Multiphysics Modeling of Activity Transport and Evolution of CRUD and Steam Generator Oxides in Pressurized Water Reactors

by

Alicia M. Elliott

B.E., Stony Brook University (2015)

Submitted to the Department of Nuclear Science and Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Nuclear Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2018

Author .....
Department of Nuclear Science and Engineering
August 23, 2018

Certified by .....
Michael P. Short
Associate Professor of Nuclear Science and Engineering
Thesis Supervisor

Certified by .....
Kord Smith
KEPCO Professor of the Practice of Nuclear Science and Engineering
Thesis Reader

Accepted by .....
Ju Li
Battelle Energy Alliance Professor of Nuclear Science and Engineering and
Professor of Materials Science and Engineering
Chairman, Department Committee on Graduate Students

Multiphysics Modeling of Activity Transport and Evolution of CRUD and Steam Generator Oxides in Pressurized Water Reactors

by

Alicia M. Elliott

Submitted to the Department of Nuclear Science and Engineering
on August 23, 2018, in partial fulfillment of the
requirements for the degree of
Master of Science in Nuclear Science and Engineering

Abstract

Fouling deposits of corrosion products on fuel cladding, known as crud, in the core of light water reactors can cause a variety of operational issues. Buildup of radioactive crud and corrosion products on ex-core structures, such as steam generators and piping, can cause increased radiation fields and higher dose exposures for plant workers. To better understand the mechanisms of corrosion product activity transport and evolution in the primary coolant loop, a crud source term and activity transport code that can predict the concentration of active isotopes in a primary loop over time and plant operating parameters was developed, implementing mechanistic models for soluble corrosion product dissolution and precipitation. The code described in this thesis tracks activated isotope deposition throughout the primary loop with spatial and temporal resolution, without the use of empirical rate constants derived from plant measurements, to predict primary loop activity buildup. Developed in `C++` using the MOOSE Framework, this code can be easily coupled to other multiphysics codes through the MOOSE `MultiApp` system. A set of input file generation scripts, written in Python, were developed to calculate thermodynamic parameters for chemical reactions added to the simulation, and easily set up simulation input files in a "user-friendly" format. The open source code described in this work, Ouroboros, is available freely for future improvements and adaptations to implement additional mechanisms and more rigorous models. This code is the first step towards a long term effort to develop an open source, fully mechanistic crud source term model including all mechanisms for activity transport in pressurized water reactors.

Thesis Supervisor: Michael P. Short
Title: Associate Professor of Nuclear Science and Engineering

Acknowledgments

First and foremost, I would like to thank Mitsubishi Heavy Industries (MHI) of Japan for funding the work presented in this thesis. I also wish to thank Professor Kord Smith, my thesis reader, for his feedback and time spent reviewing my thesis.

I'd like to thank my advisor, Professor Mike Short, for guiding me throughout this thesis work. Mike, your advice and input throughout my time at MIT has been of immeasurable value to me. You provided a supportive environment for me to develop my computational skills and explore a research niche that I love. I truly cannot thank you enough; your impact on my development as a scientist will never be forgotten.

I want to thank Marina Dang for her mentoring and guidance throughout my time at MIT. Marina, you have been so much more than just my manager in the NSE Comm Lab. I consider you my "MIT Mom" for all the nurturing you've provided me. I wouldn't be completing this degree without your guidance and kindness over these years. Thank you.

I also want to acknowledge the rest of the Comm Lab team, especially Leigh Ann Kesler, Patrick White, and Cody Dennett. Thanks for the impromptu coaching sessions and all of the laughs. I can always count on you guys to make me smile.

A huge thanks to Briana Hiscox and Travis Labossiere-Hickman; you two have been there for me through so many late nights in NW12, struggling with psets, and dealing with life in general. I love you guys. I don't know what I'd do without you.

Thanks also to AJ Pawel, Monica Gehrig, and Landon Crawford, who helped keep me sane during the final months of writing this thesis. From debating Buzzfeed quiz results, to impromptu trail runs, to rock climbing adventures, your company and support (and many laughs) have been paramount towards the completion of this thesis.

Finally, I want to thank my wonderful family - Mom, Dad, Jonas, and Emma. You have been my biggest supporters from day one, I can always count on you to be there for me. Whether it's spontaneous visits home to escape the hustle and bustle of Cambridge, or making up weird inside jokes (that Mom and Dad don't quite understand), I'm always sure to gain some perspective and feel how much you care for me. Thank you. I love you, always.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

304SS	Type 304 Stainless Steel
316SS	Type 316 Stainless Steel
A600	Inconel 600/Alloy 600
A690	Inconel 690/Alloy 690
AOA	Axial Offset Anomaly
BC	Boundary Condition
BWR	Boiling Water Reactor
CILC	CRUD-Induced Localized Corrosion
CIPS	CRUD-Induced Power Shift
CRUD	Chalk River Unidentified Deposits
ECP	Electrochemical Corrosion Potential
FEM	Finite Element Method
IC	Initial Condition
INL	Idaho National Laboratory
LWR	Light Water Reactor

MOOSE	Multiphysics Object Oriented Simulation Environment	
NPP	Nuclear Power Plant	
ODE	Ordinary Differential Equation	
PDE	Partial Differential Equation	
PETSc	Portable, Extensible Toolkit for Scientific Computation	
PWR	Pressurized Water Reactor	
SNB	Subcooled Nucleate Boiling	

Physics Constants

N_A	Avogadro Constant	$6.022140857 \times 10^{23}\, mol^{-1}$
R	Gas Constant	$8.3144598\, J/mol - K$

Other Symbols

ΔG_f^0	Standard Gibbs free energy of formation	
ρ	Density	
D_i	Diffusivity of species i	m^2/s
G	Mass flux	$kg/m^2 - s$
V	Constant Volume	

Chemical Notation

$[Fe^{2+}]$	Molar concentration of Fe^{2+}	mol/m^3
^{54}Fe, Fe-54	Iron isotope with 54 nucleons	
Fe^{2+}	Iron ion with 2+ charge	
M^{i+}, M^{i-}	Metal ion with i+ or i- charge	

Chapter 1

Introduction

1.1 Motivation (The CRUD problem)

Global energy production is central to much of the modern world, and increasing energy demands necessitate the use of a diverse mix of energy sources to provide reliable, affordable electricity. However, not all energy sources are equal, especially as concerns over environmental carbon emissions and global climate change increasingly dictate the mix of energy sources utilized. The second largest source of low-carbon electricity generation is nuclear fission, which accounts for approximately 11 percent of the worldwide energy portfolio [1].

The International Atomic Energy Agency (IAEA) reported that as of 2017, 448 nuclear power plants (NPPs) were operational worldwide. Of these, pressurized water reactors (PWRs) make up approximately 65 percent, with 289 plants total. As of December 2016, an additional 51 new reactors under construction (of 61 total) are PWRs. The second most abundant NPPs are boiling water reactors (BWRs) (17.4%, 78 plants), followed by pressurized heavy-water reactors (PHWRs) (10.9%, 49 plants). With 20 percent of annual electricity generation in the United States coming from 99 operating nuclear plants - including 65 PWRs - finding solutions to common operational problems is crucial to maintaining this component of the US clean energy portfolio [2].

1.1.1 Pressurized water reactors

Pressurized water reactors are a type of "light water" cooled and moderated nuclear reactors (commonly referred to as light water reactors, or LWRs). "Light" water refers to the use of "normal" water (H_2O) in contrast to "heavy" water (D_2O or ^2H_2O) - water in which the hydrogen isotope contains a neutron, making it "heavy" hydrogen (also called deuterium). In a BWR, the water is allowed to boil in the core to release steam, which goes through a turbine to convert the steam energy into electricity. The water coolant in PWRs is pressurized to approximately 15-16 MPa to prevent boiling in the reactor core.

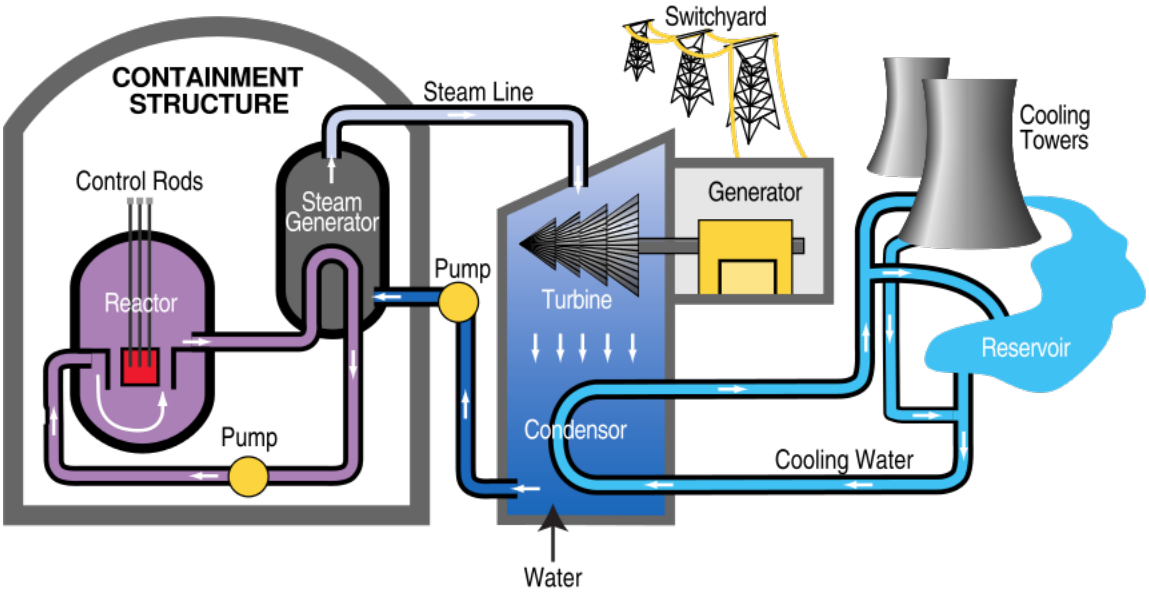

Figure 1-1: Schematic of PWR systems (primary and secondary loops) [3]

The main components of a PWR make up the primary and secondary coolant loops, which work together to take heat produced by fission in the reactor core and convert it to electricity with turbine systems; Figure 1-1 presents a schematic of these components. The primary loop is entirely inside of the containment structure, and consists of the reactor core, steam generator, and piping between these two components (referred to as the hot leg and cold leg, for the segment bringing hot coolant water from the core to the steam generator and the segment bringing cooler water from the steam generator back to the core, respectively). The secondary loop interfaces with the primary loop in the steam generator, where heat from the primary loop coolant is transferred to the secondary loop coolant. The separation of

these two loops allows for containment of radioactive materials and fission products within the primary loop and containment structure, decreasing the risk for radioactive material contamination or release in the turbine and generator systems.

1.1.2 Crud and steam generator oxides in PWRs

In a pressurized water reactor (PWR), corrosion of materials in the primary loop, including steam generator components, hot leg piping, core components, and cold leg piping, is constantly occurring due to the exposure of these materials to the coolant. Primary loop materials are dissolved into the coolant, forming aqueous metal ions and releasing particulate oxides into the coolant. Particulate and ionic release is dependent on many environmental factors, including temperature, electrochemical potential, pH, local fluid properties, and solubilities of different chemical species in the coolant [4, 5]. Crud (Chalk River Unidentified Deposits), the name of fouling deposits made of these corrosion products, can deposit on the fuel rod clad surfaces. The predominant phases of crud found in PWRs are non-stoichiometric nickel ferrite spinel ($Ni_xFe_{3-x}O_4$) and nickel oxide or nickel metal [4]. Interactions between the crud and coolant can dissolve or dislodge particulates and ions into the coolant, which then circulate the primary loop.

Crud deposits in the core of nuclear reactors cause a variety of operational issues, particularly in PWRs. These issues include CRUD-induced power shift (CIPS)/axial offset anomaly (AOA) and CRUD-induced localized corrosion (CILC), which have been the subject of many industrial and academic efforts to understand, predict, and reduce the impacts of these problems [6]. CIPS/AOA is a phenomenon when the axial power distribution shifts towards the bottom of the core; this occurs when crud deposits preferentially on the upper portions of PWR fuel rods due to subcooled nucleate boiling (SNB). Boron then deposits in pores of the crud. Boron has a large thermal neutron capture cross section (3600 barns), allowing it to absorb neutrons easily [7]. This causes local flux depressions in areas with more crud (thus, more boron), and the overall downward shift of the power distribution [8]. This shift in the power distribution reduces the control rod worth during the initial stages of control rod insertion. The downward shift also causes "fresher" fuel near the top of the core, which increases reactor power unexpectedly during restart because deposited boron dissolves

Reaction	$t_{1/2}$ **(days)**	$\sigma_{capture}$ (b)	μ **(at. %)**
^{54}Fe(n,γ)^{55}Fe	1001.56	2.25	5.845
^{58}Fe(n,γ)^{59}Fe	44.495	1.314	0.2819
^{50}Cr(n,γ)^{51}Cr	27.7025	15.37	4.3452
^{58}Ni(n,p)^{58}Co	71.3	0.105	68.0769
^{58}Co(n,γ)^{59}Co	stable	1855	n/a
^{59}Co(n,γ)^{60}Co	1925.28	37.5	100
^{94}Zr(n,γ)^{95}Zr	64.032	0.04987	17.380

Table 1.1: Half-life, thermal neutron capture cross sections (fast neutron capture for ^{58}Ni), and natural abundance (μ) of activated species [9–16]

out of the crud during reactor shutdown. Crud deposition onto fuel cladding also leads to increased temperatures at the clad-crud interface, which causes sharp temperature gradients in the clad. These gradients induce faster local corrosion rates in areas with thicker crud growth, and this process is called CILC [8].

However, there is another operational problem caused by crud that has not been examined with the same mechanistic treatment and level of detail in modeling in the open literature: buildup of radioactive corrosion products outside of the core. Nickel based alloys and austenitic stainless steels (such as 316 or 304 grade) are commonly used materials for PWR steam generators, and austenitic stainless steel is commonly used for hot and cold leg piping [5]. Zirconium-based alloys (such as Zircaloy-4) are typically used in the fuel cladding and other in-core structures [5]. These alloys contain iron, nickel, chromium, zirconium, and cobalt - all of which can undergo neutron capture to form activated isotopes with half-lives ranging from 1 month to over 5 years (see Table 1.1). As corrosion products formed from these materials are transported through the core, they are subject to high neutron flux and can undergo neutron capture to form radioactive isotopes ("activated species"). These aqueous corrosion products and crud particulates, both activated and nonactivated, can be transported out of the core and deposit on the walls of the cold leg, hot leg, and steam generator components. Corrosion products then deposit on primary loop structures; oxides formed on these structures become radioactive as activated species are deposited.

Deposition of activated species outside of the core yields higher radiation dose rates for workers who maintain the primary loop systems, as the accumulated radioactivity outside of the core remains long after the plant is shut down [5, 6]. Typical radiation protection

guidelines emphasize the principle of ALARA ("as low as is reasonably achievable"), which "means making every reasonable effort to maintain exposures to radiation as far below the dose limits in this part as is practical" [17]. The annual occupational dose limit, specified in NRC Regulations, are 5 rem per person [17]. In 1979, degradation of steam generator tubing from corrosion product buildup at Surry Unit 2 necessitated their replacement [18]. According to a report from the US Environmental Protection Agency, 150 people received occupational exposure above the 5 rem limit in 1976 due to PWRs; 72 of these occurred at Surry [19]. Similarly, in 1977, there were 93 exposures above the dose limit due to PWRs, with 64 of those at Surry [19]. Maintenance of the steam generators in the years prior to this replacement, as well as during the replacement operations, led to significantly higher exposures for workers at Surry.

1.1.3 Modeling activity accumulation in primary loop oxides

Models to predict the behavior that creates these problems are of great interest to the nuclear energy community, and have traditionally been understood through careful experimental evaluations and development of correlations or semi-empirical relations to predict crud buildup and behavior in reactors. However, mechanistic models are lacking - and semi-empirical models do not hold up well for reactors with substantially different geometries or operating conditions than the reactor for which the model was optimized. To obtain a better understanding of activity buildup on the primary side for modern PWRs, a model for activity transport that is based upon the *mechanisms* of processes is needed [5]. By simulating this process using the mechanisms, instead of empirical relations from measurements, new plants can be modeled with greater accuracy. Such models can aid in the design process as well, by choosing construction materials based on minimization of the corrosion products that lead to high accumulated activity in ex-core primary loop structures. Physics-informed designs and renovations to plants can help to minimize the exposure of primary-side workers to high doses of radiation, which is increasingly important as regulations on accumulated worker doses become stricter.

Current activity transport codes are based on empirical or semi-empirical models and correlations that are highly plant-specific, making it difficult to adapt these codes to new

reactor styles, geometries, or coolant chemistry conditions. As such, these codes are no longer adequate for accurately predicting behavior in modern PWRs with transient operating conditions. Additionally, many of these codes are not open-source, which makes it difficult to study and modify the models to simulate new plants or add new models for mechanisms of physical processes being simulated.

1.2 Thesis objectives

The objective of this thesis is the development of a code that can predict the concentration of active isotopes in a primary loop as function of time and plant operating parameters - with a mechanistic treatment instead of empirical - for versatility to apply to a broad range of PWR geometries. This code was developed in `C++` using the Multiphysics Object Oriented Simulation Environment (MOOSE). MOOSE is an open source finite element framework developed by the Idaho National Laboratory, designed to solve tightly coupled sets of partial differential equations on arbitrary geometries using implicit numerical methods and finite element methods. MOOSE is not specifically adapted for any particular field or application, and the MOOSE MultiApp Transfer System facilitates coupling of different MOOSE Application codes to obtain high-fidelity multiphysics simulations of large-scale systems [20, 21]. An activity transport model developed by Macdonald et al. [5] was adapted for implementation using modular code design in a MOOSE application; this facilitates future coupling to high-fidelity thermal hydraulic, fuel performance, neutron transport, and coolant chemistry codes for transient multiphysics feedback. This model, as well as the ability to track the isotopic inventory, was added to an existing crud source term code, Ouroboros, a `C++` code that uses the MOOSE Framework.

The model implemented in this code uses plant-specific parameters such as temperature, chemical characteristics, hydrodynamic properties, and electrochemical properties, minimizing the use of empirical-based models and parameters, to determine activity transport in the PWR primary loop. This treatment allows for detailed modeling of a broader style of plants, whose geometries and coolant chemistry do not need to have been previously studied in terms of activity transport, in order to provide better estimates of isotope uptake and

deposition throughout the primary loop. Mechanisms for precipitation or dissolution rates are represented - not using empirical rate constants, which vary plant to plant dependent upon the many parameters mentioned previously.

Operating parameters for PWRs are carefully considered, as it is a delicate balancing act to optimize plant performance while also achieving conditions to minimize crud, activity buildup, and other corrosion mechanisms (and stress corrosion cracking). For example, the pH must be controlled within a narrow window; too low pH can cause more crud release via accelerated primary side corrosion, while too high pH can increase Zircaloy corrosion and impact stress corrosion cracking by destabilizing the passive oxide film normally responsible for protecting the underlying alloy [4]. By studying and modeling these processes to obtain a deeper understanding of the mechanisms that drive them, plant parameters, materials, and designs can be chosen to reduce the activity buildup in PWRs.

Development of an activity transport code implementing multiple physical processes for mass transport is not a trivial task; developing a fully mechanistic modeling code, with high spatial and temporal resolution on scales that vary by orders of magnitude, is even less so. As such, it is vital to note that the end goal for developing a fully mechanistic model is a long-term goal, and will require many iterations upon the code with improved assumptions and equations as complexity is added and modification of physics for non-ideal conditions are considered. To complete such a model, it is necessary to begin with many assumptions that simplify the complexities greatly and may remove some of the nuances of the real processes that occur; for example, many codes reviewed in this thesis assume a uniform neutron flux distribution in the core to obtain an estimate on the correct order of magnitude, with activation by thermal neutrons alone. Neutronics models can obtain high levels of detail for the transient neutron flux over the fuel cycle, which could be considered in the model. However, the truly interesting physics and chemistry - the mechanisms not yet understood - are those of kinetics of dissolution and precipitation, of mass transport, and of isotopic exchange. By simplifying the neutronics (crudely, yet sufficiently for this type of model), the effects of modifying the mechanisms represented in the model can be easier studied and understood. Once a better understanding for the mechanisms is gained, and the model is no longer limited by that understanding, more detailed secondary physics can be

implemented to gain better spatial and temporal resolution that is closer to reality.

Chapter 2

Background

This chapter presents a summary of prior work that has been performed to understand activity transport and buildup in PWRs, as well as a brief introduction to the Multiphysics Object Oriented Simulation Environment (MOOSE) framework.

2.1 Mechanisms for activity transport

Figure 2-1 details the mechanisms responsible for activity transport in the primary loop. The mechanisms of mass transport differ based on the state of the corrosion product (i.e. soluble ions vs. nonsoluble particulates). Metallic ions are released into the coolant from metal corrosion reactions, oxide dissolution reactions, and dissolution of particulates within the coolant. Ions are removed by precipitation of oxides onto primary loop structures, incorporation of ions into existing or growing oxides, and precipitation to form particulates within the coolant. Particulates are released into the coolant through erosion of oxides or formation by precipitation of ions in the coolant; removal of particulates occurs by dissolution into ions or deposition onto primary loop structures.

Currently, the mechanisms believed to be responsible for ion mass transport are coolant convection (transport throughout loop) and chemical solubility (dissolution/precipitation). Turbulent flow in the coolant is responsible for transporting particles throughout the loop, and erosion from coolant turbulence is governed by a force balance between hydrodynamic forces and adherent forces between the particle and the wall. Multiple possible mechanisms

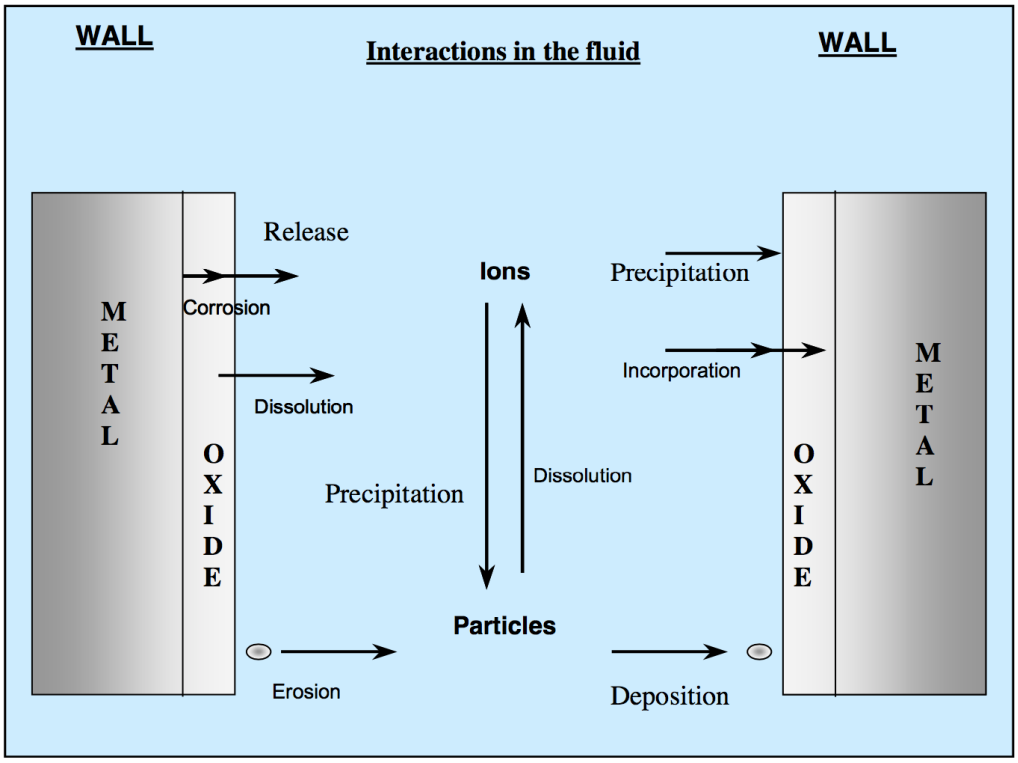

Figure 2-1: Mechanisms for activity transport at metal-oxide and oxide-coolant interface of primary loop structures [22].

have been described for particulate deposition; these include turbulent diffusion (momentum transfer), Brownian diffusion (for small particles), inertial impaction (large particles), sedimentation in horizontal piping (due to gravity), and thermophoresis. These vary depending on parameters and assumptions or simplifications considered in each model.

2.2 Previous models and codes

Activity accumulation in PWRs is certainly not a new issue to the nuclear power industry; multiple models have previously been established to quantify the buildup of radioactivity in the primary loop systems [5, 22]. These codes have been developed over many decades, and approaches and assumptions used in each differ vastly depending on previous models and on the desired end use for which each particular code was developed. Many of the models were developed or calibrated for a particular PWR plant design or operating conditions. The models/codes are briefly described in the following subsections.

2.2.1 Castelli model (corrosion source term)

Castelli published a crud source term model in *Nuclear Corrosion Modeling: The Nature of CRUD* in 2009 [23]. The model represents a one dimensional PWR primary coolant loop, with four types of mass balance equations for each element modeled to account for the corrosion oxide sublayers and surface layers, aqueous solubles, and aqueous particulates. Four nuclide activation reactions are included ($^{59}\mathrm{Co}(n,\gamma)^{60}\mathrm{Co}$, $^{64}\mathrm{Zn}(n,\gamma)^{65}\mathrm{Zn}$, $^{58}\mathrm{Ni}(n,p)^{58}\mathrm{Co}$, $^{58}\mathrm{Fe}(n,\gamma)^{59}\mathrm{Fe}$). The mass balance equations are loosely coupled nonlinear partial differential equations, which are solved through linearization and iterative finite differencing methods.

