Stoichy

Final Report

Professor Ramakrishna Upadrasta

K Vamshi Akash Tadwai Vinta Reethu Havya K Sai Varshittha

December 18, 2020

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Contents

Introduction

Stoichy is a language that can be used to solve problems in different fields of chemistry , like stoichiometric calculations, oxidation-reduction reactions, acid-base reactions, thermodynamics, and electrochemistry. The questions in chemistry are generally procedural and there's a common approach to understand each specific type of issue.
For example, to find the molar mass of a molecule, we use the molar mass of an element and the number of moles of each element present in 1 mole of the compound and based on this info, we then calculate the mass of 1 mole of the compound. Sometimes, these calculations become very complex to

be done by hand, lets say, where we need to balance an equation with consisting many elements and complex compounds, where we need to solve for many variables using linear equations. All these kind of problems can generally be solved in a unique stepwise procedure involving calculations, solving linear equations,etc, which can even be applied to complex models and equations too.

Solving all such problems can be done simply by using Stoichy, where we use data types and functions that are explictly designed to suit such needs, handles all such complex calculations and users can save a lot of time by making use of this language.

We have implemented this by using a multi-paradigm language Ocaml by using it's functional features.

Chapter 1

Language Tutorial

1.1 Pre-requisites

The whole project needs **java** and some other libraries. First one need to install java and then run the bash script **prereqs.sh** in **sudo** mode to install all dependenices.

1.2 Program Execution

We have written a *make* file, which on compiling would give an executable file named **stoichy**. The extension for our language is **sthy**, in order to compile a .sthy file we simply have to run the following command :

```
./stoichy <path-to-file.sthy>
```

Upon compilation, the sthy files are converted into java file *(.java)*, which is run using **JVM**.

1.3 Function Declaration

Functions can be declared using the keyword **"def"** in stoichy. One function named **main** should be present in every stoichy file. We may pass any number of arguments along with the type specifier inside the function as functional arguments. However the passed arguments values in the function can't be changed.

Syntax to declare a function with no arguments.

```
def main( )
{

...

}
```

Syntax to declare a function with three arguments.

```
def message( int a, double b, string c )
{

...

}
```

1.4 Variable Declarations

We support **3** types in primitive and **2** types in non primitive datatype. Stoichy is a **Statically typed** language, therefore you should explicitly mention the type of the variable when you are declaring it. In stoichy you can't declare and assign the value at the same time.

Syntax to declare a variable :

```
int number;
string message;
boolean check;
element O(8,16.0,-2);
molecule H2O(H,H,O);
```

1.5 Print Statement

We support print statement with minimal syntax.

Syntax to print to stdout :

```
print " Hello, World! ";
print " Stoichy is awesome! ";
```

In stoichy we even support string concatenation hence you can also use print statement like this.

```
print "Hello" ^ "," ^ "World" ^ "!";
```

1.6 Call Keyword

In stoichy you can use **call** keyword to call a function inside another function.
Syntax to call keyword :

```
def message(string s)
{
print "Hi" ^ s;
}


def main()
{
string name = "Justin";
call message(name);
}
```

1.7 Statements

In stoichy we support if-else,while and for loops.

Syntax to declare if-else statement :

```
if(a==0)
{
 print "Number is zero : (";
}
else
{
 print "Number is not zero!!";
}
```

Syntax to declare while loop :

```
while(i!=10)
{
    temp = i%5;
    print temp;
    i=i+1;
}
```

Syntax to declare for loop :

```
for(i=0;i<10;i=i+1)
{
    temp = i%5;
    print temp;
}
```

Chapter 2

Language Reference Manual

2.1 Types

2.1.1 Primitive Types

The four basic types in *Stoichy* are : string, int, double, and boolean.

String :

String is a basic type, its not a collection of characters. A string is a sequence of characters surrounded by double quotes **"..."**. Stings can be concateneated by using $\hat{}$ to form a new string.

Integers :

int data type is represented in 32 - bits and in signed two's complement form. It has a minimum value of -2^{31} and maximum value of 2^{31}. Type conversion is not allowed in stoichy(**strongly typed**) hence error will be reported if we try to change the type.

Double :

double is a double-precision 64 -bit IEEE 754 floating point. Double should be used when we declare decimal values. We didn't support type conversion between a type of int and type of double.

Boolean

The **boolean** data type has binary(two) possible values: true and false.

2.1.2 Non primitive Types

The language comes built-in with **elements, molecules**.

Element :

Element is declared as a tuple (atomic number, mass number, charge). Here atomic number is of int type, mass is of double type and charge is of int type. The element type is fundamental in this language and could be used to create compounds, molecules, etc. Elements are **immutable**.
$^{14}_{7}N$ - **element** N $= (7, 14.0, 0)$;
$^{15}_{7}N$ - **element** N $= (7, 15.0, 0)$

Molecule :

We have implemented a **molecule** using dictionary where the keys of the dictionary are the elements and the values are their frequencies. In our language, there is no difference in the way, a molecule or a compound is declared.
molecule HCl $= (\mathrm{H}, \mathrm{Cl})$

2.1.3 Type Inference

This language is statically typed language hence, we should specify it's type when we are declaring a variable.

