Software Requirements Specification for Chemistry Code

Samuel J. Crawford

March 31, 2023

Contents

1 Reference Material

This section records information for easy reference.

1.1 Table of Symbols

The symbols used in this document are summarized in the Table of Symbols along with their units. The symbols are listed in alphabetical order.

Symbol	Description	Units
$\mathbf{0}$	Zero vector	–
$\mathbf{1}$	Unary vector	–
$\mathbf{A}$	Generic matrix	–
$\mathbf{b}$	Generic vector	–
C	Compound data type	–
c	Generic compound	–
$\mathbf{c}$	Generic vector	–
count	Count of an element in a compound	–
E	Element data type	–
$\mathbf{E}$	Matrix representation of a chemical equation	–
e	Generic element	–
R	Reaction data type	–
r	Representation of a chemical equation	–
t_C	Generic tuple of a compound	–
$\mathbf{x}$	Generic vector	–

Table 1: Table of Symbols

1.2 Abbreviations and Acronyms

Abbreviation	Full Form
ChemCode	Chemistry Code
DD	Data Definition
IM	Instance Model
R	Requirement
SRS	Software Requirements Specification
TM	Theoretical Model
UC	Unlikely Change

Table 2: Abbreviations and Acronyms

2 Introduction

Chemical equations are common ways of representing chemical reactions but they must be balanced [**lund2023**]. This process of balancing a chemical equation involves introducing coefficients before each chemical formula such that there are the same number of atoms of each element on the reactant and product sides of the chemical equation. Because balancing must be done before a given chemical reaction can be used [**lund2023**], it is useful to have a tool to automatically do this. This would improve the productivity of scientists and engineers and reduce the potential for human error. This program should balance a given chemical equation if it is feasible and if it is not, it should provide a descriptive message communicating this to the user. The program that performs these tasks as documented here will be called Chemistry Code (ChemCode).

The following section provides an overview of the Software Requirements Specification (SRS) for ChemCode. This section explains the purpose of this document, the scope of the requirements, the characteristics of the intended reader, and the organization of the document.

2.1 Purpose of Document

The primary purpose of this document is to record the requirements of the Chemistry Code. Goals, assumptions, theoretical models, definitions, and other model derivation information are specified, allowing the reader to fully understand and verify the purpose and scientific basis of ChemCode. With the exception of system constraints, this SRS will remain abstract, describing what problem is being solved, but not how to solve it.

This document will be used as a starting point for subsequent development phases, including writing the design specification and the software verification and validation plan. The design document will show how the requirements are to be realized, including decisions on the numerical algorithms and programming environment. The verification and validation plan will show the steps that will be used to increase confidence in the software documentation and the implementation. Although the SRS fits in a series of documents that follow the so-called waterfall model, the actual development process is not constrained in any way. Even when the waterfall model is not followed, as Parnas and Clements point out [**parnasClements1986**], the most logical way to present the documentation is still to "fake" a rational design process.

2.2 Scope of Requirements

The scope of the requirements includes all chemical equations with at most one more compound than element. Furthermore, it also includes all inputted chemical formulas that describe real chemical compounds, are formatted following a set of conventions, and only consist of atomic symbols and subscripts.

2.3 Organization of Document

The organization of this document follows the template for an SRS for scientific computing software proposed by [**koothoor2013**] and [**smithLai2005**]. The presentation follows the standard pattern of presenting goals, theories, definitions, and assumptions. For readers that would like a more bottom up approach, they can start reading the instance models and trace back to find any additional information they require.

The goal statements are refined to the theoretical models and the theoretical models to the instance models.

3 Specific System Description

This section first presents the problem description, which gives a high-level view of the problem to be solved. This is followed by the solution characteristics specification, which presents the assumptions, theories, and definitions that are used.

3.1 Problem Description

A system is needed to balance chemical equations with the smallest possible whole number coefficients so they can be useful for other computations.

3.1.1 Terminology and Definitions

This subsection provides a list of terms that are used in the subsequent sections and their meaning, with the purpose of reducing ambiguity and making it easier to correctly understand the requirements.

- Compound: A molecule made up of more than one atom, which may or may not be of different elements.
- Element: The group of all atoms with the same number of protons in the atomic nucleus. For example, all atoms with one proton are hydrogen atoms.
- Equation: A textual representation of a chemical reaction.
- Product: A substance formed by a chemical reaction.
- Reactant: A substance involved in and changed by a chemical reaction.
- Reaction: An interaction between different types of matter that results in at least one new substance being formed [**lund2023**].

3.2 Solution Characteristics Specification

The instance models that govern ChemCode are presented in the Instance Model Section. The information to understand the meaning of the instance models and their derivation is also presented, so that the instance models can be verified.