The corrosion source term includes corrosion growth (conversion of base metals/alloys to oxides) and corrosion release (release of metal ions into the reactor coolant) [23]. Elements oxidize proportional to the weight percents in the reactor materials; the oxides formed are typically spinels (AB_2O_4, where A is a divalent metal ion and B is a trivalent metal ion), regardless of the base metal being oxidized. The spinel oxides most commonly found in PWRs are magnetite (Fe_3O_4), nickel ferrite ($NiFe_2O_4$), ferrous chromite ($FeCr_2O_4$), and nickel chromite ($NiCr_2O_4$) [23]. The oxide-based crud source term on 300- and 600-series alloys forms with two distinct layers of oxides: a chromium-rich sublayer at the alloy interface, and an iron-rich surface layer of "large tetrahedral crystals," which are easily dislodged into the coolant (see Fig. 2-2) [23].

The expression for rate of change of corrosion growth, using "thick film" growth kinetics, is as follows:

$$\frac{\partial w}{\partial t} = \frac{k_p}{2\sqrt{t}}$$

where w is the mass density of corrosion products on the surface (in milligrams of corrosion products per square decimeter of wetted area) in the primary loop, k_p is the corrosion growth rate for the particular alloy, and t is the exposure time [23].

This model includes 4 mass balance equations per nuclide; two solid phases (one for each oxide layer - $^{\mathrm{el}}\mathrm{W}_{\mathrm{sub}}$ and $^{\mathrm{el}}\mathrm{W}_{\mathrm{sur}}$), and two aqueous phases (solubles, $^{\mathrm{el}}\mathrm{C}_{\mathrm{s}}$ and particulates, $^{\mathrm{el}}\mathrm{C}_p$). The physicochemical processes that couple these phases are represented by rate constants ($k_s, \bar{k_r}, k_{dp}, k_e, k_d$); descriptions of the processes can be found in Fig. 2-3.

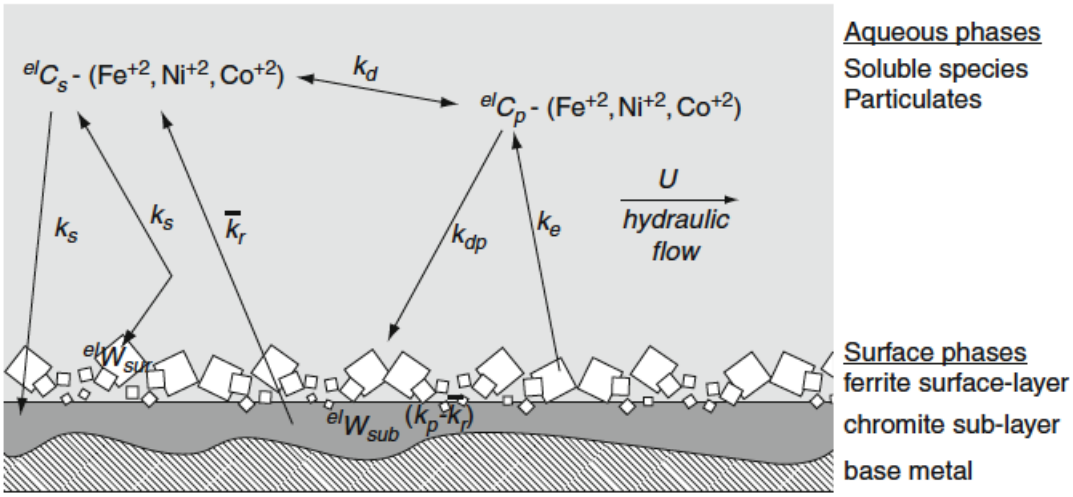

Figure 2-2: Variables and processes represented in Castelli corrosion source term model [23].

Process	Rate Constant	Sublayer	Surface Layer	Solubles	Particulates	Comment
Corrosion growth	$(k_p - \overline{k}_r)$	Source				
Corrosion release	$\overline{k}_r$			Source		
Hydrothermal crystallization dissolution	k_s	Source or loss	Source or loss	Source or loss		A bidirectional flux to solubles and surface films Mechanism for cobalt substitution
Particulate deposition	k_{dp}		Source		Loss	
Particulate erosion	k_e		Loss		Source	
Particulate crystallization dissolution	k_d			Source or loss	Source or loss	Bidirectional flux that couples solubles and particulates
Hydraulic flow	U			Source and loss	Source and loss	U is coolant velocity (cm/sec)

Figure 2-3: Processes and rate constants in Castelli corrosion source term model [23].

The Castelli model consists of the following general set of equations. Hydraulic diameter is represented as d_h, the coolant equilibrium saturated concentration is C_{sat}, natural abundance of an isotope is μ, and radioactive decay parameter is λ.

Corrosion growth/release

Subsurface source:

$$\frac{\partial W}{\partial t} = \frac{k_p - \bar{k_r}}{\sqrt{t}} \tag{2.1}$$

Soluble source:

$$\frac{\partial C_s}{\partial t} = \frac{4\bar{k_r}}{d_h\sqrt{t}} \tag{2.2}$$

Particulate deposition/erosion

Surface layer source:

$$\frac{\partial W}{\partial t} = k_{dp}C_p - k_e W \tag{2.3}$$

Coolant source:

$$\frac{\partial C_p}{\partial t} = \frac{4}{d_h}\left(k_e W - k_{dp}C_p\right) \tag{2.4}$$

Hydrothermal crystallization/dissolution

Surface layer source:

$$\frac{\partial W}{\partial t} = k_s(C_s - C_{sat}) \tag{2.5}$$

Soluble source:

$$\frac{\partial C_s}{\partial t} = -\frac{4}{d_h}(C_s - C_{sat}) \tag{2.6}$$

Radioactive source/decay

$$\frac{\partial C_{active}}{\partial t} = \mu C_{nonactive}\sigma_{capture}\phi - \lambda C_{active} \tag{2.7}$$

This set of equations is written for each element/isotope of interest in the system. Rate coefficients/constants are used from measurements reported in published literature, according to Castelli. The coolant equilibrium saturated concentration, C_{sat}, is dependent on temperature, pH, solid surface phases facing the coolant, and concentration of dissolved hydrogen in the coolant [23]. This is calculated from equilibrium thermodynamic equations to relate the solid phase to the dissolved phase in the coolant using the dissolution reaction

equilibrium constant. Though it is not obvious from a simple inspection of this equation set, "the ability to predict C_{sat} is key to the success of the entire fundamental modeling system" according to Castelli.

2.2.2 CPAIR-P

CPAIR-P is a time-dependent corrosion product activity code, originated in Pakistan by Mirza et al [24–26]. The model for the activated species mass balance involves five processes, as can be seen in Equations 2.8. Corrosion products are activated from neutron capture in the core, and removed by radioactive decay, water purification processes, deposition on primary loop structures, and coolant leakage. Local concentration gradients in the primary loop are not considered in this model, and precipitation/dissolution is modeled as being proportional to the concentration fraction of a given species in the coolant. Empirical removal rates are used in this model, based on published data on PWRs. Dissolution and deposition rates from experimental data are used. This code implements an implicit, fourth-order Runge-Kutta scheme to solve the coupled set of ODEs. The mechanisms for rates of the physical processes considered in the mass balance are not modeled, and are empirically determined [5, 26].

$$\frac{dn_w}{dt} = \sigma\Phi_E N_w - \left[\sum_j \frac{\epsilon_j Q_j}{V_w} + \sum_k \frac{l_k}{V_w} + \lambda\right] n_w + \frac{K_p}{V_w} n_p + \frac{K_c}{V_w} n_c \tag{2.8a}$$

$$\sum_j \epsilon_j Q_j = \epsilon_I Q_I + \epsilon_p Q_p + \epsilon_c Q_c + \epsilon_F Q_F \tag{2.8b}$$

$$S_w(t) = \frac{C(t) S N_A}{V_w A} f_n f_s \tag{2.8c}$$

Empirical removal rates are used in this model, based on published data on PWRs. Dissolved boron concentration is a function of time based on measured plant operational data. Group constant cross sections are calculated by another code, based on plant-specific core design parameters including power density, core geometry, number of assemblies, linear heat rate, coolant pressure, and inlet/outlet coolant temperatures. Precursor species concentrations are considered to have a source term from the corrosion rates producing each species (see Eq. 2.8c).

n_w	activated concentration in coolant
N_w	nonactive concentration in coolant
σ	group constant cross section
Φ_e	effective 1-group neutron flux
n_p	activated concentration in piping
n_c	activated concentration in core
$\epsilon_I Q_I$	removal rate in coolant from ion exchanger
$\epsilon_p Q_p$	deposition rate in piping
$\epsilon_c Q_c$	deposition rate in core
$\epsilon_F Q_F$	removal rate in coolant from filters
l_k	leakage rate from kth leak
K_p	removal rate from piping deposits
K_c	removal rate from core deposits
S_w	Source term for precursor concentration in coolant
$C(t)$	corrosion rate as function of time
S	wetted area of primary loop
A	atomic weight of precursor
f_n	natural abundance of isotope
f_s	fraction of element present in construction material

Table 2.1: Variables in CPAIR-P

2.2.3 ACE-II

ACE-II is an empirical activity transport code that was developed for modeling Japanese PWRs [5, 22, 27]. The model considers formation of an inner and outer oxide layer from corrosion in the primary loop, and both particulates and metal ions are released into the coolant from erosion and dissolution of the outer oxide layer, respectively. These species are activated from neutron flux in the core (either in the aqueous state or by activation and release of CRUD from core internal structures), and both activated and non-activated species precipitate throughout the primary loop. This code also includes isotopic exchange, where the activated species that precipitate diffuse through the outer oxide, inner oxide, and into the base construction materials of the primary loop. Mass transport is considered by using empirically determined solubilities for corrosion products. Rates of various kinetic processes are represented by experimentally measured rate constants, which are specific to a particular plant for which the code is optimized. The code implements iron, nickel, and cobalt elemental mass balance calculations and tracks activated ^{58}Co and ^{60}Co [22]. ACE-II includes models to account for the effect of pH on erosion and deposition rates, which

can be significant for particulates due to surface charge effects [22]. However, the empirical nature of ACE-II could lead to difficulty when modeling reactors with different operating parameters, geometries, or coolant chemistry [5, 6]. Corrosion rates, soluble species mass transfer coefficients, and aqueous solubilities of nickel and cobalt used were estimated from measured plant data [22].

2.2.4 MIGA-RT

MIGA-RT is a code developed by Dinov for predicting activity buildup in PWRs and their Russian counterpart, water-water energetic reactors (VVERs) [5,22,28,29]. The code models steady-state conditions, as temporal variation is not included, and transients are considered by changing the water chemistry conditions in the simulation at each statepoint desired throughout the fuel cycle. Soluble and particulates are considered, though particulates are considered more thoroughly; Dinov implemented an analytical model for determining mass transfer coefficients of particulates based on sticking probabilities and surface conditions within the primary loop [5,27]. This code includes activation of ionic and particulate corrosion products for two nuclides: $^{58}\mathrm{Ni}(n,p)^{58}\mathrm{Co}$, and $^{59}\mathrm{Co}(n,\gamma)^{60}\mathrm{Co}$. Solubilities of corrosion products in nonstoichiometric nickel ferrite are calculated using thermodynamic stability models. Magnetite (Fe_3O_4), nickel oxide, and nickel metal phases are also included in thermodynamic solid-phase stability calculation [22]. MIGA-RT was used to determine the activity buildup in the primary loop of Cruas-1, and was validated with data from EBO-1 (Bohunice NPP), Loviisa 1 and 2, and Beznau NPP [22].

2.2.5 PACTOLE

Developed in France by Commissariat a l'Energie Atomique (CEA) with collaboration from Electricite de France (EDF) and Framatome, the PACTOLE code for activity transport implements many analytic solutions to equations for calculating concentrations [5, 22, 30]. The main assumption in this code is that corrosion products are released from oxides directly into the coolant. The corrosion oxides, as in the ACE-II code, are treated as an inner and outer oxide layer, and the rate of dissolution of ions is proportional to the inner ox-

ide thickness, while the rate of particulate erosion is proportional to the outer oxide layer thickness. The mechanism for precipitation of dissolved species is modeled by considering concentration gradients and saturated concentrations of species in the coolant. Mechanisms for precipitation of particulates are also implemented, including gravitational settling, turbulent diffusion, and thermophoresis [5]. Main advantages of this code include the use of predominantly analytic or mechanisms-based models with many physicochemical processes represented, including treatment of both soluble and particulate corrosion products.

2.2.6 Macdonald Model

Macdonald et al. developed an activity transport model with the intent of avoiding the use of empirical data, which would prevent it from being used for a variety of plants and conditions [5]. This model focuses on the impact of physicochemical, hydrodynamic, and electrochemical properties of the system, in addition to thermal hydraulic parameters. The primary loop is discretized into 15 nodes, and these properties are calculated at each node along the primary loop based on local conditions [5]. A detailed model for calculating the local electrochemical corrosion potential (ECP) is implemented; this potential varies with temperature, pH, thermal hydraulic parameters, and local concentrations of electroactive species from water radiolysis reactions (see Table 2.2) that includes lithium and boron, which are commonly used to control the primary loop pH. In this code, local pH is calculated based on a system of 8 chemical reactions that includes lithium and boron, which are commonly used to control the primary loop pH. The local ECP is determined using a "mixed potential model" that takes into consideration the electrochemical potentials of each electroactive species calculated, and accumulates the total ECP by summing each species' potential weighted by the local species concentration [5].

The dissolution and precipitation of corrosion products in the coolant is modeled by considering the saturation concentration of corrosion products in the coolant near the coolant-clad/oxide interface along the primary loop.

	Chemical Reaction
1	$B(OH)_3 + OH^- = B(OH)_4{}^-$
2	$2\,B(OH)_3 + OH^- = B_2(OH)_7{}^-$
3	$3\,B(OH)_3 + OH^- = B_3(OH)_{10}{}^-$
4	$4\,B(OH)_3 + 2\,OH^- = B_4(OH)_{14}{}^{2-}$
5	$5\,B(OH)_3 + 3\,OH^- = B_5(OH)_{18}{}^{3-}$
6	$Li^+ + OH^- = LiOH$
7	$Li^+ + B(OH)_4{}^- = LiB(OH)_4$
8	$H_2O = H^+ + OH^-$

Table 2.2: Reactions considered by Macdonald et al. for calculating local pH at each primary loop node

Model Equations

Processes represented in each equation of this model can be found in Table 2.4, and nomenclature can be found in Table 2.3.

The dissolution and precipitation of corrosion products in the coolant is modeled by considering the saturation concentration of corrosion products in the coolant near the coolant-clad/oxide interface along the primary loop. Concentrations of an aqueous species (B) released by an electrochemical reaction between an oxide or metal (A) (Eq. 2.9a) are determined by the Nernst equation (Eq. 2.9b) under the assumption of local equilibrium conditions.

$$aA + xH^+ + ze^- \rightarrow bB + cH_2O \tag{2.9a}$$

$$E = E^0 - \frac{2.303RT}{zF}\log Q_{rxn} \tag{2.9b}$$

$$C_{s,i,j} = 10^{\left[\frac{z}{b}\frac{F}{2.303RT}\left(\frac{-\Delta G^0_{f,i}}{zF} - \mathrm{ECP}\right) - \frac{x}{b}\mathrm{pH}\right]} \tag{2.9c}$$

The reaction quotient Q_{rxn} for the reaction in Eq. 2.9a can be written as:

$$Q_{rxn} = \frac{a_B^b a_{H_2O}^c}{a_A^a a_{H^+}^x}$$

The activity of water (a_{H_2O}) and the activity of the solid oxide (a_A) can be set equal to 1 by convention. This model lets E = local ECP, $E^0 = \frac{-\Delta G^0_{f,i}}{zF}$, and expands the logarithm term to

Symbol	Description
$C_{s,i,j}$	Surface/saturation concentration of ion i at j
$C_{b,i}$	Bulk coolant concentration of ion i
$C^{n}_{b,iso,p}$	Bulk coolant concentration of isotope
$C_{b,element}$	Bulk coolant concentration of element
$\widetilde{C}_{precip,iso,j}$	Concentration of activated, precipitated isotope at section j
$\widetilde{C}_{b,iso,p}$	Bulk coolant concentration of activated isotope
$R_{i,j}$	Release rate of ion i at section j
$R^{n,-}_{i,j}$	Negative values of release rate (precipitation)
$R^{n,+}_{i,j}$	Positive values of release rate (dissolution)
$D_{i,j}$	Diffusivity of ion i at section j
Sh_j	Sherwood number at j
$\Delta G^{0}_{f,i}$	Change in Gibbs energy of formation of i
ϕ_j	Neutron flux of section j
A_j	Wetted area of section j
L_j	Length of section j
$\sigma_{capture}$	Neutron capture cross-section of isotope
μ	Natural abundance of isotope
λ_p	Decay constant for isotope from modeled reaction p

Table 2.3: Variables in Macdonald et al. model

substitute in $pH = -\log a_{\mathrm{H}^+}$. The activity of ion B is replaced with its molar concentration by assuming a dilute solution. Some additional algebraic rearrangement yields an expression for the saturation or surface concentration of species B that is used in this model, see Eq. 2.9c.

Concentrations of the aqueous species due to chemical reaction is determined using rate theory of equilibrium reactions. For a reaction $a\mathrm{A} + x\mathrm{H}^+ \rightarrow b\mathrm{B} + c\mathrm{H_2O}$, the reaction rate constant K is calculated by:

$$K = \frac{a_B^b a_{\mathrm{H_2O}}^c}{a_A^a a_{\mathrm{H}^+}^x} = \exp\left(\frac{-\Delta G^0_{f,i}}{RT}\right)$$

Applying the same conventions on the activity of water and the solid oxide as used for electrochemical reactions and using the definition of pH to substitute $a^x_{\mathrm{H}^+} = 10^{-xpH}$, this expression can be rearranged to Eq. 2.10:

$$C_{s,i,j} = \left[10^{-xpH}\exp\left(\frac{-\Delta G^0_{f,i}}{RT}\right)\right]^{-b} \tag{2.10}$$

Eqn.	Processes represented
2.10	Surface concentration of ion i at location j - dissolution due to chemical reaction
2.9c	Surface concentration of ion i at location j - due to electrochemical reaction
2.11	Diffusivity of ion i at location j
2.12	Rate of dissolution/precipitation of ion i at location j
2.13	Total rate of change of ion i in bulk coolant
2.14	Dissolution, activation, and precipitation of nonactivated isotope
2.15	Activation, decay, precipitation of activated isotope
2.16	Precipitation build-up of activated isotope

Table 2.4: Equations in model developed by Macdonald et al.

The rate of release or deposition of ionic species into and out of the bulk primary coolant is driven by the concentration gradient between the local surface concentration and the bulk coolant. This rate is impacted by the species diffusivity in the coolant at the local temperature, which is also a function of the mass transfer coefficient, k_c. The mass transfer coefficient is important in describing mass transport in non-steady state conditions of fluids with turbulent flow and mixing [31]. The effects of turbulent mixing are significant in this system and cannot be neglected, so the mass transfer coefficient is implemented in calculating the species diffusivity to account for this effect. Diffusivity is calculated using a typical Arrhenius-type expression, taking into account the local temperature; see Eq. 2.11.

$$D_{i,j} = D_i^0 \exp\left[\frac{k_{c_{i,j}}}{R}\left(\frac{1}{T_j} - \frac{1}{298.15}\right)\right] \tag{2.11}$$

The species rate of release and deposition is determined by a mass transport equation, driven by the concentration gradient from the local surface concentration and the bulk coolant, that considers diffusive and convective transport properties; see Eq. 2.12. This expression incorporates effects from the hydrodynamic properties of the coolant through the Sherwood number, Sh. The Sherwood number is a dimensionless number that represents the ratio of convective to diffusive mass transfer rates.

$$R_{i,j} = D_{i,j}\frac{Sh_j A_j c}{L_j}\left(C_{s,i,j} - C_{b,i}\right) \tag{2.12}$$

If the surface concentration is greater than the bulk concentration, the species will be released and move away from the interface to the bulk coolant. Likewise, if the bulk concen-

tration is greater than the saturation/surface concentration, the species will precipitate out of the coolant at this interface. The release rate is also dependent on the wetted area (A_j) and length of the node (L_j), which varies by material and primary loop component.

The total bulk concentration of an ionic species is calculated by integrating the release/deposition rates of the species along each node of the primary loop (Eq. 2.13).

$$\frac{dC_{b,i}^{n}}{dt} = \sum_{j} R_{i,j} \tag{2.13}$$

The isotopic species balance equations contain terms accounting for dissolution into the coolant, loss to activation by neutron capture, and loss from precipitation out of the coolant (Eq. 2.14). Isotopes are dissolved in proportion to their natural abundance, μ, and precipitate in proportion to the ratio of the bulk concentration of the isotope to the bulk concentration of the parent element in the coolant.

$$\frac{dC_{b,iso,p}}{dt} = \mu_p \left(\frac{\sum_{elems} \sum_{j} R_{i,j}^{n,+}}{\sum_{j} Vol_j} \right) - \sum_{j} C_{b,iso,p}^{n} \phi_j \sigma_{capture} - \left(\frac{C_{b,iso,p}^{n}}{C_{b,elem}^{n}} \right) \left(\frac{\sum_{elems} \sum_{j} R_{i,j}^{n,-}}{\sum_{j} Vol_j} \right) \tag{2.14}$$

Similarly, activated isotopes are modeled with balance equations that account for activation via neutron capture, loss by radioactive decay, and loss by precipitation (Eq. 2.15) Isotopes are assumed to precipitate in proportion to the ratio of the bulk concentration of the isotope to the bulk concentration of the parent element.

$$\frac{d\widetilde{C}_{b,iso,p}}{dt} = \sum_{j} C_{b,iso,p}^{n} \phi_j \sigma_{capture} - \lambda_p \widetilde{C}_{b,iso,p} - \left(\frac{\widetilde{C}_{b,iso,p}^{n}}{C_{b,element}^{n}} \right) \left(\frac{\sum_{elements} \sum_{j} R_{i,j}^{n,-}}{\sum_{j} Vol_j} \right) \tag{2.15}$$

Using these balance equations, a full system mass balance is represented, and the activity buildup is quantified from the concentrations of precipitated activated isotopes (Eq. 2.16).

$$\widetilde{C}_{precip,iso,j}^{n+1} = (1 - \lambda_p \Delta t) \widetilde{C}_{precip,isotope,j}^{n} + \frac{\widetilde{C}_{b,iso,p} N_V \Delta t \lambda_p}{C_{b,element} A_j} \sum_{i \in elem} R_{i,j}^{n,-} \Delta t \tag{2.16}$$

2.2.7 Comparison of previous models

With the exception of PACTOLE and the model by Macdonald et al., these codes are all based on empirical or semi-empirical models and correlations that are highly plant-specific, making it difficult to adapt these codes to new reactor styles, geometries, or coolant chemistry conditions. Many of the models were developed or calibrated for a particular PWR plant design or operating conditions, which limits the versatility in applying the models to other PWRs. Development of codes with more mechanistic-based models to replace empirical models could improve the accuracy of modeling modern PWRs with transient operating conditions and new designs.

2.3 MOOSE Framework

The Multiphysics Object Oriented Simulation Environment (MOOSE) is an open source finite element framework developed by the Idaho National Laboratory (INL). The MOOSE framework uses fully-coupled and implicit multiphysics solvers (such as the Portable, Extensible Toolkit for Scientific Computation (PETSc) developed by Argonne National Laboratory), and harnesses the finite element library libMesh for automatic parallelization and mesh adaptivity, among other features [20, 21].

The MOOSE framework is physics-agnostic; that is, the framework is designed to solve tightly coupled sets of partial differential equations (PDEs) on arbitrary geometries using implicit numerical methods and finite element methods, and is not specifically adapted for any particular field or application. MOOSE-based applications have been developed for a variety of physics problems, including nuclear reactor fuel performance modeling (BISON), hydrothermal and geothermal systems (FALCON), and phase field modeling of microstructural evolution (MARMOT). Length scales of these physics vary from micrometers to hundreds of meters, and the timescales can vary from fractions of a second to decades [20, 21].

Developed in `C++`, the MOOSE framework uses an object-oriented approach to develop extensible sub-systems to represent physics, set up meshes, applying boundary conditions and material properties, and execute solves (Figure 2-4 presents a high-level overview of the MOOSE framework architecture). MOOSE development is focused on providing "plug-

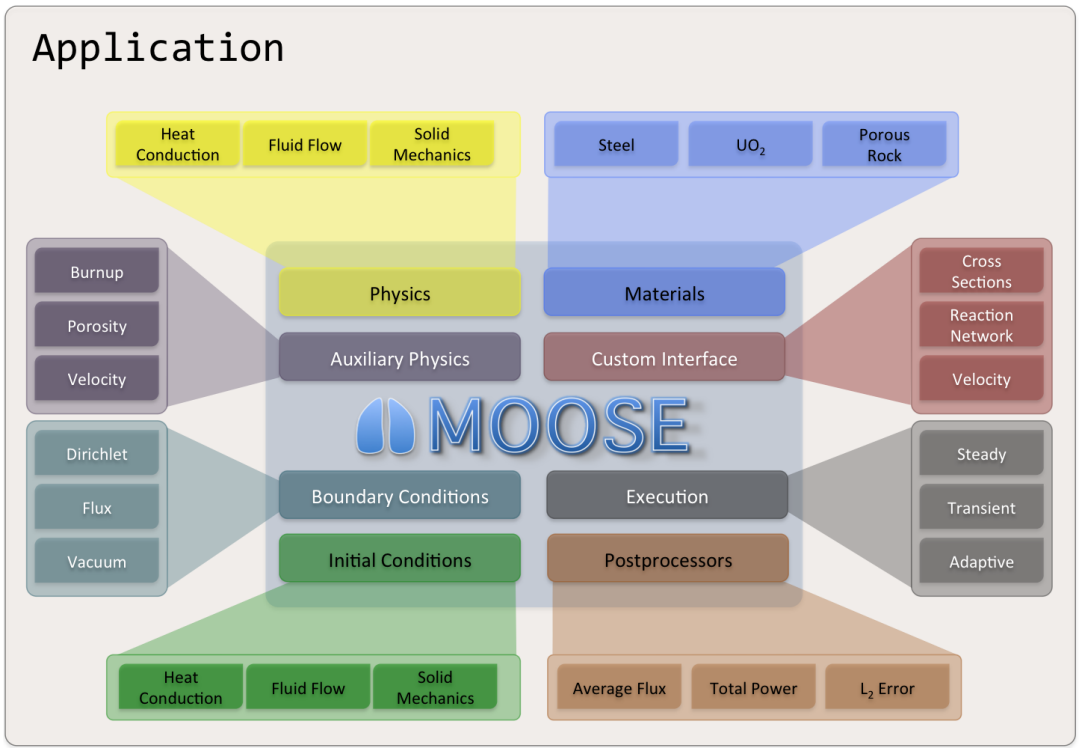

Figure 2-4: MOOSE Application architecture [32]

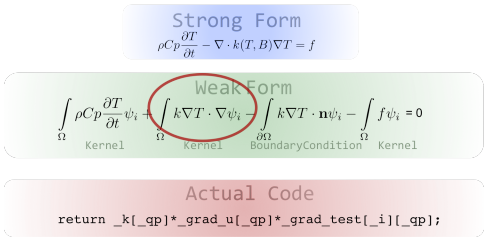

Figure 2-5: MOOSE code breakdown example [32]

and-play" capabilities for modeling different physics, materials, initial/boundary conditions, and solve types to allow scientists and engineers to develop models efficiently by removing the need for users to develop or implement nonlinear solvers, discretization schemes, or parallelization capabilities. Users can add new physics by creating a "Kernel," a MOOSE object that represents the "weak form" of a single term in a PDE (see Figure 2-5) [32]. Each term is added individually to the MOOSE Application input file, making it simple to modify the physics being solved.

