRT);
            tmp->right->right=new FNode(T_OP,0,MINUS);
            tmp->right->right->left=new FNode(T_VAL,1);
            tmp->right->right->right=new FNode(T_OP,0,POWER);
            tmp->right->right->right->left=right->Copy();
            tmp->right->right->right->right=new FNode(T_VAL,2);
            break;
        case ACOS:
            tmp->op=DIVIDE;
            tmp->left=new FNode(T_OP,0,TIMES);
            tmp->left->left=new FNode(T_VAL,-1);
            tmp->left->right=right->Derive(_var);
            tmp->right=new FNode(T_OP,0,SQRT);
            tmp->right->right=new FNode(T_OP,0,MINUS);
            tmp->right->right->left=new FNode(T_VAL,1);
            tmp->right->right->right=new FNode(T_OP,0,POWER);
            tmp->right->right->right->left=right->Copy();
            tmp->right->right->right->right=new FNode(T_VAL,2);
            break;
        case ATAN:
```

```
                                tmp->op=DIVIDE;
                                tmp->left=right->Derive(_var);
                                tmp->right=new FNode(T_OP,0,PLUS);
                                tmp->right->left=new FNode(T_VAL,1);
                                tmp->right->right=new FNode(T_OP,0,POWER);
                                tmp->right->right->left=right->Copy();
                                tmp->right->right->right=new FNode(T_VAL,2);
                                break;
                        case LOG:
                                tmp->op=DIVIDE;
                                tmp->left=right->Derive(_var);
                                tmp->right=new FNode(T_OP,0,TIMES);
                                tmp->right->left=right->Copy();
                                tmp->right->right=new FNode(T_VAL,log(2.0));
                                break;
                        case LN:
                                tmp->op=DIVIDE;
                                tmp->left=right->Derive(_var);
                                tmp->right=right->Copy();
                                break;
                }
        }
        return tmp;
}

bool FNode::AddNode(FNode*node){
        if(type!=T_OP)return 0;
        if(op>=PLUS && op<=POWER){
                if(left==NULL){
                        left=node;
                        return 1;
                }
                if(left->AddNode(node))return 1;
        }
        if(right==NULL){
                right=node;
                return 1;
```

```
        }
        return right->AddNode(node);
}

class FTree{
private:
        vector<string> variable;
        FNode*root;
        double
GetIntegal(vector<double>_begin,vector<double>_end,vector<double>_now,int _var);
public:
        FTree(vector<string>_variable, FNode*_root);
        FNode*Copy();
        void AddNode(FNode*node);
        void Print();
        FTree*Derive(int _var);
        FTree*Derive(string var);
        double GetIntegal(vector<double>_begin, vector<double>_end);
        double GetValue(vector<double>_var);
        void Reduce();
};

FNode*FTree::Copy(){
        return root->Copy();
}

void FTree::Reduce(){
        root->Reduce();
}

FTree::FTree(vector<string>_variable,
FNode*_root=NULL):variable(_variable),root(_root){}

void FTree::AddNode(FNode*node){
        if(root==NULL)root=node;
        else root->AddNode(node);
```

```
}

void FTree::Print(){
        root->Print(-1,variable);
}

FTree* FTree::Derive(int _var){
        vector<string> tmp_variable;
        tmp_variable.push_back(variable[_var]);
        FTree*tmp=new FTree(tmp_variable,root->Derive(_var));
        tmp->root->Reduce();
        return tmp;
}

FTree* FTree::Derive(string var){
        int pos = std::find(variable.begin(), variable.end(), var) -
variable.begin();
        return Derive(pos);
}

double FTree::GetValue(vector<double>_var){
        return root->GetValue(_var);
}

double FTree::GetIntegal(vector<double>_begin, vector<double>_end,vector<double>_now,
int _var){
        int n=_begin.size();
        double ans=0;
        int cnt=(int)pow(TOT,1.0/n);
        _now.push_back(_begin[_var]);
        double last=(_var==n-
1?GetValue(_now):GetIntegal(_begin,_end,_now,_var+1)),tmp;
        for(int i=1;i<=cnt;i++){
                _now[_var]=_begin[_var]+i*(_end[_var]-_begin[_var])/cnt;
                tmp=(_var==n-1?GetValue(_now):GetIntegal(_begin,_end,_now,_var+1));
                ans+=(_end[_var]-_begin[_var])*(tmp+last)/2/(cnt);
```

```
                last=tmp;
        }
        return ans;
}

double FTree::GetIntegal(vector<double>_begin, vector<double>_end){
        vector<double>_now;
        return GetIntegal(_begin,_end,_now,0);
}
```

8.8 calcul2.helper