3.2.1 Assumptions

This section simplifies the original problem and helps in developing the theoretical models by filling in the missing information for the physical system. The assumptions refine the scope by providing more detail.

elemCompDiff: For all chemical equations, there is at most one more compound than element. (RefBy: UC:allEqsPermitted.)

validForms: All inputted chemical formulas describe real chemical compounds. (RefBy: UC:checkValidForms.)

validEqns: All inputted chemical equations describe real chemical reactions. (RefBy: UC:checkValidEqns.)

3.2.2 Theoretical Models

This section focuses on the general equations and laws that ChemCode is based on.

Refname	**TM:canonIntLinProg**
Label	Canonical integer linear program
Equation	$$\mathbf{cx}$$ $$\mathbf{Ax} \leq \mathbf{b}$$ $$\mathbf{x} \geq \mathbf{0}$$ $$\mathbf{x} \in \mathbb{Z}^n$$
Description	**c** is the generic vector (Unitless) **x** is the generic vector (Unitless) **A** is the generic matrix (Unitless) **x** is the generic vector (Unitless) **b** is the generic vector (Unitless) **x** is the generic vector (Unitless) **0** is the zero vector (Unitless) **x** is the generic vector (Unitless)
Notes	The above equation gives the canonical form of an integer linear program, which is "a mathematical optimization or feasibility program in which some or all of the variables are restricted to be integers [and] the objective function and the constraints (other than the integer constraints) are linear" [**ilpWiki**] The values of **x** are unknown and will be solved for
Source	[**ilpWiki**]
RefBy	IM:chemEqIntLinProg

3.2.3 General Definitions

There are no general definitions.

3.2.4 Data Definitions

This section collects and defines all the data needed to build the instance models.

Refname	**DD:countFunc**
Label	Count of an element in a compound
Symbol	count
Equation	$$\text{count} = \begin{cases} t_{C_1}, & t_C \in c \wedge t_{C_0} = e \\ 0, & \neg t_C \in c \wedge t_{C_0} = e \end{cases}$$
Description	count is the count of an element in a compound (Unitless) t_C is the generic tuple of a compound (Unitless) c is the generic compound (Unitless) e is the generic element (Unitless)
Notes	The output represents the number of atoms of a given element e (of type E) in a given compound c (of type C)
Source	–
RefBy	

Refname	**DD:elementType**
Label	Element data type
Symbol	E
Equation	$E = \{\text{H}, \text{He}, \text{Li}, \text{Be}, \text{B}, \text{C}, \text{N}, \text{O}, \text{F}, \text{Ne}, \text{Na}, \text{Mg}, \text{Al}, \text{Si}, \text{P}, \text{S}, \text{Cl}, \text{Ar}, \text{K}, \text{Ca}, \text{Sc}, \text{Ti}, \text{V}, \text{Cr}, \text{Mn}, \text{Fe},$
Description	E is the element data type (Unitless)
Notes	A type representing each element from [**elemListWiki**]
Source	[**smithChemSpec**]
RefBy	

Refname	**DD:compoundType**
Label	Compound data type
Symbol	C
Equation	$C = \text{sequence of tuple of (elem} \in E\text{, count} \in \mathbb{R})$
Description	C is the compound data type (Unitless)
Notes	A type representing a compound
Source	–
RefBy	

Refname	**DD:reactionType**
Label	Reaction data type
Symbol	R
Equation	$R =$ tuple of (prod $\in$ sequence of tuple of (comp $\in C$, coeff $\in \mathbb{Z}$) , reac $\in$ sequence of tuple
Description	R is the reaction data type (Unitless)
Notes	A type representing a reaction
Source	–
RefBy	

3.2.5 Instance Models

This section transforms the problem defined in the problem description into one which is expressed in mathematical terms. It uses concrete symbols defined in the data definitions to replace the abstract symbols in the models identified in theoretical models and general definitions.

Refname	**IM:chemEqIntLinProg**
Label	Integer linear program for a chemical equation
Input	**E**
Output	**x**
Equation	minimize $\mathbf{1} \cdot \mathbf{x}$ subject to $\mathbf{Ex} = \mathbf{0}$, $\mathbf{x} > \mathbf{0}$, and $\mathbf{x} \in \mathbb{Z}^n$
Description	**1** is the unary vector (Unitless) **x** is the generic vector (Unitless) **E** is the matrix representation of a chemical equation (Unitless) **0** is the zero vector (Unitless)
Notes	This is a specific instance of the ILP from TM:canonIntLinProg with $\mathbf{A} = \mathbf{E}$, $\mathbf{b} = \mathbf{0}$, and $\mathbf{c} = \mathbf{1}$. The goal and constraints are also modified.
Source	–
RefBy	

4 Requirements

This section provides the functional requirements, the tasks and behaviours that the software is expected to complete, and the non-functional requirements, the qualities that the software is expected to exhibit.