Chapter 3

Chemistry/Physics of Model (Methods)

This chapter presents the chemistry and physics considered in this model, the set of equations and their assumptions/simplifications, and a comparison of this model with prior models. The MOOSE implementation of this model is discussed in detail in the next chapter.

3.1 Original Ouroboros code

At the outset of this work, a bare-bones crud source term modeling code had previously been developed using the MOOSE framework. This code, Ouroboros, implemented two ordinary differential equations (ODEs) that determine the concentration and release rate of soluble species from construction materials into the coolant and the release of soluble species from crud, and a calculation of oxide thickness, all of which evolve in time. These equations, while included in the modified version of Ouroboros, are not implemented in the model/simulations presented in this thesis. The equations have been included here for completeness and documentation of the code.

The release rate of soluble species from construction materials is calculated using Equation 3.1, where D_i is diffusivity of species i, C_i is concentration of species i, N_D is number density, `th` is oxide thickness, N_A is Avogadro's number, and `wp` is the wetted perimeter of the section.

$$R_i = \frac{D_i \cdot C_i \cdot \texttt{wp} \cdot N_D}{\texttt{th} \cdot N_A} \tag{3.1}$$

Oxide thickness is determined using Equation 3.2, where `th` is the oxide thickness, `offset` is the offset for starting oxide thickness, `prefact` is the prefactor for growth, and P is the thickness power. Oxide thickness is independent of concentrations, just the thickness power, which is constant in each region.

$$\texttt{th} = \texttt{offset} \cdot \texttt{prefact}(t^P) \tag{3.2}$$

The crud concentration ODE is represented by Equation 3.3, where R is deposition rate, which is from the growth of the outer oxide crystal layer.

$$\frac{dC}{dt} = R \cdot \texttt{wp} \tag{3.3}$$

The soluble species coolant concentration ODE is Equation 3.4, where R is soluble metal release rate from diffusion of soluble species through the oxide (Eqn. 3.1).

$$\frac{dC}{dt} = R \cdot \texttt{wp} = \frac{D_i \cdot C_i \cdot \texttt{wp} \cdot N_D}{\texttt{th} \cdot N_A} \cdot \texttt{length} \tag{3.4}$$

3.2 Modifications and additions to Ouroboros

3.2.1 Corrosion growth source term

The corrosion growth model was implemented from the Castelli model:

$$\frac{\partial w}{\partial t} = \frac{k_p}{2\sqrt{t}}$$

where w is the mass density of the surface (in milligrams of alloy per square decimeter of wetted area) in the primary loop, k_p is the corrosion growth rate for the particular alloy, and t is the exposure time [23]. For simplicity, it is assumed that the corrosion growth rate k_p is constant, regardless of alloy, for each element considered. This is easily modified by using a function to set the value of k_p in each region of the primary loop. For consistency in units, the calculation implemented in Ouroboros also uses the molar mass of the oxide created (MM_{oxide}) and the wetted area of the section to output the concentration of solid oxide phase in moles per unit time.

$$\frac{dm_{oxide}}{dt} = \frac{k_p w a}{2\sqrt{t}}$$

Note that the units of k_p are typically expressed in mass per unit area per $\sqrt{time}$. Elements oxidize proportional to their weight percent in the alloy; the rate is thus weighted by the weight percent of the alloy to get the elemental concentration [23].

$$\frac{dm_{elem}}{dt} = wt\% \times \frac{k_p w a}{2\sqrt{t}}$$

To get this to a volumetric concentration of moles, this must be converted from mass to moles. The concentration is also converted from a "surface" concentration ($mols/m^2$) to a "volumetric" concentration ($mols/m^3$) by multiplying by $\frac{d_h}{4}$.

$$R_{corr} = \frac{dn_{elem}}{dt} = \frac{k_p \cdot wa}{2\sqrt{t} \cdot MM_{oxide}} \frac{d_h}{4} \tag{3.5}$$

3.2.2 Corrosion release source term

Derivation of metal ion release rate calculation

Metal ions are assumed to be released by solid state diffusion from the base metal through the oxide layer of corrosion products into the coolant. The rate is a flux across the top oxide surface times the area of that surface. The rate is derived using Fick's first law for the flux, with a constant source of the element C_{metal} at $x = 0$ and assuming $C = 0$ at $x = thickness$.

$$J = -D\frac{dC}{dx} = -D\frac{C_{metal} - 0}{0 - thickness} = D\frac{C_{metal}}{thickness}$$

So the rate can be expressed as:

$$R = J \cdot wa = D\frac{C_{metal}}{thickness}wa$$

If the concentration is given in $mols/m^3$, then the units on this rate is $mols/s$. Solving the ODE over all nodes will get the total number of moles released from metal ion diffusion into the coolant. The concentration of element in the alloy can be calculated from the mole fraction of the element in the alloy ($x_{elem} = \frac{moles_i}{moles_{alloy}}$) and the density of the alloy.

$$C_{alloy} = \frac{\rho_{alloy}}{MM_{alloy}}$$

$$C_{elem} = C_{alloy} \times x_{elem}$$

The alloy composition is usually given by weight percents; this can be converted easily to mole fractions. Note that molar mass is $MM_i = \frac{mass_i}{moles_i}$

$$wt\% = \frac{mass_{elem}}{mass_{alloy,tot}} \longrightarrow \frac{mass_{elem}}{mass_{alloy,tot}} \times \frac{\frac{mass_{alloy}}{moles_{alloy}}}{\frac{mass_{elem}}{moles_{elem}}} = \frac{moles_{elem}}{moles_{alloy}} = x_{elem}$$

$$x_{elem} = wt\% \times \frac{MM_{alloy}}{MM_{elem}}$$

To express the element concentration in terms of known quantities:

$$C_{elem} = \frac{\rho_{alloy}}{MM_{alloy}} \times x_{elem} = \frac{\rho_{alloy}}{MM_{alloy}} \times wt\% \times \frac{MM_{alloy}}{MM_{elem}}$$

$$\therefore C_{elem} = \frac{\rho_{alloy}}{MM_{elem}} \times wt\%$$

And finally the full rate expression is:

$$R_{corr,MR} = D\frac{\rho_{alloy} \cdot wt\% \cdot wa}{thickness \cdot MM_{elem}} \tag{3.6}$$

This gives the moles of metal ions released from diffusion through the corrosion product oxide layer per unit time. An alternative, less simplified assumption would be to make the driving force for the diffusion flux (concentration gradient across the oxide) use the coolant ion concentration as the concentration at the oxide-coolant interface. However, for this preliminary model, the assumption of zero concentration at the surface of the oxide will suffice, because typical ion saturated concentrations range from parts per billion to fractions of one part per billion in PWRs [33].

It is important to note that the diffusivity in this expression is that of a metal ion diffusing through the solid oxide layer, and differs from the diffusivity used in other parts of this model. The solid state diffusivity is determined using an Arrhenius relation, $D_i = D_0 \exp(-E_A/RT)$.

3.2.3 Surface/saturated concentration

The dissolution of ions from oxides into the coolant is driven by the coolant saturated concentration; this concentration is determined from equilibrium relations of chemical and electrochemical dissolution reactions between the metal, oxides, and coolant. It is a function of temperature, pH, electrochemical potential (ECP), the Gibbs free energy of formation of the reaction, and the stoichiometry of each reaction. This gives the equilibrium concentration of the ions in the coolant near the oxide surface.

The saturated concentration calculation is implemented from the Macdonald model, using the Nernst equation for electrochemical equilibrium and expressions from equilibrium thermodynamics for chemical reactions between solid oxides and dissolved ions (dissolu-

tion/precipitation reactions). These equations are derived in the following sections.

Nernst Equation Derivation

The Nernst equation can be derived from expressions for Gibbs free energy under standard conditions; E is the potential difference of redox cell, F is the Faraday constant, and z is the number of electrons transferred in the electrochemical reaction considered.

$$\Delta G = -zFE \tag{3.7}$$

At standard conditions, this is expressed as:

$$\Delta G^0 = -zFE^0 \tag{3.8}$$

The Gibbs free energy of a reaction is determined by the following expression:

$$\Delta G = \Delta G^0 + RT \ln Q \tag{3.9}$$

where Q is the chemical reaction quotient. For reaction $a\mathrm{A} + x\mathrm{H}^+ + ze^- \rightarrow b\mathrm{B} + c\mathrm{H_2O}$, where A is a solid oxide species and B is the ionic species formed, Q is written as:

$$Q = \frac{(a_B)^b (a_{\mathrm{H_2O}})^c}{(a_A)^a (a_{\mathrm{H}^+})^x} \tag{3.10}$$

a_B is the activity of species B - the numerator is the products, and the denominator is reactants (note that electrons are not included in this expression).

Converting the natural log to a base 10 log is trivial; substituting the expressions for ΔG, ΔG^0 (eqns. 3.7, 3.8) into equation 3.9, with some algebraic rearrangement, gives the Nernst equation:

$$E = E^0 - \frac{2.303RT}{zF} \log_{10} Q \tag{3.11}$$

Derivation of electrochemical equilibrium concentration

Substituting the expressions for ΔG, ΔG^0, and Q from equations 3.7, 3.8, and 3.10 into equation 3.9 yields:

$$-zFE = -zFE^0 + RT\ln\left[\frac{(a_B)^b(a_{\mathrm{H_2O}})^c}{(a_A)^a(a_{\mathrm{H^+}})^x}\right] \tag{3.12}$$

For this reaction, the activity of water ($a_{\mathrm{H_2O}}$) can be set equal to 1 by convention. The activity of the solid oxide (a_A) is set equal to 1 for consistency with the assumptions presented by Macdonald et al. Future work could revisit these assumptions to implement a more rigorous approximation of the activities for the oxide and aqueous ions.

This, along with some algebraic rearrangement, allows the expression to be simplified to

$$E = E^0 - \frac{RT}{zF}\ln\left[\frac{(a_B)^b}{(a_{\mathrm{H^+}})^x}\right] \tag{3.13}$$

After converting the natural log to a base 10 log, this expression then becomes

$$E = E^0 - \frac{2.303RT}{zF}\log_{10}\left[\frac{(a_B)^b}{(a_{\mathrm{H^+}})^x}\right] \tag{3.14}$$

The log term can be expanded by the properties of logarithms:

$$\log_{10}\left[\frac{(a_B)^b}{(a_{\mathrm{H^+}})^x}\right] = b\log_{10}(a_B) - x\log_{10}(a_{\mathrm{H^+}}) \tag{3.15}$$

Substituting this expression into equation 3.14 then gives:

$$E = E^0 - \frac{2.303RT}{zF}\Big[b\log_{10}(a_B) - x\log_{10}(a_{\mathrm{H^+}})\Big] \tag{3.16}$$

For this model, the potential E is substituted with the local electrochemical potential, ECP. Also, from the definition of pH, one can substitute in $\mathrm{pH} = -\log_{10}(a_{\mathrm{H^+}})$ to get the following:

$$ECP = E^0 - \frac{2.303RT}{zF}\Big[b\log_{10}(a_B) + x\mathrm{pH})\Big] \tag{3.17}$$

Algebraic rearrangement to isolate the activity term of ion B yields:

$$\log^{10}(a_B) = \frac{z}{b}\frac{F}{2.303RT}\Big(E^0 - ECP\Big) - \frac{x}{b}\text{pH} \tag{3.18}$$

Assuming a dilute solution, the activity of ion B can be replaced with its molar concentration $[B]$, and by inverting the logarithm (exponentiate with base 10), the following expression is used to calculate the molar concentration of ionic species B:

$$[B] = 10^{\left[\frac{z}{b}\frac{F}{2.303RT}\Big(E^0 - ECP\Big) - \frac{x}{b}\text{pH}\right]} \tag{3.19}$$

If the reaction is reversed

The only modification to reverse the reaction is the reaction quotient, Q - the numerator and denominator are swapped - so $Q = \frac{(a_{\text{H}^+})^x}{(a_B)^b}$. When the logarithm of this term is expanded, it becomes:

$$\log_{10}\left[\frac{(a_{\text{H}^+})^x}{(a_B)^b}\right] = x\log_{10}(a_{\text{H}^+}) - b\log_{10}(a_B) \tag{3.20}$$

Substituting in the definition of pH ($\text{pH} = -\log_{10} a_{\text{H}^+}$) this becomes

$$\log_{10}\left[\frac{(a_{\text{H}^+})^x}{(a_B)^b}\right] \quad = \quad -x\text{pH} - b\log_{10}(a_B) \quad = \quad -1 * \big[x\text{pH} + b\log_{10}(a_B)\big] \tag{3.21}$$

Putting this expression into the Nernst equation gives:

$$E = E^0 + \frac{2.303RT}{zF}\Big[b\log_{10}(a_B) + x\text{pH}\Big] \tag{3.22}$$

Rearranging this to get an expression for the concentration of species B gives:

$$[B] = 10^{\left[\frac{z}{b}\frac{F}{2.303RT}\Big(ECP - E^0\Big) - \frac{x}{b}\text{pH}\right]} \tag{3.23}$$

Note that equation 3.23 is almost identical to equation 3.19 - the only difference is the electrochemical potential difference. To make these mathematically equivalent, the sign on

the stoichiometric coefficient, z, must be flipped. This should also modify the calculation of $E^0 = \frac{-\Delta G_f^0}{zF}$; however, this is easily mitigated by taking the absolute value of z in this expression.

$$\frac{z}{b}\frac{F}{2.303RT}\Big(\frac{-\Delta G_f^0}{|z|F} - ECP\Big) \quad \rightarrow \quad \frac{-z}{b}\frac{F}{2.303RT}\Big(\frac{-\Delta G_f^0}{|z|F} - ECP\Big)$$

Distributing the negative sign from the outside gives

$$\frac{z}{b}\frac{F}{2.303RT}\Big(\frac{-(-\Delta G_f^0)}{|z|F} + ECP\Big) \rightarrow \frac{z}{b}\frac{F}{2.303RT}\Big(ECP - \frac{-\Delta G_f^0}{|z|F}\Big) \equiv \frac{z}{b}\frac{F}{2.303RT}\Big(ECP - E^0\Big)$$

So to reverse the reaction (such that product B is now a reactant), the sign of z for that reaction must be negated, and the final expression is written as:

$$[B] = 10^{\left[\frac{-z}{b}\frac{F}{2.303RT}\left(E^0 - ECP\right) - \frac{x}{b}\mathrm{pH}\right]} \tag{3.24}$$

Surface concentration calculation implemented

The calculation for the surface concentration of ion i due to an electrochemical reaction is implemented using Equation 3.25.

$$a\mathrm{A} + x\mathrm{H}^+ + ze^- \rightarrow b\mathrm{B} + c\mathrm{H_2O}$$

$$C_{s,i} = 10^{\left[\frac{z}{b}\frac{F}{2.303R\cdot T}\left(\frac{-\Delta G_{f,i}^0}{zF} - \mathrm{ECP}\right) - \frac{x}{b}\mathrm{pH}\right]} \tag{3.25}$$

The calculation for the surface concentration of ion i due to a chemical reaction is implemented using Equation 3.26.

$$aA + xH^+ \rightarrow bB + cH_2O$$

$$C_{s,i} = \left[10^{x \cdot pH} \exp\left(\frac{-\Delta G^0_{f,i}}{R \cdot T}\right)\right]^{-b} \tag{3.26}$$

3.2.4 Rate of dissolution/precipitation of soluble species

The dissolution/precipitation (ion release) rate from the Macdonald model was implemented in Ouroboros, as the mechanisms behind the dissolution/precipitation reaction are included. It considers the concentration gradient from the coolant-oxide interface (saturated concentration) to the bulk coolant, the mass transfer coefficient k_c, the ion diffusivity (as a function of temperature and k_c), and the Sherwood number (based on coolant properties like density, viscosity, Reynolds number, and also the species diffusivity).

This rate is implemented using Equation 3.27. A positive value for R_i indicates that the bulk concentration in the coolant is less than the saturated concentration, thus driving dissolution of oxides (source term to coolant concentration, loss term to oxide concentration). A negative value indicates that the bulk concentration has exceeded the saturated concentration, which drives precipitation of oxides (loss term to the coolant concentration, source term to oxide concentration).

$$R_i = D_i \cdot Sh \cdot \texttt{wa} \left(C_{s,i} - C_{b,i}\right) \tag{3.27}$$

The diffusivity of species is calculated using an expression from the Macdonald model, Equation 3.28.

$$D_i = D^0_i \exp\left[\frac{k_{c_i}}{R}\left(\frac{1}{T} - \frac{1}{298.15}\right)\right] \tag{3.28}$$

The release rate of isotopes into bulk coolant is calculated by adding the total release rate (positive values) for all ions of that element, and weighting that value by the natural

abundance of that isotope. For example, the natural abundance of ^{54}Fe is 5.85%, so the release rate of ^{54}Fe added to the coolant is the total release rate of all Fe ions in coolant multiplied by 0.0585.

The precipitation rate of isotopes is weighted by the ratio of concentration of that isotope in coolant to the concentration of all isotopes of the same element in coolant (the elemental bulk concentration). For example, if calculating the precipitation rate of ^{54}Fe and the coolant also contains ^{55}Fe, ^{58}Fe, and ^{59}Fe, then the precipitation rate of ^{54}Fe is calculated by adding together the total precipitation rate of all Fe ions, and multiply that by the ^{54}Fe concentration divided by the total Fe concentration in coolant, which includes ^{54}Fe, ^{55}Fe, ^{58}Fe, and ^{59}Fe.

3.3 Mass balance equation set

This model includes global ordinary differential equations (ODEs) to determine values in the primary loop coolant, and nodal ODEs (solved on each node to obtain a value at each node) to determine values in the oxide layer.

3.3.1 Coolant mass balance (Global ODEs)

Coolant concentrations are implemented using global ODEs (single equation solved for the entire system) to determine a single value for the bulk coolant. A single scalar value is calculated to represent the "bulk" concentration of each constituent species, as it is assumed that the coolant concentration is approximately uniform throughout the loop, due to the high flow rates and coolant velocities of a typical primary loop. Spatial resolution in the coolant would require a fine temporal resolution to capture the effects of fluid transport affecting local coolant concentrations. For this model, larger timesteps are desirable as the system quickly approaches an equilibrium-like state where chemical kinetics slows significantly. Finer timesteps would not improve the accuracy of this model significantly, given the many approximations and simplifications made to study the chemical kinetics of interest. Future work could implement this finer spatial and temporal resolution as physics coupling to include fluid dynamics/thermal hydraulics effects are implemented.

Elements

Elemental coolant source terms include metal release from corrosion and dissolution from the oxide layer; loss terms include precipitation to the oxide layer.

$$\frac{dC_b^{elem}}{dt} = \sum_{nodes} R_{corr,MR} + \sum_{nodes} R_{dissol(+)} - \sum_{nodes} R_{precip(-)} \tag{3.29}$$

Isotopes

Nonactivated isotopes are added to the coolant through dissolution of oxides (proportional to the isotope's natural abundance, μ), and removed by precipitation (proportional to the isotope's concentration in the coolant) and activation by neutron capture.

$$\frac{dC_b^{iso}}{dt} = \mu \sum_{nodes} R_{dissol(+)} - \frac{C_b^{iso}}{C_b^{elem}} \sum_{nodes} R_{precip(-)} - \sum_{nodes} C_b^{iso} \sigma_{capture} \phi \tag{3.30}$$

Activated isotopes

Activated isotopes in the coolant are produced from activation of nonactive parent isotopes, and removed by radioactive decay and precipitation to the oxide layer (proportional to the coolant concentration).

$$\frac{dC_b^{act}}{dt} = \sum_{nodes} C_b^{iso} \sigma_{capture} \phi - \frac{C_b^{act}}{C_b^{elem}} \left[\sum_{nodes} R_{precip(-)} \right] - \lambda_{act} C_b^{act} \tag{3.31}$$

3.3.2 Oxide mass balance (Nodal ODEs)

An oxide concentration value for each species of interest is calculated at every node to obtain good spatial resolution. Equations for elemental and activated isotope concentrations are implemented; nonactivated isotopes are not computed in the oxide, as it is assumed that all activated species are created in the coolant due to neutron flux activation in the core, and activation of oxides in the core is negligible compared to that in the coolant.

Elements

Oxide elemental sources in this model include corrosion growth and precipitation from the coolant, and loss terms represented are due to corrosion metal release and oxide dissolution. Note that in the case of precipitation, $R_{dissol/precip}$ will be negative, and will be a source term in this expression.

$$\frac{dC_{ox}^{elem}}{dt} = R_{corr} - R_{corr,MR} - R_{dissol/precip} \tag{3.32}$$

Activated isotopes

Activated isotopes in the oxide layer are only added through precipitation of activated isotopes from the coolant, and are lost only through radioactive decay. Dissolution of activated oxides is neglected.

$$\frac{dC_{ox}^{act}}{dt} = \frac{C_b^{act}}{C_b^{elem}} R_{precip(-)} - \lambda_{act} C_{ox}^{act} \tag{3.33}$$

3.3.3 Accumulated activity

Activity of an oxide at any quadrature point is determined by Equation 3.34 (with units of Becquerels, or s^{-1}). Integrating this over the entire primary loop yields the total accumulated activity.

$$Activity = \sum_{isotopes} mols_{ox}^{act} \times N_A \times \lambda_{act} \tag{3.34}$$

3.4 Comparison with previous models

One of the major differences between this model and prior codes is the inclusion of spatial resolution for the oxide composition and activity. Additionally, rates for ion solubility (dissolution/precipitation) source term are implemented using mechanistic models adapted from Macdonald, and this model also implements a corrosion source term (oxide growth and metal release). This is an improvement over models using empirical or semi-empirical kinetic

rates determined from experiments or plant measurements, such as CPAIR-P and ACE-II.

This model lacks the detailed water chemistry and electrochemical potential calculation included in the Macdonald model. The electrochemical potential across the loop is approximated from results of Macdonald et al. and implemented using a function for each region of the primary loop [5]. The value for pH is also assumed constant. Future work to calculate local values for pH and ECP can be implemented into the code for this model with little difficulty (see Chapter 4 for details). This applies to thermal hydraulic and neutronic parameters as well, which are represented by single values or by mathematical functions to vary the parameter by region. Particulates are not treated in this model (included in PACTOLE-2), and the oxide is treated as a single layer instead of a separate inner and outer oxide layer (included in ACE-II, PACTOLE-2). Future work to add a particulate model could be easily integrated into this code due to its modular nature (see Chapter 4 for further discussion). The thickness of the oxide is calculated based on local oxide elemental concentrations, an improvement over the previous model used in Ouroboros (Eq. 3.2).

Though the oxide growth term due to corrosion (adapted from the Castelli model) uses an empirical rate constant, this can be easily modified in the future, and still presents an improvement to models that do not consider this source term. As Castelli writes, "With so much uncertainty in our ability to describe these fundamental source terms, one wonders how or why anyone would choose to proceed from this point. The answer is quite clear, at least to me. Modeling of these phenomena must start somewhere, and even if the initial set of data is somewhat flawed, it will always be possible to improve it, in time, as new investigations are performed in the future" [23]. Though this model is not entirely mechanistic, using empirical rates and crude assumptions to simplify the problem, it implements small improvements to prior models, replacing empirical rates with more mechanistic-based physics where possible. Over time, the aim is to replace more and more of these empirical components and simplifications to develop a model that is increasingly representative of the physics and chemistry driving the system.

Chapter 4

Computational Methods & Code Structure

The MOOSE Framework was developed to solve large systems of tightly coupled PDEs using implicit numerical methods. MOOSE `Kernels` are "PDE operators representing physics," which are applied to `Variables` in the MOOSE problem [32]. MOOSE also provides a system for solving systems of coupled global and nodal ODEs by using MOOSE `ODEKernels` and `NodalKernels`, respectively.

The equations used in this model involve many coupled ODEs - including global ODEs and nodal ODEs (to obtain a fine spatial resolution). While this set of ODEs could have been solved and implemented in another language or framework (such as `Python` or `FORTRAN`), the C++-based MOOSE framework offers a system to "loosely couple" multiple MOOSE-based applications using the `MultiApp`/`Transfer` systems. The `MultiApp` system provides the ability to easily transfer data between physics occurring on different length scales or time scales [21]. Thus, the MOOSE framework was chosen to facilitate future integration of this code with existing codes, such as the MOOSE-based crud chemistry code, MAMBA-BDM [8].

This chapter details the structure and MOOSE implementation of these ODEs in the Ouroboros source term code. Coupling global and nodal ODEs can be quite challenging, particularly when using an implicit numerical scheme to solve the coupled system. As such, some assumptions and simplifications were made to facilitate this coupling as these equations were implemented using MOOSE `Kernels`. These assumptions are presented in the following

sections.