2.2 Lexical Conventions

2.2.1 Comments

The syntax for comments in this language is similiar to that of C/C++ language.
Single line comment : // Stoichy is awesome!!
Multiline comment : /* Stoichy is awesome!! */

2.2.2 Identifiers

An identifier is a sequence of digits and characters in which the 1st character must be a lowercase letter. This language is case sensitive.
Examples :
message
name
check
name_1

2.2.3 Keywords

The following words are reserved words and can't be used for any other purposes.

int	element	if	def
double	molecule	else	true
string	call	for	false
boolean	print	while	

2.2.4 Literals

Literals are values written in conventional form whose value can be directly understood . Literals do not change in value. They are constants. An double or integer literal is a sequence of digits. A boolean literal has 2 possible values: true (or) false. A string literal is a sequence of characters. An element literal is a Upper case followed by a optional lowercase. An molecular literal is group of element literals followed by a optional digit.

2.2.5 Punctuation

Punctuation	**Use**	**Example**
,	Dunction name, def parameters	def message(string a,int b)
;	Statements	x=1;
""	String declaration	string x = "Hello, World!"
{}	Statement list and element/molecule declaration	while (expression) statements
()	Conditional parameters, expression precedence	if(x == 0)

2.2.6 Operators

Operator	Use	Associativity
+	Addition	Left
−	Subtraction	Left
∗	Multiplication	Left
/	Division	Left
%	Modulo	Left
^	Concatenate	Left
==	Test equivalence	Left
! =	Test inequality	Left
>	Greater than	Left
<	Less than	Left
>=	Greater than or equal to	Left
<=	Less than or equal to	Left
&&	AND	Left
||	OR	Left
&	BITWISE AND	Left
|	BITWISE OR	Left
.	Access	Left
=	Assignment	Right
if	if	Non associative
else	else	Non associative

2.3 Syntax :

2.3.1 Expressions:

Identifiers:

An identifier can be a primitive type, function or a non primitive type. The type of the identifier should be declared when you are calling it for the first time. The value an identifier can take is restricted by type of identifier but can be changed anywhere inside a program. However identifiers passed as functional arguments can't be changed. Renaming of identifiers is not allowed within the scope of the program.

```
int a ;
double b;
a = b ; -> this is not valid
```

Binary Operators:

- **Logical operators :** Logical operators that we support are **and**(|&|&),**or**(||).

- **Relational operators :** Relational operators that we support are <, >, <=, >=, ==, and !=.
- **Arithmetic operators:** Arithmetic operators that we support are +,-,*, /, %. The operands to an arithmetic operator must be numbers.
- **Assignment operator:** The assignment operator (=) assigns whatever is on the right side of the operator to whatever is on the left side of the operator.
- **Access operator :** If the expression is of non-primitive type, access operator of the form Exp.value will return the value associated with that particular parameter.

Function call:

A function can be called by listing the parameter list inside the parenthesis.
The way to call function is :

```
def NameofFunction{parameter1,parameter2,....}
```

When a user calls a function, the parameters types passed into the function must be the same as those in the function declaration.

2.3.2 Statements:

Selection statements:

The only selection statement in Stoichy is **if-else** statement. It has following syntax :

```
if( expression)
{
 /* Statements to be executed are given here */
}
else
{
 /* Statements to be executed are given here */
}
```

The expression inside if condition should be of Boolean type. If the expression inside if evaluates to true, then the statements within the first set of curly brackets is evaluated. If the expression inside if evaluates to false, then the statements in the curly brackets following else is evaluated.
We also support nested if-else statements.

Example :

```
if ( )
{
    if ( )
    {
         /* Statements to be executed are given here */
    }
    else
    {
        * Statements to be executed are given here */
    }
}
else
{
/* Statements to be executed are given here */
}
```

Iteration statements:

We support two types of iteration statements **for** loop,**while** loop
The expression in the while loop is evaluated, if the expression is evaluated to true then the statements inside the while loop is evaluated. When expression is evaluated to false the loop breaks.

```
while ( expression )
{
/* Statements to be executed are given here */
}
```

The for loop executes the statements inside it as long as for loop condition is satisfied.

```
for (initialisation ; condition ; increment)
{
/* Statements to be executed are given here */
}
```

2.3.3 Built in functions

Balancing Equations

Given an chemical equation(imbalanced), this method computes the correct coefficients for each compound in the chemical equation.

```
balance [H1,Cl2 --> HCl];
```

The output of this balance in built function is the balanced chemical reaction.
Output :

```
2H1,Cl2 --> 2HCl
```

Molar Mass Calculation

Given a molecule, this method calculates and returns the molar mass of the molecule.

```
element H (1,1.0,1);
element O (8,16.0,-2);
molecule H2O(H,H,O);
mass1 = H2O.mass;
print "Mass of H2O molecule is " ^ mass1;
\textbf{Output :}
```

Output of this will the total molar mass of the molecule.

```
Mass of H2O molecule is 18
```

Charge Calculation

Given a molecule, this method calculates and returns the charge of the molecule.

```
element H (1,1.0,1);
element O (8,16.0,-2);
molecule H2O(H,H,O);
charge1 = H2O.charge;
print "Charge of H2O molecule is " ^ charge1;
```

Output of this will the total charge of the molecule. **Output :**

```
Charge of H2O molecule is 0.
```

Chapter 3

Project Plan

- **Weak 0**
  - Translator
  - Stoichiometric calculations like balancing equations,molar mass calcuations etc
  - Planning to have a graphical representation of molecules.
  - We support basic features in cpp like int, bool, list, string, classes lexer and parser - Ocaml lex and Ocaml Yacc because Ocaml has many inbuilt functionalities(like array-list,map,etc) which we need.
  - identification of warnings and errors in representation of eqns and compounds.
- **Week 1**
  - Learnt Ocamllex, Ocamlyacc syntax
  - Completed lexer
  - Written makefile for executing lexer,parser file
  - Written grammar rules
- **Week 2**
  - Language discussions
  - Written parser (running tests)
  - Started white paper draft
  - Reading about AST
- **Week 3**
  - Parser debugging done
  - Completed white paper
  - Updated make file
- **Week 4**
  - Refined white paper draft
  - Fixed more bugs in Parser