```
##input for calcul2

int main(){
  f(x,y)=x*2+y*3;
  :f(1,2);
}
##

##output for calcul2

#include <cstdio>
#include "calcul2.h"
using namespace std;

int main(){
  double printer;

  vector<string> f_var;
  f_var.push_back("x");
  f_var.push_back("y");
  vector<double> f_begin, f_end, f_now;
  FTree f(f_var);
  f.AddNode(new FNode(T_OP,0,PLUS)); //T_OP for operator
  f.AddNode(new FNode(T_OP,0,TIMES));  // when T_OP calls, the second parameter
should be 0
```

```
  f.AddNode(new FNode(T_VAR,0,0)); //T_VAR for variable, 0 for the first(x), 1 for
the second(y)
  f.AddNode(new FNode(T_VAL,2)); // T_VAL for value
  f.AddNode(new FNode(T_OP,0,TIMES));
  f.AddNode(new FNode(T_VAR,0,1)); // when T_VAR calls, the second parameter should
be 0
  f.AddNode(new FNode(T_VAL,3));

  f_now.clear();
  f_now.push_back(1);
  f_now.push_back(2);
  printer=f.GetValue(f_now);
  printf("%lf\n",printer);
  return 0;
}
```

8.9 makefile

```
all: ast.cmi compiler
compiler: parser.cmo scanner.cmo semantics.cmo codegen.cmo compiler.cmo
        ocamlc -o calcul parser.cmo scanner.cmo semantics.cmo codegen.cmo
compiler.cmo

compiler.cmo: compiler.ml
        ocamlc -c compiler.ml

codegen.cmo: codegen.ml
        ocamlc -c codegen.ml

semantics.cmo: semantics.ml
        ocamlc -c semantics.ml

scanner.cmo: scanner.ml
        ocamlc -c scanner.ml

scanner.ml: scanner.mll
        ocamllex scanner.mll
```

```
parser.cmo: parser.ml
	ocamlc -c parser.ml

parser.ml: parser.mli
	ocamlc -c parser.mli

parser.mli: parser.mly
	ocamlyacc parser.mly

ast.cmi: ast.ml
	ocamlc -c ast.ml

clean:
	rm -f parser.mli parser.ml scanner.ml ast.cmi testall.log calcul \
	output output.cpp *.cmo *.cmi *.out
```

8.10 run.sh

```
#!/bin/sh

# compile the input file
./calcul < $1

# compile cpp file
g++ -o output output.cpp

# run the cpp executation
./output | tee output.out
```

8.11 test

8.11.1 test-arith1.cul

```
main()
{
	x = 15;
```

```
        :(39 + x + 3);
}
```

8.11.2 test-arith2.cul

```
main()
{
        a = 3;
        b = 4;
        :(a + b);
}
```

8.11.3 test-fibo.cul

```
fibo(x)
{
        if (x < 2) return 1;
        return fibo(x - 1) + fibo(x - 2);
}

main()
{
        :(fibo(0));
        :(fibo(1));
        :(fibo(5));
}
```

8.11.4 test-func-while.cul

```
f(x, y) {
                while (x != y) {
                        if (x > y) x = x - y;
                        else y = y - x;
                }
                return x+y;
        }
```

```
main()
{
        :f(15, 20);
}
```

8.11.5 test-func-while2.cul

```
f(x){
        f = 0;
        while (! x == 0) {
                f = f + x;
                x = x - 1;
        }
        return f;
}

main()
{
        :f(5);
}
```

8.11.6 test-func1.cul

```
gcd(x, y)
{
        while (x != y) {
                if (x > y) x = x - y;
                else y = y - x;
        }
        return x;
}

main()
{
        :(gcd(24, 36));
}
```

8.11.7 test-mathfunc-basic1.cul

```
main()
{
        f($x) = 3 * x + 2;
        :f(4);
}
```

8.11.8 test-mathfunc-diriv1.cul

```
main()
{
        f($x) = x ^ 3 + 3 * x;
        :f'(x);
}
```

8.11.9 test-mathfunc-diriv2.cul

```
main()
{
        f($x, $y) = 2 * x + 3 * y;
        :f'(x);
}
```

8.11.10 test-mathfunc-gcd.cul

```
f(x, y) {
                while (x != y) {
                        if (x > y) x = x - y;
                        else y = y - x;
                }
                return x+y;
        }


main()
{
        :f(15, 20);
}
```

8.11.11 test-mathfunc-ifelse.cul

```
f(x){
s=0;
if (x == 0)
s = 0;
else
s = f(x - 1) + x;
return s;
}

main()
{
        :f(3);
}
```

8.11.12 test-mathfunc-integ1.cul

```
main()
{
        f($x) = 3 * x ^ 2 + x;
        :f@x(3, 10);
}
```

8.11.13 test-mathfunc-total1.cul