4.1 Coolant concentrations: ODE implementation

4.1.1 Elemental coolant concentrations

Concentration of each element in the bulk coolant is represented using a global ODE (Eq. 3.29), implemented in the code using a MOOSE `ODEKernel` to represent each component of the ODE. The equation is broken up into three pieces: the time derivative term, the corrosion release source term, and the dissolution/precipitation rate term.

$$\frac{dC_b^{elem}}{dt} = \sum_{nodes} R_{corr,MR} + \sum_{nodes} R_{dissol(+)} - \sum_{nodes} R_{precip(-)} = R_{total}^{elem}$$

Difficulty in properly coupling the summed nodal rates for each source/sink term necessitated a simplification in the implementation of this ODE. A postprocessor, which accumulates the value of a parameter over the entire spatial domain (all nodes) at each timestep, was used to compute the effective dissolution/precipitation rate per element at the beginning of each timestep. This rate (effectively a constant over each timestep) was coupled into an `ODEKernel` that returns a constant rate term. While this assumption does remove some of the implicit coupling from this system, it lifts some of the computational burden of solving the full, tightly coupled system while still allowing the use of an implicit numerical scheme. This approach still provides an improvement over a fully explicit scheme, in both computational cost and coupling, despite this simplification. The metal release term is computed in its own `ODEKernel`. Applying each of these kernels to the scalar variable for the coolant concentration fully represents this ODE in this system.

Time derivative term

The time derivative term is implemented using an existing MOOSE `ODEKernel`, named `ODETimeDerivative` (see Listing 4.1). In computing the residual, this code returns the time derivative of the scalar variable to which this kernel is applied.

Dissolution/precipitation term

The constant dissolution/precipitation rate term is implemented using an `ODEKernel` that returns `-1 * rate`. In the case where there are multiple reactions/species with dissolution/-precipitation reactions in the coolant mass balance, this kernel is simply applied once for each reaction in the mass balance (see input example in Listing 4.3).

```
Real ODETimeDerivative::computeQpResidual()
{
 return _u_dot[_i];
}
Real ODETimeDerivative::computeQpJacobian()
{  if (_i == _j)
     return _du_dot_du[_i];
   else
     return 0;
 }
```

Listing 4.1: `ODETimeDerivative` Kernel

```
Real ImplicitODEVariableRate::computeQpResidual()
{  // the term of the ODE without the time derivative term
    return -1 * _rate;
}
```

Listing 4.2: `ImplicitODEVariableRate` ODEKernel

```
[./dp_Fe2p_Fe3O4_cool_bulk]
      type = ImplicitODEVariableRate
      variable = conc_fe_cool
      pp_rate = pp_releaserate_Fe2p_Fe3O4
      loop_volume = pp_volume_tot
[../]
[./dp_Fe2p_Fe2O3_cool_bulk]
```

```
      type = ImplicitODEVariableRate
      variable = conc_fe_cool
      pp_rate = pp_releaserate_Fe2p_Fe2O3
      loop_volume = pp_volume_tot
[../]
```

Listing 4.3: Input example for using `ImplicitODEVariableRate` to calculate the release rate for Fe^{2+} from Fe_2O_3 and Fe_3O_4

Metal release term

The metal release rate due to corrosion growth is implemented in `MetalReleaseODE`. As this is a global rate, the diffusivity, weight percent (of the element within the base construction materials), wetted area, density, and thickness are volume-averaged scalar values for the entire loop, evaluated using a postprocessor at the beginning of each timestep. This rate is calculated using Equation 3.6:

$$R_{corr,MR} = D\frac{\rho_{alloy} \cdot wt\% \cdot wa}{thickness \cdot MM_{elem}}$$

```
Real MetalReleaseODE::computeQpResidual()
{
   Real release;
    release = (_diffusivity[_i] * _wt[_i] * _wa[_i] * _dens[_i])
                            / (_thickness[_i] * _MM_elem);

   return -1*release;
}
```

Listing 4.4: `MetalReleaseODE` ODEKernel

```
[./mr_fe_cool_bulk]
      type = MetalReleaseODE
      variable = conc_fe_cool
```

```
      diffusivity = diffusivity_elemavg_fe
      wt_percent = wtpercent_elemavg_fe
      thickness = thick_elemavg
      wetted_area = WA_elemavg
      alloy_density = density_elemavg
      loop_volume = pp_volume_tot
      MM_elem = 0.055845
[../]
```

Listing 4.5: Input example for using `MetalReleaseODE` to calculate the release rate of Fe due to corrosion growth and release

4.1.2 Isotopic coolant concentrations (nonactivated)

Nonactive isotope bulk coolant concentration mass balance is computed using Equation 3.30. Each component of this ODE is included in a single `ODEKernel`, called `ImplicitODENonActive`.

$$\frac{dC_b^{iso}}{dt} = \mu \sum_{nodes} R_{dissol(+)} - \frac{C_b^{iso}}{C_b^{elem}} \sum_{nodes} R_{precip(-)} - \sum_{nodes} C_b^{iso} \sigma_{capture} \phi$$

```
Real ImplicitODENonActive::computeQpResidual()
{
   // the term of the ODE without the time derivative term
   Real my_var = _mu * _dissol / _vol - _u[_i] * _nc / _vol - (_u[_i] / ↩
       _elemconc[_i])* abs(_precip / _vol);

     //to prevent unphysical negative concentrations - cannot remove ↩
        from coolant if coolant concentration is zero
     if (_u[_i] < 0.)      my_var = _mu * _dissol / _vol;

    return -1*my_var;
}
```

Listing 4.6: `ImplicitODENonActive` ODEKernel

```
[./conc_fe54_cool_bulk_nonact]
      type = ImplicitODENonActive
      variable = conc_fe54_cool
      elem_conc = conc_fe_cool
      natural_abundance = 0.00585
      ncapture_pp = ncapture_fe54_pp
      dissolrate = pp_total_dissol_rate_fe
      preciprate = pp_total_precip_rate_fe
      loop_volume = pp_volume_tot
[../]
```

Listing 4.7: Input example for using `ImplicitODENonActive` to calculate the bulk coolant concentration of ^{54}Fe

4.1.3 Activated isotopic coolant concentrations

Activated isotopes are implemented similarly to the nonactive isotopes. Equation 3.31 is implemented in a single `ODEKernel` that contains the neutron activation source term and both the precipitation and radioactive decay loss terms.

$$\frac{dC_b^{act}}{dt} = \sum_{nodes} C_b^{iso}\sigma_{capture}\phi - \frac{C_b^{act}}{C_b^{elem}}\left[\sum_{nodes} R_{precip(-)}\right] - \lambda_{act}C_b^{act}$$

```
Real ImplicitODEActive::computeQpResidual()
{ Real  my_var;
  if (_elemconc[_i] > 0. && _u[_i] > 0.)
   my_var = _nonact[_i] * _nc/_vol - _lambda * _u[_i] - (u[_i] / _elemconc[_i])
       * abs(_precip/_vol);
  //cannot remove if concentration is zero:
  else if (_u[_i] <= 0.)     my_var = _nonact[_i] * _nc/_vol;
  else my_var = 0;
  return -1*my_var;
}
```

Listing 4.8: `ImplicitODEActive` ODEKernel

```
[./conc_fe55_cool_bulk]
      type = ImplicitODEActive
      variable = conc_fe55_cool
      elem_conc = conc_fe_cool
      nonactive_conc = conc_fe54_cool
      lambda = 8.01e-09
      ncapture_pp = ncapture_fe54_pp
      preciprate = pp_total_precip_rate_fe
      loop_volume = pp_volume_tot
[../]
```

Listing 4.9: Input example for using `ImplicitODEActive` to calculate the bulk coolant concentration of ^{55}Fe

4.1.4 Sample input block

To demonstrate how these `ODEKernels` are used to set up these ODEs in `Ouroboros`, a sample input block that implements all three of these ODEs for iron is provided (Listing 4.10). Two ion-oxide reactions are included ($Fe^{2+} \rightarrow Fe_2O_3$ and $Fe^{2+} \rightarrow Fe_3O_4$) and one isotope activation reaction ($^{54}Fe(n,\gamma)^{55}Fe$). The primary loop is spatially represented from $x = 0$ to 78 meters. The equations represented by this input sample are shown below.

Bulk coolant concentration of element Fe:

$$\frac{dC^{\mathrm{Fe}}_{cool}}{dt} = \sum_{x=0}^{x=78m} R_{corr,\mathrm{Fe}}(x) + \sum_{x=0}^{x=78m} R^{\mathrm{Fe}^{2+}\rightarrow\mathrm{Fe_2O_3}}_{dissol/precip}(x) + \sum_{x=0}^{x=78m} R^{\mathrm{Fe}^{2+}\rightarrow\mathrm{Fe_3O_4}}_{dissol/precip}(x)$$

Bulk coolant concentration of nonactivated isotope ^{54}Fe:

$$\frac{dC^{\mathrm{Fe54}}_{cool}}{dt} = \mu^{\mathrm{Fe54}} \sum_{x=0}^{x=78m} \left[R^{\mathrm{Fe}^{2+}\rightarrow\mathrm{Fe_2O_3}}_{dissolution}(x) + R^{\mathrm{Fe}^{2+}\rightarrow\mathrm{Fe_3O_4}}_{dissolution}(x)\right] - C^{\mathrm{Fe54}}_{cool} \sum_{x=0}^{x=78m} \sigma^{\mathrm{Fe54}}_{(n,\gamma)}\Phi(x) - \frac{C^{\mathrm{Fe54}}_{cool}}{C^{\mathrm{Fe}}_{cool}} \sum_{x=0}^{x=78m} \left[R^{\mathrm{Fe}^{2+}\rightarrow\mathrm{Fe_2O_3}}_{precipitation}(x) + R^{\mathrm{Fe}^{2+}\rightarrow\mathrm{Fe_3O_4}}_{precipitation}(x)\right]$$

Bulk coolant concentration of activated isotope ^{55}Fe:

$$\frac{dC^{\mathrm{Fe55}}_{cool}}{dt} = C^{\mathrm{Fe54}}_{cool} \sum_{x=0}^{x=78m} \sigma^{\mathrm{Fe54}}_{(n,\gamma)}\Phi(x) - \lambda^{\mathrm{Fe55}} C^{\mathrm{Fe55}}_{cool} - \frac{C^{\mathrm{Fe55}}_{cool}}{C^{\mathrm{Fe}}_{cool}} \sum_{x=0}^{x=78m} \left[R^{\mathrm{Fe}^{2+}\rightarrow\mathrm{Fe_2O_3}}_{precipitation}(x) + R^{\mathrm{Fe}^{2+}\rightarrow\mathrm{Fe_3O_4}}_{precipitation}(x)\right]$$

```
[ScalarKernels]
   [./td_conc_fe_cool]
      type = ODETimeDerivative
      variable = conc_fe_cool
   [../]
   [./mr_fe_cool_bulk]
      type = MetalReleaseODE
      variable = conc_fe_cool
      diffusivity = diffusivity_elemavg_fe
      wt_percent = wtpercent_elemavg_fe
      thickness = thick_elemavg
      wetted_area = WA_elemavg
      alloy_density = density_elemavg
      loop_volume = pp_volume_tot
      MM_elem = 0.055845
   [../]
   [./dp_Fe2p_Fe3O4_cool_bulk]
      type = ImplicitODEVariableRate
      variable = conc_fe_cool
      pp_rate = pp_releaserate_Fe2p_Fe3O4
      loop_volume = pp_volume_tot
   [../]
   [./dp_Fe2p_Fe2O3_cool_bulk]
      type = ImplicitODEVariableRate
      variable = conc_fe_cool
      pp_rate = pp_releaserate_Fe2p_Fe2O3
      loop_volume = pp_volume_tot
   [../]

   [./td_conc_fe54_cool]
      type = ODETimeDerivative
      variable = conc_fe54_cool
   [../]
   [./conc_fe54_cool_bulk_nonact]
      type = ImplicitODENonActive
      variable = conc_fe54_cool
      elem_conc = conc_fe_cool
      natural_abundance = 0.00585
```

```
39      ncapture_pp = ncapture_fe54_pp
40      dissolrate = pp_total_dissol_rate_fe
41      preciprate = pp_total_precip_rate_fe
42      loop_volume = pp_volume_tot
43    [../]
44
45    [./td_conc_fe55_cool]
46      type = ODETimeDerivative
47      variable = conc_fe55_cool
48    [../]
49    [./conc_fe55_cool_bulk]
50      type = ImplicitODEActive
51      variable = conc_fe55_cool
52      elem_conc = conc_fe_cool
53      nonactive_conc = conc_fe54_cool
54      lambda = 8.01e-09
55      ncapture_pp = ncapture_fe54_pp
56      preciprate = pp_total_precip_rate_fe
57      loop_volume = pp_volume_tot
58    [../]
59  []
```

Listing 4.10: Input example for global ODE coupling for iron

4.2 Oxide concentrations: nodal ODE implementation

Oxide concentrations are implemented using `NodalKernels`. These MOOSE `Kernels` are used to solve ODEs on each quadrature point (node) of the problem domain, thus giving spatial resolution not present when solving global ODEs for scalar variable values. Implementing these ODEs with`NodalKernels` requires the use of nodal variables and auxvariables.

The time derivative piece of nodal ODEs is implemented almost exactly the same way as in `ODETimeDerivative`. The `TimeDerivativeNodalKernel` differs slightly as it computes on each quadrature point (`[_qp]`) for the nodal variable; the residual computation for `ODETimeDerivative` and `TimeDerivativeNodalKernel` can be found in Listing 4.1 and 4.11, respectively.

```
Real TimeDerivativeNodalKernel::computeQpResidual()
{
   return _u_dot[_qp];
}
```

Listing 4.11: `TimeDerivativeNodalKernel` Kernel

4.2.1 Nonactive oxides

Oxide concentrations of nonactivated elements are represented by Equation 3.32 and implemented using three `NodalKernels` to represent each process in the expression.

$$\frac{dC_{ox}^{elem}}{dt} = R_{corr} - R_{corr,MR} - R_{dissol/precip}$$

Corrosion growth source term

`NodalCorrosionGrowth` calculates an oxide growth rate due to corrosion, Equation 3.5 .

$$R_{corr} = \frac{dn_{elem}}{dt} = \frac{k_p \cdot wa}{2\sqrt{t} \cdot MM_{oxide}} \frac{d_h}{4}$$

```
Real NodalCorrosionGrowth::computeQpResidual()
{
      Real tot_rate = 0.5 * _rateconst[_qp] * _wa[_qp] * (std::pow(_t, -0.5)) / (_MM_ox );
      tot_rate = tot_rate * (_dh[_qp]/4);

      return -1 * tot_rate;
}
```

Listing 4.12: `NodalCorrosionGrowth` Kernel

```
[./corr_Fe2O3]
      type = NodalCorrosionGrowth
      variable = conc_Fe2O3_ox_nodal
      rate_constant = cg_rate_const_fe
      wetted_area = wet_area
      MM_oxide = 0.15969
[../]
```

Listing 4.13: Input example for using `NodalCorrosionGrowth` to calculate the oxide growth rate of Fe_2O_3 due to corrosion

Corrosion metal release loss term

`NodalMetalRelease` implements Equation 3.6 for the corrosion release loss term. This loss term kernel is the complement to the `MetalReleaseODE` coolant concentration source term kernel, and is calculated on each individual node instead of a scalar metal release term for the entire loop.

$$R_{corr,MR} = D\frac{\rho_{alloy} \cdot wt\% \cdot wa}{thickness \cdot MM_{elem}}$$

```
Real NodalMetalRelease::computeQpResidual()
{
   Real release = 0;
   release = (_diffusivity[_qp] * _wt[_qp] * _wa[_qp] * _dens[_qp]) / (_thickness[_qp] * ↩
       _MM_elem)*(_vol[_qp] / _loopvol);
   release = release * (_dh[_qp]/4);
   if (_oxconc[_qp] <= 0.) release = 0.;

   return release;
}
```

Listing 4.14: `NodalMetalRelease` Kernel

```
[./mr_Fe2O3_ox]
      type = NodalMetalRelease
      variable = conc_Fe2O3_ox_nodal
      diffusivity = base_fe_diffusivity
      wt_percent = base_fe
      thickness = aux_thick
      wetted_area = wet_area
      equiv_vol = my_volume
      loop_volume = pp_volume_tot
      alloy_density = metal_density
      oxide_conc = conc_Fe2O3_ox_nodal
      MM_elem = 0.055845
[../]
```

Listing 4.15: Input example for using `NodalMetalRelease` to calculate the nodal release rate of Fe from Fe_2O_3

Dissolution/precipitation release term

`NodalDissolPrecip` simply applies the appropriate source/loss rate from dissolution or precipitation. It also checks to ensure oxide concentrations or coolant concentrations are not zero to prevent nonphysical solutions with negative concentrations. This is implemented using a Heaviside function; if the solid oxide concentration at this point is zero and the rate indicates dissolution, the Heaviside function returns 0, otherwise it returns 1. This coefficient is multiplied onto the rate to force the rate to zero if dissolution cannot occur. Likewise, the function is also used to ensure that if the rate indicates precipitation, there is a nonzero bulk coolant concentration that will allow for precipitation to occur.

```
Real NodalDissolPrecip::computeQpResidual()
{
   int H =  1;
    Real rr = _release[_qp] * (_dh[_qp]/4);
   //check oxide conc for dissolution to prevent nonphysical negative concentrations
    H = evaluateHeaviside(-1*rr, _u[_qp]);
    //check coolant conc for precipitating
    if (H==1)  H = H * evaluateHeaviside(rr, _cbs[0]);
```

```
    rr = H * rr;

    return rr;
}
int NodalDissolPrecip::evaluateHeaviside(Real rate, Real my_conc)
{
    int _heaviside;
   //if concentration (of solid) is 0 and rate is negative, evaluate to zero.
    if (my_conc <= 0. && rate <= 0.)      _heaviside = 0;
     //else, evaluate to 1.
    else       _heaviside = 1;

  return _heaviside;
}
```

Listing 4.16: `NodalDissolPrecip` Kernel

```
[./dp_Fe2p_Fe2O3]
      type = NodalDissolPrecip
      variable = conc_Fe2O3_ox_nodal
      release_rate = releaserate_Fe2p_Fe2O3
      bulk_conc = conc_fe_cool
[../]
```

Listing 4.17: Input example for using `NodalDissolPrecip` to calculate the nodal dissolution/precipitation rate for Fe_2O_3

4.2.2 Activated oxides

The concentration of activated isotopes in oxides is represented by Equation 3.33. This ODE is implemented in a single `NodalKernel`, `PrecipRate`, that includes both the precipitation source term and radioactive decay loss term.

$$\frac{dC_{ox}^{act}}{dt} = \frac{C_b^{act}}{C_b^{elem}} R_{precip(-)} - \lambda_{act} C_{ox}^{act}$$

```
Real PrecipRate::computeQpResidual()
{
  Real my_rate;
  Real my_precip = abs(_precip[_qp]) * (_dh[_qp]/4);

  if (_cbs[0] > 0. && _iso[0] >= 0. )
  {
    if (_u[_qp]>0)    my_rate =  (_iso[0]/_cbs[0]) * my_precip  -  _lambda * _u[_qp];
    else my_rate = (_iso[0]/_cbs[0]) * my_precip ;
    //ensure the concentration does not go negative
    if (_u[_qp] <=0 && my_rate <0.) my_rate =0.;
  }
  else my_rate = 0.;

  return -1*my_rate;
}
```

Listing 4.18: `PrecipRate` Kernel

```
[./precip_fe55_act_ox_fe54]
      type = PrecipRate
      variable = conc_fe55_oxide_active
      precip_rate = total_precip_rate_fe
      elem_conc = conc_fe_cool
      iso_conc = conc_fe55_cool
      lambda = 8.01e-09
[../]
```

Listing 4.19: Input example for using `PrecipRate` to calculate the nodal precipitation and decay rate of ^{55}Fe in the oxide

4.2.3 Sample input block

A sample input block to demonstrate use of these `NodalKernels` to implement nodal ODEs for iron is provided (Listing 4.20). One ion-oxide reactions is included ($Fe^{2+} \rightarrow Fe_2O_3$) and one isotope activation reaction ($^{54}Fe(n,\gamma)^{55}Fe$).

```
[NodalKernels]
   [./td_conc_Fe2O3_ox_nodal]
      type = TimeDerivativeNodalKernel
      variable = conc_Fe2O3_ox_nodal
   [../]
   [./corr_Fe2O3]
      type = NodalCorrosionGrowth
      variable = conc_Fe2O3_ox_nodal
      rate_constant = cg_rate_const_fe
      wetted_area = wet_area
      MM_oxide = 0.15969
   [../]
   [./dp_Fe2p_Fe2O3]
      type = NodalDissolPrecip
      variable = conc_Fe2O3_ox_nodal
      release_rate = releaserate_Fe2p_Fe2O3
      bulk_conc = conc_fe_cool
   [../]
   [./mr_Fe2O3_ox]
      type = NodalMetalRelease
      variable = conc_Fe2O3_ox_nodal
      diffusivity = base_fe_diffusivity
      wt_percent = base_fe
      thickness = aux_thick
      wetted_area = wet_area
      equiv_vol = my_volume
      loop_volume = pp_volume_tot
      alloy_density = metal_density
      oxide_conc = conc_Fe2O3_ox_nodal
      MM_elem = 0.055845
   [../]
   [./td_conc_fe55_oxide_active]
```

```
33      type = TimeDerivativeNodalKernel
34      variable = conc_fe55_oxide_active
35    [../]
36    [./precip_fe55_act_ox_fe54]
37      type = PrecipRate
38      variable = conc_fe55_oxide_active
39      precip_rate = total_precip_rate_fe
40      elem_conc = conc_fe_cool
41      iso_conc = conc_fe55_cool
42      lambda = 8.01e-09
43    [../]
44  []
```

Listing 4.20: Input example for nodal ODE coupling for iron

4.3 Auxiliary calculations (`AuxKernels`)

Intermediate calculations are implemented using `AuxKernels`. These compute a value for an intermediate variable (`AuxVariable`) upon each timestep or each solver iteration (specified in the input block).

Surface concentrations

The most crucial calculation for this model is the saturated/surface concentration for each reaction, Equation 3.25. This calculation is implemented in `EchemSurfaceConcentration`.

$$C_{s,i} = 10^{\left[\frac{z}{b}\frac{F}{2.303R\cdot T}\left(\frac{-\Delta G^0_{f,i}}{zF} - \mathrm{ECP}\right) - \frac{x}{b}\mathrm{pH}\right]}$$

Note that the Gibbs energy for the reaction is computed at each quadrature point based upon the enthalpy change and entropy change for the reaction, and the temperature. This auxkernel calculation must be executed at the beginning of the simulation, when the system is initialized, to ensure that values for saturated concentrations are set before the first dissolution/precipitation rate calculation is executed. This calculation is also only needed on initialization, as the temperature, pH, and electrochemical corrosion potential variables do

not vary temporally in this version of the code. This is set in the input block through the `execute_on` parameter (see Listing 4.22). Should temporal evolution of temperature, pH, or electrochemical potential be added to future versions of the code, this option will need to be modified to execute on each linear or nonlinear solver iteration or on each timestep, in addition to execution upon initialization.

```
Real EchemSurfaceConcentration::computeValue()
{ Real R = 8.314;
  Real F = 96485.33289; //Faraday constant , Coulomb/mol
  Real gibbs = _dH - _T[_qp] * _dS;
  Real E0 = (-1 * gibbs ) / (abs(_z) * F);
  Real temp1 = (_z / _b) * F / (2.303 * R * _T[_qp]);
  Real temp2 = (_x / _b) * _pH;
  Real expon = temp1 * (E0 - _ECP[_qp]) - temp2;
  Real value = std::pow(10.0, expon);

  return value;
}
```

Listing 4.21: `EchemSurfaceConcentration` Kernel

```
[./surface_conc_Fe2p_Fe3O4_aux]
      type = EchemSurfaceConcentration
      variable = surfconc_Fe2p_Fe3O4
      x = 8
      b = 3
      z = 2
      delta_H = -292220.0
      delta_S = -279.86
      execute_on = 'initial'
[../]
```

Listing 4.22: Input example for using `EchemSurfaceConcentration` to determine the saturated/surface concentration for Fe^{2+} in equilibrium with Fe_3O_4

Dissolution/precipitation release rates

Dissolution and precipitation rates for each species are calculated on each node using Equation 3.27 , which is implemented in the `ReleaseDepositionRate AuxKernel`.

$$R_i = D_i \cdot Sh \cdot \mathtt{wa} \, (C_{s,i} - C_{b,i})$$

```
Real ReleaseDepositionRate::computeValue()
{
  Real  _release_rate =  _diffusivity[_qp]  * _sherwood[_qp]
        *  _wet_area[_qp]  * (_surf_conc[_qp]*(4/_dh[_qp])- (_cbs[0]));

 return _release_rate;
}
```

Listing 4.23: `ReleaseDepositionRate` Kernel

```
[./releasedepositionrate_Fe2p_Fe3O4]
      type = ReleaseDepositionRate
      variable = releaserate_Fe2p_Fe3O4
      bulk_conc = conc_fe_cool
      surf_conc= surfconc_Fe2p_Fe3O4
      species_diffusivity = diffusivity_Fe2p
      sherwood_number = sherwood_Fe2p
      wetted_area = wet_area
      hydraulic_diameter = hydr_diameter
      execute_on = 'initial timestep_begin'
[../]
```

Listing 4.24: Input example for using `ReleaseDepositionRate` to calculate the release rate of Fe^{2+} from Fe_3O_4

Local oxide activity

The activity contribution for each active isotope present in oxides at each node is calculated using Equation 3.34 and implemented in `ActivityOxideCalc`. The value output is in units

of Becquerels per square meter.

$$Activity = mols_{ox}^{act} \times N_A \times \lambda_{act} \times \frac{1}{wa}$$

```
Real ActivityOxideCalc::computeValue()
{ //number of moles of active oxide * NA * lambda = 1/s -> becquerels
Real NA = 6.022E23; //Avogadro's number, 1/mols

//returns value in Becquerel/m^2
  return _c[_qp] * NA * _lambda / _wa[_qp];
}
```

Listing 4.25: `ActivityOxideCalc` Kernel

```
[./activity_aux_fe55]
      type = ActivityOxideCalc
      variable = oxide_active_fe55
      oxide_conc = conc_fe55_oxide_active
      wetted_area = wet_area
      decay_parameter = 8.01e-09
      execute_on = timestep_end
[../]
```