  - Completed testcases

- **Week 5**
  - Played with yacc (Mini assignment)
  - Learnt about higher order functions to using in AST.
- **Week 6**
  - Read more about on how to link yacc and lexer
  - trying to figure out how to integrate semantic with parser,lexer
- **Week 7**
  - Discussed about design decisions of semantic analyser
  - Writing recursive procedures for type checking
  - Read about how to do type checking
- **Week 8**
  - Debugged parser issues
  - Semantic version 0 done
  - Testing correctness of semantic analyser
- **Week 9**
  - Working on code generation.
  - Writing balance file
  - Checking correctness of semantic analyser
- **Week 10**
  - Writing Testcases
  - Reviewing code
  - Fixed minor bugs in lexer,parser,semantic
  - Written bash script to evaluate testcases
- **Week 11**
  - Working on ppt,video
  - Demo files written
  - Working on report

We thought of adding graphics , enable typecasting , print line numbers for errors but couldn't do because of time constraints

Chapter 4

Language Evolution

We started off our project keeping some targets that we wanted to achieve like some really cool in-built functions. We wanted to do string error reporting when it finds unterminated string constant. In the first version of lexer, we left this part but later on when were doing code review we added that part. We wanted to declare and initialize at the same place like this :

```
int a = 2;
```

When we were trying to implement this we faced some which we couldn't figure out. We thought that this feature isn't of much importance and hence we left this out. As we were running out of time we felt that, we could leave out on printing line numbers while reporting error. One feature that we wanted to add was type casting. As we were using Ocaml, a functional based language that itself doesn't support typecasting, it became difficult for us to analyse and add the code for the semantic part. After many failed attempts, we began thinking why would we need this feature as we were writing a DSL for stoichiometric calculations. So we left this part out.

We did our code generation in Java. Initially we thinking to do the balance inbuilt function code generation in Java itself, but we realised that it may became too lengthy and we were in search of some way in which we could reduce the code efficiently. We thought of exploring a method, where python script is run in a java file and returns output to the java console and we were successful in doing it. As python has many inbuilt functions, which makes our implementation very simple.

We wanted to implement many built-in function like graphical representation of molecules,elements,moles calculations, but due to the time constraint we couldn't implement them

Chapter 5

System Architecture

The system architecture of stoichy can be divided into:

1. Lexer
2. Parser
3. Semantic Analyser
4. Code generator
5. java compiler

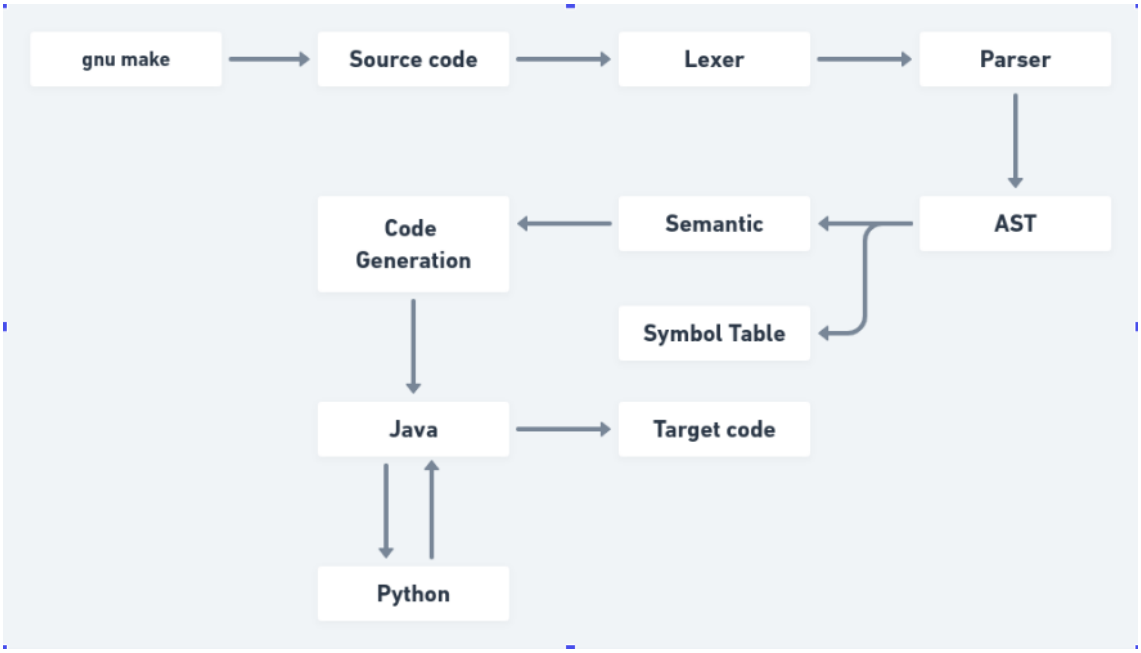

Figure 5.1: Architecture

5.1 Lexer

Stoichy lexer takes input stream from a .sthy program and converts them into tokens. It removes the white spaces and comments in the program. At this stage, errors like invalid characters, incomplete string and incomplete multi line comments are handled. Lexer was written using Ocamllex.

