Chemlogic: Project Notes

Nicholas Paun

May 7, 2015

Contents

Part I
Implementation Notes

1 Input/Output Formatting

The new output formats work almost perfectly.
Just a few this are left to deal with.

Polyatomic groups:

1. Polyatomic groups like NH4 are built once at runtime.
2. They have to be in user format, so that someone can actually enter them in.
3. But, we cannot render subscripts for them, as they are pre-built.
4. Possible solutions:
   (a) One polyatomic group fact per output mode:
      i. Slow?
   (b) The polyatomic group facts should dynamically call the correct output formatter?
      i. Really slow.

LaTeX mode will require a wrapper program to allow Chemlogic to be used to simplify document and paper writing.
Interfaces should set the correct output format on startup.

2 Parser Improvements and Efficiency

- Exceptions seem like a good way of raising errors.
  - They are clear and systematic.
  - They can be handled at the right level, and then re-thrown to be formatted for output.
- Current parser code triggers several exceptions during the course of normal operation.
  - This should never happen.
- We must avoid triggering redos while parsing. It causes a massive amount of repetition, which slows everything down.

- When the user enters a compound of unknown type, some issues occur:
  - Ionic expects a metal followed by a non-metal. If it is actually covalent, an error occurs.
  - When formulas are entered, the same sort of thing occurs.
- Parsers can be made more efficient by using tail rules, that could either add another formula, for example, or expand to nothing. This avoids redos.

3 Code Style

3.1 File Naming

ALL CAPS = plain-text documentation
all lowercase, no extension = user interfaces, or programs that could be embedded into other systems
*.pl = Prolog code files, libraries

3.2 Variable and Predicate Naming

- Prefix internal variables and functions used for formatting, debugging, workarounds, etc., with a _.
- Keep variable names consistent:
  - 'Sym' for symbol
  - 'Formula' for formula
  - 'Elem' for element
  - 'Mol' for molecule
  - 'Coeff' for coefficient
  - 'Subs' for subscript
  - 'Qty' for quantity (stoichiometry)
  - 'Fmt' for format
- No singletons.
- When processing a list, use the pattern X, XS where possible.
  - Otherwise use H, T (with a prefix like OutH, OutT)
- Do not give functions meaningless names.
  - No functions ending in _2, _real, etc.

- Some terms might be ambiguous: Result, Output, Value, Coeff, Vars, Qty.
  - Try to be precise about what they mean.
- Rest, R, R0, FinalRest, and other variables are just noisy. They should be abstracted away somehow.

3.3 Web Interface

- x_page routes the request.
- x_input defines input parameters.
- x_process sets up the query to the underlying APIs and converts the result to a HTML structure.
- x_do_process actually queries the APIs.
- x_nop shows the prompt message instructing the user to enter their input.
- x_html generates the page.
- Do not use raw HTML (except in formatting functions, as they may be used by other interfaces), use SWI-Prolog HTML structures instead.

3.4 Indentation in Prolog Code

- One line between different clauses of the same predicate.
- Two lines between different predicate.
- Spaces around all operators.
- No trailing spaces
- Clause body is indented with a tab.
- One line per call.
  - Excluding !, fail, true, etc.
  - Sometimes it is better to combine ';' and '->' clauses on one line.
- Parenthesis or brace blocks:
  - Blank line before and after
  - Indent contents one level

4 Ideas for the Use of Metaprogramming

- An operator to avoid the need to repeat the arguments of predicates:
  - Current:

```
name(Sym,Rest,Formula) --> ionic(Sym,Rest,
    Formula).
name(Sym,Rest,Formula) --> covalent(Sym,Rest,
    Formula).
```

  - Simplified:

```
name" --> ionic".
name" --> covalent".
```

- Simple syntax for creating maplist-type functions:
  - Current:

```
double([],[],_).
double([InH|InT],[OutH|OutT],Factor) :-
        OutH is Factor * InH,
        double(InT,OutT,Factor).
```

  - Simplified:

```
double([In],[Out],Factor) :[]
        Out is Factor * In.
```

- Something should be done to reduce the complexity of DCG syntax when lots of structures are created.
  - Example of highly complex code:

```
expr(Fmt,Coeff,CoeffR,Elems,ElemR,Formula,
    FormulaR,[SideH|SideT],Stoich,Qty,QtyR) -->
        balanced_formula(Fmt,Coeff,CoeffR0,
            Elems,ElemR0,Formula,FormulaR0,SideH
            ,Stoich,Qty,QtyR0),
        expr_tail(Fmt,CoeffR0,CoeffR,ElemR0,
            ElemR,FormulaR0,FormulaR,SideT,
            Stoich,QtyR0,QtyR), !.
```

  - Perhaps consider applying attributed grammar syntax.

5 Performance Testing

Input	Before		Experiment 1		After	
	FWD	REV	FWD	REV	FWD	REV
CH_4	28	28	32	28	26	20
CH_4O	40	39	44	40	34	26
HNO_2	41	63	40	38	19	15
NO_2	29	52	57	28	30	20
NH_4Cl	25	28	21	19	19	15
$NaCO_3 \cdot 10H_2O$	54	74	54	50	27	28

All values are number of inferences. Improvements to parsers have made Chemlogic significantly more performant.

6 Experiment: Recognizing Double Replacement Reactions

May 1, 2014

```
Welcome to the Chemlogic CLI interface! (Work in
   progress)
Enter a rule in the form:
        input type - 'input' :: output type -X.
For default output, use:
        input type - 'input' :: -X.
To quit the program:
        halt.

Input and Output Types:
        formula Chemical formula
        name            Chemical name
        symbolic        Symbolic equation
        word            Word equation

Example:        formula - 'CuCl2' :: -Name.
                        symbolic - 'H2 + O2 --> H2O'
                           :: word - Result.
Complex:        name - 'baking soda' :: -F, formula -
   F :: -CanonicalName.

ALL LINES MUST END WITH A .
* Fact DB compiled
?- formula(user,E,[],F,[],"HCH3COO",[]).
        E = [['H'], ['C'], ['H'], ['C'], ['O'], ['O
            ']],
```

```
        F = [[['H'], 1], [[[['C'], 1], [['H'], 3], [['
            C'], 1], [['O'], 1], [[...]|...]], 1]].
?- formula(user,E,[],F2,[],"NH4OH",[]).
        E = [['N'], ['H'], ['O'], ['H']],
        F2 = [[[[['N'], 1], [['H'], 4]], 1], [[[['O'],
             1], [['H'], 1]], 1]].
?- EQN = [$F,$F2].
        EQN = [[[['H'], 1], [[[['C'], 1], [['H'], 3],
            [['C'], 1], [[...]|...], [...|...]], 1]],
            [[[[['N'], 1], [['H'], 4]], 1], [[[['O'],
            1], [['H'], 1]], 1]]].
?- $EQN = [[["H",_],[AcidIon,_]],[[BaseIon,_],[[["O
    ",1],["H",1]],_]]].
        AcidIon = [[['C'], 1], [['H'], 3], [['C'], 1],
             [['O'], 1], [['O'], 1]],
        BaseIon = [[['N'], 1], [['H'], 4]].
?- EQNR = [[[$BaseIon,x],[$AcidIon,y]],[["H",2],["O
    ",1]]].
        EQNR = [[[[[['N'], 1], [['H'], 4]], x], [[[['C
            '], 1], [['H'], 3], [['C'], 1],
            [[...]|...], [...|...]], y]], [[['H'],
            2],[['O'], 1]]].
```