Listing 4.26: Input example for using `ActivityOxideCalc` to calculate the activity contribution of precipitated ^{55}Fe

4.4 Input syntax and variable names

Variable	Description	Input type	Input syntax
`_u[_i]`	variable this kernel acts on	Variable name	`variable = var_name`
		`ImplicitODEVariableRate`	
`_rate`	rate of change (constant)	Postprocessor name	`pp_rate = pp_name`
		`MetalReleaseODE`	
`_diffusivity`	Average diffusivity		`diffusivity = pp_name`
`_wt`	Weight percent of elem in base material	Postprocessor name	`wt_percent = pp_name`
`_wa`	Avg wetted area		`wetted_area = pp_name`
`_dens`	Avg base alloy density		`alloy_density = pp_name`
`_thickness`	Avg oxide thickness		`thickness = pp_name`
`_MM_elem`	Molar mass of alloy	Float	`MM_elem = 0.055845`
		`ImplicitODENonActive`	
`_mu`	Natural abundance of isotope	Float	`natural_abundance = 0.00585`
`_dissol`	Total dissolution rate along loop	Postprocessor name	`dissolrate = pp_name`
`_nc`	Total neutron capture along loop		`ncapture_pp = pp_name`
`_precip`	Total precipitation rate along loop		`preciprate = pp_name`
`_vol`	Total loop volume		`loop_volume = pp_name`
`_elemconc`	Element bulk coolant concentration	Variable name	`elem_conc = bulk_conc_var`
		`ImplicitODENonActive`	
`_lambda`	Radioactive decay parameter for isotope	Float	`lambda = 8.01e-9`
`_nonact`	Nonactive isotope coolant concentration	Variable name	`nonactive_conc = var_name`

Table 4.1: `ODEKernel` variable names and descriptions

Variable	Description	Input type	Input syntax
_u[_qp]	variable this kernel acts on	Variable name	variable = var_name
NodalCorrosionGrowth			
_wa[_qp]	Nodal wetted area	AuxVariable name	wetted_area = name
_rateconst[_qp]	Corrosion rate		rate_constant = name
_dh[_qp]	Hydraulic diameter		hydraulic_diameter = name
_MM_ox	Molar mass of oxide	Float	MM_oxide = 0.15969
NodalMetalRelease			
_diffusivity[_qp]	Ion diffusivity through oxide	AuxVariable name	diffusivity = var_name
_wt[_qp]	Weight percent of element in base alloy		wt_percent = var_name
_dens[_qp]	Density of base alloy		alloy_density = var_name
_thickness[_qp]	Oxide thickness		thickness = var_name
_vol[_qp]	Node volume		equiv_vol = var_name
_loopvol	Total volume	Postprocessor Name	loop_volume = pp_name
_MM_elem	Base alloy molar mass	Float	MM_elem = 0.055845
_oxconc[_qp]	Oxide concentration	Variable name	oxide_conc = var_name
NodalDissolPrecip			
_release[_qp]	Ion release rate	Variable name	release_rate = var_name
_cbs[0]	Element bulk coolant concentration		bulk_conc = var_name
PrecipRate			
_precip[_qp]	Nodal precipitation rate	Variable name	precip_rate = var_name
_iso[0]	Nonactive isotope bulk concentration		iso_conc = var_name
_lambda	Radioactive decay parameter for isotope	Float	lambda = 8.01e-9

Table 4.2: NodalKernel variable names and descriptions

Variable	Description	Input type	Input syntax
`_u[_qp]`	variable this kernel acts on	Variable name	`variable = var_name`
		`EchemSurfaceConcentration`	
`_dH`	Enthalpy change from reaction	Float	`delta_H = -292220.0`
`_dS`	Entropy change from reaction		`delta_S = -279.86`
`_T[_qp]`	Temperature	Variable name	`temp = var_name`
`_z`	Electrons tranferred in reaction	Integer	`z = 2`
`_b`	Stoichiometry coefficient of ion		`b = 3`
`_x`	Stoichiometry coefficient of H^+		`x = 8`
		`ReleaseDepositionRate`	
`_diffusivity[_qp]`	Ion diffusivity	AuxVariable name	`species_diffusivity = name`
`_sherwood[_qp]`	Sherwood number		`sherwood_number = var_name`
`_surf_conc[_qp]`	Surface concentration of ion		`surf_conc = var_name`
		`ActivityOxideCalc`	
`_c[_qp]`	Activated oxide concentration	Variable name	`oxide_conc = var_name`
`_lambda`	Radioactive decay parameter for isotope	Float	`decay_parameter = 8.01e-9`

Table 4.3: `AuxKernel` variable names and descriptions

4.5 Calculation of Gibbs energies

The most crucial component of this model is the saturated/surface concentration equilibrium calculation for each dissolution reaction; this calculation requires the Gibbs energies for that reaction. The Gibbs energy of a system, as derived from the second law of thermodynamics, is defined by Equation 4.1 [34].

$$G = H - TS \tag{4.1}$$

For a constant temperature system that undergoes a state change (such as a chemical reaction), the Gibbs energy changes with the following relation [34]:

$$\Delta G = \Delta H - T\Delta S \tag{4.2}$$

Accurate thermodynamic data can be difficult to find for all reactions of interest at appropriate temperatures. For this work, the Gibbs energies for each reaction were approximated using the fundamental thermodynamic definitions based on the enthalpy and entropy of reactants and products, and the temperature.

$$\begin{aligned}\Delta G_{rxn} &= G_{products} - G_{reactants} \\ &= H_{products} - H_{reactants} - T(S_{products} - S_{reactants}) \\ &= \Delta H_{rxn} - T\Delta S_{rxn}\end{aligned}$$

The Gibbs energy is calculated at each node with local temperature dependence included. This calculation is included in `EchemSurfaceConcentration`, which requires the user to provide values for enthalpy and entropy of the reaction. As part of this work, a Python module was created to calculate the enthalpy, entropy, and Gibbs free energy for reactions of interest, and output the results as a list of Python dictionary objects containing the desired properties. This output object is used in a script to automatically create input files that include multiple sets of reactions and adds all the necessary variables, auxvariables, kernels,

auxkernels, and postprocessors to set up the system of ODEs necessary. See Listing 4.31 for code and demonstration of how this module/script is used.

4.6 Input file generation script

The structure of this problem, with sets of elements/isotopes/ions coupled, naturally lends itself to an object-oriented approach for setting up the input file/problem. As such, an object-oriented Python script for input file generation was written that provides users a simpler way to set up new problems while ensuring the proper coupling is maintained between elements, isotopes, and reactions within the problem. Users input the elements, isotopes, ions, and oxides, along with certain properties (molar mass, decay constant, neutron capture cross section, etc.) and the set of ion-oxide dissolution reactions to include; see Listing 4.27. The input generation script uses the Python module written to generate the thermodynamic data for reactions, and gets the enthalpy and entropy change for each reaction from the output. The ions, oxides, and isotopes are linked to ensure the proper variables, kernels, and postprocessors are added to the input file. The file created from this process can then be input to Ouroboros to run the simulation. Codes for object classes and sample input file generated from this script can be found in Appendices C and B.

Note that this script differs from the MOOSE `CustomActions` system, which can be used to automatically add MOOSE objects to a simulation. The `CustomActions` system does not write out a file with all of the inputs to MOOSE; instead, it sets up the solve directly within MOOSE, and the user must look at the code for each `CustomAction` to see what objects and parameters are being added to the simulation. In contrast, this Python script writes out a full Ouroboros input file including all auxkernels, kernels, variables, and other parameters. The input files created are long, but it is simple to understand exactly which physics is being applied to which variables and how everything is coupled. The script is fairly straightforward, mostly full of "write" statements and calls to methods that write a generic MOOSE block and fills it with the desired components (see Listing 4.28 for examples of these methods and their outputs). As additional MOOSE kernels, auxkernels, etc. are created to add new physics to Ouroboros, this script can be easily modified to set up the

input file desired.

```
from scripting import Element, Isotope, Ion, Oxide
# add element object instances
fe = Element('fe', 55.845E-3)
ni = Element('ni', 58.6934E-3)
co = Element('co', 58.933E-3)
#add isotope object instances
fe54 = Isotope('fe54', 'fe55', 8.01E-9, 0.00585, 2.9E-28)
fe58 = Isotope('fe58', 'fe59', 1.81E-7, 0.000028, 1.3E-28)
ni58 = Isotope('ni58', 'co58', 1.08E-7, 0.68077, 4.6E-28)
co58 = Isotope('co58', 'co59', 0, 1E-5, 1.9E-25)
co59 = Isotope('co59', 'co60', 4.17E-9, 0.9999999, 2.07E-27)
#add ion object instances
fe2p = Ion('Fe2p')
ni2p = Ion('Ni2p')
co2p = Ion('Co2p')
#add oxide object instances
fe3o4 = Oxide('Fe3O4', 231.533E-3)
fe2o3 = Oxide('Fe2O3', 159.69E-3)
nio = Oxide('NiO', 74.6928E-3)
coo = Oxide('CoO', 74.9326E-3)

#add isotope objects to the element
fe.addIsotope([fe54, fe58])
ni.addIsotope([ni58])
co.addIsotope([co58, co59])
# add ions to element objects
fe.addIon([fe2p])
ni.addIon([ni2p])
co.addIon([co2p])
# add oxides to element objects
fe.addOxide([fe2o3, fe3o4])
ni.addOxide([nio])
co.addOxide([coo])

my_reactions= [   'NiO + 2H+ -> Ni2+ + H2O',
                      'Fe3O4 + 8H+ + 2e- -> 3Fe2+ + 4H2O',
                       'Fe2O3 + 6H+ + 2e- -> 2Fe2+ + 3H2O',
                       'CoO + 2H+ -> Co2+ + H2O'   ]
#call method to create a new input file using these elements, isotopes, and reactions
generateFile(my_elements = ['fe'], reactions_included = my_reactions, my_ph = 6.9)
```

Listing 4.27: User input sample for Ouroboros input file generation script

```
addVariable(f, var_name, my_order, my_family, my_initial_condition, my_scaling):
   """   adds variable block that looks like:
   [./var_name]
      order = my_order
      family = my_family
      initial_condition = my_initial_condition
      scaling = my_scaling
   [../]

   example use:
   addVariable(f, oxide.ox_conc, 'FIRST', 'LAGRANGE', 1E-9, oxide.scaling)
   """

addTD(my_type, f,  var_name):
   """   adds appropriate time derivative block:
   [./td_var_name]
      type = TimeDerivativeNodalKernel
      variable = var_name
   [../]

   example:
   addTD('Nodal', f, oxide.ox_conc)
   """

addFunctionIC(f, var_name, my_function):
   """ adds FunctionIC block:
   addFunctionIC(f, 'metal_density', metaldensity_func)

   [./metal_density_IC]
      type = FunctionIC
      variable = metal_density
      function = 'if(x<4.587, 6550, if(x>22.798&x<28.493, 8470, 8000))'
   [../]
   """

addPP(f, var_name, pp_type, block_name, execute_string):
   """ adds postprocessor block

   addPP(f, 'metal_density', 'ElementIntegralVariablePostprocessor', 'avg_dens', '\'↩
       initial timestep_begin\'')

   output:
```

```
   [./avg_dens]
      type = ElementIntegralVariablePostprocessor
      variable = metal_density
      execute_on = 'initial timestep_begin'
   [../]
   """

addGenericBlock(f, blockname, var, mytype, params, varparams):
   """ adds a block as specified; parameters needed for a particular block are

myparams = ['oxide_conc', 'release_rate', 'bulk_conc']
myvars = [oxide.ox_conc, rxn.releaserate, elem.cool_conc]
addGenericBlock(f, 'dp_' + rxn.name, oxide.ox_conc, 'NodalDissolPrecip', myparams, myvars)

    output:
    [./dp_Ni2p_NiO]
      type = NodalDissolPrecip
      variable = conc_NiO_ox_nodal
      oxide_conc = conc_NiO_ox_nodal
      release_rate = releaserate_Ni2p_NiO
      bulk_conc = conc_ni_cool
   [../]
   """
```

Listing 4.28: Methods used for Ouroboros input file generation script. These methods, written in Python, output different types of MOOSE blocks to write a custom Ouroboros input file for desired variables, initial conditions, functions, auxkernels, kernels, and postprocessors.

```
f.write("[Variables]\n\n")                    # add Variables block

# create variables for each element desired
for elem in elements:
    # create oxide nodal element variable for each oxide
    for oxide in elem.oxides:
        oxide.setOxideConcentration('conc_' + oxide.name + '_ox_nodal')
        addVariable(f, oxide.ox_conc, 'FIRST', 'LAGRANGE', 1E-9, oxide.scaling)

    # create element coolant scalar variable
    elem.setCoolConcentration('conc_' + elem.name + '_cool')
    addVariable(f, elem.cool_conc, 'FIRST', 'SCALAR', 1E-9, elem.scaling)

    for isotope in elem.isotopes:
        #  create isotope coolant scalar variable
        isotope.setCoolConc('conc_' + isotope.name + '_cool')
        #  'if' prevents multiple of same isotope variable for Ni/Co decay chains
        if not (isotope.name == 'co59'):
            addVariable(f, isotope.cool_conc, 'FIRST', 'SCALAR', 1E-50, isotope.↩
                cool_scaling)

        # create oxide nodal activated isotope variable
        isotope.setActOxConc('conc_' + isotope.activated + '_oxide_active')
        addVariable(f, isotope.act_ox_conc, 'FIRST', 'LAGRANGE', 0.0 , isotope.↩
            act_ox_scaling)

        #  create activated isotope coolant scalar var
        isotope.setActCoolConc('conc_' + isotope.activated + '_cool')
       #  'if' prevents multiple of same isotope variable for Ni/Co decay chains
        if not (co_and_ni and isotope.name == 'ni58'):
            addVariable(f, isotope.act_cool_conc, 'FIRST', 'SCALAR', 0.0, isotope.↩
                act_scaling)
f.write("[]\n\n")                             # close Variables block
```

Listing 4.29: Code snippet to add variables automatically

4.6.1 ChemicalThermodynamics module

```
class ChemicalThermodynamics:
    """
    simple chemical thermodynamics "library" with enthalpies, entropies, and gibbs ↩
        energies of reactions
    """
    def __init__(self):
        self.reactions = []
        self.data = []
        self.convertEnthalpy = True
        print "\nCHEMICAL THERMODYNAMICS LIBRARY"
        print "Created for use with Ouroboros CRUD Chemistry Code"
        print "https://github.com/shortlab/ouroboros"
        print "Author: Alicia M. Elliott, aliciae@mit.edu\n"
        print "Initializing.....\n"
        self.generate()

    def addThermoData(self, data):
        # format of data = [species, enthalpy, entropy, gibbs]
        my_data = {
                    'species': data[0],
                    'H':data[1],
                    'S': data[2],
                    'G':data[3],
        }
        self.data.append(my_data)

    def getSpeciesData(self, species):
        #returns the dictionary of thermo data input for a given species name
        my_species = filter(lambda data: data['species'] == species, self.data)
        return my_species[0]

    def createThermoDatabase(self):
       #creates the lookup dictionary for species thermo data
       # format of list inputs:
       # ['Ni2+', dH =-54.0 kJ/mol, S = -128.9 J/mol-K, dG =-45.6 kJ/mol]
        print "Generating thermodynamic property dictionaries..."
        my_list = [
            ['H2O', -285.83, 69.91, -237.18],
            ['H+', 0., 0., 0.],
```

```
            ['Ni2+', -54.0, -128.9, -45.6],
            ['NiO', -239.7, 37.99, -211.7],
            ['Ni', 0., 29.87, 0.],
            ['Fe2+', -89.1, -137.7, -78.9],
            ['Fe3+', -48.5, -315.9, -4.7],
            ['Fe3O4', -1118.4, 146.4, -1015.5],
            ['Fe2O3', -824.2, 87.4, -742.2],
            ['Cr2O3', -1139.7, 81.2, -1058.1],
            ['Co2+', -58.2, -113.0, -54.4],
            ['Co3+', 92.0, -305.0, 134.0],
            ['Co', 0., 30.4, 0.],
            ['CoO', -237.94, 52.97, -214.22],
            #all data above this comment were found in "Principles of Modern Chemistry, ↩
                Fourth Edition"
            # Appendix D, "Standard Chemical Thermodynamic Properties"
            # Authors: Octoby, Gillis, Nachtrieb
            # ISBN 0-03-024427-7

            ['Cr2+', -143.5, -100.0, -146.0],
            ['Cr3+', -238.0, -317.0, -194.5],
            # chromium ion data derived from:
            #Dellien I., Hall F. M. and Hepler G. L. (1976) Chemical Reviews, 76, 283.
            #doi: 10.1021/cr60301a001

            # values for zr4+  from:
            #Paul Scherrer Institut, Authors Tres Thoenen, Enzo Curti, report #TM-44-14-04
            #Title: The PSI/Nagra Chemical Thermodynamic Database 12/07 (Update of the ↩
                Nagra/PSI TDB 01/01): Data Selection for Zirconium
            # date: 05/05/2014
            ['Zr4+', -608.5, 39.08, -528.5],

            #ZrO2 data from:
            # US Department of Commerce, National Bureau of Standards (NBS) Selected ↩
                Values of Chemical Thermodynamic Properties
            # Tables for Elements 54 through 61 in the Standard Order of Arrangement,  NBS↩
                 Technical Note 270-5
            # Nat. Bur. Stand. (U.S.), Tech. Note 270-5, 49 pages (Mar. 1971)
            ['ZrO2', -1100.56, 50.38, -1042.82]
        ]
        for sp in my_list:
            my_data = []
            for item in sp:
                my_data.append(item)
```

```
78              self.addThermoData(my_data)
79          #Generating thermodynamic property dictionaries...
80           print "\t\t\t\t\t\t\t\t\t\t\t\t ....DONE\n"
81
82      ####################
83
84      def addReaction(self, label, reactants, products, ion=None, oxide=None, x=None, b=None
            , electrons=0):
85          rxn = {
86                    'label': label,
87                    'reactants': reactants,
88                    'products': products,
89                    'e-':electrons,
90                    'ion':ion,
91                    'solid':oxide,
92                    'x':x,
93                    'b':b,
94          }
95          self.reactions.append(rxn)
96
97      def getReaction(self, reaction):
98          my_reaction = filter(lambda rxn: rxn['label'] == reaction, self.reactions)
99          return my_reaction[0]
100
101     def addRP(self, react, stoich):
102         # input format: ['NiO', 'H+'], [1, 2]
103         my_list = []
104         for i,j  in zip(react, stoich):
105             my_list.append({'name': i, 'coeff': j})
106         return my_list
107
108     def createReactionDatabase(self):
109         # a cumbersome, yet effective, method for creating the thermodynamic lookup
                dictionary for reactions of interest
110         print "Generating reactions database.....................\n"
111         #add 'NiO + 2H+ -> Ni2+ + H2O'
112         reactants = self.addRP(['NiO', 'H+'], [1, 2])
113         products = self.addRP(['Ni2+', 'H2O'], [1, 1])
114         self.addReaction('NiO + 2H+ -> Ni2+ + H2O', reactants , products, 'Ni2p', 'NiO',
                2, 1, electrons=0)
115
116         # add 'Ni2+ + 2e- -> Ni'
117         reactants = self.addRP(['Ni2+'],[1])
```

```
118         products = self.addRP(['Ni'],[1])
119         self.addReaction('Ni2+ + 2e- -> Ni', reactants, products,'Ni2p', 'Ni',0 , 1, ↩
                electrons=-2)
120 
121         #add Fe3O4 + 8H+ + 2e- -> 3Fe2+ + 4H2O
122         reactants = self.addRP(['Fe3O4', 'H+'], [1, 8])
123         products = self.addRP(['Fe2+', 'H2O'], [3, 4])
124         self.addReaction('Fe3O4 + 8H+ + 2e- -> 3Fe2+ + 4H2O', reactants, products,'Fe2p', ↩
                'Fe3O4', 8, 3, electrons=2)
125 
126         #add 'Fe2O3 + 6H+ + 2e- -> 2Fe2+ + 3H2O'
127         reactants = self.addRP(['Fe2O3', 'H+'],[1, 6])
128         products = self.addRP(['Fe2+', 'H2O'],[2, 3])
129         self.addReaction('Fe2O3 + 6H+ + 2e- -> 2Fe2+ + 3H2O', reactants, products,'Fe2p', ↩
                'Fe2O3', 6, 2, electrons=2)
130 
131         # add 'Fe2O3 + 6H+ -> 2Fe3+ + 3H2O
132         reactants = self.addRP(['Fe2O3', 'H+'],[1, 6])
133         products = self.addRP(['Fe3+', 'H2O'],[2, 3])
134         self.addReaction('Fe2O3 + 6H+ -> 2Fe3+ + 3H2O', reactants, products,'Fe3p', 'Fe2O3↩
                ', 6, 2, electrons=0)
135 
136         # add '3Fe3+ + 4H2O + e- -> Fe3O4 + 8H+'
137         reactants = self.addRP(['Fe3+', 'H2O'],[3, 4])
138         products = self.addRP(['Fe3O4', 'H+'],[1, 8])
139         self.addReaction('3Fe3+ + 4H2O + e- -> Fe3O4 + 8H+', reactants, products,'Fe3p', '↩
                Fe3O4', 8, 3, electrons=-1)
140 
141         #add 'Cr2O3 + 6H+ + 2e- -> 2Cr2+ + 3H2O'
142         reactants = self.addRP(['Cr2O3', 'H+'],[1, 6])
143         products = self.addRP(['Cr2+', 'H2O'],[2, 3])
144         self.addReaction('Cr2O3 + 6H+ + 2e- -> 2Cr2+ + 3H2O', reactants, products, 'Cr2p',↩
                 'Cr2O3', 6, 2, electrons= 2)
145 
146         # add 'Cr2O3 + 6H+ -> 2Cr3+ + 3H2O'
147         products = self.addRP(['Cr3+', 'H2O'],[2, 3])
148         self.addReaction('Cr2O3 + 6H+ -> 2Cr3+ + 3H2O', reactants, products, 'Cr3p', '↩
                Cr2O3', 6, 2, electrons=0)
149 
150         #add 'ZrO2 + 4H+ + Zr4+ + 2H2O'
151         reactants = self.addRP(['ZrO2', 'H+'], [1, 4])
152         products = self.addRP(['Zr4+', 'H2O'],[1, 2])
```

```
153         self.addReaction('ZrO2 + 4H+ -> Zr4+ + 2H2O', reactants, products, 'Zr4p', 'ZrO2',↩
                4, 1, electrons=0)
154 
155         # add 'Co2+ + 2e- -> Co'
156         reactants = self.addRP(['Co2+'], [1])
157         products = self.addRP(['Co'], [1])
158         self.addReaction('Co2+ + 2e- -> Co', reactants, products, 'Co2p', 'Co', 0, 1, ↩
                electrons=-2)
159 
160         # add 'CoO + 2H+ -> Co2+ + H2O'
161         reactants = self.addRP(['CoO', 'H+'], [1,2])
162         products = self.addRP(['Co2+', 'H2O'],[1, 1])
163         self.addReaction('CoO + 2H+ -> Co2+ + H2O', reactants, products, 'Co2p', 'CoO', 2,↩
                1, electrons=0)
164 
165         #add 'Co3+ + 3e- -> Co'
166         reactants = self.addRP(['Co3+'], [1])
167         products = self.addRP(['Co'],[1])
168         self.addReaction('Co3+ + 3e- -> Co', reactants, products, 'Co3p', 'Co', 0, 1, ↩
                electrons=-3)
169 
170         #add 'Co3+ + H2O + e- -> CoO + 2H+'
171         reactants = self.addRP(['Co3+', 'H2O'], [1, 1])
172         products = self.addRP(['CoO', 'H+'], [1,2])
173         self.addReaction('Co3+ + H2O + e- -> CoO + 2H+', reactants, products, 'Co3p', 'CoO↩
                ',2, 1, electrons=-1)
174 
175         print "\t\t\t\t\t\t\t\t\t\t\t\t ....DONE\n"
176 
177     def getH(self, my_species):
178         return my_species.get('H')
179 
180     def getS(self, my_species):
181         return my_species.get('S')
182 
183     def getG(self, my_species):
184         return my_species.get('G')
185 
186     def calculateEnthalpy(self, rxn):
187         rdata = 0.
188         pdata = 0.
189         for r, p in zip(rxn.get('reactants'), rxn.get('products')):
190             #get the enthalpies for each species
```

```
191             rdata = rdata + (r['coeff']) * self.getSpeciesData(r['name'])['H']
192             pdata = pdata + (p['coeff']) * self.getSpeciesData(p['name'])['H']
193         delta_H = pdata - rdata
194         if self.convertEnthalpy:
195             # convert units from kJ/mol to J/mol
196             delta_H = delta_H * 1000
197         return delta_H
198 
199     def calculateEntropy(self, rxn):
200         rdata = 0.
201         pdata = 0.
202         for r, p in zip(rxn.get('reactants'), rxn.get('products')):
203             # get the enthalpies for each species
204             rdata = rdata + (r['coeff']) * self.getSpeciesData(r['name'])['S']
205             pdata = pdata + (p['coeff']) * self.getSpeciesData(p['name'])['S']
206         delta_S = pdata - rdata
207         return delta_S
208 
209     def generate(self):
210         self.createThermoDatabase()
211         self.createReactionDatabase()
212         print "************************************************************"
213         print "Reactions Available:\n"
214         for r in self.reactions:
215             print r['label']
216             r['dH'] = self.calculateEnthalpy(r)
217             r['dS'] = self.calculateEntropy(r)
218         print "\nTo add additional reactions, modify the library code."
219         print "************************************************************\n\n"
```

Listing 4.30: `ChemicalThermodynamics` code

```
#  This code snippet is from the method generateFile()
#  reactions_included is set from the input parameters of the method
#  (see user input sample for input file generation script)

thermo = chem.ChemicalThermodynamics()          #initialize ChemicalThermodynamics

for reaction in reactions_included:             # loop over all reactions from user input
   #get dictionary with calculated thermodynamic parameters for reaction
    r = thermo.getReaction(reaction)

    #create instance of Reaction object, and set parameters from dictionary
    r_obj = Reaction(r['solid'], r['x'], r['e-'], r['b'], r['ion'], r['label'], r['dH'], r['dS'])
    reactions.append(r_obj)               #add  Reaction object to the list of reactions

 for rxn in reactions:              # loop over all Reaction objects
    rxn.setRxnName()             # sets name of reaction object from label

    for element in elements:                          # loop over all elements
        for oxide in element.oxides:                  # loop over all oxides
            if rxn.oxide == oxide.name:               # find oxide for this reaction
                for ion in element.ions:
                  if ion.name == rxn.ion:       # ensure reaction ion exists
                     oxide.addRxn(rxn)          #  link oxide to reaction object
                     print '\t\tReaction added to Oxide %s' % (oxide.name)
                     ion.reactions.append(rxn)     #  link ion to reaction object
                      print '\t\tReaction added to Ion %s\n\n' % (ion.name)
```

Listing 4.31: Code for linking oxide/ion/reactions and using `ChemicalThermodynamics` module output

Chapter 5

Results

5.1 Problem Setup

The test case used for validation and sensitivity studies was set up based largely upon the test case presented in Macdonald. General parameters can be found in Tables 5.1 and 5.2.