```
main()
{
        f($x) = 2 * x;
        g($x) = 3 * sin(x);

        h($x) = f * g'(x);
        :h;
        :h'(x);
        :h@x(1, 3);

        q($x) = sin(x);
        ##:h(1.57);

        a = 3;
        :a ^ 2;
```

```
}
```

8.11.14 test-mathfunc-while.cul

```
f(x){
        f = 0;
        while (! x == 0) {
                f = f + x;
                x = x - 1;
        }
}
main()
{
        :f(5);
}
```

8.11.15 test-total.cul

```
main()
{
        f($x) = 2 * x;
        g($x) = 3 * sin(x);

        h($x) = f * g'(x);
        :h;
        :h'(x);
        :h@x(1, 3);

        q($x) = sin(x);
        ##:h(1.57);

        a = 3;
        :a ^ 2;
}
```

8.11.16 test-total1.cul

```
main(){
```

```
        f($x)=x^x-1;
        g($x)=f'(x);
        :f;
        :g;
        :f'(x);
        :g@x(1,5);
        :f(5);
}
```

8.12 semantic test

8.12.1 setest1.cul

```
#* Failure("Duplicate parameter in function add") *#
add(x,x){
        return x+y;
}

main(){
x=3;
}
```

8.12.1 setest2.cul

```
#* Failure("No Main Function exist!") *#



add(x,y){
        return x+y;
}
```

8.12.1 setest3.cul

```
#* Failure("Function whose name is add has been defined more than once") *#
add(x,y){
        return x+y;
}

add(x){
```

```
        return x;
}

main(){
x=3;
}
```

8.12.1 setest4.cul

```
#* Failure("Undefined function or math function: add1 is used!")*#

add(x,y){
        return x+y;
}
main(){
x=3;
add1(x,y);
}
```

8.12.1 setest5.cul

```
#*exception Failure("Math function has duplicate parameter!")*#

main(){
x=3;
f=1;
h($y,$y)=y+1;
}
```

8.12.1 setest6.cul

```
#* exception Failure("Using math function that parameter not match!")*#

main(){
x=3;
h($x)=x+1;
g($y)=y+1;
f($x)=h+g+1;
}
```

8.12.1 setest7.cul

```
#*exception Failure("ID : t not valid!")*#
```

```
main(){
x=3;
y=t+1;
}
```

8.12.1 setest8.cul

```
#*Failure("function parameter not match!")*#
add(x,y){
  return x+y;
}

main(){
x=3;
y=1+add(3,5,2);
}
```

8.12.1 setest9.cul

```
#*Failure("Undefined ID : x!")*#
main(){
{ x=3; }
y=x+1;
}
```

8.12.1 setest10.cul

```
#*Failure("Derivation is not properly used!")*#
main(){
f($x)=x*x+2;
:f'(y);
}
```

8.12.1 setest11.cul

```
#*Failure("Integral is not properly used!")*#
main(){
f($x)=x*x+2;
:f'(x);
```

```
:f@y(1,3);
}
```

8.12.1 setest12.cul

```
#*Failure("Integral is not properly used!")*#
main(){
f($x)=x*x+2;
:f'(x);
:f@x(1,3,5);
}
```

8.12.1 setest13.cul