7 Building Chemlogic for Prolog Cafe (A Prolog Implementation in Java)

October 1, 2014

1. Environment variables

```
PLCAFEDIR="/home/npaun/Downloads/PrologCafe1.2.5/"
ANT_HOME="/home/npaun/Downloads/apache-ant-1.9.4"
JAVA_HOME="/usr/lib/jvm/java-1.6.0/jre/" PATH="/
   home/npaun/Downloads/apache-ant-1.9.4/bin/:/
   home/npaun/Downloads/PrologCafe1.2.5/bin/:/usr/
   lib64/qt-3.3/bin:/usr/local/bin:/usr/bin:/bin:/
   usr/local/sbin:/usr/sbin:/sbin:/home/npaun/bin"
```

2. Create an ordinary project in Eclipse Name: xxx.app
3. Copy plcafe.jar to libs/
4. Write Java bindings Package: xxx.bindings
5. Create symlink <bindings> to src/xxx/bindings/

6. Write Prolog code
7. Run pljava to create Java source code
8. Copy everything in the java/ folder to some location
9. Use ed to create package xxx.prolog

```
for i in *.java; do
        echo '0a package ca.nicholaspaun.plcafe.
           prolog; . w' | ed $i
done
```

10. Create symlink <prolog> to src/xxx/prolog/
11. Update project using android

```
~/vboxshare/adt-bundle-linux-x86_64-20140702/sdk/
   tools/android update project -p . --subprojects
    --target android-19
```

12. Build using ant debug
13. Install with adb

```
~/vboxshare/adt-bundle-linux-x86_64-20140702/sdk/
   platform-tools/adb install -r bin/MainActivity-
   debug.apk
```

8 Cross-Compiling Chemlogic for Android

November 24, 2014

In the end, I had to give up on converting Chemlogic's code to run on a Prolog that is made for Android. Cross-compiling (building on a PC to run on Android) SWI-Prolog (the one Chemlogic currently uses) and many, many other Prologs using Google's tools would not work because the C library supplied with Android does not conform to international standard. I realized that I would have to get a real C library (GLIBC) to work on Android. The process looked so convoluted that I did not want to even try it. Then, by chance, I stumbled on a method that I'd tried out earlier during the summer but could not get working (without requiring copious amounts of disk space): running the code of a real Linux distribution, compiled for ARM, using the Android kernel. This is named the chroot process. The excellent article I read suggested using Angstrom Linux, which is a distribution specifically intended for embedded Linux and is only 80MB. It also explained how to bootstrap the chroot, using Angstrom's linker to re-link the shell to the provided GLIBC library, which then starts the

chroot, in order to bypass Android's crummy C library and start the Linux distribution.

It worked so well! The only thing left to do was to compile SWI-Prolog for ARM, using my Intel laptop, with the real GLIBC. Angstrom used to make an excellent toolchain, but they stopped distributing it. Luckily, someone on a forum had a link to a copy floating around at the Internet Archive. The toolchain worked perfectly and I had to repair only a silly short-sighted piece of code in SWI-Prolog's build utility. I did also have to replace some ARM compiled tools, used only during building, with pre-compiled Intel versions, so that the build process could actually use these files. Prolog could then be bootstrapped on the Android itself, to make its PRC file, thus finishing the build process.

Finally, Chemlogic was loaded and it could correctly balance chemical equations – even the built-in linear algebra system worked properly! I then decided to trim down the 80 MB installation by selecting only the essential files and libraries Chemlogic needed, using another fascinating article that discusses using the linker, again, (which I never knew could be used directly by a user, before this project) to retrieve dependency information. This got the whole system down to 5.1 MB.

Now, I have to make this process systematically repeatable, so that I am not dependent on random files and computers that might break down, leaving me with no clue as to why this stuff even works. I will need to write a simple bootstrap program that automatically starts Chemlogic, so it can communicate with the Android app, which is written in Java. I will also write a little bit of code to make the output of Chemlogic easy to transmit to Java, parse and insert into the app. Most of the code is already written: just a few small changes to the command-line interface, a few subroutines to deal with identification and error handling and a new formatter for errors, based mostly on the one from the Web interface, interestingly enough.

9 Repeatable Builds for the Cross-Compiling Chemlogic

November 30, 2014

1. angstrom distribution

   (a) Unpack it and grab the necessary files

2. angstrom toolchain

   (a) Unpack it.

3. Patch SWI-Prolog's configure script to stop complaining when cross-compiling, run autoreconf

4. Build it with angstrom toolchain.

5. When trouble strikes, repatch with defatom and mkvmi from x86_64 Prolog.
6. Build libgmp with angstrom toolchain.
7. Copy clpqr to clp library locally
8. Copy library and boot to angstrom environment
9. Tell SWI-Prolog to build the boot file.
10. Tell SWI-Prolog where clp is located.
11. Load chemlogic files
12. QSave
13. Finally, copy only necessary files to production angstrom environment.

10 The Android App, Stoichiometry and Reaction Type Analysis

January 23, 2015

I've finished the Android app for Chemlogic last week. It's been a major challenge and has consumed nearly all of my time. It works quite well on the emulator, but I still have not tried it on a real phone. I also haven't put it on the Google Play app store yet, because the payment setup is rather convoluted – I will probably do this sometime before the science fair.

Amazingly, though, stoichiometry is working very well: In about 30 minutes, I added molar mass calculations, conversions to/from mass, moles, volume (gases). I've started work on implementing molar concentrations (added calculations involving solution volume and concentrations to/from moles), but I've forgotten how exactly the rest of the concentration calculations work. I have to re-read some sections from the Chem 11 book.

After finishing stoichiometry, I hope to implement reaction type analysis, a quiz feature and net ionic equations. I'm not quite sure how difficult these features are.

Simply detecting the type of reaction (for a few common types) an equation represents, will be as easy as the work I started on stoichiometry, completing the reaction is easy too (if I stick to double replacement, combustion, etc.) The major problem is deciding whether or not a reaction will take place. Last year I discussed using the electrochemical series of standard electrode potentials, which is almost the same as the activity series, but with a numerical quantity associated with it. I was worried that it was a bit different from the activity series in the textbook, until my teacher found 3 different activity series, all of which have slightly different orders. I'm not sure which version to use. The

reactivity series on Wikipedia disagrees with all of the other ones I've seen; is lithium more reactive than sodium: Wikipedia says no, the others say yes.

Non-metals and polyatomic ions are also a massive problem: I cannot find any series which shows the reactivities of different non-metals, except for the halogens. My teacher says that there aren't any high school Chemistry problems that would involve deciding whether one non-metal would replace another, except for the halogens. Maybe Chemlogic should just say "I don't know – try it yourself!" in those cases.

The new features are very complex, but they don't change any of the fundamentals of Chemlogic. I can explain that because of the modular design of Chemlogic I implemented, stoichiometry and reaction type analysis can simply be implemented by manipulating the abstract syntax trees the equation and formula parsers produce and the android app can communicate with the rest of Chemlogic through the user interface systems. I'm not sure how much new material this is – but I don't think I should worry about this: after all, I wrote a 30 page paper where a 5 page one was required!