5.2 Parser and AST

Parser takes the tokens generated by the lexer and matches them with grammar rules to form an abstract syntax tree. In parser grammar, there are also exclusive rules for non primitive data types. Some of the non terminals in the grammar are element list ,molecule list.. . Any errors in the syntax will be caught here. Parser was written using Ocamlyacc, hence it is a LALR parser.

5.3 Semantic Analysis

Semantic Analyzer takes the abstract syntax tree generated by the parser, checks if statements and expressions are semantically and syntactically correct. A semantically checked AST is generated. If there are no errors in this stage too, we can use this AST to generate the Java code.

5.4 Intermediate code generation

Most of the code generation is in java and we also used python for balance function in python. Whenever balance function is called, the python script is run and it returns the output to the java console. All the code is put into a java file which contains the Java source code.The generated java code is compiled by javac command in makefile. The primary reason for choosing Java for doing our code generation is as it is machine independent.

Chapter 6

Development environment

6.1 GNU Make

We used GNU make as one of the tool as we had many files and many things to compile and **make** made our work very easy from compiling and converting all the files into objects to cleaning all the intermediaries, **make** did a very good job. Before knowing about the make file (for first 2 weeks), we used to compile and type each and every command where we used to do a lot of mistakes and hence thought of an alternative, we were amazed by the idea of make and chose to include in our Development environment. The advantages that we learned about make are:

- Make enables the end user to build and install your package without knowing the details of how that is done – because these details are recorded in the makefile that you supply.
- Make figures out automatically which files it needs to update, based on which source files have changed. It also automatically determines the proper order for updating files, in case one non-source file depends on another non-source file.
  As a result, if you change a few source files and then run Make, it does not need to recompile all of your program. It updates only those non-source files that depend directly or indirectly on the source files that you changed.
- GNU Make has many powerful features for use in makefiles, beyond what other Make versions have. It can also regenerate, use, and then delete intermediate files which need not be saved.

Our final dependency graph looks like the one in the below figure.

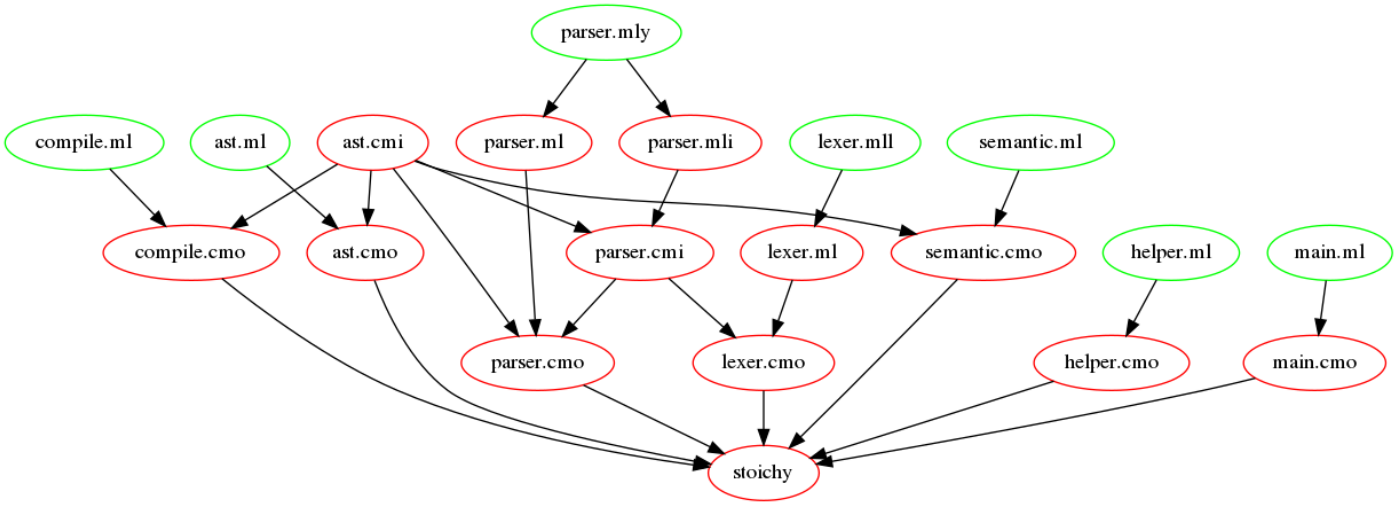

Figure 6.1: Makefile Dependencies

6.2 Bash

We used bash as a part of our testing to run and check correctness of all testcases that we wrote.As we used many modules and libraries from Ocaml and sympy from python. We had written a bash script where a user can relentlessly just run a bash script and it does it's job of downloading all the required things.

6.3 Git

We used Git as our version control system. Where we pushed our work into a remote on GitHub whenever one of us finished working on some feature, we have done some code review and then modified pulled, pushed etc. Our commits are shown in Appendix 9.1

6.4 VSCode

We got into many problems and some of them wanted to be solved at very quickly. All of us Used VSCode particularly **liveshare** when we ran into problems. We were sometimes successfull in rectiying the mistakes. As VSCode provided liveshare which had many nice featues such as Terminal sharing and Code sharing, syntax highlighting etc, we included it in one of our Tools.