5.1.1 Simulation parameters

A primary loop with total length of 78m, core length of 4.587m, and steam generator length of 5.695m was used for all test cases, and each simulation represents a 12 month fuel cycle. While fuel cycles are typically 18 to 24 months long, 12 months was chosen for consistency with Macdonald et al. to more easily compare results. A piecewise constant function was used to input the temperature, electrochemical corrosion potential, hydraulic diameter, wetted area, and coolant velocity along the loop. The electrochemical corrosion potential profile for the loop is presented in Figure 5-1, and the temperature profile is presented in Figure 5-2. Using the built-in solvers from the MOOSE framework, a preconditioned Jacobian-Free Newton Krylov method was used to solve the coupled system, and an implicit midpoint time integration scheme was used. Tolerances for linear and nonlinear solve steps can be found in Table 5.1.

Loop length	78.0 m
Core length	4.587 m
Steam generator length	5.695 m
nx	78
ny	1
Finite element type	QUAD9
Solve type	PJFNK
Preconditioner	hypre boomerAMG
Time integration scheme	Implicit midpoint
End time	31536000 s (1 year)
Linear tolerance	1.0e-5
Linear max iterations	15
Nonlinear relative tolerance	1.0e-4
Nonlinear absolute tolerance	1.0e-6
Nonlinear max iterations	10

Table 5.1: Solve parameters

End (m)	**Region**	d_h **(m)**	**Velocity** (m/s)	**Wet area** (m^2)
0.854	Core	0.004	7.93	1400
3.416		0.004	5.06	4200
4.270		0.004	5.32	1400
4.587		0.011	5.00	235.2
8.137	HL	0.400	4.00	452.96
14.982		0.736	15.98	50
22.797		0.736	5.5	200
28.493	SG	0.0169	5.25	4050
35.767	CL	0.7874	5.12	200
43.178		0.7874	12.40	150
64.355		0.6985	15.73	100
78.124		0.5200	5.50	911.94

Table 5.2: Parameters

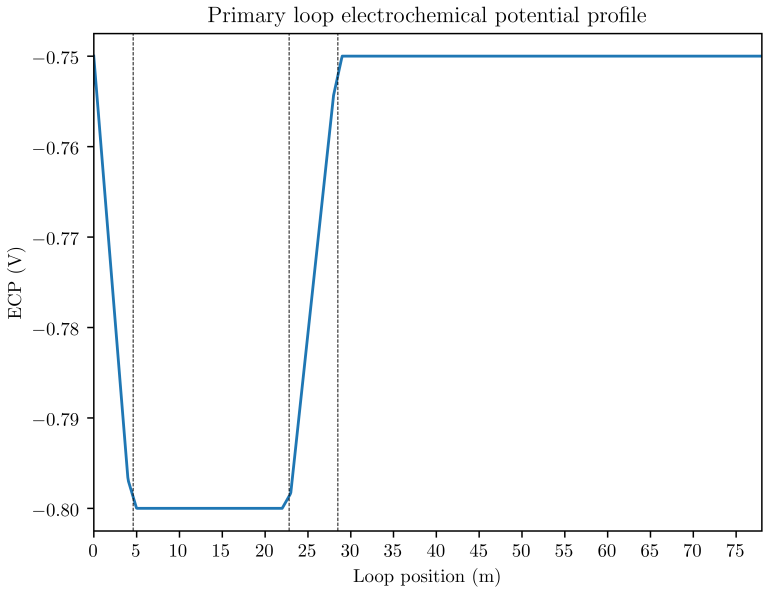

Figure 5-1: Electrochemical corrosion potential profile used for test case (based on Macdonald)

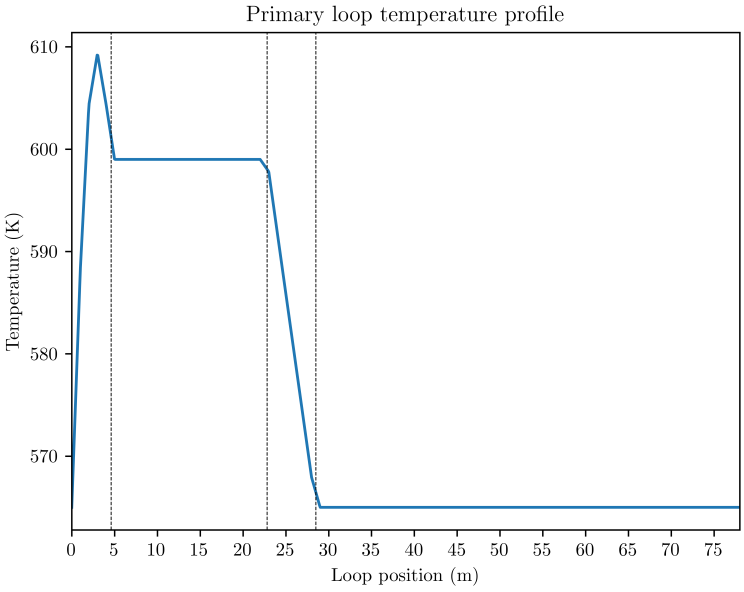

Figure 5-2: Temperature profile used for test case

5.1.2 Chemistry

The set of dissolution-precipitation reactions used in the test cases are presented in Table 5.3. To fully model the complex chemistry present in an actual plant, hundreds of reactions would be required. As such, these reactions are a representative selection to demonstrate the functionality of the code for multiple elements and oxides.

No.	Oxide	Ion	Reaction	ΔH_{rxn} (kJ/mol)	ΔS_{rxn} (J/mol-K)
1	Fe_3O_4	Fe^{2+}	$Fe_3O_4 + 8\,H^+ + 2\,e^- \rightarrow 3\,Fe^{2+} + 4\,H_2O$	-292.22	-279.86
2	Fe_2O_3	Fe^{2+}	$Fe_2O_3 + 6\,H^+ + 2\,e^- \rightarrow 2\,Fe^{2+} + 3\,H_2O$	-211.49	-153.07
6	NiO	Ni^{2+}	$NiO + 2\,H^+ \rightarrow Ni^{2+} + H_2O$	-100.13	-96.98
9	ZrO_2	Zr^{4+}	$ZrO_2 + 4\,H^+ \rightarrow Zr^{4+} + 2\,H_2O$	-79.6	128.52
11	CoO	Co^{2+}	$CoO + 2\,H^+ \rightarrow Co^{2+} + H_2O$	-106.09	-96.06

Table 5.3: Reactions considered for dissolution/precipitation

At each node, the following oxide concentrations are tracked:

$$Fe_3O_4, Fe_2O_3, NiO, ZrO_2, CoO$$

5.1.3 Corrosion growth rates

A value for the corrosion growth rate constant, k_p, for 304SS was provided by Castelli [23]. To make a rough approximation for other materials, it was assumed that the corrosion growth (oxidation of metal) is limited by the diffusion of oxygen into the base metal. For example, to adjust the rate for A600 and Zr4, the rate from 304SS was weighted by a ratio of the diffusivity of oxygen in the main element of the new alloy, to the diffusivity of oxygen in iron (the main constituent of 304SS). This would scale the corrosion oxide growth rate by some orders of magnitude to provide an approximate corrosion rate; future work should include the mechanisms of corrosion growth instead of an empirical rate constant. An intermediate improvement would be to use a measured or calculated corrosion growth rate constant for A600 and Zr4.

Assuming the oxidation growth rate is proportional to the diffusivity of $O_2{}^-$ in the base metal of construction materials:

$$k_P^{alloy} = \frac{D_{alloy}^{ox}}{D_{304ss}^{ox}} \times k_p^{304ss}$$

Diffusivities were computed using an Arrhenius relationship at a temperature of 582 K.

$$D = D_0 \exp\left(\frac{-E_a}{k_b T}\right)$$

Values used for prefactors, activation energies, and calculated diffusivities can be found in Table 5.4.

Alloy	D_0 (m^2/s)	E_A (eV)	Source	D_{calc} (m^2/s)	D_{alloy}/D_{304ss}	k_p^{mod}
304SS	2.91e-7	0.93	[35]	2.75e-15	1	1.16
A600	4.9e-6	1.7	[36]	9.41e-21	3.42e-6	3.97e-6
Zr-4	9.4e-6	2.267	[37]	2.23e-25	8.11e-11	9.41e-11

Table 5.4: Diffusivities for O_2^- in base metals; calculated using an Arrhenius relationship at a temperature of 582 K.

5.2 Validation and sensitivity studies

Validation was performed by replicating certain results presented by Macdonald et al. These include spatial surface concentration trends, temporal bulk coolant concentration evolution, temporal and spatial activity accumulation.

5.2.1 Surface concentration trends

Surface concentration is dependent upon local temperature, pH, electrochemical potential, and Gibbs energy values for each element's dissolution reaction. These results were determined assuming no local variation in pH, and a fixed electrochemical potential profile along the primary loop.

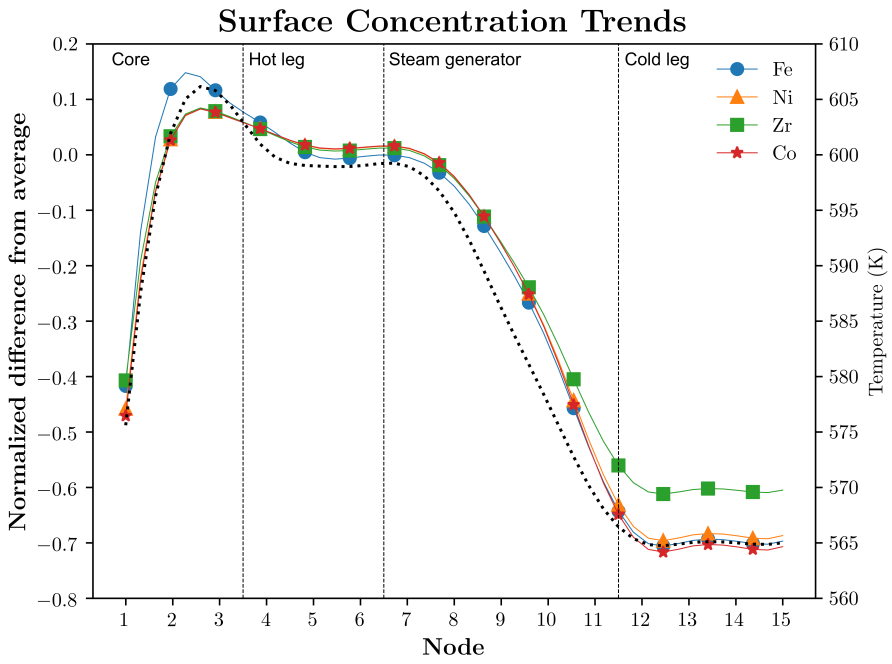

Figure 5-3: Normalized surface concentration trends along primary loop; the temperature dependence is well represented by all elements (nodal temperature profile is represented by the dashed line).

The surface concentration values presented in Fig. 5-3 were calculated by averaging the values over the elements whose positions correspond to those of the nodes described in Macdonald et al. Nodes 1-3 include the core, 4-6 include the hot leg, 7-11 include the

steam generator, and 12-15 are the cold leg. The temperature dependence for saturation of each element is apparent. Nodes in higher temperature regions (hot leg, nodes 4-7) have greater surface concentrations than the average along the primary loop, and lower surface concentrations in areas with lower temperatures (cold leg, nodes 12-15).

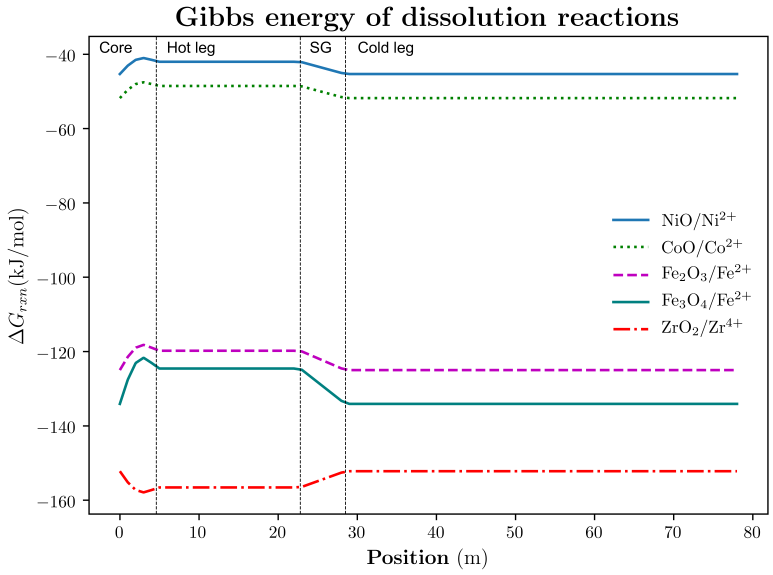

Figure 5-4: Gibbs energy values along loop for each reaction

The Gibbs energy for each reaction was calculated at each node as a function of the node temperature.

$$\Delta G_{rxn}(T) = \Delta H_{rxn} - T\Delta S_{rxn}$$

As the Gibbs energy scales linearly with temperature, the values calculated along the loop should yield approximately the same trend and shape as the temperature profile along the loop. Figure 5-4 presents the calculated Gibbs energies along the loop for each of the 5 chemical reactions represented. The trends for $NiO/Ni_2{}^+$, $CoO/Co_2{}^+$, $Fe_2O_3/Fe_2{}^+$, and $Fe_3O_4/Fe_2{}^+$ follow the same shape as the temperature profile, differing in magnitude due to the different entropy changes for each reaction. The Gibbs energy profile for $ZrO_2/Zr_4{}^+$ is inverse in the slopes; this is explained by the opposing sign on the reaction entropy as compared with the other reactions, which negates the slope in this linear relationship.

5.2.2 Isotope concentrations in bulk coolant

Temporal changes in bulk coolant concentrations of precursor isotopes during the first hours of the fuel cycle are presented in Figure 5-5. Concentrations for each isotope rise sharply during the first 3 hours. This increase is followed by a slight decrease as the coolant concentrations become saturated and converge to a stable value. This slight dip is not present in the results presented by Macdonald (Fig. 5-6); the concentrations here also converge to a stable coolant concentration much sooner than in those presented by Macdonald. These differences could be attributed to the additional corrosion release source term that is present in this model and not included in the Macdonald model.

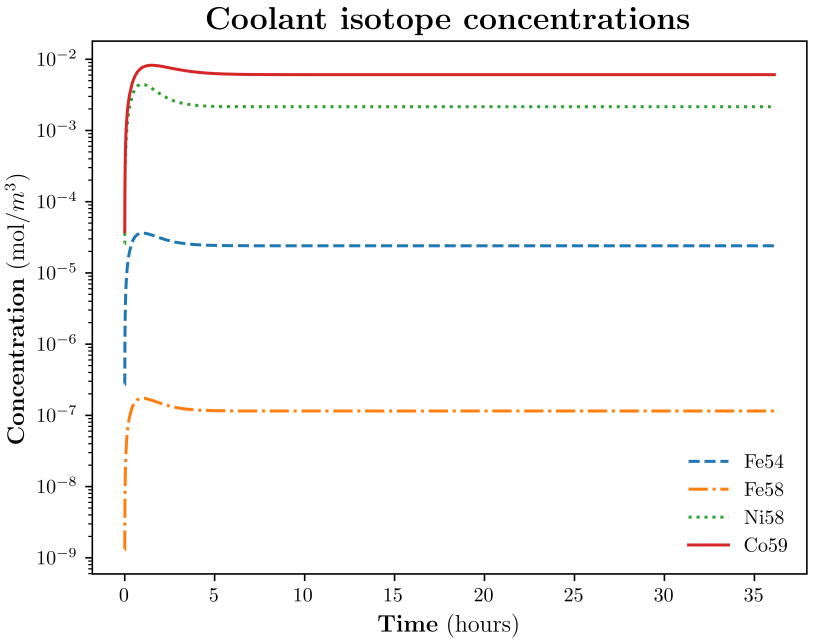

Figure 5-5: Bulk coolant isotope concentrations

Note the difference in magnitude between each precursor isotope; the ordering of these isotopes appear to correlate with the Gibbs energies of different element reactions. Gibbs energies decrease in magnitude in the following order: $NiO > CoO > Fe_2O_3 > Fe_3O_4 > ZrO_2$.

^{59}Co is the exception to this; the nickel reaction has a higher Gibbs energy, so one might anticipate a higher saturated concentration and bulk coolant concentration as a result. However, ^{59}Co differs from the other precursor isotopes in this system as there are 2 source

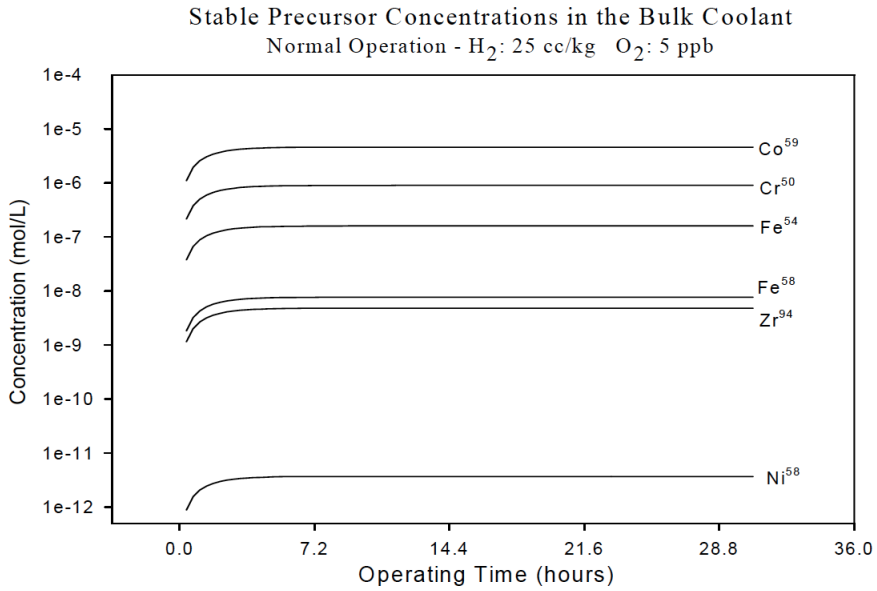

Figure 5-6: Bulk coolant isotope concentrations reported by Macdonald et al. [5]

terms. As with the other isotopes, there is a source term from corrosion release/dissolution. Dissolution for each isotope is proportional to the natural abundance of that isotope; ^{59}Co has a natural abundance of approximately 100%, while ^{58}Ni is approximately 68%. As such, the dissolution source term of ^{58}Ni is reduced. The additional source term for ^{59}Co is from the neutron capture reaction from ^{58}Co. The large capture cross section for ^{59}Co (1855 barns) makes this source term significant.

The bulk concentration of activated isotopes in the coolant, shown in Figure 5-7, shows similar trends. The concentrations rise sharply at the beginning of the fuel cycle before reaching a stable value after the first 10 to 20 hours. Note that the concentrations for activated isotopes are related to those of precursors; these activated isotopes reach stable values in fewer hours than the results presented by Macdonald (Fig. 5-8), and this could be attributed to the faster convergence of the precursor isotope concentrations from the additional source term.

The behavior of ^{58}Co during the first 15 hours of the fuel cycle does not follow the same trend as for other isotopes. The concentration drops to a seemingly converged value, then rises to a new converged value between hours 10 and 20. This could be a result of numerical stability and convergence issues from the coupling of variables in the system. If the residual

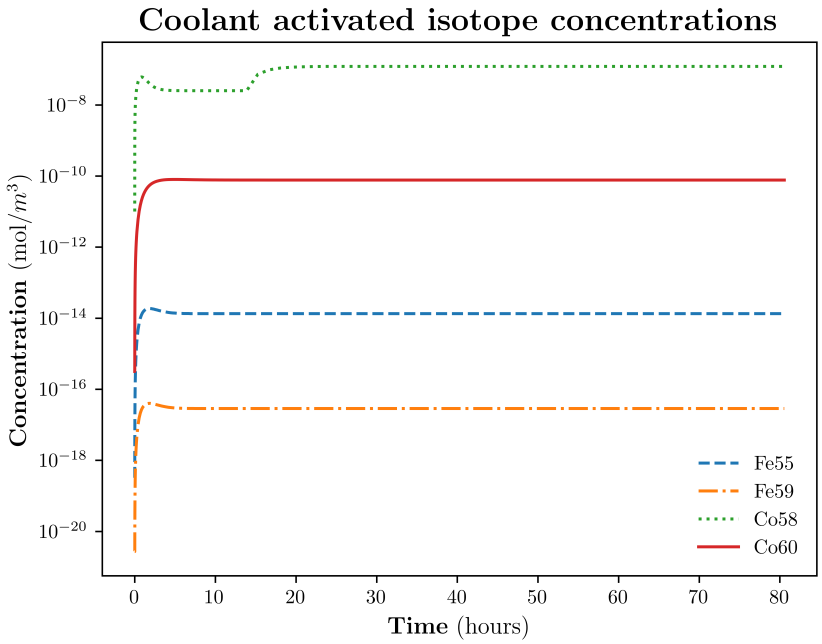

Figure 5-7: Bulk coolant activated isotope concentrations

for certain variables is orders of magnitude smaller than the largest variable residual, some variables can be "oversolved" as the numerical solver tries to reduce the residuals of all coupled variables in order to find a well-converged solution.

5.2.3 Spatial accumulated activity

Figure 5-9 presents the nodal accumulation of activity per activated isotope. Each isotope appears to follow a similar trend, with higher accumulation near nodes 6, 13, and 14, and the least accumulation in the core. Note that ^{55}Fe and ^{59}Fe accumulate activity in the core, while the remaining isotopes do not. One possible explanation for this behavior is that there are two oxides (and two dissolution reactions) present for the iron isotopes; the release rates for $Fe^{2+} \rightarrow Fe_2O_3$ are positive, indicating dissolution in the core, while $Fe^{2+} \rightarrow Fe_3O_4$ precipitates. The remaining isotopes have positive release rates within the core; the absence of oxide precipitation prevents the accumulation of activity within oxides.

Figure 5-11 presents the total accumulated activity per node at the end of one 12 month fuel cycle. Note the maximum activity accumulated at node 6 (end of hot leg) and node 14

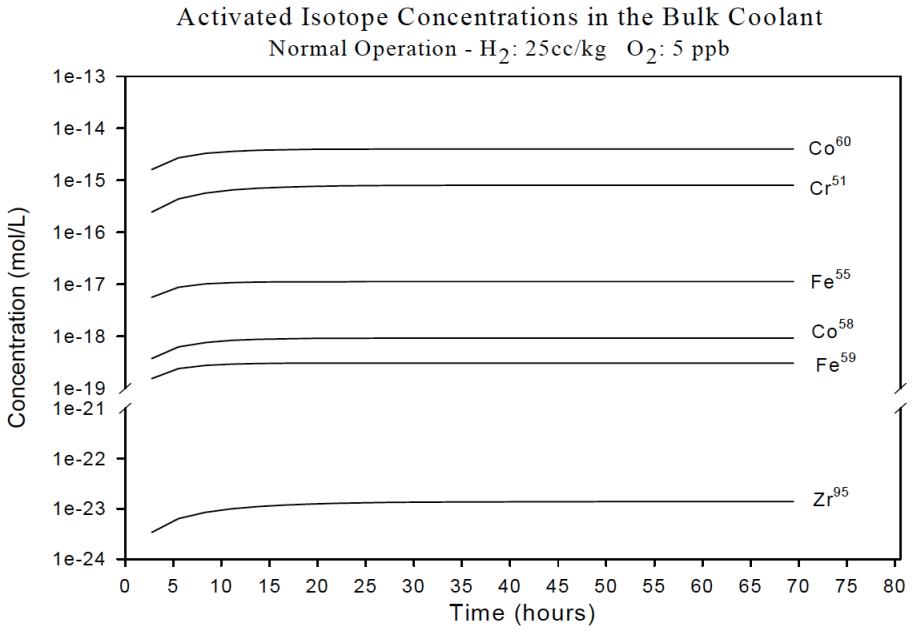

Figure 5-8: Bulk coolant activated isotope concentrations reported by Macdonald et al. [5]

(cold leg). The largest contributions to the total activity come from ^{58}Co.

Figures 5-12, 5-13, and 5-14 present the activity accumulation per element over the primary loop. Precipitation rates impact the activity accumulation, and are directly proportional to the saturated concentration at each point. Lower saturated concentrations drive precipitation, which would suggest that more activity would accumulate in regions with lower saturated concentrations, such as the cold leg. This is seen in for iron, cobalt, and zirconium as shown in Figures 5-12, 5-13, and 5-14. However, this does not account for the accumulation in the hot leg. The precipitation rate is also dependent on thermal hydraulic parameters (such as the Sherwood number, coolant velocity, etc.) and the wetted areas in each region. The hydraulic diameter and coolant velocity in the hot leg are greater than in the core and steam generator, which might explain this accumulation. The sharp increases and decreases can be explained by the use of piecewise functions to represent certain system properties in different regions along the primary loop.