11 More Updates on the Android App

January 30, 2015

I've also been able to finally try it on a real device! Mr. Leeming has lent me a junky old Galaxy Tab (a tablet thingy). The app worked!

There were a few issues, but all of them had to do with the appearance of the app, rather than the functionality. Some of the styling has been changed by Google between the really old version of Android that the Tab runs and newer versions of Android – this caused such wonderful behaviour as white text on a white background. I was able to quickly correct the unreadable parts and get the app to at least look normal on all Android devices. Although I'm not going to bother too much with styling, there are one or two rather irritating display issues that I will fix when I get around to it.

I'm not quite sure how to give a technical discussion of the App — everything seemed so complex and arcane in December, when I last explained it. So, here is a very long summary of how it works:

The App consists of a Java (the language Android uses) user interface and the cross-compiled Chemlogic code, written in Prolog.

The user interface renders the design of the app (textboxes, buttons, etc.), it then filters the input to make sure that there isn't anything in it that will crash Prolog and then communicates with Chemlogic through a UNIX pipe. Upon recieving the response, it renders the formatting – subscripts, red color for errors, etc. — (Android was supposed to do this for me, but it wasn't very good at it so I had to extend it) and displays it. The user interface also adds an extra row of keys to the keyboard so that you don't have to repeatedly enter menus on the phone to type an arrow or a plus.

I've greatly simplified the way that Chemlogic runs on the Android. Basically, Chemlogic is now compiled as a stand-alone application, meaning that it

includes the Prolog interpreter in its file. It is distributed with the libraries it depends on. Chemlogic is started by Java by running a program that runs the dynamic linker to locate Chemlogic's libraries and then executes it.

The need for a chroot mechanism has been eliminated (and it didn't actually work on a real Android phone), as well as a shell for running inside the chroot. A few of the libraries have also become unnecessary, as has an extra copy of the Prolog interpreter. The total size of Chemlogic is now 4 MB. Prolog and Chemlogic must still be cross-compiled by a convoluted process, but I have now finished automating it and it works correctly every time (...I think).

To make the app work, I had to implement a way for the user interface to install Chemlogic onto the device, run it and then communicate with it. Since the Android system will not install Chemlogic's files, I had to implement some Java code that verifies whether the correct version of Chemlogic is installed on the device. If it is, the app will just start. If not, it extracts Chemlogic and copies it file by file, megabyte by megabyte (thanks Java!)

I've already explained a little bit about how the app is started and how it communicates. The user interface writes a command to the pipe, when Chemlogic reads. Chemlogic writes its answer to the pipe, which the interface reads. It would be nicer if bindings between Prolog and Java could be implemented, so that running Chemlogic is the same as running a subroutine, but I did not have time to experiment with this and it would greatly increase the complexity of the cross-compilation process. The pipe model is, however, one of the most interesting parts of the UNIX model: everything is a file, everything is text, anything can read text and anything can write text. It allows many utilities to be piped to each other, each reading the previous one's output and writing to the next one. User input/output is, in fact, implemented using this model as well: the keyboard is piped to the input and the output is piped to the screen. The funny issue with this, which took me a long time to grasp, is: How do the interface and Chemlogic know when the other end of the pipe finishes transmitting data (like a query or a response.)? Unlike a subroutine which returns a variable, the two programs must look for a specific pattern — the same way a user would.

e.g. When the user enters a command, he presses ENTER. Now Chemlogic knows that the user has finished typing and it displays the response. The App does this as well. The output was a bit harder. Chemlogic might produce blank lines as output, for readability, error correction or other reasons. Therefore another mechanism must be found for determining the end of the message. Creating an "END OF MESSAGE" marker seemed just ridiculous. I realized that a human user also needs a cue to know when to type another query — this is the interactive prompt. In the case of the App, the user interface now continues reading until it sees "?- CL [query n]" (Chemlogic's interactive prompt), which lets it know that Chemlogic has finished.

I should add that although the pipe model seems to imply transmission and communication between computers (and it often can), in this case the communication is entirely virtual – it is between multiple processes on the same computer. The mechanism is almost exactly the same.

....And now, with all that mostly done, I will continue to work on stoichiometry.

12 Some Notes on Stoichiometry

April 29, 2015

- The chemical equation parsers were extended to allow the user to provide the quantity of each molecular species in an equation.
- A new grammar was written for the purposes of manipulating quantities.
- The queries which could be made when limiting reactant analysis is performed, are very complex.
- A separate list of queries is provided by the user.
- Stoichiometry begins by manipulating the ASTs generated by balancing.
- First, it is determined whether limiting reactant analysis is necessary. (It is necessary if the quantity of more than compound is provided).
- If so, the limiting reactant is determined by a simple systematic algorithm, which applies even if the units of the input quantities are different. (UC Davis Chemwiki).
  - The quantities of each reactant are converted to moles, then divided by the coefficient of the compound.
  - The compound with the lowest ratio is the limiting reactant.
  - The number of moles of the limiting reactant are used to answer the queries.
- If there is no limiting reactant, then the moles of the provided compound are used to answer the queries.
- Procedures are implemented for performing all of the unit conversions necessary.
  - Support for concentrations of chemicals is implemented.
  - Many patterns are supported.
- Chemlogic will always produce results with the correct number of significant figures. The number of significant figures in each input value is determined and recorded. Each unit conversion procedure determines the maximum number of significant figures that can be preserved. When user output is produced, numbers are rounded to the correct precision.
- Producing a user-friendly Web interface that can handle complex stoichiometric calculations is a significant challenge.

- Unlike the interfaces for balancing and converting formulas to names, which are just simple wrappers around the underlying API, the stoichometry user interface is very user interactive and much code had to be added to prepare the correct queries to send to the API.

Part II
Program Documentation

13 README

13.1 Usage and Installation

Chemlogic comes with two interfaces: a command-line interface and a Web interface.

To run the CLI:

1. cd cli/
2. ./chemcli

The CLI implements an extemely simple DSL for querying chemical information (a help message will appear when chemcli is run). It is also a Prolog top-level (shell), so you program more complicated things.

To run the Web interface:

1. cd web/
2. ./chemweb

The Web interface runs on port 8000 by default, at http://localhost:8000/chemlogic/. The Web interface provides a simple interface for all of the Chemlogic features.

To compile the interfaces, use make:

1. 1. make <interface>
2. 2. make install

Run make help for more details.
<interface> is either cli or web, or all to build both interfaces.

The program will be installed by default to /usr/local. This can be changed by passing PREFIX=<directory> to make to install Chemlogic under a UNIX-style filesystem structure or by passing DEST=<directory> to install all Chemlogic files to the specified directory.

NOTE: These options must be specified when running both the build and install targets, if they are run separately.

The executables built are still dependent on a copy of SWI-Prolog.
NOTE: If the fact/ database is edited, the program must be re-compiled.
NOTE: The Web interface will automatically search for its style files in <PREFIX>/share/chemlogic, or if it is not found, the current directory.

13.2 Dependencies

Chemlogic is written in Prolog and requires a Prolog interpreter. The code is mostly conformant with ISO Prolog and requires DCG features and the CLP(q) library. The Web interface depends on SWI-Prolog and is built using the Web framework.