Chapter 7

Test Plan and Test suites

7.1 Introduction

We have made a good no. of test cases, which handles all syntax structures (like arithmetic, loops, conditional statements, etc) by running which, we can make sure that the project is in a good state.
To be fast and accurate in testing, we have decided to do our testing an automated process.
So we have also created expected output files for each test case and had wrote a bash shell script(run_test.sh), which takes all test cases, executes them and matches the output with the expected output files using the diff command.

```
#!/usr/bin/env bash

demo_files="demo/*.sthy"
test_files="test_cases/*.sthy"
ran=0
success=0

Compare() {
  diff -bq "$1" "$2" && {
    (( success++ ))
    echo "PASS"
  } || {
    cat "$1"
    echo "FAILED:- not matching with expected output"
  }
}


for f in $test_files
do
  (( ran++ ))
  name=${f%.sthy}     # remove .sthy from the end
  name=${name#test_cases/}  # remove ./test_cases/ from the beginning
  exp=${f%$name.sthy}"expected/$name.out"   # insert expected/ into file path
  echo "====================="
  echo "Testing: $name"
  ./stoichy "$f" > "test_cases/$name.out" 2>&1 && {
  Compare "test_cases/$name.out" "$exp"
  } || {
    cat "test_cases/$name.out"
```

```
    echo "FAILED: did not compile"
  }
done


for f in $demo_files
do
  (( ran++ ))
  name=${f%.sthy}     # remove .sthy from the end
  name=${name#demo/}  # remove ./demo/ from the beginning
  exp=${f%$name.sthy}"expected/$name.out"   # insert expected/ into file path
  echo "====================="
  echo "Testing: $name"
  ./stoichy "$f" > "demo/$name.out" 2>&1 && {
  Compare "demo/$name.out" "$exp"
  } || {
    cat "demo/$name.out"
    echo "FAILED: did not compile"
  }
done

echo "====================="
echo "SUMMARY"
echo "Number of tests run: $ran"
echo "Number Passed: $success"
```

Listing 7.1: run_tests.sh bash script

7.2 Demo files

```
def main()
{
    print "Hello, World!";  // Print statement
}
```

Listing 7.2: Hello World test

```
Hello, World!
```

Listing 7.3: Expected output

```
def checkeven(int a)
{
    if(a%2==0)              // Checing the modulous value
    {
        print "Hurray! " ^ a ^ " is an even number : )";
    }
    else
    {
        print "Oh no! " ^ a ^ " is a odd number : (";
```

```
    }
}
def main()
{
    int a;
    int b;
    a = 4;
    b = 5;
    call checkeven(a);      // Calling checheven function
    call checkeven(b);      // Calling checheven function
    print a>2;
    print b<=7;
}
```

Listing 7.4: Methods, Booleans, conditional operators test

```
Hurray! 4 is an even number : )
Oh no! 5 is a odd number : (
true
true
```

Listing 7.5: Expected output

```
def main()
{
    int a;
    int i;
    int j;
    int temp;
    a = 10;
    i = 0;
    j = 0;

    while (i<=a)
    {
        temp = i-a;
        print temp;
        i=i+1;
    }
    print "-----";
    for (i=1;i<=a;i=i+1)
    {
        temp = i*1;
        print temp;
    }

}
```

Listing 7.6: for and while loops

```
-10
-9
```

```
-8
-7
-6
-5
-4
-3
-2
-1
0
-----
1
2
3
4
5
6
7
8
9
10
```

Listing 7.7: Expected output

```
def main()
{
    int charge1;
    int charge2;
    double mass1;
    double mass2;
    element Na (11,23.0,1);
    element Cl (17,35.5,-1);
    element H (1,1.0,1);
    element O (8,16.0,-2);
    molecule NaCl(Na,Cl);
    molecule H2O(H,H,O);
    charge1 = NaCl.charge;
    charge2 = H2O.charge;
    print "Charge of NaCl molecule is " ^ charge1;
    print "Charge of H2O molecule is " ^ charge2;
    mass1 = NaCl.mass;
    mass2 = H2O.mass;
    print "Mass of NaCl molecule is " ^ mass1;
    print "Mass of H2O molecule is " ^ mass2;
}
```

Listing 7.8: Non primitve data types and basic unbuilt functions test

```
Charge of NaCl molecule is 0
Charge of H2O molecule is 0
Mass of NaCl molecule is 58.5
Mass of H2O molecule is 18.0
```

Listing 7.9: Expected output


```
def main()
{
    string a;
    string b;
    string c;
    string d;
    string e;

    // Balancing unbalanced chemical equations
    a = balance [H1,Cl2 --> HCl];
    b = balance [PCl5,H2O --> H3PO4,HCl];
    c = balance [KBr,KMnO4,H2SO4 --> Br2,MnSO4,K2SO4,H2O];
    d = balance [SnO2,H2 --> H2O,Sn1];
    e = balance [H2-->O2];
    print a;
    print b;
    print c;
    print d;
    print e;
}
```

Listing 7.10: Main core usage(Balancing of equations) test

```
2 H1 + 1 Cl2 --> 2 HCl
1 PCl5 + 4 H2O --> 1 H3PO4 + 5 HCl
10 KBr + 2 KMnO4 + 8 H2SO4 --> 5 Br2 + 2 MnSO4 + 6 K2SO4 + 8 H2O
1 SnO2 + 2 H2 --> 2 H2O + 1 Sn1
Invalid Equation!!!
```

Listing 7.11: Expected output

7.3 Test cases

```
// Test 1 : Empty main block

def main(int a)
{

}
```

Listing 7.12: test1

Listing 7.13: Expected output

```
// Test 2 : Primitive Datatypes declaration and printing

def main()
{
  int a;
  int b;
```

```
  string s;
  boolean c;

  a = 0;
  b = 1;
  s = "Stoichy is awesome !";
  c=true;

  print a;
  print b;
  print s;
  print c;
}
```

