Chemical Programming: Models, Concepts, and Designs

Joshua Weinstein

2018

Contents

0.1 Preface

This book is intended to serve as a reference and guide toward programming in a chemical paradigm. It serves three main purposes. The first, and most important, is addressing the methods and models in which chemical processes can be utilized for programming. Chemical programming is an extremely young paradigm, with much to explore and create. Parts of this text will teach concepts of how to program chemically in existing programming languages.

The second is to illustrate design patterns that can be implemented in software and programming systems. Chemical programming offers an array of algorithms that can simplify common tasks such as sorting, filtering, searching and much more. The hope is to promote the development of chemical programming as a prominent paradigm in modern software development. The third, is to detail the creation of machines that can process entirely chemical programs. This includes the creation and design of a chemical virtual machine, and paths to create chemical programming languages.

This book is *not* a tutorial in how to design programs that deal with real chemical process or chemical engineering. Programming techniques for dealing with problems in chemical analysis is also not a concern of this book. Some principles of chemistry will be discussed, but not developing programs to deal with actual chemical compounds or data. The focus here is not framed around scientific computing.

Chapter 1

Introduction

Chemical Programming is a paradigm of programming that is modeled after the mechanics of chemical reactions. This style focuses on utilizing the features of chemical components, such as atoms, elements, compounds and reactions. The intention of this programming paradigm is to allow the functionality and mechanisms in chemical processes to be applied to computational values. Chemical computation permits highly customizable abstraction over traditional types like booleans, integers, and strings.

Purely modeling chemical reactions is not the goal, rather defining and modeling the components of chemistry to make them useful in programming. Computer science and chemistry are well defined, yet very different fields. The fashions and organizations of chemical reactions provide unique interfaces for program design. However, moles, molecular orbitals, and energy levels are not fruitful or applicable to abstract program designs, or a programming paradigm. Another goal in this book is to illustrate models and definitions such that components of chemicals can exist more fruitfully as a programming construct.

Traditionally, programming languages use typed values or data, such as integers, booleans, or characters. These are commonly referred to as *primitive* types. Such a type is normally the most basic level of abstraction in a programming language. They cannot be further decomposed into simpler types. The higher level abstractions such as functions or classes are composed of primitive types, which allows the creation of *user-defined* types. Similarly, elements, the most basic *type* of matter, exists in it's smallest form as an atom, which compose molecules and more sophisticated compounds.

Some fundamentals in chemistry are rather limiting when judged in a computational perspective. Reactions usually require an activation energy, E_a, in order to being and transition from reactants to products. This energy arises from a combination of heat, pressure, acidity, and other environment factors. This pushes the reactants toward a transition state, allowing the reaction to occur.

If a set of reactants does not exist in an environment or under conditions *favorable* to the products, a reaction will not occur. Similarly, a reaction can

$$A + B \xrightarrow{e} AB$$
$$B + C \rightleftharpoons BC$$

Figure 1.1: A chemical reaction with energy, and a reversible reaction

reverse itself if conditions or available energy favor the reactants. This flow of requirements and barriers does not truly relate to computational models, as restrictions in programming paradigms are self-imposed, they are not technological or physical restrictions. To adhere to the loose abstractions that a chemical program could offer, these restrictions from actual chemistry are not obeyed.

One of the main differences between chemistry and chemical programming is that reactions are *definite* transitions. A reaction can occur in a mutable or immutable state, but a reactant cannot spontaneously revert or undergo more reactions. The useful part of loose abstraction is giving a programmer more control over occurring events. *Conservation* of resources like mass, does not literally apply in chemical programming. New data is simply written to new blocks of memory in any computer, just like new objects are created in object-oriented programming languages. The return to some equilibrium or entropy is not required.

In chemistry, reactions deal with one or more molecules transforming into new molecules. As discussed in later chapters, the term *compounds* is used exclusively in this text, since a value in chemical programming does not possess real physical size. Similarly to real molecules, compounds are composed of one or more *elements*. A 32-bit integer, for example, is an element. The number zero, of any size, can also be an element. Like mentioned earlier, think of primitive types in other paradigms as equivalent to elements.

An element can be modeled in a structured or unstructured context. The structure, or *order* of the elements in a structure may or may not matter in a particular use case. Modeling for both structured or unstructured compounds can provide an array of possible reactions and computations in chemical programming. Both approaches will be described later on.

$$B_2C_2$$

$$\begin{array}{ccc} B & \text{——} & C \\ | & & | \\ B & \text{——} & C \end{array}$$

Figure 1.2: Compounds in structured versus unstructured forms

Lastly, and most importantly, the purpose in using chemical phenomena as a

backbone to programming is to allow the application of *self-organization*. Nearly any language used today only allows instructed organization of a program.

Chapter 2

Sets

In order to understand the modeling behind chemical programming, you must first be acquainted with set theory and operations that can be performed on sets. The designs and models will utilize several types of sets, including partially ordered sets and multi-sets. This chapter is a fundamental guide toward understanding set theory relevant for chemical programming, by defining key relationships and properties. It defines sets, operations like unions, intersections, complements, and more.

Sets serve as an efficient and simple representation of most concepts in chemical programming. Sets can be ordered or unordered. They can be distinct or indistinct. A set can be a subset or superset of another set. Set representation is a convenient method of expressing chemical processes in mathematical terms. Types of sets that are unique to chemical program models will also be defined and discussed in this chapter.

2.1 What is a set?

A set is a collection of one or more elements. It is usually denoted with curly braces, {}. A set's elements can be expressed as some series of E_n elements.

$$S = \{E_n \dots E_k\}$$

If an element, E is contained in a set S, then,

$$E \in S$$

and if a set S has only one member, element E, then,

$$E \equiv S,$$
$$E \equiv \{E\}$$

Similarly, if an element is not in some set, this is written as $E \notin S$.

Chapter 3

Compounds

Compounds are foo.