Prolog distributions:

- SWI-Prolog: Developed and tested on this platform. (See <http://www.swi-prolog.org/> for information.)
- YAP, CIAO and XSB Prolog: Will probably work.
- GNU Prolog: Might work, but CLP(q) must be ported.
- Prolog Cafe, tuProlog and GNU Prolog for Java: Will not work (tested). Signficant parts of the ISO standard are missing or incorrectly implemented.

13.3 Features

- Chemical formulas (incl. hydrates)
- Chemical names
  - Retained names
  - Acids
  - Ionic compounds
  - Covalent compounds
  - Some organic compounds
  - Pure substances/allotropes
  - Common names
- Chemical equations
  - Symbolic equations
  - Word equations

- Stoichiometric calculations
  - Limiting reactant analysis
  - Excess quantity calculations
  - Support for concentration units
- Reaction type analysis
  - Completion of neutralization, double replacement and single replacement reactions.
  - Predicts whether reactions of any of the above types will occur (using the reactivity series.)
  - Identification of neutralization, double replacement, single replacement, decomposition and synthesis reactions.

13.4 Changelog

- Version 2.0
  - Added support for reaction type analysis (identification, completion and prediction)
  - Added support for stoichiometric calculations (incl. limiting reactant analysis, concentration calculations)
  - Developed an Android user interface (packaged separately)
- Version 1.0
  - First release

14 TODO

14.1 Chemistry features

- Support for structural formulas:
  - The covalent parser will have to be extended very much, to handle structures of the compounds it supports
  - The formula parser will need to have some sort of input and output representation for structural formulas
  - Each output format will have its own ways of rendering structural formulas. This will have to be extended.
  - A module will be needed to convert structural formulas to molecular formulas for balancing and other processes.

- More organic naming:
  - It will be useful to implement organic compound naming at least for Chemistry 11 to 12.
- Complex Redox reactions:
  - Sometimes, for a few very complex redox reactions, Chemlogic gives an answer that satisfies the system of linear equations (i.e. is balanced) but will not actually occur in real life.
  - A new balancing process, with a separate module should be implemented. Perhaps based on oxidation numbers or half-reactions
- Diagramming:
  - Show structural formulas of compounds
  - Bohr models, Lewis diagrams
  - Periodic tables?
- Equilibrium calculations
  - Extend stoichiometry to enable the user to perform equilibrium calculations.
  - This requires applying multiple stoichiometric calculations in a new simplified user interface.
  - Reaction, solubility and acid-base equilibria could be supported.

14.2 Program features

- Extend the `chemcli` DSL to make it more useful.
  - Offer a way to query the chemical information database.
  - More constructs/operators.
  - A standard library?
  - This will all depend on the sorts of programs that someone will actually want to write
- Quiz program
  - Allow for questions to be generated with selectable options (the parser and perhaps major sub-parts)
  - Interactive and non-interactive usage depending on output format
  - Configurable marking (allow retry, show correct answer at end, etc.)
  - Possible to produce the same questions if passed the same seed. Some sort of intelligence, focusing on problem areas when giving questions.

- Expose the chemical information database.
- Better error messages for equation balancing errors
  - Test to ensure that all elements appear in both products and reactants
  - Perhaps explain why some charge shifts are unsatisfied
  - Explain which element makes the system unbalancable, if possible.

14.3 Organization and Structure

- Take advantage of SWI-Prolog's new string type. This makes tokenization, concatenation and many other things more efficient.
  - Chemlogic needs a simple tokenizer to break up chemical names and equations (especially)
  - At minimum, it should split on spaces and remove extraneous spaces
  - Potentially deal with character types?
  - Potentially deal with the insides of parentheses?
- Rewrite the current error tokenizers to use the new and better functions
- Tokenizers make parser much, much nicer:
  - It is now quick and simple to see if something is an acid without having to go through all of the tests
  - It can be easy to distinguish between ionic and covalent
- Make the oxyanion functions less messy and hacky. There must be a better way to tell the user what's wrong with the oxyanion names.
- The ugly hacks around pure substances can be removed with a better tokenizer
- Some things will need to be renamed and reorganized
- Use more meta-programming to remove boilerplate code from the web interface.

14.4 Bugs

- The program will get very upset if a substance is repeated:
  - e.g. H2O + H2O -> H2O
  - There is not much of a valid reason to enter this, but the program should handle this correctly

– An error message explaining that this is junk is probably a good idea

Part III
Development Information

15 Branches

Variants of the code were produced to allow for new features to be tests without interfering with already written code.

android Extended the command-line interface to allow its use in the Android application, by making simple changes, including: adding an API mode to the chemcli program to allow the Android App to communicate with Chemlogic over pipes, extending the output formatting system to process error displays.

build Developed an improved building system, using make instead of shell scripts, that could support more complex building configurations. This was useful when developing the Android App.

combine_solving An experiment: Removing the matrix and system of linear equation creation steps, and converting the tabulation module to directly solve the equations. This version was not used because the current process is easier to debug.

complex_redox Various experiments to determine under what circumstances Chemlogic produces a valid balanced solution that is not actually the experimentally-determined balancing.

db_add Used to introduce new information into the chemical facts database.

explain Introduced the error explanation feature, which highlights the erroneous parts of user input and provides an appropriate error message.

interface Developed the Chemlogic Web interface, written using SWI-Prolog's HTML processing libraries.

master The main development branch.

modularize Separated code into logical parts using Prolog modules.

parser The performance of various parsers was improved by rewriting DCG clauses. Some incorrect behavior in special cases was also corrected.

parser_error The parsers were modified to raise syntax errors when user input was invalid. This changes allowed for the implementation of error handling.

parser_tokenizer An experiment: Implementing a tokenizer would make parsers simpler and more performant.

plcompat An experiment: Made the Chemlogic code fully compliant with the ISO Prolog standard, and provided implementations for some non-portable predicates. This branch was not used because the Android App was realized by cross-compiling SWI-Prolog for Android, rather than compiling the Chemlogic code using a JVM based Prolog implementation.

solve_explain Various experiments to explain the solving process and its errors more clearly to users.

stoichiometry Implemented the new stoichiometry feature, including limiting reactant analysis, a quantity grammar, significant figures handling, unit conversion and the molar and stoichiometer interfaces.

type_domain_error Introduced handling for type and domain errors which occur when input is syntactically valid, but malformed. This can occur due to implementation errors, but are usually caused by a user inputting a logically incorrect equation.

type_info Developed the new reaction type analysis feature, based on pattern match. Reaction completion, prediction and type identification were implemented.

unbalancable The balancing process was modified to detect equations that are unbalanceable because elements are missing from the reactants and/or the products and identify the missing elements.

16 Changelog

An abbreviated version of the Git version control log for the Chemlogic repository is reproduced here.

- New changelog and features for README.
- Updated TODO to reflect 2014-2015 changes.
- Finished merging stoichiometry with type_info.
- Merge branch 'stoichiometry'
- Remove ridiculous debugging message.
- Merge in stoichiometry!!!
- Stoichiometry is done.
- The full calculation can now be performed. We simply require input boxes.
- Render results as HTML.