Listing 7.14: test2

```
0
1
Stoichy is awesome !
true
```

Listing 7.15: Expected output

```
// Test 3: Arithmetic Expressions

def main()
{
  print 100;
  print 2020;


  print 1+2;
  print 1- 2;
  print 1*2;
  print 1/2;
  print 1%2;

  print -3- 39;
  print 14*-3;
}
```

Listing 7.16: test3

```
100
2020
3
-1
2
0
1
-42
-42
```

Listing 7.17: Expected output

```
// Test 4 : Printing strings

def main()
{
    string first;
  string a;
  string b;
  string c;
    print " Printing without using concatenation";
    first = "Hello, world!";
    print first;
  a = "Hello";
  b = "world";
  c = "!";
    print " Printing by using concatenation";
  print a ^ ", " ^ b ^ c;

}
```

Listing 7.18: test4

```
 Printing without using concatenation
Hello, world!
 Printing by using concatenation
Hello, world!
```

Listing 7.19: Expected output

```
// Test 5 : If-Else statements

def main()
{
    int a;
  a=10;
  if(a==0)
  {
    print "Number is zero : (";
  }
  else
  {
    print "NUmber is not zero!!";
  }
}
```

Listing 7.20: test5

```
NUmber is not zero!!
```

Listing 7.21: Expected output

```
// Test 6 : Nested if-else statements

def main()
```


```
{
    int a;
    int b;
    a=2;
    b=1;
    if(a!=b)
    {
        print "a is not equal to b";
        if(a>b)
        {
            print "a is greater than b";
        }
        else
        {
            print "b is greater than a";
        }
    }
    else
    {
        print "a is equal to b";
    }

}
```

Listing 7.22: test6

```
a is not equal to b
a is greater than b
```

Listing 7.23: Expected output

```
// Test 7 : While statement

def main()
{
    int i;
    int temp;
    i=0;
    while(i!=10)
    {
        temp = i%5;
        print temp;
        i=i+1;
    }
}
```

Listing 7.24: test7

```
0
1
2
3
```


```
4
0
1
2
3
4
```

Listing 7.25: Expected output

```
// Test 8 : For loop

def main()
{
    int i;
    int temp;
    for(i=0;i<10;i=i+1)
    {
        temp = i%5;
        print temp;
    }
}
```

Listing 7.26: test8

```
0
1
2
3
4
0
1
2
3
4
```

Listing 7.27: Expected output

```
// Test 9 : Call key word

def multipleof5(int a)
{
    if(a%5==0)              // Checing the modulous value
    {
        print "Hurray! " ^ a ^ " is multiple of 5 : )";
    }
    else
    {
        print "Oh no! " ^ a ^ " is not multiple of 5 : (";
    }
}
def main()
{
```


```
    int a;
    int b;
    a = 4;
    b = 10;
    call multipleof5(a);      // Calling multipleof5 function
    call multipleof5(b);      // Calling multipleof5 function
}
```

Listing 7.28: test9

```
Oh no! 4 is not multiple of 5 : (
Hurray! 10 is multiple of 5 : )
```

Listing 7.29: Expected output

```
// Test 10 : Non primitive datatyped declarations

def main()
{
    int a;
  element Na(11,23.0,1);
  element Cl(17,35.5,-1);
    element H(1,1.0,0);
    element O(8,16.0,-2);
  molecule NaCl(Na,Cl);
  molecule H2O(H,H,O);
    print "Elements and molecules are created succesfully!";
}
```

Listing 7.30: test10

```
Elements and molecules are created succesfully!
```

Listing 7.31: Expected output

7.4 Error cases

```
// Test 13 : Error file shld report error

def main()
{
    /* Opened a comment but didn't close it
}
```

Listing 7.32: test8

```
// Test 14 : Error file shld report error

def main()
{
    string a;
    // Didn't put ' " ' in string
    a="hi
}
```

Listing 7.33: test9

```
// Test 15 : Error file shld report error

def main()
{
    print " Didn't place ';' after print statement "
}
```

Listing 7.34: test10

Chapter 8

Conclusion & Lessons Learned

It was wholly a good experience for us. It firstly weird when we heard that, we didn't know much about all the parts of the compiler along with it we even had to write some new language. Then came the period where sir asked us to form groups and come up with an idea of what we write, i.e a DSL or a compiler or Translator. We felt that it would be very exciting of writing something thoughtown. In first week we searched for so many ideas on what to do. Being inspired from the functional language from POPL course, we thought in that direction. Then we thought of problems we faced in college and all of us where like it would be nice if we do something related to chemical equation calculations. There came the Idea of **Stoichy**. We started working on it. There were times when we weren't able to debug the code and looked to many blogs and websites such as stackoverflow. Majorly we worked toghether on weekends. As we were a group of 5, arranging time for project was a difficult task.

As we used Github for version control it became easier to see what others have done and review/modify the code. Our major learnings were about bash scripting, writing make files. Working as a team member was a pretty good experience for all of us. When we were working on balancing function, we thought of doing the code generation in Java and we later realised that it was too lengthy. We started thinking of is there any way to make it simpler. Thats where we learnt about executing the Python code from java by passing the equation as argument. We felt pretty awesome after the Python code was working as it was easier to write in python using its vast linear algebra libraries. Finally we understood about the difficulty in writing a language and design decisions involved in it.