4.1 Functional Requirements

This section provides the functional requirements, the tasks and behaviours that the software is expected to complete.

Input-Values: Input the values from Tab:ReqInputs.

Convert-to-Matrix: Convert the inputted chemical equation to matrix form.

Feasible-Output: If the inputted chemical equation is feasible, output its balanced form with the smallest positive integer coefficients possible.

Infeasible-Output: If the inputted chemical equation is infeasible, output a descriptive message.

Symbol	Description	Units
r	Representation of a chemical equation	–

Table 3: Required Inputs following FR:Input-Values

4.2 Non-Functional Requirements

This section provides the non-functional requirements, the qualities that the software is expected to exhibit.

Accurate: Chemical equations are only useful if they are balanced, so computed coefficients should be exact.

Verifiable: The code is tested with complete verification and validation plan.

Reusable: The code is modularized.

Portable: The code is able to be run in different environments.

5 Likely Changes

This section lists the likely changes to be made to the software.

6 Unlikely Changes

This section lists the unlikely changes to be made to the software.

allEqsPermitted: A:elemCompDiff assumes...

checkValidForms: A:validForms assumes...

checkValidEqns: A:validEqns assumes...

7 Traceability Matrices and Graphs

The purpose of the traceability matrices is to provide easy references on what has to be additionally modified if a certain component is changed. Every time a component is changed, the items in the column of that component that are marked with an "X" should be modified as well. Tab:TraceMatAvsA shows the dependencies of assumptions on the assumptions. Tab:TraceMatAvsAll shows the dependencies of data definitions, theoretical models, general definitions, instance models, requirements, likely changes, and unlikely changes on the assumptions. Tab:TraceMatRefvsRef shows the dependencies of data definitions, theoretical models, general definitions, and instance models with each other. Tab:TraceMatAllvsR shows the dependencies of requirements on the data definitions, theoretical models, general definitions, and instance models.

	A:elemCompDiff	**A:validForms**	**A:validEqns**
A:elemCompDiff			
A:validForms			
A:validEqns			

Table 4: Traceability Matrix Showing the Connections Between Assumptions and Other Assumptions

	A:elemCompDiff	**A:validForms**	**A:validEqns**
DD:countFunc			
DD:elementType			
DD:compoundType			
DD:reactionType			
TM:canonIntLinProg			
IM:chemEqIntLinProg			
FR:Input-Values			
FR:Convert-to-Matrix			
FR:Feasible-Output			
FR:Infeasible-Output			
NFR:Accurate			
NFR:Verifiable			
NFR:Reusable			
NFR:Portable			
UC:allEqsPermitted	X		
UC:checkValidForms		X	
UC:checkValidEqns			X

	A:elemCompDiff	A:validForms	A:validEqns

Table 5: Traceability Matrix Showing the Connections Between Assumptions and Other Items

	DD:countFunc	DD:elementType	DD:compoundType	DD:reactionType	TM:canonIn
DD:countFunc					
DD:elementType					
DD:compoundType					
DD:reactionType					
TM:canonIntLinProg					
IM:chemEqIntLinProg					

Table 6: Traceability Matrix Showing the Connections Between Ite

	DD:countFunc	DD:elementType	DD:compoundType	DD:reactionType	TM:canonIn
FR:Input-Values					
FR:Convert-to-Matrix					
FR:Feasible-Output					
FR:Infeasible-Output					
NFR:Accurate					
NFR:Verifiable					
NFR:Reusable					
NFR:Portable					

The purpose of the traceability graphs is also to provide easy references on what has to be additionally modified if a certain component is changed. The arrows in the graphs represent dependencies. The component at the tail of an arrow is depended on by the component at the head of that arrow. Therefore, if a component is changed, the components that it points to should also be changed. Fig:TraceGraphAvsA shows the dependencies of assumptions on the assumptions. Fig:TraceGraphAvsAll shows the dependencies of data definitions, theoretical models, general definitions, instance models, requirements, likely changes, and unlikely changes on the assumptions. Fig:TraceGraphRefvsRef shows the dependencies of data definitions, theoretical models, general definitions, and instance models with each other. Fig:TraceGraphAllvsR shows the dependencies of requirements on the data definitions, theoretical models, general definitions, and instance models. Fig:TraceGraphAllvsAll shows the dependencies of dependencies of assumptions, models, definitions, requirements, goals, and changes with each other.

Figure 1: TraceGraphAvsA

Figure 2: TraceGraphAvsAll

For convenience, the following graphs can be found at the links below:

- TraceGraphAvsA
- TraceGraphAvsAll
- TraceGraphRefvsRef
- TraceGraphAllvsR
- TraceGraphAllvsAll

8 References

Figure 3: TraceGraphRefvsRef

Figure 4: TraceGraphAllvsR

Figure 5: TraceGraphAllvsAll