- Stoichiometer will kinda sorta show results.
- It still did not crash – we are now rendering our results.
- Added query_result for the Web interface.
- The Web Stoichiometer can actually perform queries in the background now!!!
- More error handling for chemical quantities.
- And now, add conversion to determine M from mols.
- The unit and format conversion routines must handle all logic errors.
- Allow requesting answers in M.
- Try to support molar UI tail units (and dilutions!)
- Re-fix querying with one part results.
- Simplify the tail unit for the molar UI.
- Merge branch 'type_info'. Import reaction type analysis features for the science fair!!!
- Actually, activity_check_multiple must be written differently.
- Also allow 2>1 and 4>3 for double replacement.
- The other form of neutralization (reaction_types).
- Ensure that all errors are handled in reaction_types.
- OOPS: Formula to name should actually run formula to name.
- Do not crash when reaction type info is unavailable.
- Removed now unecessary query conversion.
- Allow Chemlogic interfaces to access the conversion interface without having to create grammatical input.
- Just an experiment to hackily allow actual answers with a tail.
- Allow enter to submit on the molar form
- Add support for two-part output (concentration chemistry).
- Simplify the output list production procedure, so as to make dealing with a large number of units actually reasonable.
- Allow entering volume/concentration quantities in grammatical input.
- HACK: Chemlogic should not fix up ethanol and ethanoic acid.

- Make the quantity DCG actually output with sigfigs.
- Make the quantity DCG produce sigfigs output.
- Allow useless reactions to take place (we tell the user that they wont happen).
- One more place to fix the odd acid issue.
- Partial correction for strange bug relating to acids with bi- anions.
- Cut away nonsensical error messages.
- Fully integrate the logic errors for ionic compounds.
- Experiment: if the correctors are declared as logic errors, we can more clearly identify where the issue lies, because syntax error handling will be unable to locate the problem input.
- Experiment: allows error messages produced while completing reactions to appear.
- Make requesting reaction info dependent upon actually successfully balancing an equation.
- No reaction when the same anion or the same cation is present in both reactants.
- Add a useful answer for reactions between acids and bases.
- Improved styling of reaction information.
- Add activity info to the web interface.
- Added a simple predicate to explain the results of activity checking.
- Try to guess whether a reaction will occur.
- Add some new headers and begin to check against the activity series.
- Registered the activity series in the database.
- Allow conversion between units via two step conversion to mols
- Allow the molar feature to provide answers inline.
- Make the molar page actually do something.
- Reverted some changes to stoich_queries.
- Mark Iodine as diatomic. Fixed long-term bug.
- Make the Molar feature actually... um... do something (still very, very buggy)

- Enable the Molar UI screen.
- Some quick patches to make the stoichiometer work like a... fancy version of the balancer.
- Place the balancer in stoichiometry mode.
- Begin to implement the stoichiometer Web GUI.
- Preliminary change to allow for stoichiometric calculation using a pre-balanced structure.
- Wowee, we can complete single replacement reactions!!!
- Recognize single replacement reactions.
- Complete neutralization reactions, too.
- For completeness sake, the remaining neutralization reaction combinations.
- Match neutralization, another double replacement ordering, synthesis, and decomposition reactions!!!
- Expose reaction type info in the Web interface!
- Enable compound calculations in the CLI, too.
- Fixed name to formula as well.
- Some bugfixes to allow for conversions involving a single compound.
- And add support for stoichiometry with no output type.
- Stoichiometry works in the chemcli DSL.
- Started to add stoichiometry to command-line interface.
- Wow. The complete stoichiometry API!
- Use the simplified way of entering input quantities.
- Made sure the existing balancing logic continues to work.
- Added stoichiometric input to the word equation grammar.
- Inserted stoichiometric support into the symbolic equation grammar.
- Added DCG rules to allow combining compounds with stoichiometric information.
- The stoich_limited calculation has now been converted, too.
- stoich_simple now re-unified. Things are much simpler.

- I think the units module has finally been unified.
- Fixed up the query format when producing output.
- Convert simple units to the new format.
- Made the format more consistent.
- Adjusted format for the quantity DCG to match the rest of Chemlogic.
- BACK-OUT the new quantity and query DCG formats. They add unnecessary complexity. AAAH!
- Try to change stoich_limited to the new format.
- Made stoich_simple use the AST.
- Made the units module use the new ASTs. Maybe it is easier to understand and manipulate?
- A start of some simplified ideas involving the quantity struct.
- Modularized the quantity grammar.
- Finished the quantities grammar.
- Created a new quantity grammar to allow expressing user queries.
- Made some adjustments to the structure of the stoichiometry features in order to develop the user interface.
- Revised menu items for future features.
- Replaced the future development menu items with the latest planned features.
- Fixed a slight mistake in the word_equation grammar: formatting is not used in this case.
- Corrected the same dumb mistake with the missing quotes but, now, in the word equations.
- Improved wording of some symbolic equation messages and enabled reaction completion for word equations!
- Added a new error message to explain problems with reaction type analysis/completion.
- Improved code formatting and removed debugging statements.
- Improved the wording of some solver error messages.

- Improved the linear equation evaluation error message to more clearly explain the mistake in question and removed examples of issues that are in fact detected and explained by the unbalancable equations feature.
- Revised the symbolic equation grammar messages to reflect the new reaction type analysis features.
- Added a corrector for a common mistake: Ionic compound formula subscripts that don't actually make any sense.
- Whew... Fixed (I think) the mysterious stop in the middle of a formula. I hope it is only a case of optimization gone too far.
- Some experiments to identify why the symbolic equation grammar explodes when a list of elements is not provided.
- Chemlogic can now complete double replacement reactions (just a few bugfixes to make it actually fit in the user interface.
- Experimental code to allow for generating double replacement reactions (and possibly single replacement reaction).
- Merge branch 'stoichiometry' into type_info We need some new extensions to balancer added by stoichiometry.
- Added the empty new reaction type analysis.
- Added a new error message for nonsensical unit conversions.
- Wrote error message to explain why you cannot calculate excess quantities if you do not know the original quantity.
- Started to add error checking for stoichiometry.
- Combine the significant figures with the stoichiometry structs to simplify a lot of code and allow for mostly correct significant figures interpretation in excess quantities problems.
- Unified calculation for almost all of the stoichiometric features supported in Chemlogic.
- Added a file block for excess.pl
- Improved the Chemlogic README, to discuss actually running it.
- Cut useless choicepoints.
- One function solving for stoichiometry of excess quantities.
- Chemlogic is now able to perform stoichiometry of excess quantities (in pieces).

- Oops, what if we do not want to know something?
- Now, we can correctly calculate the amount actually reacted/produced.
- Broken attempt at stoichiometry of excess quantities.
- Modify the limiting reactant calculation to produce the mols of each substance (which is generated anyway).
- Move the stoichiometry of excess quantities code experiments into their own file.
- Removed unecessary cut operations.
- A tiny change, turning on optimization, speeds up some parts of Chemlogic dramatically.
- Clarified comments and removed unused rules.
- Simplified the nonzero sequence to, in fact, represent a non-null sequence (leading zeros means that zeros would never reach this rule, in the first place)
- Made the DCG more efficient by testing for nonzero because this will only happen once per number, as opposed to retrying.
- [M] Fixed formatting in the README
- Updated TODOs: It's getting there!
- Mark stoichiometer executable.
- A minimal interface for entering coherent stoichiometric calculations.
- Added PLDoc comments for two more balancer functions.
- Clarified PLDoc to add that some numbers are expected to be integers.
- Improved PLDoc blocks by adding types (from memory).
- Corrected and improved PlDoc comments for the balancer (changes necessitated by the implementation of stoichiometry.
- Added documentation blocks to stoichiometry code.
- Corrected documentation blocks for many different files.
- Added a file block for the chemcli API, and correct a few other blocks.
- Updated Chemlogic copyright year!!!
- Fixed the generic calculation predicates: in this clause, some aspects of the result must be known, while others must be unknown. The variable may be in either term.