```
#*Failure("Integral is not properly used!")*#
main(){
f($x)=x*x+2;
:f'(x);
:h@x(1,3,5);
}
```

8.13 Project log file from Github

commit 68b9fc88a7f2668651fc499d888729ef3152ba8f
Author: jandyhuang <jndhuang8@gmail.com>
Date: Fri Dec 20 14:13:39 2013 -0500

Update again

commit 693c4f452ede5908263bcb53c5ef39a559c100c8
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Fri Dec 20 14:06:27 2013 -0500

Updated the slides.

commit 24bfa6b6881230114dcd5e8ac9547fb659363d77
Author: wingsking <jz2540@columbia.edu>
Date: Fri Dec 20 13:44:53 2013 -0500

Update codegen.ml

commit 9c4dde4020ad738fa1ee1efd02c32c8a2f9ba3bf
Author: jandyhuang <jndhuang8@gmail.com>
Date: Fri Dec 20 11:19:42 2013 -0500

Removed Again

commit 95777f91d2edc38f79cf940afa27425e379dc02e
Author: jandyhuang <jndhuang8@gmail.com>
Date: Fri Dec 20 11:16:00 2013 -0500

Useless Files Removed

Don know why these files are commited automatically

commit 6c57aece92bcb85d7c18ba44827ed46a01049ea8
Author: jandyhuang <jndhuang8@gmail.com>
Date: Fri Dec 20 11:13:03 2013 -0500

Slides Updated

commit c0a10fd55bfd8bbd6528986ecdad1b1893acc01d
Author: jandyhuang <jndhuang8@gmail.com>
Date: Fri Dec 20 10:48:21 2013 -0500

Reupdated Tests

commit 84d03566e66a66e5901f499ba04f547f7abfa4d6
Author: WtYin <yyin0909@gmail.com>
Date: Fri Dec 20 01:38:42 2013 -0500

Update semantics and add semantics test

commit e9f9e9527e8c5837661891d4756bdc2b93dfe52e
Author: jandyhuang <jndhuang8@gmail.com>
Date: Fri Dec 20 01:28:50 2013 -0500

Deleted False Test

commit 161a29baed7ed2e99d59b88eb6cfcbc4565b95f5
Author: wingsking <jz2540@columbia.edu>
Date: Fri Dec 20 00:12:16 2013 -0500

Update test-mathfunc-while.cul

commit feea3cf9cc3761c54b3272742ccd816ef4eb2a24
Author: WtYin <yyin0909@gmail.com>
Date: Thu Dec 19 23:53:20 2013 -0500

Sementics Updated

commit 49d1f5d2d0475ddb27b5dbf34db501680b57ec78
Author: WtYin <yyin0909@gmail.com>
Date: Thu Dec 19 20:45:23 2013 -0500

Sementics updated

commit ae7727d9f22f916043e2b3b8ce4a71272f4b2d64
Author: wingsking <jz2540@columbia.edu>
Date: Thu Dec 19 23:02:10 2013 -0500

Update test-mathfunc-ifelse.cul

commit caeaa4a3ec89d69858ec0f75f6c09afdb4131a17
Author: wingsking <jz2540@columbia.edu>
Date: Thu Dec 19 23:00:44 2013 -0500

Update test-mathfunc-gcd.cul

commit 017962a81c0d013a9ebc976939c4f3492ed92d4c
Author: wingsking <jz2540@columbia.edu>
Date: Thu Dec 19 22:59:19 2013 -0500

Update test-func1.cul

commit fb7e6c17c5c3a671783291ab30f4960bb840e7b9
Author: wingsking <jz2540@columbia.edu>
Date: Thu Dec 19 22:58:43 2013 -0500

Update test-fibo.cul

commit cbdfcf43a227eb1238e8c84491fab03b15a75bdb
Author: wingsking <jz2540@columbia.edu>
Date: Thu Dec 19 22:58:14 2013 -0500

Update test-arith2.cul

commit 29b8aacd62ffdad66dbf18b8ee1d7b235666fd73
Author: wingsking <jz2540@columbia.edu>
Date: Thu Dec 19 22:57:51 2013 -0500

Update test-arith1.cul

commit 040d911ab19ea2bcab8fe4ebdbbe0f5e021c2c2f
Author: wingsking <jz2540@columbia.edu>
Date: Thu Dec 19 22:56:05 2013 -0500

Update ast.ml

commit 9b709e805692eeda68608c9b4c3557cc6b9ad230
Author: wingsking <jz2540@columbia.edu>
Date: Thu Dec 19 22:55:26 2013 -0500

Update parser.mly

commit d5c8879d04a65d82a2ce23cc806acb4296413591
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Wed Dec 18 21:54:55 2013 -0500

Updated codegen.ml, compiler.ml, makefile.

Added comments for codegen.ml.

commit 68a3e4bcad70ec0cc07f327fcdcc35aca4b4c25e
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Wed Dec 18 21:20:24 2013 -0500

Updated codegen.ml and compile.ml, Added run.sh, Added a test case.

1. Reduced duplicate codes to improve code quality.
2. Compile takes input from standard input.
