Assignment 2 Solution

Your name here

February 16, 2020

This report outlines the results of implementing a simplified chemical equation balancing program as specified by the provided MIS. The MIS includes a mix of interfaces and modules with generability and modularity in mind. This report will also discuss test results, critiques of the given design specification, and answer questions about software principles/design in general.

1 Assumptions

- To make testing simple, I assumed that the calculated reaction coefficients must be in their lowest whole number form.
- Since python is weakly typed langauge, some of the specifications weren't directly addressed. Technically, ElmSet and MolecSet are the same as just Set. For example, ElmSet is suppose to only contain elements of type ElementT as per the MIS, but there are actually no restrictions on the element types that may be added to the set.
- Although ElmSet and MolecSet aren't strict as per what their set elements contain, any functions that are suppose to accept parameters of type ElmSet or MolecSet are required to have inputs of that type, otherwise a ValueError is raised (for example, CompoundT's input must be of type MolecSet).
- If the linear system for the reaction coefficients was not solvable, raise a ValueError.

2 Testing of the Original Program

3 Results of Testing Partner's Code

4 Critique of Given Design Specification

5 Answers

a)

F Code for ChemTypes.py

```
## @file ChemTypes.py
#  @author Jay Mody
#  @brief Provides module for chemistry types (ElementT).
#  @date 08/02/20 (dd/mm/yy)

from enum import Enum, auto

## @brief A module that represents the elements in the periodic table.
class ElementT(Enum):
    H = auto()
    He = auto()
    Li = auto()
    Be = auto()
    B = auto()
    C = auto()
    N = auto()
    O = auto()
    F = auto()
    Ne = auto()
    Na = auto()
    Mg = auto()
    Al = auto()
    Si = auto()
    P = auto()
    S = auto()
    Cl = auto()
    Ar = auto()
    K = auto()
    Ca = auto()
    Sc = auto()
    Ti = auto()
    V = auto()
    Cr = auto()
    Mn = auto()
    Fe = auto()
    Co = auto()
    Ni = auto()
    Cu = auto()
    Zn = auto()
    Ga = auto()
    Ge = auto()
    As = auto()
    Se = auto()
    Br = auto()
    Kr = auto()
    Rb = auto()
    Sr = auto()
    Y = auto()
    Zr = auto()
    Nb = auto()
    Mo = auto()
    Tc = auto()
    Ru = auto()
    Rh = auto()
    Pd = auto()
    Ag = auto()
    Cd = auto()
    In = auto()
    Sn = auto()
    Sb = auto()
    Te = auto()
    I = auto()
    Xe = auto()
    Cs = auto()
    Ba = auto()
    La = auto()
    Ce = auto()
    Pr = auto()
    Nd = auto()
    Pm = auto()
    Sm = auto()
    Eu = auto()
    Gd = auto()
    Tb = auto()
    Dy = auto()
    Ho = auto()
```

```
Er = auto()
Tm = auto()
Yb = auto()
Lu = auto()
Hf = auto()
Ta = auto()
W = auto()
Re = auto()
Os = auto()
Ir = auto()
Pt = auto()
Au = auto()
Hg = auto()
Tl = auto()
Pb = auto()
Bi = auto()
Po = auto()
At = auto()
Rn = auto()
Fr = auto()
Ra = auto()
Ac = auto()
Th = auto()
Pa = auto()
U = auto()
Np = auto()
Pu = auto()
Am = auto()
Cm = auto()
Bk = auto()
Cf = auto()
Es = auto()
Fm = auto()
Md = auto()
No = auto()
Lr = auto()
Rf = auto()
Db = auto()
Sg = auto()
Bh = auto()
Hs = auto()
Mt = auto()
Ds = auto()
Rg = auto()
Cn = auto()
Nh = auto()
Fl = auto()
Mc = auto()
Lv = auto()
Ts = auto()
Og = auto()
```

G Code for ChemEntity.py

```
## @file ChemEntity.py
#  @author Jay Mody
#  @brief Provides module interface for chemical entities (molecules, compounds, etc ...).
#  @date 08/02/20 (dd/mm/yy)

## @brief A module interface for chemical entities.
#  @details A module interface for chemical entities like molecules and compounds.
class ChemEntity:

    ## @brief Counts the number of atoms of a specific element.
    #  @param e The ElementT atom to count.
    #  @return An integer for the number of atoms e.
    def num_atoms(self, e):
        raise NotImplementedError

    ## @brief Gets the set of all elements that constitute this entity.
    #  @return An ElmSet of all the ElementT's in this entity.
    def constit_elems(self):
        raise NotImplementedError
```