The activity accumulation of iron isotopes (Fig. 5-12) shows some accumulation in the core, with a sharp decrease along the steam generator. Cobalt and zirconium, however, have little to no accumulation in the core. This is likely due to the absence of precipitation reactions in that region. Note the difference in the activity scales between these isotopes;

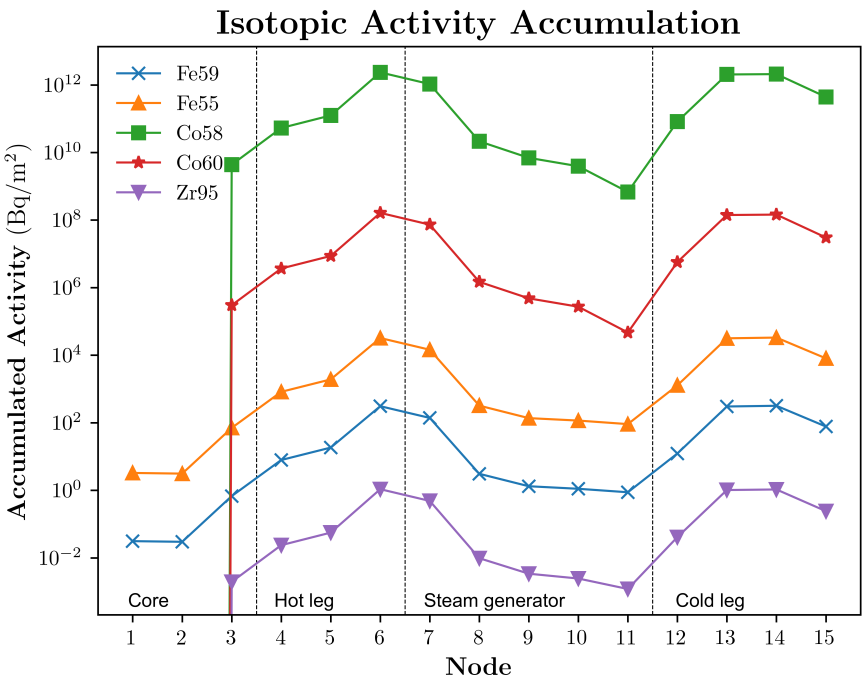

Figure 5-9: Accumulated activity per node per isotope

cobalt dominates, and zirconium activity buildup is negligible in comparison.

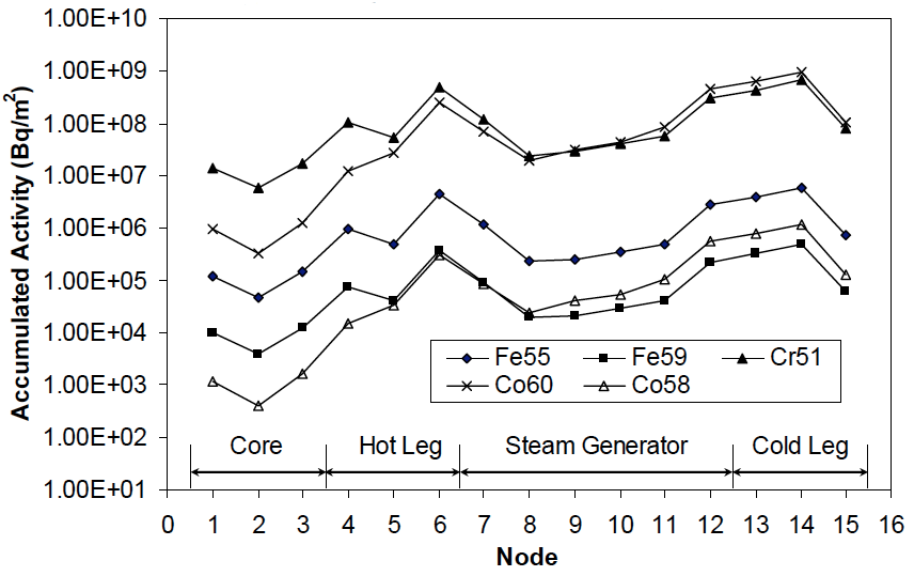

Figure 5-10: Accumulated activity per node per isotope reported by Macdonald et al. [5]

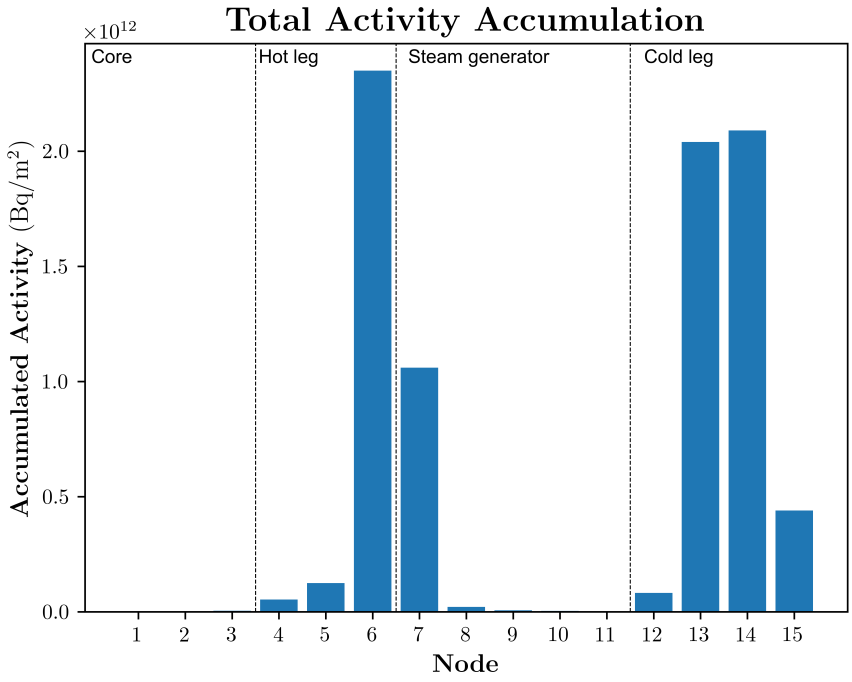

Figure 5-11: Total accumulated activity per node

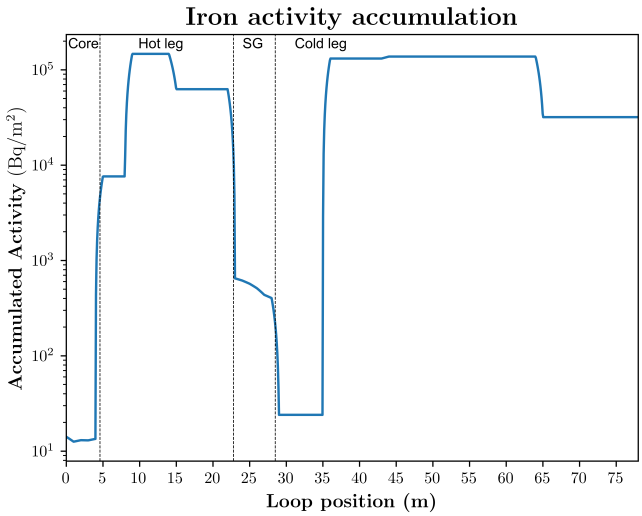

Figure 5-12: Spatial distribution of accumulated activity from iron isotopes

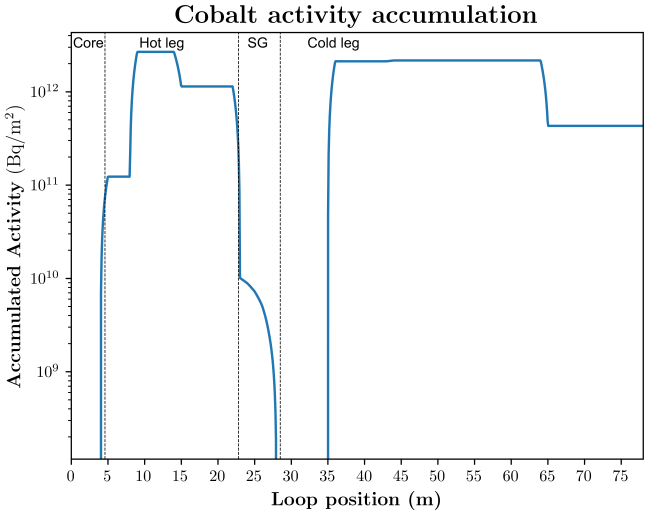

Figure 5-13: Spatial distribution of accumulated activity from cobalt isotopes

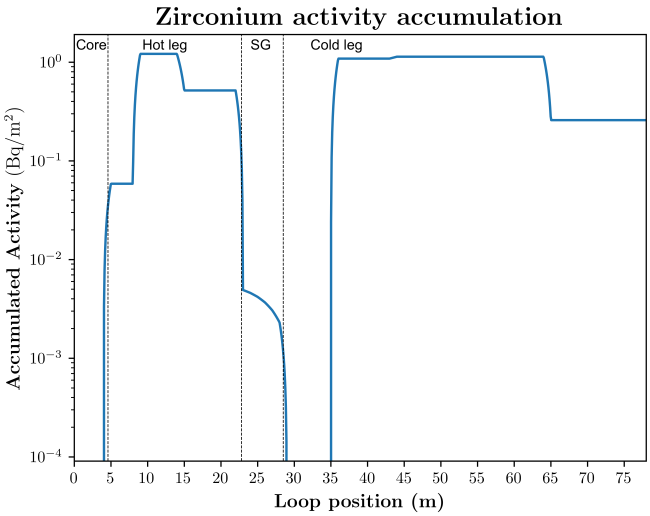

Figure 5-14: Spatial distribution of accumulated activity from zirconium isotopes

5.2.4 Temporal activity accumulation

Activity accumulation increases exponentially during the first month of the fuel cycle, then begins to level off to a nearly linear increase for the remainder of the fuel cycle. The difference in magnitude for each element is as would be expected, due to the significantly longer half life of cobalt.

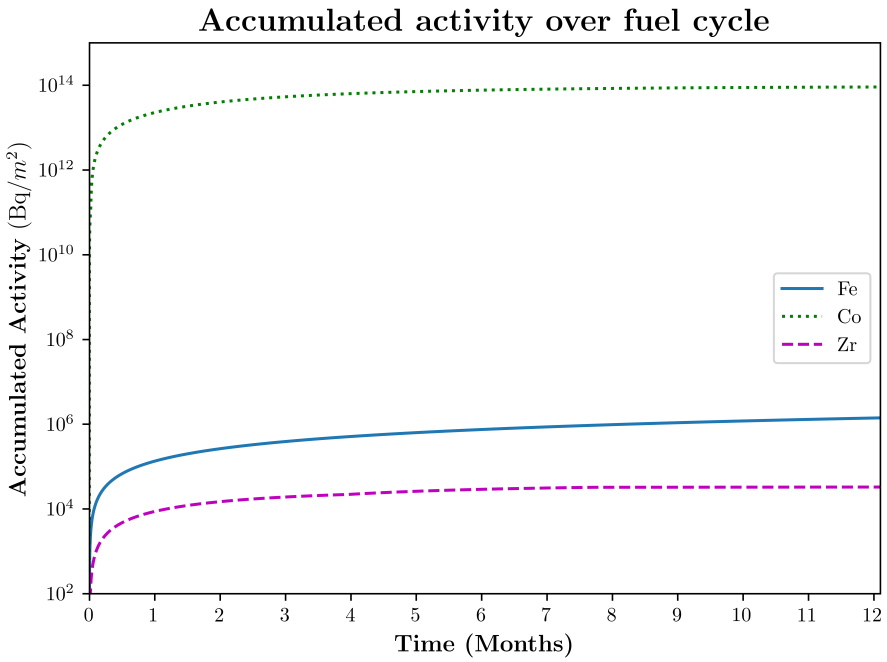

Figure 5-15: Activity accumulation over fuel cycle

5.3 Sensitivity studies

5.3.1 Activity accumulation pH sensitivity study

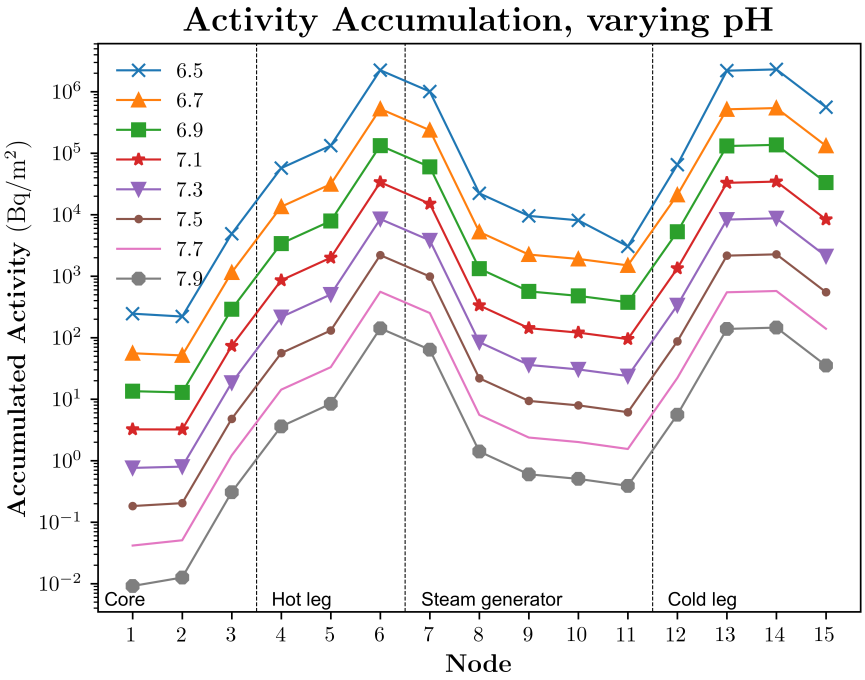

Figure 5-16: Iron accumulated activity per node with varied pH

Figures 5-16 and 5-17 present results of varying pH on activity accumulation for iron and cobalt. Each of these results show a decrease in activity accumulation with increasing pH.

The relationship between the saturated concentration and pH is $\propto 10^{-\mathrm{pH}}$. Increasing pH would cause the saturated concentration to decrease, which should then drive the precipitation rate. As the precipitation rate increases, activity accumulation should also increase. However, this does not account for the effect of pH and electrochemical corrosion potential (ECP), which has the same relationship with saturated concentration as pH, and could potentially impact this behavior significantly if it were included.

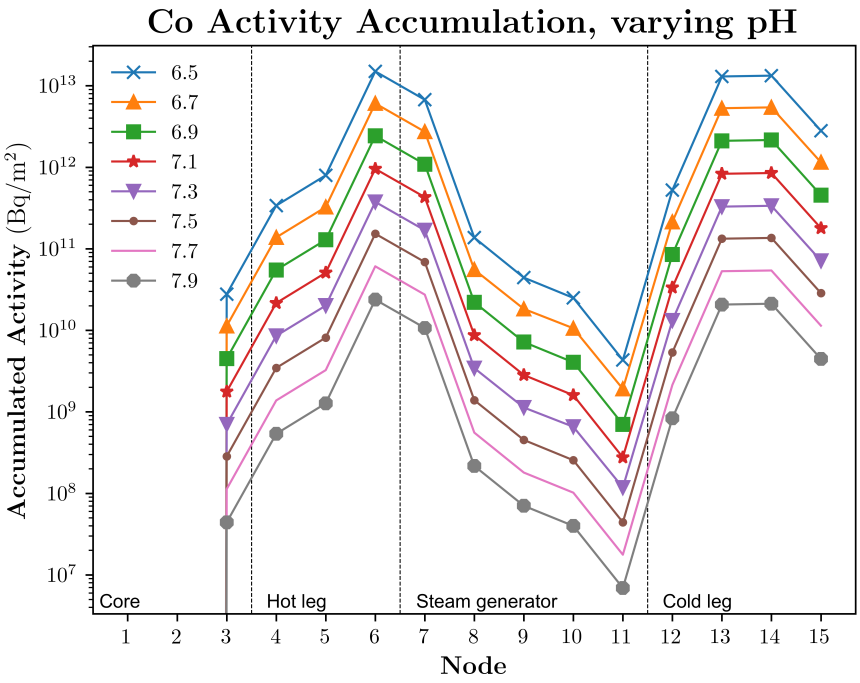

Figure 5-17: Cobalt accumulated activity per node with varied pH

5.3.2 Growth rate sensitivity study

To look at the impact of pH and of corrosion growth rates on temporal activity accumulation, a sensitivity study was performed. Corrosion growth rates were selected to obtain particular oxide thicknesses in the absence of dissolution or precipitation reactions; the values and thicknesses are presented in Table 5.3.2.

Element	20μm	40μm	60μm	80μm	100μm
Fe	6.6	9.0	11.01	12.7	14.1
Ni/Co	4.3	6.0	7.4	8.5	9.55
Zr	2.7	5.3	7.8	10.3	13.1

Table 5.5: Corrosion rate constants used to obtain fixed thicknesses of oxide growth from corrosion

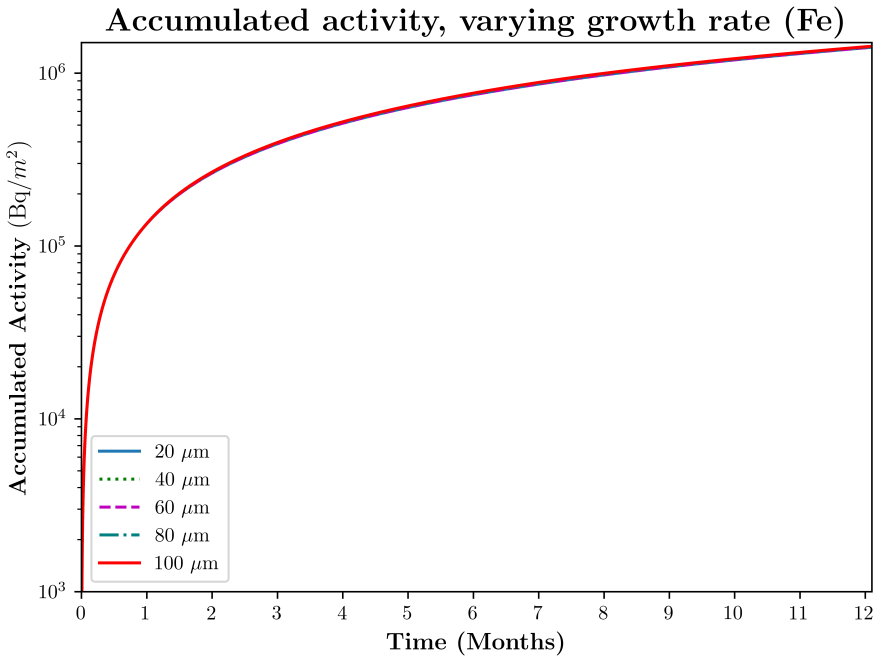

Figure 5-18: Accumulated activity, varying corrosion growth rates of iron, full fuel cycle

Figure 5-18 presents the accumulated activity over a full fuel cycle for each growth rate of iron. At this scale, the activity accumulation appears to be identical for each. Figure 5-19 presents a zoomed in view of these results over months 6 to 8. This view clearly shows some differences between each growth rate, with increasing activity accumulation as growth rate increases, as would be expected. Higher growth rates should provide a larger source term

for elements, which could yield higher precipitation as the coolant concentration saturates and excess material precipitates, and also a higher concentration of activated species in the coolant.

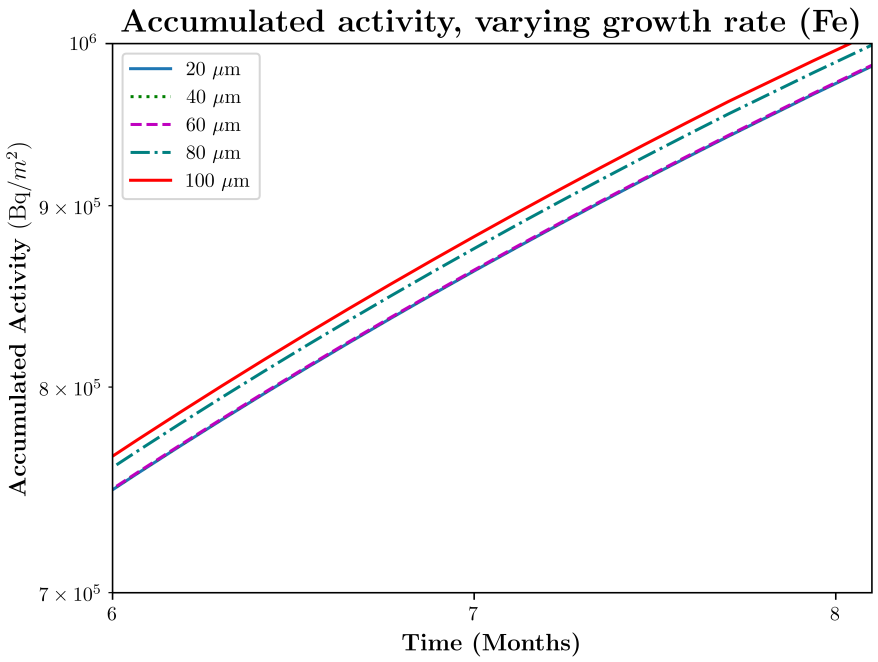

Figure 5-19: Accumulated activity, varying corrosion growth rates of iron, over two months

Figures 5-20 and 5-21 present the same results for nickel growth rates (cobalt is included in these results, but cobalt growth rate was not changed). Figure 5-20 is similar to 5-18, as the difference in accumulation cannot be seen on the long time scale. Figure 5-21 does not show any clear trend in activity accumulation from cobalt with changing nickel growth rate. A possible explanation for this is that the growth rate for nickel source is changed without changing the cobalt source terms; as the majority of activity accumulation does come from ^{58}Co, increasing the ^{58}Ni should increase activity accumulation. However, increasing ^{58}Co can also increase ^{59}Co as more activation occurs. Activity is highly sensitive to the decay parameter of the activated isotope; since ^{59}Co is stable, an increase in the activation of ^{58}Co to ^{59}Co could lead to less activity accumulation. Future studies to investigate this result could investigate the impact of varying nickel growth rate without including the cobalt source terms.

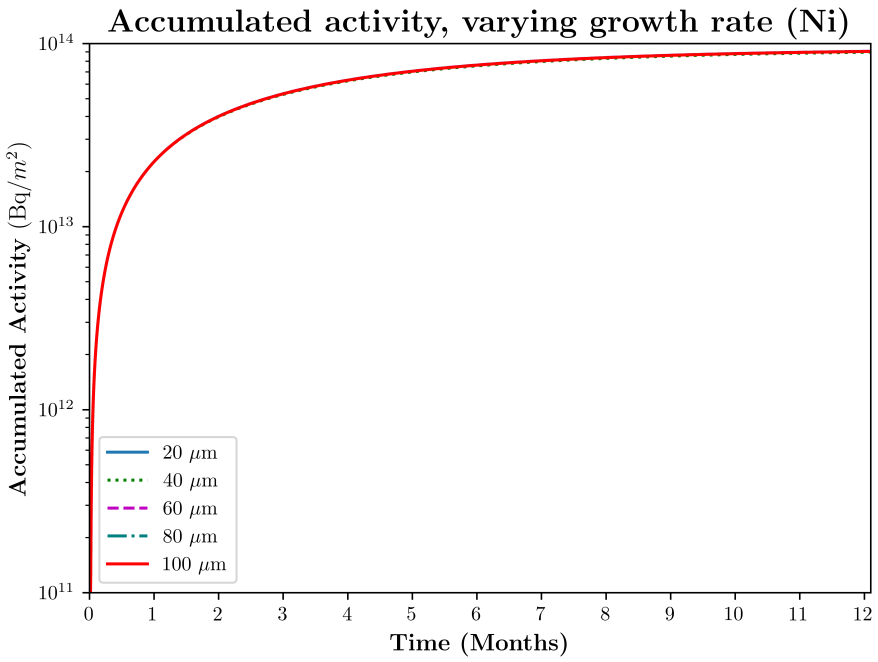

Figure 5-20: Accumulated activity from cobalt, varying corrosion growth rates of nickel

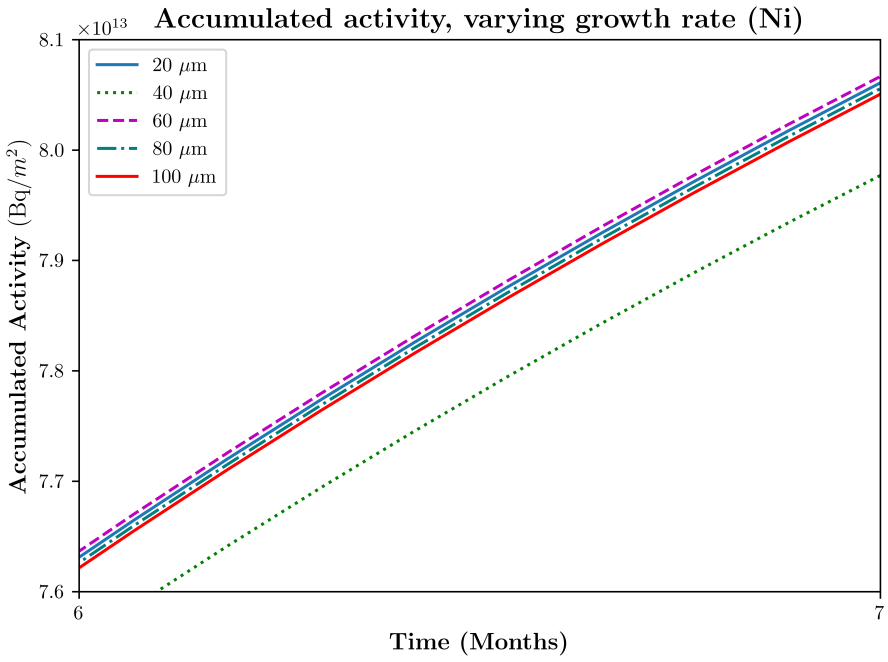

Figure 5-21: Accumulated activity from cobalt, varying corrosion growth rates of nickel, over one month

Chapter 6

Conclusions and Future Work

As stated in the introduction of this thesis, the objective of this work was to develop a code that can predict the concentration of active isotopes deposited throughout the primary loop over time and varied operating parameters.