3. Run.sh accepts argument of .cul input files and will compite it, run it and print the output.
4. Added a test case.

commit a7f9b2bcba80ccd952be38fb8d0158d1e799b568
Author: wingsking <jz2540@columbia.edu>
Date: Wed Dec 18 20:48:55 2013 -0500

Update codegen.ml

commit a4c92e97c33f5251d2040ba60b0abec0a9542bc4
Author: wingsking <jz2540@columbia.edu>
Date: Wed Dec 18 20:33:02 2013 -0500

Update parser.mly

commit b22180dde936f209494714c76ea4e4e90bfbea59
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Wed Dec 18 20:32:08 2013 -0500

Updated codegen.ml

Fixed some bugs. It supports arithmathic operations for derivation of functions now, e.g. h($x) = f + g'(x).

commit 63a6d5e96796d3ead816edb873c066f69ed9eede
Author: jandyhuang <jndhuang8@gmail.com>
Date: Wed Dec 18 19:43:02 2013 -0500

Updated calcul2.h again

Repair some errors in Reduce.

commit fc7e4e2c39771bcc20cb38257187b80afac20de4
Merge: 48dc37c 0678377
Author: jandyhuang <jndhuang8@gmail.com>
Date: Wed Dec 18 19:12:54 2013 -0500

Merge branch 'master' of https://github.com/jandyhuang/Calcul2

commit 48dc37c8106addfe2ccabd9408266d7d843726da
Author: jandyhuang <jndhuang8@gmail.com>
Date: Wed Dec 18 19:12:48 2013 -0500

Updated calcul2.h

Repair some errors in Integral

commit 067837744f11de9d4230b1f7c0324d1225dca622
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Wed Dec 18 19:09:08 2013 -0500

Updated codegen.ml

It supports infinite integal now.

commit c6d1bc3fa9eace7cbadb64e876323cf5fdd3fe87
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Wed Dec 18 16:57:58 2013 -0500

Updated calcul2.h, codegen.ml, compiler.ml

1. Added a function in calcul2.h to call the Derive() with string argument.
2. Updated codegen.ml. Now it supports the derivation and arithmathic operations for math functions.
3. Modified compiler.ml, commited out the SEMANTICS part temporarily.

commit 7d96b49ac1bfdfb23a48212f30d3300b6ffa4519
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Wed Dec 18 13:21:58 2013 -0500

Fixed typoes in codegen.ml

commit 66413ae32e80133297dfe1cf52ffc792ecf18e54
Author: michaelzs <shuzhan.swjtu@gmail.com>
Date: Wed Dec 18 10:29:14 2013 -0500

semantics update

commit 7b61ab067ed2d6916a7e95c170d05d9416d72175
Author: wingsking <jz2540@columbia.edu>
Date: Wed Dec 18 05:40:07 2013 -0500

update codegen

commit 9af57a7ab90b5ed65a18d7077013f46f9a3a963a
Author: michaelzs <shuzhan.swjtu@gmail.com>
Date: Wed Dec 18 01:11:29 2013 -0500

Semantics update

Modify some functions and debug the errors

commit 21c17ab0e4957217c864c8e75526f7a252bc1121
Author: WtYin <yyin0909@gmail.com>
Date: Tue Dec 17 17:40:57 2013 -0500

sementics

commit 738d1718e487f862faf10bb701d27bcada11964d
Author: WtYin <yyin0909@gmail.com>
Date: Tue Dec 17 17:39:09 2013 -0500

sementics change

commit 43af2054d8f395064c09b8275cfe389e46dc0d7b
Author: WtYin <yyin0909@gmail.com>
Date: Tue Dec 17 17:20:49 2013 -0500

sementics update

commit a9aed1a9576362d8349b112a740a0017de50f846
Merge: 7c6653a 60495bc
Author: jandyhuang <jndhuang8@gmail.com>
Date: Tue Dec 17 02:30:59 2013 -0500

Merge branch 'master' of https://github.com/jandyhuang/Calcul2

commit 7c6653a83d818f24cdb6ab2cc0996553280360fa
Author: jandyhuang <jndhuang8@gmail.com>
Date: Tue Dec 17 02:30:53 2013 -0500

Update calcul2.h

Derivation added. Fixed every problem known. Much more powerful than I can ever imagine.

commit 60495bcba0fc3e60cca470a3060c8b87719d3187
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Tue Dec 17 01:26:30 2013 -0500

Updated codegen.ml

Added code generation for math functions.

Now it can generate cpp code correctly, which matches the data structure that our cpp library uses.

Here is an example:
1. Input:

