Chemlogic:

A Logic Programming Computer Chemistry System

Nicholas Paun

∗ Abstract

CHEMLOGIC IS A LOGIC PROGRAM for balancing chemical equations and converting chemical formulas to and from chemical names, using a database of chemical element and polyatomic group information, a set of grammars, and a linear equation solver. Chemlogic can detect and provide guidance for resolving syntax and other errors and has a user-friendly Web interface.[6]

Background

IN HIGH SCHOOL CHEMISTRY, students learn to write formulas and names for chemical compounds and to write and balance chemical equations. These concepts are simple, but implementing a program to do this was an interesting task. Algorithms were researched, adapted to chemistry problems and implemented in Prolog to create a program that could be useful in education.

Design and Implementation

USER INPUT IS PARSED into a form that can be easily manipulated and transformed. A parser recognizes a formal grammar describing valid user input. In Prolog, parsers are implemented using DCGs (Definite Clause Grammars), which provide a simplified syntax for creating logical clauses that process a grammar using difference lists, an efficient representation (e.g. concatenation in O(1)).[4]

Difference lists consist of an instantiated part (the head) and an uninstantiated part (the tail), which is always paired with the rest of the lists. Difference lists are a complex concept and are usually abstracted away by DCG syntax.

In DCG clauses, the head of a grammatical rule is unified with the input and any remaining data is unified with the tail, which is passed to the next clause until the parsing is completed. If any clause fails, Prolog will backtrack to find another clause that can satisfy the grammar. If no clause is found, then the parsing fails.

To make a useful program it is not enough to simply decide whether or not a given input conforms to a grammar — internal representations must be created. Commonly, Abstract Syntax Trees are used for this purpose. Chemlogic uses a pseudo-AST to record the structure of an equation, as well as lists containing useful information (e.g. the elements contained in an equation). DCGs provide extra arguments for this purpose.

Balancing of chemical equations is usually done by inspection.[3] This process cannot easily be used in a program because it is unsystematic: as coefficients may be corrected again during the balancing, the order and steps performed may vary from equation to equation.

Chemlogic uses an elegant process, using a system of linear equations. One linear equation is created for every element in a chemical equation, with the number of occurrences of the element in each formula representing a coefficient, multiplied by an unknown (the chemical equation coefficient).[5] To make the system solvable, the first coefficient is set to 1. The solution is always reduced to lowest whole number terms.

This process can be made even simpler to program by creating a homogeneous linear system, where the terms representing reactants have positive sign and terms representing products have negative sign. These equations are all equal to 0. These systems are commonly solved by converting them to a matrix and applying Gaussian elimination.[2]

In Chemlogic, a matrix is produced from structures created by the parser and lookup tables. The matrix is converted into a system of linear equations, which is then provided to the built-in CLP(q) facility, which can solve constraints over rational numbers.[1] This process is less efficient, but allows for code reuse, saving programmer work.

Syntax errors cannot simply cause a program to fail — clear identification and explanation of an error is necessary. When a predicate that must succeed for a given input to be valid fails, a syntax error exception is thrown, containing a code name for the error and whatever remains in the tail (what could not be parsed). The exception aborts the execution of the program and is then passed to the error handling module. It first attempts to localize the error by highlighting only the problematic part within the tail. Different rules are used depending on the type of the first character (e.g. an invalid letter suggests a chemistry mistake, while an invalid symbol suggests a typo). The combination of an error code and character type is used to find the correct error message to provide to the user.

Multiple interfaces are supported in Chemlogic. Currently, command-line and Web interfaces have been implemented. In order to show correct symbols and formatting (e.g. subscripts) in each interface, there is an output formatting module that allows each parser to automatically use the correct symbols. The error handling module re-throws its syntax errors to a simple error handler for each interface.

Metaprogramming is an excellent feature in Prolog. Metaprogramming allows a program to write or manipulate parts of itself. An important aspect of metaprogramming is the ability to manipulate code as a data structure — this is used to translate simple facts from a database into various grammatical rules.

