ChemLab Project Proposal
COMS W4115

Alice Chang (Syst. Architect, Lang Guru), Gabriel Lu (PM, Lang Guru), Martin Ong (Syst. Architect, Tester)
avc2120, ggl2110, mo2454

September 24, 2014

1. Description

ChemLab is a functional language that allows users to conveniently manipulate chemical elements. It can be used to solve physical and organic chemistry problems including, but not limited to, stoichiometeric calculations, oxidation-reduction reactions, acid-base reactions, gas stoichiometry, chemical equilibrium, thermodynamics, stereochemistry, and electrochemistry. It may also be used for intensive study of a molecule's properties such as chirality or aromaticity.
The language will make solving such problems easier by using data types and functions that are specifically designed to suit the needs of a chemist. A functional language was implemented because of the way it treats computation as the evaluation of mathematical functions and avoids changing state and mutable data. It makes sense for the output value of a function to depend only on the arguments' inputs in a chemical context because, given a set of input elements, a reaction will always produce the same output molecule no matter how many times the reaction is run. Overall, we hope to use the simplicity of a functional language to help chemists and students easily solve problems. While the language currently has a focus on chemistry, one may easily foresee later versions including utilities for other scientific areas such as biology or physics.

2. Proposed Uses

ChemLab is used to easily solve a variety of chemistry and organic chemistry related problems. These questions are mostly procedural and there is a general approach to solving each specific type of problem. For example, to determine the molecular formula of a compound: 1) use the mass percents and molar mass to determine the mass of each element present in 1 mole of compound 2) determine the number of moles of each element present in 1 mole of compound. Albeit these problems can generally be distilled down to a series of plug-and-chug math calculations, these calculations can become extremely tedious to work out by hand as molecules and compounds become more complex (imagine having to balance a chemical equation with Botox: $C_{6760}H_{10447}N_{1743}O_{2010}S_{32}$). Our language can be used to easily create programs to solve such problems through the use of our specially designed data types and utilities.

3. Syntax

3.1 Comments
ChemLab allows for single and multiline comments

Operator	Description	Example
//	Single-line comment	// hello world, except not really

/* */	Multi-line comment	/* Hello world, except not really*/

3.2 Variable Declarations

Variables can only be declared globally. A global variable is declared using the assignment operator (=). The name on the left hand side of the assignment operator is given the value of whatever is on the right side of the operator. When you declare a variable, you must also initialize it.

```
int n = 8;
int n; //(invalid)
```

3.3 Types

Strings use only double quotes.

```
String word1 = "chemistry";
String word2 = " lab"
String concat = "chemistry lab";
//syntactic sugar allows easy concatenation of Strings
```

Numbers can either be int or doubles. If an integer and a double are in the same expression, the integer is automatically typecasted into a double

```
int n = 8;
double k = 10.0
double z = n + k // (z = 18.0)
```

Boolean can only be true or false, there is no null value;

```
boolean b1 = true;
boolean b1 = false;
boolean b1 = null; // (invalid)
```

Arrays declared using [ ] in which the elements of the array must all be of the same type. When assigning a value to an array, put the values in parenthesis in which the elements are separated by commas.

```
double[] k = [1.0,2.0,3.0,4.0,5.0];
String[] lastNames = ["Chang", "Lu", "Ong"];
```

3.3.1 Element

Since there are only 118 elements, it could have been possible to hard code each element into the language. However, we chose not to do this to give the user a greater degree of flexibility in terms of declaring the properties of the element they want to consider because isotopes of elements have different amounts of neutrons and some elements can exist in more than one state. Element is declared with (atomic number, mass number, charge). The element type is the basic building block provided by the program that can be used to create molecules, compounds, etc.

$^{12}_{6}C$

```
element c = {6, 12, 0};
```

$^{14}_{6}C$

```
element c = {6, 14, 0};
```

3.3.2 Molecule

For the purpose of the language, there is no distinction between molecule or compound and both are declared the same way. Molecule is declared as a Hashmap with the elements as keys and the number of each element as the values.

```
molecule NaCl = {Na: 1; Cl:1} //declared as a Hashmap
```

Note that the only time curly brackets { } are syntactically accepted is in the declaration of an element, molecule, or equation.

3.3.3 Equation

Equation is declared in the following way: An array consisting of an array of elements/molecules on left side of reaction and an array of elements/molecules on right side of reaction.