H Code for Equality.py

```
## @file Equality.py
#  @author Jay Mody
#  @brief Provides generic interface module for equality.
#  @date 08/02/20 (dd/mm/yy)

## @brief A generic interface for modules with equality properties.
class Equality:
    ## @brief Determines if this object is equivalent to object T.
    #  @param T The object to be compared with.
    #  @returns A boolean that is True if T is equivalent to this object, else False
    def equals(self, T):
        raise NotImplementedError

    ## @brief Allows equivalency comparisons via ==
    #  @details If A and B are objects that implement Equality, then this function makes A == B the
        same as A.equals(B).
    #  @returns A boolean that is True if T is equivalent to this object, else False
    def __eq__(self, T):
        return self.equals(T)
```

I Code for Set.py

```
## @file Set.py
#  @author Benjamin Kostiuk
#  @brief Module that defines the Set ADT
#  @details Assumes that the Set constructor is called for each object instance
#           before any other access methods are called.
#  @date 02/01/2020

from Equality import Equality


## @brief An abstract data type that represents a set
class Set(Equality):

    ## @brief Set constructor
    #  @details Initializes a Set object whose state consists of a set of elements
    #  @param s Sequence of elements with which to initialize the Set
    def __init__(self, s):
        self.S = set(s)

    ## @brief Add an element to the set
    #  @param Element to be added to the set
    def add(self, e):
        self.S = self.S.union({e})

    ## @brief Remove an element from the set
    #  @param e Element to be removed from the set
    #  @throws ValueError if element to be removed cannot be found in the set
    def rm(self, e):
        if not self.member(e):
            raise ValueError("Cannot remove element not found in set.")
        self.S = self.S.difference({e})

    ## @brief Determine whether an element is in the set
    #  @param e Element to check whether in the set
    #  @return True if the element is in the set, otherwise false
    def member(self, e):
        return e in self.S

    ## @brief Get the size of the set
    #  @return The size of the set
    def size(self):
        return len(self.S)

    ## @brief Determine if the Set is equal to another set
    #  @details A Set is considered equal if all elements in one set are in another
    #  @param r Set to compare with
    #  @return True if the two Sets are equal, otherwise false
    def equals(self, r):
        if self.size() != r.size():
            return False

        for element in self.S:
            if not r.member(element):
                return False
        return True

    ## @brief Returns a sequence of all elements in the set
    #  @return A sequence of all elements in the set
    def to_seq(self):
        return list(self.S)

    def __eq__(self, value):
        return self.equals(value)
```

J Code for ElmSet.py

```
## @file ElemSet.py
#  @author Jay Mody
#  @brief Provides Set module for ElementT objects.
#  @date 08/02/20 (dd/mm/yy)

from Set import Set

## @brief A module for Sets of elements with type ElementT.
class ElmSet(Set):
    pass
```

K Code for MolecSet.py

```
## @file MolecSet.py
#  @author Jay Mody
#  @brief Provides Set module for MoleculeT objects.
#  @date 08/02/20 (dd/mm/yy)

from Set import Set

## @brief A module for Sets of elements with type MoleculeT.
class MolecSet(Set):
    pass
```

L Code for CompoundT.py

```
## @file CompoundT.py
#  @author Benjamin Kostiuk
#  @brief Module defines the CompoundT ADT for chemical compound representation
#  @date 02/01/2020

from MoleculeT import *
from MolecSet import *
from ElmSet import *


## @brief An abstract data type that represents a chemical compound
class CompoundT(ChemEntity, Equality):

    ## @brief CompoundT constructor
    #  @details Initializes a CompoundT object whose state consists of a MolecSet
    #  @param m MolecSet of molecules in the chemical compound
    def __init__(self, m):
        self.C = m

    ## @brief Get the MolecSet of molecules in the chemical compound
    #  @return The MolecSet of molecules in the chemical compound
    def get_molec_set(self):
        return self.C

    ## @brief Get the number of atoms of a given ElementT in the chemical compound
    #  @param e ElementT to check for in chemical compound
    #  @return The number of atoms of the specified ElementT in the chemical compound
    def num_atoms(self, e):
        count = 0
        for m in self.C.to_seq():
            count += m.num_atoms(e)
        return count

    ## @brief Return an ElmSet of the ElementTs in the chemical compound
    #  @return An ElmSet of the ElementTs in the chemical compound
    def constit_elems(self):
        return ElmSet([m.get_elm() for m in self.C.to_seq()])

    ## @brief Determine if the chemical compound is equal to another chemical compound
    #  @details Two chemical compounds are considered equal if they have
    #           all the same molecules in them
    #  @param d CompoundT to compare with
    #  @return True if the chemical compounds are equal, otherwise false
    def equals(self, d):
        return self.C.equals(d.get_molec_set())

    def __eq__(self, value):
        return self.equals(value)
```