Chapter 9

Appendix

9.1 Project Log

```
commit 55302df623075b0d8a5d6885da2de696e00ab6c4
Author: K Vamshi Krishna Reddy <70333890+kvkr3003@users.noreply.github.com>
Date:   Wed Dec 16 22:53:10 2020 +0530

    Add files via upload

commit db6e040ba84c30dc411d284fc1bbc1200582a2da
Author: havya7 <61291792+havya7@users.noreply.github.com>
Date:   Wed Dec 16 22:48:21 2020 +0530

    Add files via upload

    Formatted code

commit effa7eab5ec161c60c0ccd32ebc7122b61a0cd6c
Author: havya7 <61291792+havya7@users.noreply.github.com>
Date:   Wed Dec 16 22:46:45 2020 +0530

    Delete semantic.ml

commit ff2644384782223fd9c3626902a7a4fd6709398e
Author: havya7 <61291792+havya7@users.noreply.github.com>
Date:   Wed Dec 16 22:46:10 2020 +0530

    Create semantic.ml

commit b9d281bcffef7ed2acf3d8686aa771b3b23dc533
Author: havya7 <61291792+havya7@users.noreply.github.com>
Date:   Wed Dec 16 22:44:53 2020 +0530

    Delete compile.ml

commit e04b182138d3e1fa486452abc3221e2f3be855d1
Author: havya7 <61291792+havya7@users.noreply.github.com>
Date:   Wed Dec 16 22:43:53 2020 +0530

    Delete semantic.ml
```

```
commit b820fde284e783d52c1f0b786eaac9429e20a315
Author: K Vamshi Krishna Reddy <70333890+kvkr3003@users.noreply.github.com>
Date:   Wed Dec 16 22:40:28 2020 +0530

    Add files via upload

commit 21fee92f911dc914d9ddb885ceeeb7ad248d9cb1
Author: sai_varshittha <saivarshitthaponnam7@gmail.com>
Date:   Wed Dec 16 21:26:35 2020 +0530

    formatting done

commit 25b88c296d061c8a9db014c9423120c05a422293
Author: sai_varshittha <saivarshitthaponnam7@gmail.com>
Date:   Wed Dec 16 21:26:18 2020 +0530

    added comments

commit 0222c632249d1f5b1ad0fb2c938d1577734fac3d
Author: Akash Tadwai <akashadarsh.tadwai@gmail.com>
Date:   Wed Dec 16 21:02:51 2020 +0530

    finalising....

commit 4445616be826248b068f6b7565cb916a6d568ae0
Author: Akash Tadwai <akashadarsh.tadwai@gmail.com>
Date:   Wed Dec 16 20:07:06 2020 +0530

    Changed README.md

commit 792e8df785dae92de02f67db2e28ec39e40beaab
Author: ReethuVinta <es18btech11028@iith.ac.in>
Date:   Wed Dec 16 18:29:55 2020 +0530

    Added test cases

commit 6cfc647595e7abe9a36f2ac8d1183de5f413f8a3
Author: Akash Tadwai <akashadarsh.tadwai@gmail.com>
Date:   Wed Dec 16 16:59:52 2020 +0530

    minor changes in code

commit cd725f3d96a183903892bf6d278c17f6dde7a695
Author: Akash Tadwai <akashadarsh.tadwai@gmail.com>
Date:   Wed Dec 16 00:24:56 2020 +0530

    Pre reqs bash script added

commit db1815a1c3d958f80bcecc604fe2b7701c47c489
Author: ReethuVinta <es18btech11028@iith.ac.in>
```


```
Date:   Tue Dec 15 21:37:36 2020 +0530

    Final Demo Version

commit 7a69e210ab9f323e1f455e41bb00925c92be3b31
Author: ReethuVinta <es18btech11028@iith.ac.in>
Date:   Tue Dec 15 20:07:57 2020 +0530

    Fixed yet another bug :P

commit 324cc797d27a9ed29c5bf098dc16ada01aa857e0
Author: ReethuVinta <es18btech11028@iith.ac.in>
Date:   Mon Dec 14 23:34:36 2020 +0530

    Fixed another bug :PP

commit a323f122b1b14846273c90dda42f0d0b1dc8813b
Author: ReethuVinta <es18btech11028@iith.ac.in>
Date:   Mon Dec 14 23:04:40 2020 +0530

    Fixed small bug in lexer :p

commit 384e151fbc4f56edec236e2bf1f06524ff6c757d
Author: Akash Tadwai <akashadarsh.tadwai@gmail.com>
Date:   Mon Dec 14 21:34:49 2020 +0530

    Demo Version done

commit c5911f9c36b3a599ed56d628184695ee1dfbc94d
Author: Akash Tadwai <akashadarsh.tadwai@gmail.com>
Date:   Wed Dec 9 12:00:12 2020 +0530

    Checking for invalid eqns done

commit b9ce3b0789620720691dffadf32ff155af5ae29f
Author: Akash Tadwai <akashadarsh.tadwai@gmail.com>
Date:   Wed Dec 9 01:16:30 2020 +0530

    Balancing using Python3 :snake:

commit e99da62b3dfbc06389ceb4bd0b77322d242ac461
Author: Akash Tadwai <akashadarsh.tadwai@gmail.com>
Date:   Wed Dec 9 01:16:05 2020 +0530

    Balancing using Python3 :snake:

commit 651891cf71837042e21af7cfa8cd0683537958f2
Author: ReethuVinta <es18btech11028@iith.ac.in>