```
main()
{
	f($x, $y) = 2 * x ^ 2 + y * 3;
	:f(1, 2);
}
```

2. Generated:

```
#include <cstdio>
#include "calcul2.h"
using namespace std;

int main()
{
	double printer;

	vector<string> f_var;
	f.push_back("x");
	f.push_back("y");
	vector<double> f_now;
	FTree f(f_var);

	f.AddNode(new FNode(T_VAL,2.));
	f.AddNode(new FNode(T_OP,0,TIMES));
	f.AddNode(new FNode(T_VAR,0,0));
	f.AddNode(new FNode(T_OP,0,POWER));
	f.AddNode(new FNode(T_VAL,2.));
	f.AddNode(new FNode(T_OP,0,PLUS));
	f.AddNode(new FNode(T_VAR,0,1));
	f.AddNode(new FNode(T_OP,0,TIMES));
	f.AddNode(new FNode(T_VAL,3.));

	f_now.clear();
	f_now.pushback("1.");
	f_now.pushback("2.");
	printer = f.GetValue(f_now);
	printf("%lf\n",printer);

	return 0;
}
```

commit 2a97ac7a088a607b824cd1ac9707ac3034fb92be
Author: michaelzs <shuzhan.swjtu@gmail.com>
Date: Sun Dec 15 21:01:10 2013 -0500

Update semantics

add function and variable name check and function body check
debug the errors

commit e9a8a7e1e464d06df3f1839db488e066a46b5ea3
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Sun Dec 15 20:50:26 2013 -0500

Updated codegen.ml

Unified the output format of functions, make it more clear.

commit 733fd0796b6df6bbf0c7d3f2f3cea7094fdb3be0
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Sun Dec 15 20:15:58 2013 -0500

Updated codegen.ml

1. Added the "#include .." header file in the beginning of the generated code file.
2. Print the generated code in screen and also output it to the "output.cpp" file.

commit ed099a737948c2b3500fdec66e874997d071bfa5
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Sun Dec 15 19:55:18 2013 -0500

Added codegen into makefile

Compiler can call codegen now.

commit 4cccdc64c7b55cb43e5519860abdf3a1203007a6
Author: wingsking <jz2540@columbia.edu>
Date: Sun Dec 15 19:54:18 2013 -0500

add DOLLAR

commit de10817e4c3e8235bc2a12c6baf16d02bace1189
Author: wingsking <jz2540@columbia.edu>
Date: Sun Dec 15 19:53:40 2013 -0500

update DOLLAR

commit 6e0ec4ed85ec0acda121fa99f522e23f65b13ec7
Author: wingsking <jz2540@columbia.edu>
Date: Sun Dec 15 19:29:11 2013 -0500

part of codegen, still need works(read comments)

commit e7f802f30f9778f808c4e7acfd77ba5643a6848d
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Sun Dec 15 19:26:57 2013 -0500

Added codegen.ml.

Created codegen.ml file

commit a7b88dba5c8e74790b2084a632a43d0f3f5445fc
Author: wingsking <jz2540@columbia.edu>
Date: Sun Dec 15 17:48:05 2013 -0500

Update scanner.mll

commit 59e94194c129d3c75cce596035d8d1319508c63a
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Sun Dec 15 17:29:02 2013 -0500

Added testall.sh and test cases, updated makefile.

Makefile can delete all intermediate files now.
Added several test cases for both basic functions and math functions.

commit 70ba41b4aaa859046219228d79325e3a7f1c2ffa
Author: jandyhuang <jndhuang8@gmail.com>
Date: Sun Dec 15 16:02:53 2013 -0500

Updated calcul2.h, calcul2.helper

If you don't know how to use calcul2.h, read calcul2.helper for an example.
calcul2.h is almost finished except Deride.

commit f25f00888d7fc9ed7816d3e79932ca0c6ec8b63f
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Sun Dec 15 15:33:39 2013 -0500

Updated compiler.ml, makefile, semantics.ml.

1. fixed errors in using "make" command;
2. added semantics into makefile;
3. fixed bugs in semantics.ml
Now "make" should work, notice that you also need a input file "input.cul" to run the command.

commit 1f6f0e33c480f55c5a2f61252ac729312bca6f35
Author: WtYin <yyin0909@gmail.com>
Date: Fri Dec 13 22:46:19 2013 -0500

update semantics

commit 7943377ed606eaefb5fed4060533f2990ce83ffe
Author: michaelzs <shuzhan.swjtu@gmail.com>
Date: Fri Dec 13 22:21:02 2013 -0500

senmantics update

commit a44d3403a44bfcb41927941ca72e4b3a233a1c7b
Author: wingsking <jz2540@columbia.edu>
Date: Fri Dec 13 14:37:54 2013 -0500

remove INTDIVIDE, MOD, LITERAL

commit aa862eefe6fc9a5dc6c858f7fbd5708c8a8548fe
Author: wingsking <jz2540@columbia.edu>
Date: Thu Dec 12 23:06:27 2013 -0500

program is now a list of fdecl, math_func is now a stmt