M Code for ReactionT.py

```
## @file ReactionT.py
#  @author Benjamin Kostiuk
#  @brief Module defines the ReactionT ADT for representing checmical reactions
#  @date 02/05/2020

from CompoundT import *

import numpy as np
from sympy import Matrix, lcm


## @brief An abstract data type that represents a chemical reaction
class ReactionT:

    ## @brief ReactionT constructor
    #  @details Initializes a ReactionT object whose state consists of a sequence of
    #           reactants a sequence of its coefficients, a sequence of products
    #           and a sequence of its coefficients. The sequences of coefficients
    #           are computed as to balance the chemical reaction with an equal
    #           number of elements on both sides.
    #  @param reactants Sequence of CompoundT in the left-hand side of the chemical
    #                   reaction, known as reactants
    #  @param products Sequence of CompoundT in the right-hand side of the chemical
    #                  reaction, known as products
    #  @throws ValueError if the elements in the reactants do not match the elements
    #                     in the products, the two sides of the reaction cannot be
    #                     balanced, any of coefficients are non-positive or if the
    #                     the sequences of coefficients do not match their
    #                     respective side of the chemical reaction
    def __init__(self, reactants, products):
        # Get ElmSet of ElementTs in L and R
        lhs_elems = self.__elements_in_equation__(reactants)
        rhs_elems = self.__elements_in_equation__(products)

        # Check that elements in the reactants and the products are the same
        if not lhs_elems.equals(rhs_elems):
            raise ValueError("Elements in reactants must match elements in products.")

        # Get coefficient matrix to solve linear equation
        lhs_coeffs, rhs_coeffs = self.__solve_matrix__(reactants, products, lhs_elems)

        # Check if length of lists match
        if len(lhs_coeffs) != len(reactants) or len(rhs_coeffs) != len(products):
            raise ValueError("Cannot match coefficients to reactants and products.")

        # Check if coefficients are balanced
        for element in lhs_elems.to_seq():
            if not self.__is_balanced__(reactants, products, lhs_coeffs, rhs_coeffs, element):
                raise ValueError("Invalid ReactionT. Reaction cannot be balanced.")

        self.lhs = reactants
        self.rhs = products
        self.coeff_L = lhs_coeffs
        self.coeff_R = rhs_coeffs

    ## @brief Get the sequence of reactants of the chemical reaction
    #  @return The sequence of CompoundT in the left-hand side of the chemical reaction
    def get_lhs(self):
        return self.lhs

    ## @brief Get the sequence of products of the chemical reaction
    #  @return The sequence of CompoundT in the right-hand side of the chemical reaction
    def get_rhs(self):
        return self.rhs

    ## @brief Get the sequence of coefficients in the left-hand side of the chemical reaction
    #  @return The sequence of coefficients in the left-hand side of the chemical reaction
    def get_lhs_coeff(self):
        return self.coeff_L

    ## @brief Get the sequence of coefficients in the right-hand side of the chemical reaction
    #  @return The sequence of coefficients in the right-hand side of the chemical reaction
    def get_rhs_coeff(self):
        return self.coeff_R

    # Returns an ElmSet of ElementT in a list of CompoundTs
```

```
def __elements_in_equation__(self, equation):
    elems = []
    for compound in equation:
        elems += compound.constit_elems().to_seq()
    return ElmSet(elems)

# Check if a ReactionTs coefficients for reactants and products are balanced
def __is_balanced__(self, reactants, products, left_coeffs, right_coeffs, element):
    lhs_count, rhs_count = 0, 0
    # Count element for left hand side
    for i in range(len(reactants)):
        if left_coeffs[i] <= 0:
            raise ValueError("Invalid ReactionT. Coefficients must be positive.")
        lhs_count += left_coeffs[i] * reactants[i].num_atoms(element)

    # Count element for right hand side
    for i in range(len(products)):
        if right_coeffs[i] <= 0:
            raise ValueError("Invalid reaction. Coefficients must be positive.")
        rhs_count += right_coeffs[i] * products[i].num_atoms(element)

    return lhs_count == rhs_count

# Return right and left coefficients solved from a list of reactants and products
def __solve_matrix__(self, reactants, products, elems):
    # Create a coefficient matrix to solve
    coeff_matrix = []
    for e in elems.to_seq():
        row = [compnd.num_atoms(e) for compnd in reactants]
        row += [-compnd.num_atoms(e) for compnd in products]
        coeff_matrix.append(row)

    # Check if reaction is null
    if(reactants == [] and products == []):
        return [], []
    else:
        # Solve for lhs and rhs coefficients
        # Uses algorithm proposed here:
        #
            https://stackoverflow.com/questions/42637872/solve-system-of-linear-integer-equations-in-python
        matrix = Matrix(coeff_matrix)
        null_vectors = matrix.nullspace()

        if null_vectors == []:
            raise ValueError("Invalid ReactionT. Reaction cannot be balanced.")

        null_vectors = null_vectors[0]
        multiple = lcm([val.q for val in null_vectors])
        x = multiple * null_vectors
        solution = np.array([int(val) for val in x]).tolist()

        lhs_coeffs = solution[:len(reactants)]
        rhs_coeffs = solution[len(reactants):]

    return lhs_coeffs, rhs_coeffs
```