- Created generic calculation predicates for the concentration/volume calculations.
- First attempt at generic calculation predicates.
- Removed experimental code we do not need anymore.
- Some fixes to ensure that our significant figures logic actually handles the numbers first.
- Made the stoichiometric code calculate the number of significant figures and only produce answers to the lowest precision.
- Almost correct rounding logic.
- Simplified and greatly clarified the sigfigs logic, with clear comments.
- Some correct rounding logic (in PHP).
- Complete significant figures counting, except for scientific notation.
- A correct new significant figures rule.
- Solved the volume/concentration dilemma.
- Modified sigfigs logic (still does not really work).
- Try to determine whether leading zeros in decimals are significant (e.g. 0./000/5 is not significant, 3./000/5 is significant).
- An inefficient attempt at determining the number of significiant figures in a number.
- Added cut so Prolog does not try to find a different way of... uh... converting from moles to moles.
- Standardized the format of the unit predicates to allow more flexibility in stoichiometric calculations.
- Clearly indicate the units of molar volume.
- Allow both the input and output to be in mol units, by creating a conversion rule that does nothing.
- Make the spacing around operators consistent.
- Automate the use of the bsd2make script.
- Simplified Makefile and restored compatability with BSD make.
- Modify the balancer interface to expose the chemical equation structures generated by the parser.

- Added simple stoichiometry calculation: it works!
- Added reference explaining source of Vm.
- Corrected incorrect parameter usage for gas volume calculations, added cut to prevent useless alternative solutions
- Added concentration units
- Added unit calculation support (mass, volume [gas only]).
- Added cuts to ensure that only the actual molar mass is produced as a solution.
- We can now calculate molar mass.
- Declared the mass fact to be discontiguous.
- Added atomic masses for stoichiometry.
- All opened tags must be closed – this bug lead to a really hard to track down display problem in the android app.
- Implement the code to set the background-color of tokens.
- Added the html_android_textview output format, which will allow for formatted text in the App.
- Merge branch 'build' into android. This contains some important changes to the build system to make versioning work better with the Android application.
- Improved the API startup identification.
- SWI-Prolog behaves differently depending on whether the application is run using the SWI-Prolog binary (the program name is not included in the argv flag) and when it is run as a native executable (the program name *is* included). This commit will just try both possibilities.
- Ignore the Buildinfo file, it is created by the tagging process and records only version information.
- Merge branch 'master' into build. Imported API message.
- Added a simple version identification message for the API startup.
- Disambiguated prefix to cf_prefix.
- Removed some blank lines to allow room for some new directives to be added to the dist file.
- Correct incorrect spacing in the ANSI format arrow symbol.

- Moved print rule back to the main DSL file and removed the echo predicate. I do not know which method is better, but perhaps print can be used in DSL programs to communicate with the user.
- Quick attempt to add printing the output to the DSL.
- Make the API mode echo the output of the DSL rule to stdout, so that it can be captured and used.
- Added the API logic to the chemcli interface. Some work still remains.
- Added the ANSI output format and changed the CLI error handler to use the output formatting system to highlight errors.
- [BUG] BSDMakefile is out of date because the conversion script is missing.
- Support DESTDIR and do not crash when the web interface is not built.
- Modified Makefile to support alternate paths to Prolog, and the framework for supporting other Prolog systems.
- OOPS. We want make to copy the actual style contents as opposed to a symlink (we do not want to refer to the source directory.
- Moved compile.cf and other build-related tools to their own directory, to avoid littering the project root.
- Load Web Interface main files before running directives — quiets down startup
- Made chemweb-daemon use compilable clfacts and also use the same styles as chemweb.
- Oops, make sure that if prefix/1 is undefined we used ./ as the style_dir.
- Finally, a solution to compile.cf
- Catch error if prefix/1 is not configured.
- Added a new predicate to automate registering files with SWI-Prolog.
- Finished converting the Web interface to use the runtime style path detection.
- Re-generated BSD makefile and add header comments regarding autogeneration
- Added info header to Makefile
- A proper working Makefile under both BSD and GNU makes
- Make sure that compile.cf is reset to default when cleaning.

- Nearly finished properly organizing the Makefile
- Corrected dependency information to allow some archive building in parallel.
- Added a configuration file to hold the prefix determined by make, so that Chemlogic can locate its files.
- Fixed Makefile to include correct install target.
- Added dynamic location of style files. Is this the best solution?
- Barium was incorrectly listed under the alkali metals heading – moved to alkaline earth metals
- Renamed iface to interface, corrected install target and added help target
- Added some more TEX related files to gitignore and also excluded everything in bin/ except README
- .gitignore and a Makefile – just an experiment
- Added package building script — maybe replace with a Makefile?
- Added some source files for the Chemlogic papers (need to check if they correspond to PDF and produce correct PDF when compiled)
- Added cobalt.
- Added PDF documentation
- Updated references to documentation
- IUPAC-ize the program
- Various quick fixes to scripts to avoid portability problems
- Finished the TODO
- Commented up build.sh
- READ ME
- Fixed the build script.
- A much simpler building system
- Working on an install process
- Some TODOs
- Whoa! Everything has a top comment now!
- Process - General (IX)

- Process - General (IIX)
- Process - General (VII)
- Process - General (VI), Data - Debug (II)
- Process - General (V), Data - Debug (I)
- Process is overloaded (IV)
- Process to General (III)
- Process to General (II)
- All the way to 99: that should do.
- Remove manual wrapping from messages
- Merge branch 'type_domain_error' This makes everything work, at least.
- Auto-wrap message but preserve indenting and newlines!
- Mr. Tchir suggestion: people do not understand spurious, extraneous and malformed.
- Mr. Tchir correction: non-metalide (III) and my correction: Allotrope
- Mr. Tchir correction: non-metalide (II)
- Mr. Tchir Corrections: Non-metal ide (I)
- Mr. Tchir suggested correction: a consistency fix and a spelling correction
- Mr. Tchir correction: acid is not a suffix
- Mr. Tchir suggested correction: Remove obsolete commented code
- Mr. Tchir corrections: not necessarily a compound; use chemical name instead.
- Mr. Tchir suggested corrections: Avoid saying token, nobody understands
- Quick fix: consistent 1 or one
- Mr. Tchir suggested corrections: Avoid saying operator, people do not understand (2A)
- Mr. Tchir suggested corrections: Avoid saying operator, people do not understand (1A)
- Mr. Tchir suggested corrections: Do not use evaluate; people do not understand.