for math_func declaration, it should be in the format like f($x,$y)=2*x+y, out a $ in front of variable is to resolve the reduce/reduce error, unless anyone can think of a better way to deal with that

commit 7d6fa0d4d7d657f86849fb93f0015359666b879a
Author: wingsking <jz2540@columbia.edu>
Date: Thu Dec 12 23:01:08 2013 -0500

remove string list from type = program

commit f45a0d84a7443f76d19aa409dac4545a6324f77f
Author: michaelzs <shuzhan.swjtu@gmail.com>

Date: Thu Dec 12 17:49:35 2013 -0500

    semantics modified

commit 1a08f1e116ccf691a8611e6178b7cbca68a4da7b
Author: michaelzs <shuzhan.swjtu@gmail.com>
Date: Thu Dec 12 15:36:14 2013 -0500

    Semantics function checking part

commit 1adcf5a9b800d733bab17850f059db8c2b2ffd26
Author: wingsking <jz2540@columbia.edu>
Date: Thu Dec 12 15:20:57 2013 -0500

    remove int, and everything related to it

commit 9d2ec00181b5d29ecd1293cee6dae284221f02c9
Author: wingsking <jz2540@columbia.edu>
Date: Thu Dec 12 01:59:39 2013 -0500

    remove math_func and var_decl

    Not sure if I am doing this right...please check

commit 35136521846efcda26b662f5ecbae24ca3677105
Author: wingsking <jz2540@columbia.edu>
Date: Thu Dec 12 01:55:06 2013 -0500

    type mismatching error

commit 4135c3a18b2e79cbf87e08f01113ebbeeb627bfe
Author: wingsking <jz2540@columbia.edu>
Date: Tue Dec 10 22:00:51 2013 -0500

    add math_fdecl under "program:", separate fdecl into fdecl and math_fdecl

commit 4004115414ff783e4823a5cae373993e1d74648a
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Sat Nov 23 17:08:13 2013 -0500

    Removed locals in func_decl in ast.ml.

commit 221b7a538bc318a918d14b3741d4fdfa6ebff155
Author: Kewei Ge <kewei.ice@gmail.com>

Date: Sat Nov 23 17:06:18 2013 -0500

    Removed unnecessary files.

commit 35f3fcc6a507e2a5a561ba4eb0fbd9c7da3ed676
Author: jandyhuang <jndhuang8@gmail.com>
Date: Sat Nov 23 16:54:05 2013 -0500

    Updated makefile

commit 00a4331ca7d7078a9b4ce669abbc74e9e17f470b
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Sat Nov 23 16:26:00 2013 -0500

    Removed an 'Equal' in ast.ml, fixed bugs in parser.mly.

commit fbdfc7e5a6aa4986200742716d6e2a8e3571b850
Author: WtYin <yyin0909@gmail.com>
Date: Sat Nov 23 16:25:16 2013 -0500

    parser

commit 02df90c701c114974d30ece357c463a72c1660f8
Author: WtYin <yyin0909@gmail.com>
Date: Sat Nov 23 16:00:50 2013 -0500

    parser

commit 87ce25156d4816a981ba58e0452530037e5dfd1a
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Sat Nov 23 15:44:09 2013 -0500

    Fixed a typo in ast.ml, and fixed a bug in parser.mly. Removed parser.ml.

commit eb8b7057af102709864c540462cf2caf28222edf
Author: WtYin <yyin0909@gmail.com>
Date: Thu Nov 21 23:41:24 2013 -0500

    parser

commit aeec7c13f6aa2559e427aac05bb7b27a18548584
Author: WtYin <yyin0909@gmail.com>
Date: Thu Nov 21 23:37:59 2013 -0500

parser

commit a8033d2a153136a788bfbdca18cd5223dc276548
Author: WtYin <yyin0909@gmail.com>
Date: Thu Nov 21 23:31:28 2013 -0500

parser

commit d0d64a459206ec412f4522735c990bea8b2f2c2c
Author: WtYin <yyin0909@gmail.com>
Date: Thu Nov 21 23:27:13 2013 -0500

add output and a SEMI for fedcl

commit 323e48daafa8c680a80824c34b7a6f34baab0d61
Author: jandyhuang <jndhuang8@gmail.com>
Date: Thu Nov 21 20:38:50 2013 -0500

Update Compiler

Update all files with temporary RETURN.
Compiling Scanner: No Problem.
Compiling Parser:182+ shift/reduce errors.
Fix it right now!!

commit fa2b13912e3cb8c306b1dfeb966b9df0ed6f5d90
Author: wingsking <jz2540@columbia.edu>
Date: Tue Nov 19 23:32:44 2013 -0500

added vecter and float_list at the end of file

commit c96ae4981c5c7f88884ed069260319297a344fbb
Author: WtYin <yyin0909@gmail.com>
Date: Sun Nov 17 17:00:13 2013 -0500

parser

commit d1a47f33219b8f3378296ec5c3b168d2fb93d38a
Author: WtYin <yyin0909@gmail.com>
Date: Sun Nov 17 16:57:32 2013 -0500

parser

commit 5124cdcfe73f46cef852479030411b36ea6138d8

Author: WtYin <yyin0909@gmail.com>
Date: Sun Nov 17 16:56:36 2013 -0500

    scanner

commit 0c0d3705f9ae72b03397d60ec05ae86e195f805e
Author: jandyhuang <jndhuang8@gmail.com>
Date: Sun Nov 17 16:53:13 2013 -0500

    Create calcul2.h

    Prepare the tree definition for Code Generating.