N Code for test_All.py

```
## @file test_All.py
#  @author Jay Mody
#  @brief Test driver for Set, MoleculeT, CompoundT, and ReactionT.
#  @date 08/02/20 (dd/mm/yy)

from Set import Set
from MoleculeT import MoleculeT
from CompoundT import CompoundT
from ReactionT import ReactionT
from ChemTypes import ElementT
from MolecSet import MolecSet
from ElmSet import ElmSet

import pytest
import numpy as np

# Set tests
def test_Set_init():
    assert Set([0, 1, 2])
    assert Set((0, "0", 1.2))

def test_Set_add():
    s = Set([1, 2, 3])

    s.add(12)
    assert s == Set([1, 2, 3, 12])

    s.add(3)
    assert s == Set([1, 2, 3, 12])

    s = Set([])

    s.add(1)
    assert s == Set([1])

    s.add("a")
    s.add("b")
    s.add("c")
    assert s == Set(["a", "b", "c", 1])

def test_Set_rm():
    with pytest.raises(ValueError):
        Set([]).rm(None)
    with pytest.raises(ValueError):
        Set([1]).rm(2)

    s = Set([1, 2, 3])
    s.rm(2)
    assert s == Set([1, 3])

    s = Set([1, 2, 3])
    s.rm(2)
    s.rm(1)
    s.rm(3)
    assert s == Set([])

def test_Set_member():
    assert not Set([]).member(True)
    assert not Set([1, 12, 2031]).member(-1)
    assert not Set([0.00001]).member(0)

    assert Set([1, 1, 1]).member(1)
    assert Set([1, 2, 3]).member(3)
    assert Set(["abc", 0]).member("abc")

def test_Set_size():
    assert Set([]).size() == 0
    assert Set([99]).size() == 1
    assert Set([1, 1, 1]).size() == 1
    assert Set(list(range(121))).size() == 121

def test_Set_equals():
    assert not Set([1, 2, 2012]).equals(Set([1, 2012]))
    assert not Set(["apple"]).equals(Set([]))

    assert Set([1, 2, 2012]).equals(Set([1, 2, 2012]))
```

```
    assert Set([1, 1, 1]).equals(Set([1]))


def test_Set_to_seq():
    assert isinstance(Set([1,2,3]).to_seq(), list)

    s = Set(list(range(2,20,3)))
    assert len(s.to_seq()) == s.size()


# MoleculeT tests
def test_MoleculeT_init():
    with pytest.raises(ValueError):
        MoleculeT(1, "He")
    with pytest.raises(ValueError):
        MoleculeT(0, ElementT.H)
    with pytest.raises(ValueError):
        MoleculeT(-1, ElementT.H)
    with pytest.raises(ValueError):
        MoleculeT(2.2, ElementT.He)

    assert MoleculeT(2, ElementT.H)
    assert MoleculeT(19999, ElementT.C)
    assert MoleculeT(1, ElementT.Fe)

def test_MoleculeT_get_num():
    assert MoleculeT(2, ElementT.O).get_num() == 2
    assert MoleculeT(1, ElementT.O).get_num() != 0

def test_MoleculeT_get_elm():
    assert MoleculeT(20, ElementT.O).get_elm() == ElementT.O
    assert MoleculeT(20, ElementT.H).get_elm() != 1

def test_MoleculeT_num_atoms():
    assert MoleculeT(20, ElementT.O).num_atoms(ElementT.O) == 20
    assert MoleculeT(10, ElementT.H).num_atoms(ElementT.O) == 0

def test_MoleculeT_constit_elems():
    assert MoleculeT(2, ElementT.H).constit_elems() == MoleculeT(1, ElementT.H).constit_elems()
    assert MoleculeT(3, ElementT.Mg).constit_elems() == ElmSet([ElementT.Mg])

def test_MoleculeT_equals():
    assert MoleculeT(2, ElementT.H).equals(MoleculeT(2, ElementT.H))
    assert MoleculeT(19999, ElementT.C) == MoleculeT(19999, ElementT.C)


# CompoundT tests
def test_CompoundT_init():
    m1 = MoleculeT(2, ElementT.H)
    m2 = MoleculeT(1, ElementT.O)

    with pytest.raises(ValueError):
        CompoundT([m1, m2])
    with pytest.raises(ValueError):
        CompoundT(set([m1, m2]))

    s = MolecSet([m1, m2])
    assert CompoundT(s)
    assert CompoundT(MolecSet([]))

def test_CompoundT_get_molec_set():
    m1 = MoleculeT(2, ElementT.H)
    m2 = MoleculeT(1, ElementT.O)
    s1 = MolecSet([m1, m2])

    m3 = MoleculeT(2, ElementT.H)
    m4 = MoleculeT(1, ElementT.O)
    s2 = MolecSet([m3, m4])

    c = CompoundT(s1)
    assert c.get_molec_set() == s2

    s1.add(MoleculeT(1, ElementT.He))
    print(s1.size())
    print(c.get_molec_set().size())
    assert c.get_molec_set() != s1

def test_CompoundT_num_atoms():
    m1 = MoleculeT(22, ElementT.H)
    m2 = MoleculeT(1, ElementT.O)
```

```
    m3 = MoleculeT(1, ElementT.O)
    c = CompoundT(MolecSet([m1, m2, m3]))
    assert c.num_atoms(ElementT.H) == 22
    assert c.num_atoms(ElementT.O) == 1
    assert c.num_atoms(ElementT.He) == 0

    m1 = MoleculeT(1, ElementT.Na)
    m2 = MoleculeT(1, ElementT.Cl)
    c = CompoundT(MolecSet([m1, m2]))
    assert c.num_atoms(ElementT.Na) == 1
    assert c.num_atoms(ElementT.Cl) == 1
    assert c.num_atoms(ElementT.H) == 0

def test_CompoundT_constit_elems():
    m1 = MoleculeT(2, ElementT.H)
    m2 = MoleculeT(1, ElementT.O)
    c = CompoundT(MolecSet([m1, m2]))
    assert c.constit_elems() == ElmSet([ElementT.H, ElementT.O])

    m1 = MoleculeT(1, ElementT.Na)
    m2 = MoleculeT(1, ElementT.Cl)
    c = CompoundT(MolecSet([m1, m2]))
    assert c.constit_elems() != ElmSet([ElementT.Na, ElementT.C])

def test_CompoundT_equals():
    m1 = MoleculeT(2, ElementT.H)
    m2 = MoleculeT(1, ElementT.O)
    s1 = MolecSet([m2, m1])

    m3 = MoleculeT(2, ElementT.H)
    m4 = MoleculeT(1, ElementT.O)
    s2 = MolecSet([m3, m4])

    assert CompoundT(s1) == CompoundT(s2)

    s2.add(MoleculeT(42, ElementT.U))
    assert CompoundT(s1) != CompoundT(s2)


# ReactionT tests
## @cite https://www.nayuki.io/page/chemical-equation-balancer-javascript
def test_ReactionT_init():
    # HClFe -> NO, is an invalid chemical equation
    m1 = MoleculeT(1, ElementT.H)
    m2 = MoleculeT(1, ElementT.Cl)
    m3 = MoleculeT(1, ElementT.Fe)
    m4 = MoleculeT(1, ElementT.N)
    m5 = MoleculeT(1, ElementT.O)
    lhs = [CompoundT(MolecSet([m1, m2, m3]))]
    rhs = [CompoundT(MolecSet([m4, m5]))]

    with pytest.raises(Exception):
        ReactionT(lhs, rhs)

    # H2 + O2 -> H2O, is a valid chemical equation that can be balanced
    m1 = MoleculeT(2, ElementT.H)
    m2 = MoleculeT(2, ElementT.O)
    m3 = MoleculeT(1, ElementT.O)
    lhs = [
        CompoundT(MolecSet([m1])),
        CompoundT(MolecSet([m2])),
    ]
    rhs = [
        CompoundT(MolecSet([m1, m3]))
    ]

    assert ReactionT(lhs, rhs)

def test_ReactionT_get_lhs():
    # H2 + O2 -> H2O
    m1 = MoleculeT(2, ElementT.H)
    m2 = MoleculeT(2, ElementT.O)
    m3 = MoleculeT(1, ElementT.O)
    lhs = [
        CompoundT(MolecSet([m1])),
        CompoundT(MolecSet([m2])),
    ]
    rhs = [
        CompoundT(MolecSet([m1, m3]))
    ]
```

```
    assert ReactionT(lhs, rhs).get_lhs() == lhs

def test_ReactionT_get_rhs():
    # H2 + O2 -> H2O
    m1 = MoleculeT(2, ElementT.H)
    m2 = MoleculeT(2, ElementT.O)
    m3 = MoleculeT(1, ElementT.O)
    lhs = [
        CompoundT(MolecSet([m1])),
        CompoundT(MolecSet([m2])),
    ]
    rhs = [
        CompoundT(MolecSet([m1, m3]))
    ]

    assert ReactionT(lhs, rhs).get_rhs() == rhs

def test_ReactionT_get_lhs_coeff():
    # H2 + O2 -> H20, coeffs [2, 1, 2]
    h2 = MoleculeT(2, ElementT.H)
    o2 = MoleculeT(2, ElementT.O)
    o1 = MoleculeT(1, ElementT.O)
    lhs = [
        CompoundT(MolecSet([h2])),
        CompoundT(MolecSet([o2])),
    ]
    rhs = [
        CompoundT(MolecSet([h2, o1]))
    ]
    assert ReactionT(lhs, rhs).get_lhs_coeff() == [2, 1]

    # Mg(OH)2 -> MgO + H20, coeffs [1, 1, 1]
    mg1 = MoleculeT(1, ElementT.Mg)
    lhs = [
        CompoundT(MolecSet([mg1, h2, o2])),
    ]
    rhs = [
        CompoundT(MolecSet([mg1, o1])),
        CompoundT(MolecSet([h2, o1]))
    ]
    assert ReactionT(lhs, rhs).get_lhs_coeff() == [1]

    # Mg(OH)2 -> MgO + H20, coeffs [2, 3, 4, 3]
    fe2 = MoleculeT(2, ElementT.Fe)
    fe1 = MoleculeT(1, ElementT.Fe)
    o3 = MoleculeT(3, ElementT.O)
    c1 = MoleculeT(1, ElementT.C)
    lhs = [
        CompoundT(MolecSet([fe2, o3])),
        CompoundT(MolecSet([c1])),
    ]
    rhs = [
        CompoundT(MolecSet([fe1])),
        CompoundT(MolecSet([c1, o2]))
    ]
    assert ReactionT(lhs, rhs).get_lhs_coeff() == [2, 3]

def test_ReactionT_get_rhs_coeff():
    # H2 + O2 -> H20, coeffs [2, 1, 2]
    h2 = MoleculeT(2, ElementT.H)
    o2 = MoleculeT(2, ElementT.O)
    o1 = MoleculeT(1, ElementT.O)
    lhs = [
        CompoundT(MolecSet([h2])),
        CompoundT(MolecSet([o2])),
    ]
    rhs = [
        CompoundT(MolecSet([h2, o1]))
    ]
    assert ReactionT(lhs, rhs).get_rhs_coeff() == [2]

    # Mg(OH)2 -> MgO + H20, coeffs [1, 1, 1]
    mg1 = MoleculeT(1, ElementT.Mg)
    lhs = [
        CompoundT(MolecSet([mg1, h2, o2])),
    ]
    rhs = [
        CompoundT(MolecSet([mg1, o1])),
        CompoundT(MolecSet([h2, o1]))
```

```
]
assert ReactionT(lhs, rhs).get_rhs_coeff() == [1, 1]

# Mg(OH)2 -> MgO + H20, coeffs [2, 3, 4, 3]
fe2 = MoleculeT(2, ElementT.Fe)
fe1 = MoleculeT(1, ElementT.Fe)
o3 = MoleculeT(3, ElementT.O)
c1 = MoleculeT(1, ElementT.C)
lhs = [
    CompoundT(MolecSet([fe2, o3])),
    CompoundT(MolecSet([c1])),
]
rhs = [
    CompoundT(MolecSet([fe1])),
    CompoundT(MolecSet([c1, o2]))
]
assert ReactionT(lhs, rhs).get_rhs_coeff() == [4, 3]
```