- Mr. Tchir suggested corrections: Do not use parse; people do not understand.
- Mr. Tchir suggested corrections: ordering (I)
- hydrogen peroxide special case.
- Remove spaces from hydrates
- Moduarize it!
- Retrofit domain errors into the error handlers.
- Added in messages for our domain errors.
- Added type errors to web_error.
- Error handler for the web interface: type errors
- So far: a huge mess that makes messages for type errors.
- So far, the error handling is a bit messed up.
- Make the error pre-processor syntax_error specific, too.
- Oops: incorrect parenthesization.
- Made the current error handlers specific to syntax errors
- Added domain_errors for all of the current possible failure points.
- Make the subscript for polyatomic groups... actually subscripted.
- Let's be really fancy: a tooltip that lets the user know why the item is disabled.
- Fixed bugs found during user testing. 1. Example in chemcli doesn't work – types swapped. 2. baking soda does not have exactly the same struct as sodium carbonate. 3. ; instead of , in chemweb breaks everything subtly.
- Added error handling to compounder. THIS COMPLETES THE REQUIRED FEATURES FOR CHEMLOGIC!!! !!! !!!
- Styling for errors.
- Correctly indented web_error and fixed class names.
- Cleaned up the outputting with a common function.
- A new error handling setup for the web interface..
- Changed the debugging system to avoid trouble and allow for better output in web interface AND to remove three unnecessary functions.

- A new theory: rethrow, with module-specific handlers.
- Test API change for web interface.
- Converted the web interface to use the new parsers
- Make the balancer the default page if none is chosen
- A nice message for the Web Interface, too.
- Removed CLI specific functions from IO.
- An elegant command line interface for chemlogic.
- Add another type error, for the matrix builder.
- Added a type error for when the equation struct cannot be tabulated.
- Some more ugly hacks to allow pure compounds to work correctly.
- Added more retained names because I do not feel like getting to work.
- Make sure that the polyatomic ions are loaded in defined order; this will make sure that ClO4 loads before ClO.
- Added some more common names of chemicals.
- Really broken stuff — now will it balance?
- Some fixes that allow single-element covalents and that just use systematic naming for an organic that cannot be figured out (reverse).
- Allow multiple single elements; covalent
- We need something that is actually unbound for this test.
- Changed some error messages to use mark, clarified some more messages and broadened a few.
- A new type of token that just marks one character
- Nearly finished setting up the error messages for word equations.
- Some small changes to allow the equation parsers to override their compound parsers.
- Finished (I think) with all of the clear formula error messages!
- Changed to new syntax for error messages in the balancer.
- Added a new token type, for dealing with parenthesized stuff.
- Improved the quality of the error highlighting (parenthesized stuff) and also clarified a message.

- Oops, we cannot go busting the reverse mode for using an explicit 1
- Fixed up our silly interface to use the parse/3 predicate. And also removed some stuff that was troubling the parsers.
- Correct mode declarations for meta predicate
- Ionic needs yet another error message to straighten things out.
- We still need to indicate which module we want handling failures and leftovers.
- Unfortunately, we had to re-organize the catch-all guidance.
- Covalent, too.
- Switched nearly everything over to the new way of declaring error messages.
- Meta programming the error handling for lots of fun.
- Finished the ugly hack that allows the covalent parser to work properly.
- Use our brand new xx operator!
- Another quick hack to explain error and some more messages.
- The error messages for the equation parser, so far
- Checked off a lot of tasks and improved the error message guidance for all of the names
- Fixed the pure substance bug, added an error message, fixed up the equation parsers.
- Converted the word equation parser to use the new formula parser.
- The arrow is required.
- Whoa: finished adding in all of the error messages, modularized the compounder.
- Had to reduce some of the error messages in the ionic parser
- A few more "CORRECTORS" here and there.
- Finished adding in all of the syntax errors for the Chemlogic program. Just a few ugly hacks.
- The ionic parser has nearly enough error messages, now.
- This will work for now. It is still quite ugly, but it starts giving the user clear error messages.

- Some better formatting; some better error messages
- So far, so good... Ionic has two error messages.
- Added all the guidance messages for formulas, now it is on to compounds.
- Fixed up some more tokenization rules.
- Scan rules can now handle everything that has been error-ifyed so far, I think.
- Fixed up the tokenizing rules; now nearly universal and almost perfect (the bracket rules)
- Fixed an issue; element no longer blows up on those ion/compounds, the program can now clearly explain what is wrong with (NH4)2.
- The work so far: most error messages in; a few defects; better tokenization.
- The start of adding parser error messages to the formula parser.
- Added the new acid.pl file.
- Some formatting fixes to an ugly piece of code.
- Really finished fixing up the oxyanion handling.
- Fixed some massive slowness...
- Finished applying the new oxyanion handling.
- Finished the oxyanion processing for formulas; onto names.
- Almost finished fixing the faster oxyanion support.
- Restored a few more ugly hacks to improve the performance even further.
- Re-added a hack that allows for faster operation in reverse.
- The new and improved parser is faster than the old one! (Just get rid of the oxyanion stuff — everything else is just as fast.
- Another round of hacks really improves the performance.
- An ugly hack that improves the performance.
- Eliminate a rule that selects between user format and other formats – it is unnecessary if the original rule is re-written
- Finished re-writting the parser. It will no longer consider NO2 to be nitrite when it is on its own (by requiring that at least one more pair be present when an ion is detected.) It will also force hydrates to be last.
- Nearly finished re-writting the parser.

- Raise the dot in output and lower the 2 in H2O, for hydrates. Still hacky.
- Add support for hydrates in formulas and ionic compounds. A bit hacky.
- Added some missing elements and charges; allow noble gas covalent compounds.
- Commented up the web interface and output formats. Does not really require PlDoc.
- Modularized symbolic and word equations to avoid conflicts, fixed a tiny bug in the output formatting.
- Rename all sorts of functions to make them logical, add some PlDoc, modularize
- Get rid of some inconsistencies: VarS, Vars, Values, etc. plus disambiguate the term coefficient: It is for the chemical equation coefficient only.
- Change most includes to consults for clearer diagnostics; repair a corrupted font file and some indenting fixes.
- Try another parser generator, perhaps.
- Added a style guide, and a lot of wondering about error checking and handling.
- An idea re: renaming input fields
- Add some odd elements, just for fun.
- Added most common elements and ions (from list)
- Added some more elements, including the confusing mercury(I) ion
- DB extension: common name: prussic acid = HCN
- Correct the naming rules for acids: acids not containing oxygen get a prefix, regardless of whether or not they involve polyatomic groups. Also, prevent some ions from forming acids.
- Some cleanup work: remove the singleton variables and an unused file.
- Store formatted group_symbol as well, in order to allow subscripts in group formulas. Switch some things to ensure_loaded to prevent double loading problems.
- Allow the various interfaces to change the output format themselves.
- The plan for fixing a few issues related to output formats.
- Stop the menu from running over two lines

- Some code needed to be changed to allow the output formats to be used
- Finally! Use the new output formatting features. Subscripts and fancy arrows.
- Use the common definition of numbers upto 20
- Finished enough styling for now. Looks pretty fancy now.
- Improve the error messages somewhat, add some red to signify error and include some unfinished projects in the menu
- Make the problems go away (see master re: carbonite)
- Remove carbonite, because it is very rare and causes inconsistent formulas for carbon dioxide
- Added the rest of the forms and some styling
- Make the balancer shut up! Now, the web interface will work
- The Web interface gets some forms
- More reversing of adding exceptions
- Go back to the parser, before we started tinkering with exceptions.
- I'm not so happy about it. But at least we have exceptions, I guess.
- Added some exceptions
- Added a plan for error handling, some errror handling code, re-factored some common parsing rules
- Longest string must be first. Yeesh.
- Added a rough prototype of the web interface. It has no forms, shows the general structure. Also necessary: adding styling and menus
- Remove some unecessary number rules (explicit 1 and 0)
- Let's use it a bit. Not quite sure if this is efficient, or good even...
- Add a stupid output formatting library
- Added a predicate which makes all of the values in the system positive, avoiding dumb solutions that do not work
- Amazingly, the old algorithm for determing the charge on a metal given the non-metal subscript and charge is no good. Fixed it with a better one: Total charge (Subscript * Charge) of the non-metal DIVIDED by the number of metal atoms.