$NaOH + HCl \rightarrow NaCl + H_20$

```
equation NaClReaction = [[NaOH, HCl], [NaCl, H2O]];
```

3.4 Operators

Arithmetic Operators

Operator	Description	Example
=	Assignment Operator	String a = "hello"
+	Addition	2+3
-	Subtraction	4+5
%	Modulus (taking remainder)	3%4
/	Division	5/5
*	Multiplication	8*8

Comparison Operators

Operator	Description	Example
< or >	Greater than or less than	5>4
<= or >=	Greater than or equal to, less than or equal to	5>=5

Boolean Operators

Operator	Description	Example
==	Check for equality	10 == 10
&&	AND	True && True = True
| |	OR	True || False = True
!, ~	NOT	10!=5

3.5 Control Flow

ChemLab will include basic control flow. We will not include a for loop because it will be redundant when there is already a while loop.

```
//if-else
if(){
}
else{
}
//while loop
while( expression ){
}
```

3.6 Functions

Functions require name of function, type of return, and type of parameters. The keyword func must be declared ahead of the function to signify that it is a function declaration.

```
func int add(int x, int y){
       return a; (a is of type int)
}
```

3.7 Utilities

3.7.1 Balance Equations

Given an unbalanced equation, this utility will be able to compute the correct coefficients that go in front of each molecule to make it balanced and return as an array of hashmaps

```
balance([[Fe,Cl2],[FeCl3]]);
//returns [{Fe:1, Cl:3},{FeCl3:2}]
```

3.7.2 Molar Mass Calculation

Given a molecule, this utility will be able to compute the total molar mass of the molecule

```
molarMass(H);
//returns 1.00794
```

3.7.3 Naming of Molecules

Given a molecule, the utility will print out the name in correct scientific notation (ex. H_20 will be printed as Dihydrogen Monoxide)

```
name(NaCl);
//returns “Sodium Chloride”
```

3.7.4 Printing of Equations

Given an equation, the utility will print out the equation in correct scientific notation

```
equation x = [{Na:1, Cl:1},{NaCl:1}];
print(x);
//prints out Na + Cl → NaCl
```

3.7.5 Amount of Moles

Given the element and the amount of grams of the element, this utility will return the amount of moles of the element

```
numMoles(Fe, 27.9225);
//calculates number of moles of Fe in 27.9225 grams of Fe
```

3.7.6 Solubility

Given a molecule, the utility will apply pattern matching and return a boolean value of whether the molecule is soluble or insoluble

```
soluble(NaCl);
//returns True
```

4. Example Programs

```
//Oxidation Reduction Problem
element Zn = {30, 65, 0};
element Zn2 = {30, 65, 0};
```

```
element Fe = {26,56,0};
element Fe2 = {26,56, 2+};
element FeCl2 = {Fe:1;Cl:2};
equation reaction = [[FeCl2,Zn],[ZnCl2, Fe]];

func String Reduced(element x_react, element x_prod, element y_react, element y_prod)
{
	String output = “”;
	if(x_prod.charge – x_react.charge > 0)
		output = output + name(x_react) + “ is reduced and ”;
	else
		output = output + name(x_react) + “ is oxidized and ”;

	if(y_prod.charge – y_react.charge)
		output = output + name(y_react) + “ is reduced”;
	else
		output = output + name(Y_react) + “ is oxidized”;
	return output;
}

oxiRed(reaction[0][0][‘Fe’], reaction[0][1], reaction[1][0][‘Zn’], reaction[1][1]);
//returns “Zinc is oxidized and Iron is reduced”

//Equilibrium Problem solving for Kc
element O = {8, 16, 2-};
element S = {16, 32, 2-};

molecule SO3 = {S:1; O:3};
molecule SO2 = {S:1; O:3};
molecule O2 = {O:2};
equation equilibrium = [[SO2,O2],[SO3,1]];

func double calculateKc(equation equ)
{
	balanced = balance(equ);		//stores [{SO2:2;O2:1},{SO3:2}] in balance
	int[] coeff1 = balanced[0].values; //stores values of reactants hashmap in array
	int[] coeff2 = balanced[1].values; //stores values of products hashmap in array
	numerator = equ[1];
	denominator = equ[0];
	int max = equ[1].size;
	int count = 0;
	kc = 0;
	while(count < max)
	{
		kc = kc * numerator[count]^(coeff1[count]);
		count = count + 1;
	}
	count = 0;
	int max = equ[0].size;
	while(count < max)
	{
		kc = kc / denominator[count]^(coeff2[count]);
		count = count + 1;
	return kc;
```

```
        }
}

calculateKc(equilibrium); //returns value of Kc for 2SO2 + O2 → 2SO3
```

5. Looking Ahead

Although the program is currently designed to implement solutions for chemistry problems, one could foresee the implementation of solutions for other scientific subjects such as biology or physics. For example, one could implement a type called amino acid that could be any of the 20 amino acids and type called nucleotide which could be any of the 5 nucleotides (C,G,T,A,U). One could then develop useful functions such as a DNA conservation function to determine the degree of sequence conservation between two DNA sequences. One could also see using the language to study protein structure in which one would have a graph of amino acids which show which amino acids in the structure are contacting other amino acids. Although this is currently just wishful thinking, it would be interesting to implement these details in the language.