O Code for Partner's Set.py

```
## @file Set.py
#  @author Benjamin Kostiuk
#  @brief Module that defines the Set ADT
#  @details Assumes that the Set constructor is called for each object instance
#           before any other access methods are called.
#  @date 02/01/2020

from Equality import Equality


## @brief An abstract data type that represents a set
class Set(Equality):

    ## @brief Set constructor
    #  @details Initializes a Set object whose state consists of a set of elements
    #  @param s Sequence of elements with which to initialize the Set
    def __init__(self, s):
        self.S = set(s)

    ## @brief Add an element to the set
    #  @param Element to be added to the set
    def add(self, e):
        self.S = self.S.union({e})

    ## @brief Remove an element from the set
    #  @param e Element to be removed from the set
    #  @throws ValueError if element to be removed cannot be found in the set
    def rm(self, e):
        if not self.member(e):
            raise ValueError("Cannot remove element not found in set.")
        self.S = self.S.difference({e})

    ## @brief Determine whether an element is in the set
    #  @param e Element to check whether in the set
    #  @return True if the element is in the set, otherwise false
    def member(self, e):
        return e in self.S

    ## @brief Get the size of the set
    #  @return The size of the set
    def size(self):
        return len(self.S)

    ## @brief Determine if the Set is equal to another set
    #  @details A Set is considered equal if all elements in one set are in another
    #  @param r Set to compare with
    #  @return True if the two Sets are equal, otherwise false
    def equals(self, r):
        if self.size() != r.size():
            return False

        for element in self.S:
            if not r.member(element):
                return False
        return True

    ## @brief Returns a sequence of all elements in the set
    #  @return A sequence of all elements in the set
    def to_seq(self):
        return list(self.S)

    def __eq__(self, value):
        return self.equals(value)
```