Date:   Tue Dec 8 21:41:21 2020 +0530

    CodeGen V2
```


```
140
141 commit b44086cf4cb49f6df53ee5103bed995f21255eca
142 Author: ReethuVinta <es18btech11028@iith.ac.in>
143 Date:   Tue Dec 8 21:01:46 2020 +0530
144
145     CodeGen V2
146
147 commit 5f008fea34b2d6131163288b975db083139489b6
148 Author: ReethuVinta <es18btech11028@iith.ac.in>
149 Date:   Tue Dec 8 21:01:38 2020 +0530
150
151     CodeGen V2
152
153 commit cc8a7fcf465b2d475d5bd310dcd9e046f8fdadda
154 Author: ReethuVinta <es18btech11028@iith.ac.in>
155 Date:   Tue Dec 8 20:45:36 2020 +0530
156
157     Added
158
159 commit 9779b4aa62d85b320aa8579e85ea8ba507c460df
160 Author: ReethuVinta <es18btech11028@iith.ac.in>
161 Date:   Tue Dec 8 20:43:40 2020 +0530
162
163     Added
164
165 commit d0267ad9775cddaf8a3c42862fb025f1ef1d3b1b
166 Author: ReethuVinta <es18btech11028@iith.ac.in>
167 Date:   Tue Dec 8 20:22:10 2020 +0530
168
169      added
170
171 commit 11932f00f92f0070912fc071c03605c7d8557f8c
172 Author: ReethuVinta <es18btech11028@iith.ac.in>
173 Date:   Tue Dec 8 18:35:59 2020 +0530
174
175     Updated makefile
176
177 commit 2cefc1db4cbbd0998af2b3cac7cccbae098a7459
178 Author: ReethuVinta <es18btech11028@iith.ac.in>
179 Date:   Tue Dec 8 18:17:20 2020 +0530
180
181     Code generation V1
182
183 commit 11c3dc7cf618d47425a20edbd221d02600abbcf2
184 Author: ReethuVinta <es18btech11028@iith.ac.in>
185 Date:   Wed Nov 25 21:17:14 2020 +0530
186
187     Semantic V0
188
189 commit ea0cfa1ce18ed0d6858b054d672b7a4f7c8fefe4
190 Author: Akash Tadwai <akashadarsh.tadwai@gmail.com>
```


```
Date:   Wed Nov 25 18:51:40 2020 +0530

    semantic v2.0

commit 704e4fcd647aee601cb0f11e63dfcc5f0aea6c53
Author: Akash Tadwai <akashadarsh.tadwai@gmail.com>
Date:   Wed Nov 25 13:59:41 2020 +0530

    debugging...

commit 5c6936dd39e5b8a5ef912ff44aaa4a8b75bb73c3
Author: ReethuVinta <es18btech11028@iith.ac.in>
Date:   Sat Nov 21 17:59:39 2020 +0530

    Version 0 Semantic

commit 438c6003e315c603c9cc4f44a715893f194c8abe
Author: Akash Tadwai <akashadarsh.tadwai@gmail.com>
Date:   Wed Oct 28 12:40:29 2020 +0530

    Update README.txt

commit 79a408eaeaf79e9e4b5cbd615e3559ccda42a0aa
Merge: b4e347a 6b040f4
Author: ReethuVinta <es18btech11028@iith.ac.in>
Date:   Wed Oct 28 12:34:10 2020 +0530

    Merge https://github.com/akashtadwai/stoichy

commit b4e347a986f8a8b30ec5b5c8ee6c71d3a2e71f5a
Author: ReethuVinta <es18btech11028@iith.ac.in>
Date:   Wed Oct 28 12:22:43 2020 +0530

    Added incorrect programs

commit 6b040f4f4b01e4cbcf38b861f8a18f07803c1eda
Author: Akash Tadwai <akashadarsh.tadwai@gmail.com>
Date:   Tue Oct 27 21:15:56 2020 +0530

    added README.txt

commit 2d97736be3d35e4f637291f10546903f2757c9ef
Author: ReethuVinta <es18btech11028@iith.ac.in>
Date:   Tue Oct 27 21:00:45 2020 +0530

    Added correct testcases

commit b99f3e355a2d50c6d2f1b2927732fff5297a5100
Author: ReethuVinta <es18btech11028@iith.ac.in>
Date:   Tue Oct 27 17:52:10 2020 +0530
```

```
    Version1 changes

commit f314488dcaf3ab86ee2959bc981752b88e892c8f
Author: ReethuVinta <es18btech11028@iith.ac.in>
Date:   Sat Oct 24 11:45:24 2020 +0530

    Parser debugging done

commit cde3809b7f0297fcda01c5ea084b1cd79ed0c281
Author: sai_varshittha <saivarshitthaponnam7@gmail.com>
Date:   Tue Oct 13 22:50:19 2020 +0530

    parser 11

commit 04f656b3f50cc6949328c25f9b89084f909b006a
Author: Akash Tadwai <akashadarsh.tadwai@gmail.com>
Date:   Sat Oct 10 23:07:04 2020 +0530

    lexer v2.0

commit e857a646f04c5949d951fde1ad7b4a32002ca392
Author: Akash Tadwai <akashadarsh.tadwai@gmail.com>
Date:   Thu Oct 8 23:27:38 2020 +0530

    makefile for basic lex and parser testing

commit 7a311775d264cb84c6be13ff488c96c8efa39145
Author: Akash Tadwai <akashadarsh.tadwai@gmail.com>
Date:   Sat Oct 3 18:37:27 2020 +0530

    lexer v1.0
```

Listing 9.1: Project Log from GitHub

9.2 Code Listing

All the codes can be found on our Github Page https://github.com/akashtadwai/stoichy