- Add manganese (and permanganate), which seems to really show bugs in the system
- Changed the balancer to use a rational number constraint solver; fixes the issue where something with fractional coefficients won't work, also eliminates the need for that tinkering around to stop an instantiation error when the equation is already balanced.
- Add retained.pl from merge.
- Eliminate some backtracking; retained names and common names (no more hydrooxic acid!) and support for "polyatomic" acids (acetic acid) – which is incomplete (hydrocyanic is correct, not my cyanic acid)
- Initial commit of working, but disorganized program.

Part IV
Implementation History

17 Compounder (2012)

The original version of Chemlogic was the misspelled *compunder*. This version could only convert iconic compound names to formulas and vice-versa. It was written in PHP and had a very simple design.

Instead of implementing a parser with grammatical rules, compunder used two command-line arguments to accept the metal and non-metal components. If a parenthesis was detected in the metal part, a function was used to translate the valence in roman numerals to decimal, which was then used as the charge for the metal.

This version also implemented a quiz program that would create a random ionic compound and ask for its name/formula, depending on the question

18 Prolog Experiments (2013)

After a few months, I decided that I wanted to implement support for covalent compounds and balancing chemical equations. The design of compunder seemed to be unable to support features of this complexity, without a great amount of work.

Around this time, I began to experiment with Prolog, which had built-in support for parsers, through DCGs (Definite Clause Grammars). I stated writing the original parsing rules for chemical equations and formulas.

I studied algorithms for balancing chemical equations, and found a method that used systems of linear equations to represent chemical equations, which was quite convenient, as Prolog had a built-in linear equation solver.

19 Proof-of-Concept Version (2013)

In my Computer Science IDS 10 class, I decided to finish implementing these features. This work took a very long time, as I wanted to implement every naming system used in Science 10. There were very many cases where a parser would produce incorrect output under certain conditions.

I also discovered that my parser design was extremely inefficient. I used inference counting, tracing and other debugging and analysis techniques to find a better design.

After 1 month of development, I had developed all of the features I had planned, but they were not connected to each-other and were spread over many files.

20 Chemlogic 1.0 (2014)

I connected all the pieces of the program together and fixed quite a few bugs. I decided to name the program "Chemlogic". It did not have a user interface, at first, because I only intended to utilize the program as a way to study Computer Science.

My teacher for the Computer Science class, Mr. Tchir, wanted to show my program to his Science 10 class, so I decided to implement a Web interface that was user-friendly enough to be actually useful. This required extending the program to support multiple output formats.

After the presentation, I decided to add one more important feature to the program: error handling. Many times the program would simply give "no." as the answer when incorrect input was given, making it very difficult to track down the error. I wanted the program to be able to give a message explaining what was wrong with the input, and, if possible, to highlight only the incorrect part of the input. This feature was implemented in my Computer Science IDS 11 class.

As it turned out, error handling took nearly as long to implement as the rest of the program and uncovered many bugs in the parsers. I wrote error messages for nearly every syntax error possible in each grammar (e.g. formulas, names, equations, etc.)

After finishing this feature, I decided that I was done with the program and could finally move on to other experiments I had planned for the year.

Mr. Tchir then told me about the West Kootenay Regional Science Fair, which was coming up in two weeks, and suggested that I enter the program. I then spent the two weeks writing a technical paper and a display.

My project won at the local fair, and I attended the Canada-Wide Science Fair.

21 Chemlogic 2.0 (2014—2015)

The next year, I decided to continue to develop my project, and hopefully win the local science fair again. Many people suggested that I develop an Android App, so that more students could use Chemlogic as a study tool. I also considered many other features I could develop, and decided on reaction type analysis (to complete the Science 10 curriculum) and stoichiometry (an entirely new branch of chemistry). My goal was to almost double the number of features in Chemlogic. But, this year, I had only half the time to work.

Developing the Android App took an extremely long time and required a lot of research and experimentation. The most important aspect of the App development was to find a way to use the existing Prolog code of Chemlogic on a mobile phone. I first tested a few JVM-based Prolog implementations, which could allow Prolog code to be integrated directly into the Android App code, but they were all incomplete and poorly-designed. Using the development tools provided by Google to cross-compile a fully-functional Prolog implementation proved futile, as the Android C library was not compatible with any of the Prolog implementations I tested. Finally, I made a breakthrough when I discovered the Angstrom distribution and toolchain, which provided a way to cross-compile programs for Android using the real GNU C library and other Linux libraries. Once I developed a working cross-compilation process, I used Make to automate building SWI-Prolog and Chemlogic for ARM, in order to avoid making time-consuming mistakes when compiling manually.

After I managed to get Chemlogic to run on an Android device, I had to research a way for Chemlogic to be bootstrapped by the App user interface. This, too, proved complex, but I designed a simple script that used the dynamic linker to locate Chemlogic's dependencies and start the program. The user interface of the App was simple and straightforward to develop.

Next, I implemented stoichiometry. Because of the existing modular design of Chemlogic, I could immediately develop new code to manipulate ASTs (Abstract Syntax Trees) generated by parsers, without needing to rewrite existing code. The manipulation of ASTs became the focus of the new features I developed. Unlike the chemical equation balancing process, the methods required to perform stoichiometric calculations manually could be easily translated to a systematic algorithm. To perform limiting reaction analysis in an efficient way, I found a simple algorithm that would compare the ratios between the number of moles of each reactant divided by its equation coefficient, to determine the limiting reactant. Implementing support for significant figures was a significant challenge, as it required working around the built-in number types provided by Prolog and writing customized parsers to count and record the number of significant figures in a given value, before it was converted to a floating-point number. When producing a result, numbers had to be rounded to the correct number of decimal places, and then any truncated zeros would have to be restored.

The implementation of reaction type analysis was also based on manipulating the ASTs for chemical equations. Identifying reaction types based on the patterns of the formulas involved was a simple task, as Prolog's support for syntactic unification could be used to match a chemical equation structure against patterns representing various types of chemical reactions. This information could then be displayed to the user whenever a chemical equation was entered. Predicting whether a reaction would occur required detailed research. First, I compared different methods of comparing the activity of two elements numerically, and decided to simply use the position of an element in the reactivity series as a numerical index for simple testing. This code was then integrated with the reaction type identification feature.

I also wanted the program to be able to complete the equations for some chemical reactions, given only the reactants. This can only be done with a high level of accuracy for a few types of reactions: double replacement, neutralization (special case of the former), single replacement and combustion of hydrocarbons. To complete equations, the necessary re-arrangements of ions and elements were identified by matching the left-hand side of a chemical equation structure against a reaction type pattern. Then, the complete formulas for the product compounds were calculated using existing rules specific to the type of compound involved. For example, in the case of an ionic compound, the charge of the metal becomes the subscript of the non-metal, and vice-versa, then the formula is reduced to lowest terms. Once the formulas were determined, they could simply be substituted back into the equation structure.

Once again, my project won at the local science fair, and I am now preparing for the Canada-Wide Science Fair.