P Code for Partner's MoleculeT.py

```
## @file MoleculeT.py
#  @author Benjamin Kostiuk
#  @brief Module defines the MoleculeT ADT for molecule representation
#  @date 02/01/2020

from Equality import Equality
from ChemEntity import *


## @brief An abstract data type that represents a molecule
class MoleculeT(ChemEntity, Equality):

    ## @brief MoleculeT constructor
    #  @details Initializes a MoleculeT object whose state consists of
    #           an ElementT and the number of that element in the molecule
    #  @param m Number of the ElementT in molecule
    #  @param e ElementT in the molecule
    def __init__(self, n, e):
        self.num = n
        self.elm = e

    ## @brief Get the number of ElementT in the molecule
    #  @return The number of ElementT in the molecule
    def get_num(self):
        return self.num

    ## @brief Get the ElementT in the molecule
    #  @return The ElementT in the molecule
    def get_elm(self):
        return self.elm

    ## @brief Get the number of atoms of a given ElementT in the molecule
    #  @param e ElementT to check for in molecule
    #  @return The number of atoms of the specified ElementT in the molecule
    def num_atoms(self, e):
        if self.elm == e:
            return self.num
        return 0

    ## @brief Return an ElmSet of the ElementT in the molecule
    #  @return An ElmSet of the ElementT in the molecule
    def constit_elems(self):
        return ElmSet([self.elm])

    ## @brief Determine if the molecule is equal to another molecule
    #  @details Two molecules are considered equal if they are composed of the same
    #           ElementT and have the same number of atoms.
    #  @param m MoleculeT to compare with
    #  @return True if the molecules are equal, otherwise false
    def equals(self, m):
        return self.elm == m.get_elm() and self.num == m.get_num()

    def __eq__(self, value):
        return self.equals(value)

    def __hash__(self):
        return hash(str(self.num) + str(self.elm))
```