commit 21817479a7f9c20e689eefd5ccd9209ff19e1698
Author: wingsking <jz2540@columbia.edu>
Date: Sun Nov 17 16:42:31 2013 -0500

    update vdecl and vdecl_list

commit 577db945939a9e94ecc0e50cdebd9f4bb02689ca
Author: michaelzs <shuzhan.swjtu@gmail.com>
Date: Sun Nov 17 16:40:45 2013 -0500

    scanner change

commit 68d180c61b73a480b85b755e967a0727dcd6cf94
Author: WtYin <yyin0909@gmail.com>
Date: Sun Nov 17 16:34:45 2013 -0500

    parser

commit 59d667145cc3fd7ccb673375542551ae9f84d670
Author: wingsking <jz2540@columbia.edu>
Date: Sun Nov 17 16:30:22 2013 -0500

    update fdecl

commit 1e0cfb47fbc7a3e09bce357671bebd938485bd97
Author: WtYin <yyin0909@gmail.com>
Date: Sun Nov 17 16:21:29 2013 -0500

    parser

commit a98c5c7163987276ac81998c6aa0fbf0b19a2c49

Author: Kewei Ge <kewei.ice@gmail.com>
Date: Sun Nov 17 16:20:08 2013 -0500

    Fixed a type name.

commit c7afde71bffe60475fc156c6ba3267d43ff9da7a
Author: WtYin <yyin0909@gmail.com>
Date: Sun Nov 17 16:06:52 2013 -0500

    parser

commit 9941ae8931b3cc36fc0f21cd5afbfe8c0e918b24
Author: jandyhuang <jndhuang8@gmail.com>
Date: Sun Nov 17 16:01:45 2013 -0500

    BAK File Deleted

    DO NOT UPLOAD ANT TMP FILE!!!

commit 1b293c264f6aed6136e0d04fcfc1a55a619a9406
Author: jandyhuang <jndhuang8@gmail.com>
Date: Sun Nov 17 15:56:38 2013 -0500

    Scanner changed

    Add floating point number as REAL

commit d5f0dc454f7b7c5a6be216ee79394a904e0d53b5
Author: michaelzs <shuzhan.swjtu@gmail.com>
Date: Sun Nov 17 15:59:19 2013 -0500

    parser improvement

commit c93e171bef620080663a287a864475714f4f5c75
Author: WtYin <yyin0909@gmail.com>
Date: Sun Nov 17 15:54:51 2013 -0500

    parser change

commit b31a934857d617ad28ef83803e39facdb96e9704
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Sun Nov 17 15:16:44 2013 -0500

    added comments for ast.ml

commit b4872450b964ff7a7c6f26c91852918ca0d905c7
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Sun Nov 17 15:09:10 2013 -0500

    removed unary operator for derivation.

commit bc5903fd8f6e0f84182abc3ff670e0776d6b1711
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Sat Nov 16 18:25:38 2013 -0500

    removed unecessary codes, modified operators and expressions, also added varible declaration and math function declaration.

commit e6bf9783fa074ac1a7938f5bd70fd06394318a64
Author: Eric Z <jz2540@columbia.edu>
Date: Sun Nov 10 17:46:59 2013 -0500

    update ast.ml

commit 01801eb2a92fa6bb0d4f8767809c6c21b7637bc2
Author: jandyhuang <jndhuang8@gmail.com>
Date: Sun Nov 10 17:42:53 2013 -0500

    SWP file removed

commit e3108d467a7b6eb333cb320d243d32140d5bd7fa
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Sun Nov 10 17:40:36 2013 -0500

    Modified ast.ml, added operators for ast.

    Also defined before unary operator and after unary operator.

commit cbb2c493ee1b8ce7b6cfe42e7e1f92599af6ef00
Author: WtYin <yyin0909@gmail.com>
Date: Sun Nov 10 17:32:44 2013 -0500

    Scanner 2

    Updated derivation as '\'', not '`'.

commit db1300566855017dea83dfb4530ad2d1d24022a4
Author: jandyhuang <jndhuang8@gmail.com>

Date: Sun Nov 10 17:01:15 2013 -0500

    Scanner version 1

commit 7151011854ebda3a9ee9db774da40475ba938101
Author: jandyhuang <jndhuang8@gmail.com>
Date: Sun Nov 10 16:47:50 2013 -0500

    Revert "new ast & parser"

    This reverts commit 43916dbc42d9256fe8b02c5fdf4447da23d173d0.

commit c7698fa34a8b957f77d1a73674bb921519d1d4dd
Author: jandyhuang <jndhuang8@gmail.com>
Date: Sun Nov 10 16:47:36 2013 -0500

    new ast & parser

commit e2563a72bfd497b2b1fbaf67fb2a0ea8800d9f06
Author: Kewei Ge <kewei.ice@gmail.com>
Date: Sun Nov 10 16:46:47 2013 -0500

    Added Scanner.mll, parser.mly, ast.ml

commit 1fc7f99f4ad725a264d6178b9aa617273801122e
Author: jandyhuang <jndhuang8@gmail.com>
Date: Sun Nov 10 16:27:24 2013 -0500

    Adding scanner.mll

commit 7e8b060d88a53eeda61fad6569dbb2f62e4f9c92
Author: jandyhuang <jndhuang8@gmail.com>
Date: Sun Nov 10 16:18:08 2013 -0500

    Delete README.md

commit f151630daad7528fab5f731ead9b917b46707961
Author: Jandy <jndhuang8@gmail.com>
Date: Sun Nov 10 16:10:13 2013 -0500

    first commit