

Chemlogic:

A Logic Programming Computer Chemistry System

Second Edition

Nicholas Paun

* **ABSTRACT**

CHEMLOGIC IS A LOGIC PROGRAM FOR COMPUTER CHEMISTRY that performs stoichiometric calculations, balances and completes equations, and converts between formulas and names. The program has applications in education, particularly as a study tool. Features are implemented using a chemical information database, linear equation solver, and grammatical rules. Guidance is provided for resolving errors in user input. Chemlogic is available on Android and the Web.

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.

THE REACTION TYPE ANALYSIS MODULE identifies common types of chemical reactions, predicts whether a reaction will take place, and completes chemical equations for common reactions, given the reactants. These features are implemented by matching the Abstract Syntax Tree generated by parsing an equation, against a pattern representing the structure of a certain type of chemical reaction. In Prolog, this matching is performed by syntactic unification, which is the process of combining two structures by replacing variables with constant terms, so as to make the two structures equal.

Once the type of reaction is determined, a prediction can be made as to whether the reaction will occur by comparing the relative positions in the reactivity series of the elements involved in the reaction. If an element cannot be found in the reactivity series, or if there is a conflict between the tendency to react of one pair of elements and the tendency to react

of another pair of elements, the program will simply refrain from making a prediction. The type of reaction and the reaction prediction can then be displayed by the user interfaces, as additional information for the entered chemical equation.

To complete a chemical equation, simply determining the reaction type is insufficient — the complete formulas of the product compounds must be calculated, given the arrangements of elements and ions determined by reaction pattern. This process is performed using rules specific to the type of compound being formed by the reaction. For example, in the case of an ionic compound, the charge of the cation becomes the subscript of the anion, and vice-versa, then the formula is reduced to lowest terms. The complete formulas are then substituted into the equation structure, to produce the representation of the complete chemical equation. This structure can then be used as input by other modules, such as the chemical equation balancer.

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. STOICHIOMETRY IS AN IMPORTANT NEW FEATURE of Chemlogic. The program can now be used to perform a variety of complex calculations involving quantities of chemicals in a reaction. The process of performing calculations begins by parsing a chemical equation that includes quantities for some compounds. A parser is also used to interpret the queries provided by the user.

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, Android, 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.

THE ANDROID APP WAS DEVELOPED to make Chemlogic available as a study tool to more users. The App consists of a user-interface, implemented in Java, using the Android APIs, and a package consisting of the cross-compiled code of Chemlogic and its dependencies.

The user interface communicates with Chemlogic through a UNIX pipe. Upon receiving the solution from Chemlogic, the interface renders the formatting of the result and displays it.

The Chemlogic package contains a copy of the Chemlogic command-line interface, compiled as a stand-alone application with an embedded Prolog interpreter in its binary, some necessary Linux libraries and an initialization script. As cross-compiling these programs for various Android platforms and CPU architectures is a complicated process, an automated procedure is used to build the package, in order to reduce errors. When loaded, the App executes the initialization script, which runs the dynamic linker to locate and link Chemlogic with the provided libraries and then executes it.

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 ($O(n^m)$). 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 parsers currently implemented in Chemlogic process user input directly, as character lists, without a tokenization process. Using new features available in version 7 of SWI-Prolog, a simple and fast procedure could be implemented to separate user input into space-delimited parts. The correct parser for the tokenized input could then be easily identified by syntactic unification, significantly reducing the number of tests and operations required to perform parsing.

An important extension to Chemlogic, would be to add support for automatically gen-

erating and marking random chemistry problems. This feature could be used by students to review concepts and by teachers to automate the marking of simple problems. With the addition of support for reaction type analysis, the preliminary work for this feature has been completed. The reaction type, equation, formula and name grammars in Chemlogic could be extended to produce random valid structures, in addition to simply recognizing and converting them. These structures could then be included in problems presented by a new user interface.

The program could be extended to add more chemistry features, including support for: electronic structures of atoms, structural formulas and the names of more complex organic compounds, and equilibrium calculations and analysis.

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.^[6] 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.

EACH MODULE OF CHEMLOGIC is designed to transform the standardized Abstract Syntax Trees generated by the parsers, allowing for modules to be composed together, each adding a piece of functionality as structures are passed from predicate to predicate. In this way, existing modules can make use of new functionality, new features can be implemented based on existing calculations and structures and modules can be connected to as part of integrated user interfaces.

New features were integrated into the existing structure of Chemlogic without requiring significant rewriting of the code. Most changes consisted of modifying parsers and other procedures to expose more information to other modules of the program.

CHEMLOGIC WAS SUCCESSFULLY EXTENDED to implement new features, based upon the existing well-designed and modular structure, and can now perform stoichiometric calculations, convert chemical quantities between units and complete chemical reactions.

EARLIER WORK

The original version of Chemlogic was an experimental program I developed in 2012, to simplify my Science homework.

Based on these experiments, the project was extended in 2013 and 2014. The program was rewritten in Prolog, new algorithms were studied and adapted and many new features were implemented. The current modular design and structure of Chemlogic was implemented in Version 1, which supported balancing chemical equations, converting chemical names to formulas, and vice-versa. This version was presented at the 2014 CWSF in Windsor, where it received a Bronze Excellence Award.

In 2014 and 2015, Chemlogic was further developed to significantly expand the project, by continuing to improve existing features, adding a new user interface and adding support for higher levels of high school Chemistry. Version 2 of Chemlogic introduced the Android user interface and support for stoichiometric calculations, chemical unit conversions and reaction type analysis.

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, now retired, my Physics, Chemistry and 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. Holzbaaur. 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, Tom Schrijvers, Markus Triska, and Torbjörn Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012.