Q Code for Partner's CompoundT.py

```
## @file CompoundT.py
#  @author Benjamin Kostiuk
#  @brief Module defines the CompoundT ADT for chemical compound representation
#  @date 02/01/2020

from MoleculeT import *
from MolecSet import *


## @brief An abstract data type that represents a chemical compound
class CompoundT(ChemEntity, Equality):

    ## @brief CompoundT constructor
    #  @details Initializes a CompoundT object whose state consists of a MolecSet
    #  @param m MolecSet of molecules in the chemical compound
    def __init__(self, m):
        self.C = m

    ## @brief Get the MolecSet of molecules in the chemical compound
    #  @return The MolecSet of molecules in the chemical compound
    def get_molec_set(self):
        return self.C

    ## @brief Get the number of atoms of a given ElementT in the chemical compound
    #  @param e ElementT to check for in chemical compound
    #  @return The number of atoms of the specified ElementT in the chemical compound
    def num_atoms(self, e):
        count = 0
        for m in self.C.to_seq():
            count += m.num_atoms(e)
        return count

    ## @brief Return an ElmSet of the ElementTs in the chemical compound
    #  @return An ElmSet of the ElementTs in the chemical compound
    def constit_elems(self):
        return ElmSet([m.get_elm() for m in self.C.to_seq()])

    ## @brief Determine if the chemical compound is equal to another chemical compound
    #  @details Two chemical compounds are considered equal if they have
    #           all the same molecules in them
    #  @param d CompoundT to compare with
    #  @return True if the chemical compounds are equal, otherwise false
    def equals(self, d):
        return self.C.equals(d.get_molec_set())

    def __eq__(self, value):
        return self.equals(value)
```