Prolog also allows a programmer to define new operators that extend the programming language and provide simple syntax for repetitive tasks. Chemlogic defines an operator that throws a syntax error, if a predicate fails, and another operator that catches syntax errors and runs the correct handler for the current interface. Defining operators makes Chemlogic's code easier to read and understand.

Chemlogic also implements a very simple Domain Specific Language (DSL) on top of Prolog, using metaprogramming techniques. This DSL is a proof-of-concept and consists of three rules that provide simple syntax for a user to query Chemlogic. DSL clauses can be composed into very simple programs and the full features of Prolog can be combined with DSL rules, if needed.

Discussion

Performance was analyzed in Chemlogic by counting inferences (provided by `time/1`) used by different algorithms for various problem sizes. Algorithms were compared on their fixed inferences (intercept), inferences per item (slope) and to ensure that their complexities were not exponential.

The time taken by the algorithms used in Chemlogic could not be analyzed because the difference between the performance of algorithms on typical problem sizes was immeasurably small.

Further research and development — It would be interesting to compare the balancing algorithm used, with one that attempts to balance equations by trial-and-error. A brute-force algorithm is logically unsatisfying and its difficulty increases exponentially ($\mathrm{O}(n^m)$). The time taken by repeated trials may be unnoticeable on new computers, however.

Currently, Chemlogic attempts to distinguish sub-classes of errors to make error messages more specific. A topic for future development would be to substitute pieces of information from incorrect input into messages, allowing the program to explain exactly what is wrong with the input, as opposed to a general error message.

The program could be extended to add more chemistry features, including: structural formulas and the names of more complex organic compounds, stoichiometric calculations and completion of simple reaction types (a very complex task).

Conclusions

Prolog was chosen as the language for Chemlogic because it has many features that are useful for implementing this type of program.

Prolog includes support for DCGs, which allow a programmer to implement a parser using a very simple syntax, without requiring manual parser writing. Using DCGs, it is easy to write grammatical rules, test them and add more types of input. Chemlogic used many advanced features of DCGs and their underlying abstractions.

SWI-Prolog's CLP(q) facility made solving systems of linear equations extremely simple.

Using a logic programming language, such as Prolog, enables the programmer to describe the results, instead of the process.[7] In practice, writing in Prolog avoids the need for manual programming of loops and other imperative constructs, most often resulting in a well-written, succinct solution requiring few lines of code.

Prolog also has very strong support for metaprogramming, which was used in a few places in Chemlogic, where the amount of boilerplate code needed was reduced, making code easier to read.

Chemlogic was successfully implemented using Prolog, in a well-designed and modular structure, and could balance chemical equations, convert names to formulas and vice versa.

Acknowledgments

I would like to thank the many people who gave advice and helped with the project.

I am particularly grateful for the valuable assistance provided by Dr. Peter Tchir, my Physics, Chemistry and, now, Computer Science teacher. His help and advice, especially with algorithms and his support for my Computer Science projects helped make this program possible.

I would also like to thank Mr. Jason Peil for his assistance in designing and printing the display and Mr. Greg Osadchuk for his input and assistance regarding the visual presentation.

Obtaining Chemlogic / Contact

Nicholas Paun <np@icebergsystems.ca>

Chemlogic is open-source software. A copy of the program and additional information is available at http://icebergsys.ca/chemlogic

References

[1] C. Holzbaur. OEFAI clp(q,r) Manual Rev. 1.3.2. 1995.

[2] Nayuki Minase. Chemical equation balancer (JavaScript), 2013. URL: http://nayuki.eigenstate.org/page/chemical-equation-balancer-javascript.

[3] L. Sandner. *BC Science 10*. McGraw-Hill Ryerson, 2008. URL: http://books.google.ca/books?id=vEjRtgAACAAJ.

[4] Markus Triska. DCG Primer. URL: http://www.logic.at/prolog/dcg.html.

[5] Mark E. Tuckerman. Methods of balancing chemical equations. 2011. URL: http://www.nyu.edu/classes/tuckerman/adv.chem/lectures/lecture_2/node3.html.

[6] Jan Wielemaker, Zhisheng Huang, and Lourens van der Meij. SWI-Prolog and the Web. *Theory and Practice of Logic Programming*, 8(3):363–392, 2008.

[7] Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012.