R Code for Partner's ReactionT.py

```
## @file ReactionT.py
#  @author Benjamin Kostiuk
#  @brief Module defines the ReactionT ADT for representing checmical reactions
#  @date 02/05/2020

from CompoundT import *

import numpy as np
from sympy import Matrix, lcm


## @brief An abstract data type that represents a chemical reaction
class ReactionT:

    ## @brief ReactionT constructor
    #  @details Initializes a ReactionT object whose state consists of a sequence of
    #           reactants a sequence of its coefficients, a sequence of products
    #           and a sequence of its coefficients. The sequences of coefficients
    #           are computed as to balance the chemical reaction with an equal
    #           number of elements on both sides.
    #  @param reactants Sequence of CompoundT in the left-hand side of the chemical
    #                   reaction, known as reactants
    #  @param products Sequence of CompoundT in the right-hand side of the chemical
    #                  reaction, known as products
    #  @throws ValueError if the elements in the reactants do not match the elements
    #                     in the products, the two sides of the reaction cannot be
    #                     balanced, any of coefficients are non-positive or if the
    #                     the sequences of coefficients do not match their
    #                     respective side of the chemical reaction
    def __init__(self, reactants, products):
        # Get ElmSet of ElementTs in L and R
        lhs_elems = self.__elements_in_equation__(reactants)
        rhs_elems = self.__elements_in_equation__(products)

        # Check that elements in the reactants and the products are the same
        if not lhs_elems.equals(rhs_elems):
            raise ValueError("Elements in reactants must match elements in products.")

        # Get coefficient matrix to solve linear equation
        lhs_coeffs, rhs_coeffs = self.__solve_matrix__(reactants, products, lhs_elems)

        # Check if length of lists match
        if len(lhs_coeffs) != len(reactants) or len(rhs_coeffs) != len(products):
            raise ValueError("Cannot match coefficients to reactants and products.")

        # Check if coefficients are balanced
        for element in lhs_elems.to_seq():
            if not self.__is_balanced__(reactants, products, lhs_coeffs, rhs_coeffs, element):
                raise ValueError("Invalid ReactionT. Reaction cannot be balanced.")

        self.lhs = reactants
        self.rhs = products
        self.coeff_L = lhs_coeffs
        self.coeff_R = rhs_coeffs

    ## @brief Get the sequence of reactants of the chemical reaction
    #  @return The sequence of CompoundT in the left-hand side of the chemical reaction
    def get_lhs(self):
        return self.lhs

    ## @brief Get the sequence of products of the chemical reaction
    #  @return The sequence of CompoundT in the right-hand side of the chemical reaction
    def get_rhs(self):
        return self.rhs

    ## @brief Get the sequence of coefficients in the left-hand side of the chemical reaction
    #  @return The sequence of coefficients in the left-hand side of the chemical reaction
    def get_lhs_coeff(self):
        return self.coeff_L

    ## @brief Get the sequence of coefficients in the right-hand side of the chemical reaction
    #  @return The sequence of coefficients in the right-hand side of the chemical reaction
    def get_rhs_coeff(self):
        return self.coeff_R

    # Returns an ElmSet of ElementT in a list of CompoundTs
```

```
def __elements_in_equation__(self, equation):
    elems = []
    for compound in equation:
        elems += compound.constit_elems().to_seq()
    return ElmSet(elems)

# Check if a ReactionTs coefficients for reactants and products are balanced
def __is_balanced__(self, reactants, products, left_coeffs, right_coeffs, element):
    lhs_count, rhs_count = 0, 0
    # Count element for left hand side
    for i in range(len(reactants)):
        if left_coeffs[i] <= 0:
            raise ValueError("Invalid ReactionT. Coefficients must be positive.")
        lhs_count += left_coeffs[i] * reactants[i].num_atoms(element)

    # Count element for right hand side
    for i in range(len(products)):
        if right_coeffs[i] <= 0:
            raise ValueError("Invalid reaction. Coefficients must be positive.")
        rhs_count += right_coeffs[i] * products[i].num_atoms(element)

    return lhs_count == rhs_count

# Return right and left coefficients solved from a list of reactants and products
def __solve_matrix__(self, reactants, products, elems):
    # Create a coefficient matrix to solve
    coeff_matrix = []
    for e in elems.to_seq():
        row = [compnd.num_atoms(e) for compnd in reactants]
        row += [-compnd.num_atoms(e) for compnd in products]
        coeff_matrix.append(row)

    # Check if reaction is null
    if(reactants == [] and products == []):
        return [], []
    else:
        # Solve for lhs and rhs coefficients
        # Uses algorithm proposed here:
        #
            https://stackoverflow.com/questions/42637872/solve-system-of-linear-integer-equations-in-python
        matrix = Matrix(coeff_matrix)
        null_vectors = matrix.nullspace()

        if null_vectors == []:
            raise ValueError("Invalid ReactionT. Reaction cannot be balanced.")

        null_vectors = null_vectors[0]
        multiple = lcm([val.q for val in null_vectors])
        x = multiple * null_vectors
        solution = np.array([int(val) for val in x]).tolist()

        lhs_coeffs = solution[:len(reactants)]
        rhs_coeffs = solution[len(reactants):]

    return lhs_coeffs, rhs_coeffs
```