This code implements many components of the mechanistic model for activity transport by Macdonald et al. [5]. Mechanisms for dissolution and precipitation of corrosion products are represented, with an emphasis on the saturated coolant concentrations as a driving force for these processes. Calculation of Gibbs energies for dissolution reactions are temperature-dependent and calculated upon each node, instead of using constant values from experimental measurements. While the detailed electrochemical potential and pH calculation models used by Macdonald are not implemented, an additional corrosion growth/release source term was added based upon the modeling effort of Castelli [23]. Buildup of crud in the core (from subcooled nucleate boiling) and crud particulate activation/release is not explicitly included in this model; however, existing models for crud growth and chemistry (such as MAMBA-BDM) can be coupled to this source term code using the MOOSE `MultiApp`/`Transfer` system.

6.1 Implications of this work

The main result of this work is an open source activity transport code, which is available freely for future improvements and adaptations to implement additional mechanisms

and more rigorous models. This code can be easily coupled to other multiphysics codes - such as thermal hydraulic, crud microstructure/fluid interactions, and neutronics - using the MOOSE `MultiApp` system. The code is capable of providing spatial (nodal) resolution of local activity and oxide concentrations/compositions due to dissolution/precipitation and corrosion growth.

The model implemented is predominantly mechanistic, and the coupled stiff equation set is solved using implicit numerical methods instead of explicit to allow faster, more stable solves. Implementation of this model using the MOOSE framework, with modular code structures, makes it simple to add new physics couplings, make more rigorous assumptions, and investigate the effects of each piece easily by adding or removing kernels from a simulation.

This work also provides a "user-friendly" input file generation method that includes computation of thermodynamic data for reactions desired, and ensures that all required pieces of physics and variables are linked properly for each solve. Users need not write input files on the order of 1500 lines by hand, reducing the time required to set up a new simulation case.

By using a mainly mechanistic model, the main assumptions made do not require operating parameters/geometry to be limited according to a plant that has already been studied. As such, physics-informed designs for plants, looking at different operating parameters and materials/geometries, can be used to reduce dose rates for primary-side workers.

6.2 Limitations in scope

As the system considered in this model is large and complex, many assumptions had to be made in order to begin with a preliminary model. These include the assumption of constant pH and electrochemical potential, loop temperature profile, and neutron flux profile in the core. These assumptions, however, can be easily removed in the future by simply adding the appropriate physics as MOOSE `Kernels` to vary those parameters temporally.

Additionally, the model is not fully mechanistic; empirical corrosion growth rates were used in implementing the corrosion growth and release source term from the Castelli model.

Gibbs energies of dissolution reactions are calculated from thermodynamic data that is inherently difficult to determine/measure, and any assumptions made in determining those values (particularly from experimental results) are carried into this model. As better thermodynamic data becomes available, these uncertainties and limitations can be reduced. The results presented here use a small selection of dissolution/precipitation reactions to demonstrate the functionality of the code. There are many more reactions that should be added to more closely represent a real system.

This model is not fully implicit; there is some explicit behavior from the coupling between the global ODEs and nodal calculations, which uses postprocessor values (to aggregate nodal values along the whole loop and obtain a scalar value). The postprocessor value is calculated on the beginning or end of a timestep or upon each linear or nonlinear solver iteration, and therefore is not fully implicit when used as a temporal rate of change in the global ODEs, introducing some instability in the model.

Validation data for activity buildup can be difficult to come by, as plant measurements of this data are typically proprietary. At the time of this thesis, a publicly accessible dataset for primary loop activity measurements and crud chemistry in operational PWRs was not found. A dataset similar to the BEAVRS benchmark (Benchmark for Evaluation And Validation of Reactor Simulations) with plant data for crud/oxide chemistry, activity buildup, etc. would be instrumental towards future development of crud source term models and activity transport codes [38]. With such a dataset, model predictions could be validated against actual plant data, and a mechanistic approach could be used to "tune" the code and correct the underlying physics represented in the models and more closely represent the true physics occurring in real PWRs as models are developed.

6.3 Future Work

Throughout this work, many assumptions and simplifications were necessary in order to develop this preliminary model. As such, there is plenty of opportunity for future improvements to the model and the code. While there are many possibilities, a few suggestions are listed here which could significantly improve the assumptions made in this model.

Possible future directions and improvements:

- Implementation of detailed water chemistry models for calculating pH and electrochemical corrosion potential
- Adding more precise thermodynamic data for Gibbs energy calculations
- Coupling with subchannel thermal hydraulics codes for thermal hydraulic parameters, feedback
- Coupling with neutronics codes for neutron flux profiles, neutron capture cross sections
- Coupling with crud chemistry codes (MAMBA-BDM) to implement localized dissolution/precipitation from transients of fluid flow through crud
- Implementation of multiple activation and decay chains, fast neutron capture, etc.
- Implementation of a larger set of dissolution reactions
- Implementation of a two-layer crud microstructure, with an outer and inner oxide
- Addition of particulates to the crud source term

6.4 Concluding thoughts

In the words of Castelli, "With so much uncertainty in our ability to describe these fundamental source terms, one wonders how or why anyone would choose to proceed from this point. The answer is quite clear, at least to me. Modeling of these phenomena must start somewhere, and even if the initial set of data is somewhat flawed, it will always be possible to improve it, in time, as new investigations are performed in the future" [23].

The code developed and described in this thesis is capable of predicting activated isotope deposition throughout the primary loop with spatial and temporal resolution, without the constraint of a particular plant's operating parameters or geometries, to obtain reasonable estimates of loop activity.

As codes for activity transport continue to implement additional mechanistic- based physics models, the mechanisms that govern activity transport will be better understood. While there is much future work to be done, this code is the first step towards a long term effort to develop an open source, fully mechanistic model which includes all mechanisms for activity transport in PWRs.

Bibliography

[1] Key world energy statistics, international energy agency. "http://www.iea.org/publications/freepublications/publication/KeyWorld2017.pdf", 2017.

[2] International atomic energy agency, nuclear power reactors in the world: Reference data series no. 2. "https://www-pub.iaea.org/MTCD/Publications/PDF/RDS_2-37_web.pdf, 2017.

[3] Tennessee Valley Authority. Pressurized water reactor schematic. "https://commons.wikimedia.org/wiki/File:PWR_nuclear_power_plant_diagram.svg", Accessed April 12 2018.

[4] Jiaxin Chen. On the interaction between fuel crud and water chemistry in nuclear power plants. *Swedish Nuclear Power Inspectorate (SKI) Report*, 00:5(SKI Project Number 97252, ISSN 1104-1374), 2000.

[5] Digby D. Macdonald, Mirna Urquidi-Macdonald, John H. Mahaffy, Amit Jain, Han Sang Kim, Vishisht Gupta, and Jonathan Pitt. Electrochemistry of water-cooled nuclear reactors. *Nuclear Energy Education Research (NEER) Final Technical Progress Report*, 2006.

[6] Iva Betova, Martin Bojinov, and Timo Saario. Start-up and shut-down water chemistries in pressurized water reactors. *VTT Research Report, No VTT-R-00699-12*, 2012.

[7] ENDF/B-VII.1 Boron-10. "http://www.nndc.bnl.gov/sigma/getDataset.jsp?evalid=15516".

[8] M.P. Short, D. Hussey, B.K. Kendrick, T.M. Besmann, C.R. Stanek, and S. Yip. Multiphysics modeling of porous CRUD deposits in nuclear reactors. *Journal of Nuclear Materials*, 443(1-3):579–587, 2013.

[9] ENDF/B-VII.1 Iron-54. "http://www.nndc.bnl.gov/exfor/servlet/E4sGetEvaluation?EvalID=25307&req=19786".

[10] ENDF/B-VII.1 Iron-58, national nuclear data center (NNDC) sigma, brookhaven national laboratory. "http://www.nndc.bnl.gov/exfor/servlet/E4sGetEvaluation?EvalID=25311&req=19787".

[11] ENDF/B-VII.1 Chromium-50, national nuclear data center (NNDC) sigma, brookhaven national laboratory. "http://www.nndc.bnl.gov/exfor/servlet/E4sGetEvaluation?EvalID=25300&req=19789".

[12] ENDF/B-VII.1 Cobalt-59, national nuclear data center (NNDC) sigma, brookhaven national laboratory. "http://www.nndc.bnl.gov/sigma/index.jsp?as=59&lib=endfb7.1&nsub=10".

[13] ENDF/B-VII.1 Cobalt-58, national nuclear data center (NNDC) sigma, brookhaven national laboratory. "http://www.nndc.bnl.gov/sigma/getSection.jsp?evalid=15022&mf=1&mt=451".

[14] ENDF/B-VII.1 Zirconium-94, national nuclear data center (NNDC) sigma, brookhaven national laboratory. "http://www.nndc.bnl.gov/exfor/servlet/X4sShowData?db=e4&op=get_e4&req=19791&ii=157&File=E4R19791_e4.txt".

[15] Chart of nuclides, national nuclear data center (NNDC) sigma, brookhaven national laboratory. "https://www.nndc.bnl.gov/chart/chartNuc.jsp".

[16] K.J.R. Rosman and P.D.P. Taylor. Isotopic composition of the elements 1997 (technical report). *Pure Appl. Chem.*, 70(1):217–235, 1998.

[17] 10 CFR 20, standards for protection against radiation. "https://www.nrc.gov/reading-rm/doc-collections/cfr/part020/index.html".

[18] G.R. Hoenes, M.A. Mueller, and W.D. McCormack. Radiological assessment of steam generator removal and replacement: update and revision (nureg/cr-1595), 1980.

[19] US EPA Office of Radiation Programs; ORP/TAD-79-11. Summary of occupational radiation exposure at nuclear power plants 1969 through 1977, 1977.

[20] Derek Gaston, Chris Newman, Glen Hansen, and Damien Lebrun-Grandié. MOOSE: A parallel computational framework for coupled systems of nonlinear equations. *Nuclear Engineering and Design*, 239(10):1768–1778, 2009.

[21] Derek R. Gaston, Cody J. Permann, John W. Peterson, Andrew E. Slaughter, David Andrš, Yaqi Wang, Michael P. Short, Danielle M. Perez, Michael R. Tonks, Javier Ortensi, Ling Zou, and Richard C. Martineau. Physics-based multiscale coupling for full core nuclear reactor simulation. *Annals of Nuclear Energy*, 84:45–54, 2015.

[22] S. Anthoni et al. Modelling of transport of radioactive substances in the primary circuit of water cooled reactors. *Technical report, International Atomic Energy Agency (IAEA)*, (IAEA-TECDOC-1672), 2012.

[23] Roy Castelli. *Nuclear Corrosion Modeling: The Nature of CRUD*. Butterworth-Heinemann, 2009.

[24] Mirza et al. Simulation of corrosion product activity in pressurized water reactors under flow rate transients. *Annals of Nuclear Energy*, 25:331 – 345, 1998.

[25] N. Mirza et al. Computer simulation of corrosion product activity in primary coolants of a typical pwr under flow rate transients and linearly accelerating corrosion. *Annals of Nuclear Energy*, 30:831–851, 2003.

[26] Rubina Nasir, Sikander M. Mirza, and Nasir M. Mirza. Evaluation of corrosion product activity in a typical PWR with extended cycles and flow rate perturbations. *World Journal of Nuclear Science and Technology*, 07(01):24–34, 2017.

[27] K.A. Burrill and P. Menut. A description of the activity transport computer codes in the IAEA benchmarking exercise. In *Water Chemistry of Nuclear Reactor Systems 8*, pages 519 – 526, Bournemouth, UK, 2000.

[28] K. Dinov. Modeling of activity transport in PWR by computer code MIGA. In *1st meeting of the IAEA Coordinated Research Program on Activity Transport Modeling*, Toronto, Canada, 1997.

[29] K. Dinov et al. Modeling of VVER light water reactors activity buildup. In *ICONE 8: Proceedings of the Eighth International Conference on Nuclear Engineering*, Baltimore, USA, 2000.

[30] D. Tarabelli et al. Status and future plans of the PACTOLE code predicting the activation and transport of corrosion products in PWRs. In *Proceedings of 1998 JAIF International Conference on Water Chemistry in Nuclear Power Plants*, pages 301 – 305, Kashiwakazi, Japan, 1998.

[31] Maria Cristina Annesini, Luigi Marrelli, Vincenzo Piemonte, and Luca Turchetti. *Artificial Organ Engineering*. Springer-Verlag, 2016.

[32] MOOSE framework website. "http://mooseframework.org/moose/", Accessed April 20 2018.

[33] M.P. Short. The particulate nature of the crud source term in light water reactors. *Journal of Nuclear Materials*, 2018.

[34] Peter Atkins and Julio de Paula. *Physical Chemistry Volume 1: Thermodynamics and Kinetics*. W. H. Freeman, 2010.

[35] Jun Takada and Masao Adachi. Determination of diffusion coefficient of oxygen in alpha-iron from internal oxidation measurements in fe-si alloys. *Journal of Materials Science*, 21(6):2133–2137, 1986.

[36] Jong-Wan Park and Carl J. Altstetter. The diffusion and solubility of oxygen in solid nickel. *Metallurgical Transactions A*, 18(1):43–50, 1987.

[37] J. Paul Pemsler. Diffusion of oxygen in zirconium and its relation to oxidation and corrosion. *Journal of The Electrochemical Society*, 105(6):315, 1958.

[38] N. Horelik, B. Herman, B. Forget, and K. Smith. Benchmark for evaluation and validation of reactor simulations (BEAVRS), v1.0.1. *Proc. Int. Conf. Mathematics and Computational Methods Applied to Nuc. Sci. and Eng., 2013. Sun Valley, Idaho*.

Appendix A

Typical PWR Parameters

Fuel Rod Parameter	**Symbol**	**Value**	**Units**
Pellet percent of theoretical density		95	%
Rod-to-rod pitch	P	12.6	mm
Fuel rod outside diameter	D	9.5	mm
Cladding thickness	t_{clad}	0.572	mm
Fuel-cladding gap (cold)	t_{gap}	0.0826	mm
Fuel pellet diameter	D_f	8.192	mm
Fuel pellet length	L_f	9.8	mm
Total fuel rod height		3.876	m
Heated fuel height	L	3.658	m
Percent of energy deposited in fuel rods		97.4	%
Peak Linear Heat Generation Rate (LHGR)	q'_0	44.62	kW/m
Core average LHGR	$< q' >$	17.86	kW/m
Core average subchannel flow rate (interior)	$\dot{m}_{cl}$	0.335	kg/s
Core average subchannel flow rate (edge)	$\dot{m}_{cl}$	0.159	kg/s
Core average subchannel flow rate (corner)	$\dot{m}_{cl}$	0.0759	kg/s
Subchannel flow area (interior)	A_{fch}	8.79 x 10^{-5}	m^2
Subchannel flow area (edge)	A_{fch}	4.27 x 10^{-5}	m^2
Subchannel flow area (corner)	A_{fch}	2.07 x 10^{-5}	m^2
Core average subchannel mass flux (interior)	G_{ch}	3807	$\frac{kg}{m^2 s}$
Core average subchannel mass flux (edge)	G_{ch}	3734	$\frac{kg}{m^2 s}$
Core average subchannel mass flux (corner)	G_{ch}	3661	$\frac{kg}{m^2 s}$

Table A.1: Typical PWR fuel rod parameters - based on Seabrook Station Reactor (from Todreas & Kazimi, Appendix K)

Parameter	Symbol	Value	Units
Thermal power	$\dot{Q}_{th}$	3411	MWth
Net electric power	$\dot{Q}_e$	1148	MWe
Efficiency	η	33.7	%
Nominal pressure	p	15.51	MPa
Total core pressure drop	Δp_{core}	0.197	MPa
Core inlet temperature	T_{in}	293.1	°C
Core exit temperature	T_{exit}	326.8	°C
Core coolant flow rate	$\dot{m}_{core}$	17476	kg/s
Number of assemblies	N_a	193	
Active core equiv diameter		3.37	m
Coolant mass in primary circuit		354	metric tons
Fuel enrichment (initial core)	r	1.6/2.4/3.1	%
Fuel enrichment (reloads)	r	3.1/3.4/4.2	%
Number of loops		4	
Cycle length		12	months
Average discharge burnup		33000	MWd/tU
Fuel inventory		89	tHM
Fuel inventory		101	t(UO2)
Average core power density		104.5	kW_{th}/L
Average core specific power		38.3	kW_{th}/kg_{HM}
Configuration		17x17	
Fuel rods per assembly	N_{rods}	264	
Channel width	l_{ch}	214.0	mm
Assembly pitch	l	215.0	mm
Core average flow rate per assembly	$\dot{m}_a$	89.8	kg/s
Core average assembly mass flux	G_a	3675.4	$\frac{kg}{m^2 s}$

Table A.2: Typical PWR Design Parameters - based on Seabrook Station Reactor (from Todreas & Kazimi, Appendix K)

Appendix B

Code snippets (Object classes)

B.1 `Ion` class

```
class Ion(object):
   """ An ion used in ion-oxide chemical reactions

   ATTRIBUTES:
   self.name: name of ion (string)
   self.reactions: list of Reaction objects for reactions including this ion
   self.diffusivity: name of diffusivity variable for this ion (string)
   self.diff_prefactor: value of diffusivity prefactor for this ion (float)
   self.kc: name of mass transfer coefficient variable for this ion (string)
   self.schmidt: name of Schmidt number variable for this ion (string)
   self.sherwood: name of Sherwood number variable for this ion (string)
   """

    def __init__(self, name):
        self.name = name
        self.reactions = []

    def setDiffusivity(self, myname):
        self.diffusivity = myname

    def setDiffPrefactor(self, myvalue):
        self.diff_prefactor = myvalue

    def setKC(self, myname):
```

```
25          self.kc = myname
26
27      def setSchmidt(self, myname):
28          self.schmidt = myname
29
30      def setSherwood(self, myname):
31          self.sherwood = myname
```

Listing B.1: `Ion` object class

B.2 `Oxide` class

```
class Oxide(object):
    """    An oxide used in ion-oxide chemical reactions

      ATTRIBUTES:
   self.name: name of oxide (string)
   self.reactions: list of Reaction objects for reactions including this oxide
   self.molar_mass: molar mass of oxide (float)
   self.ox_conc: name of oxide concentration variable (string)
    """

    def __init__(self, name, molar_mass):
        self.name = name
        self.reactions = []
        self.molar_mass = molar_mass

    def addRxn(self, rxn):
        self.reactions.append(rxn)

    def setOxideConcentration(self, myname):
        self.ox_conc = myname
```

Listing B.2: `Oxide` object class

B.3 `Reaction` class

```
class Reaction(object):
    """ A chemical reaction between oxides and ions

      ATTRIBUTES:
      self.oxide: name of oxide in this reaction (string)
      self.x: stoichiometry coefficient for H+
      self.z: number of electrons
      self.b: stoichiometry coefficient for ion
      self.ion: name of ion in this reaction (string)
      self.label: chemical reaction for this object (string)
      self.dH: Enthalpy change of reaction
      self.dS: Entropy change of reaction
      self.surfconc: name of surface concentration variable (string)
      self.preciprate: name of precipitation rate variable (string)
      self.dissolrate: name of dissolution rate variable (string)
      self.releaserate: name of release rate variable (string)
      self.oxide_obj: Oxide object for oxide in this reaction
      self.name: string to represent reaction name (ion and oxide names)
    """

    def __init__(self, oxide, x, z, b, ion, label, dH, dS):

        self.oxide = oxide
        self.x = x
        self.z = z
        self.b = b
        self.ion = ion
        self.label = label
        self.dH = dH
        self.dS = dS

    def printRxn(self):
        print self.label
        #    string format for self.label:
        #   '\t%d %s + %d H+ + %d e- --> %d %s + %d H2O' %(self.a, self.oxide, self.x, ↩
            self.z, self.b, self.ion, self.c)

    def setSurfConc(self, myname):
        self.surfconc = myname
```

```
39
40      def setPrecipRate(self, myname):
41          self.preciprate = myname
42
43      def setDissolRate(self, myname):
44          self.dissolrate = myname
45
46      def setReleaseRate(self, myname):
47          self.releaserate = myname
48
49      def addOxideObject(self, myoxide):
50          self.oxide_obj = myoxide
51
52      def setRxnName(self):
53          self.name = self.ion + '_' + self.oxide
54
55      def setHrxn(self, myval):
56          self.dH = myval
57
58      def setSrxn(self, myval):
59          self.dS = myval
```

Listing B.3: `Reaction` object class

B.4 `Isotope` class

```
class Isotope(object):
   """    A chemical isotope of an element

      ATTRIBUTES:
   self.name: name of isotope (string)
   self.activated: name of activated isotope (string)
   self.decay: radioactive decay parameter (float)
   self.natural_abundance: natural abundance of isotope (float)
   self.sigma: neutron capture cross section of isotope (float)
   self.cool_conc: name of isotope coolant concentration variable (string)
   self.act_cool_conc: name of activated coolant concentration variable (string)
   self.act_ox_conc: name of isotope coolant concentration variable (string)
   self.ncapture: name of isotope neutron capture variable (string)
   self.active_oxide: name of oxide isotope activity variable (string)
    """

    def __init__(self, isotope, active, decay, mu, capture):
        self.name = isotope
        self.activated = active
        self.decay = decay
        self.natural_abundance = mu
        self.sigma = capture

    def setCoolConc(self, myname):
        self.cool_conc = myname

    def setActCoolConc(self, myname):
        self.act_cool_conc = myname

    def setActOxConc(self, myname):
        self.act_ox_conc = myname

    def setNCapture(self, myname):
        self.ncapture = myname

    def setActiveOxide(self, myname):
        self.active_oxide = myname
```

Listing B.4: `Isotope` object class

B.5 Element class

```
class Element(object):
   """    A chemical element

      ATTRIBUTES:
      self.name: name of element (string)
      self.MM_metal: molar mass of element (float)
      self.isotopes: list of objects of isotopes of this element
      self.ions: list of objects of ions of this element
      self.oxides: list of object of oxides of this element
      self.oxide_conc: list of names of oxide concentration variables
      self.cool_conc: name of isotope coolant concentration variable (string)
      self.preciprate: name of precipitation rate variable (string)
      self.dissolrate: name of dissolution rate variable (string)
      self.releaserate: name of release rate variable (string)
      self.wtperc_elemavg: name of average weight percent variable (string)
      self.diffusivity_elemavg: name of average diffusivity variable(string)
      self.base_diff: name of diffusivity variable (string)
      self.e0: name of diffusion activation energy variable(string)
      self.d0: name of diffusion prefactor variable (string)
      self.e0_func: function to represent e0 values along loop (string)
      self.d0_func: function to represent d0 values (string)
      self.corr_rate_const: function for element corrosion rate along loop (string)
      """

    def __init__(self, element, molar_mass):
        self.name = element
        self.MM_metal = molar_mass
        self.isotopes = []
        self.ions = []
        self.oxides = []
        self.oxide_conc = []

    def addIon(self, ions):
        for ion in ions:
            self.ions.append(ion)

    def addOxide(self, oxides):
        for oxide in oxides:
            self.oxides.append(oxide)
```

```
40
41      def addIsotope(self, isotopes):
42          for isotope in isotopes:
43              self.isotopes.append(isotope)
44
45      def setCoolConcentration(self, myname):
46          self.cool_conc = myname
47
48      def addOxideConcentration(self, myname):
49          self.oxide_conc.append(myname)
50
51      def setWtPercentElemAvg(self, myname):
52          self.wtperc_elemavg = myname
53
54      def setDiffElemAvg(self, myname):
55          self.diffusivity_elemavg = myname
56
57      def setBaseDiff(self, myname):
58          self.base_diff = myname
59
60      def setPrecipRate(self, myname):
61          self.preciprate = myname
62
63      def setDissolRate(self, myname):
64          self.dissolrate = myname
65
66      def setReleaseRate(self, myname):
67          self.releaserate = myname
68
69      def setE0andD0(self, e0, d0):
70          #used to calculate diffusivity of metal element through oxide from corrosion
71          self.e0 = e0
72          self.d0 = d0
73
74      def setFunc_E0D0(self, e0func, d0func):
75          self.e0_func = e0func
76          self.d0_func = d0func
77
78      def setCorrRateConst(self, my_const):
79          self.corr_rate_const = my_const
```

Listing B.5: `Element` object class

Appendix C

Sample input file

```
################################
######## GLOBALPARAMS #########
################################
[GlobalParams]
   pH = 6.5
   temperature = temp
   water_viscosity = viscosity
   water_density = density
   reynolds = reynolds
   hydraulic_diameter = hydr_diameter
   velocity = cool_velocity
   ECP_var = ECP
   wetted_area = wet_area
   loop_volume = pp_volume_tot
[]

################################
############ MESH #############
################################
[Mesh]
   type = GeneratedMesh
   dim = 2
   nx = 78
   ny = 1
   xmax = 78
   elem_type = QUAD9
[]
```

```
28
29 ################################
30 ############ VARS #############
31 ################################
32 [Variables]
33    [./conc_NiO_ox_nodal]
34       order = FIRST
35       family = LAGRANGE
36       initial_condition = 1e-09
37    [../]
38
39    [./conc_ni_cool]
40       order = FIRST
41       family = SCALAR
42       initial_condition = 1e-09
43    [../]
44
45    [./conc_ni58_cool]
46       order = FIRST
47       family = SCALAR
48       initial_condition = 1e-50
49       scaling = 10
50    [../]
51
52    [./conc_co58_oxide_active]
53       order = FIRST
54       family = LAGRANGE
55       initial_condition = 0
56       scaling = 100000
57    [../]
58
59    [./conc_CoO_ox_nodal]
60       order = FIRST
61       family = LAGRANGE
62       initial_condition = 1e-09
63    [../]
64
65    [./conc_co_cool]
66       order = FIRST
67       family = SCALAR
68       initial_condition = 1e-09
69       scaling = 1e+10
70    [../]
```

```
71
72    [./conc_co58_cool]
73       order = FIRST
74       family = SCALAR
75       initial_condition = 1e-50
76       scaling = 1e+10
77    [../]
78
79    [./conc_co59_oxide_active]
80       order = FIRST
81       family = LAGRANGE
82       initial_condition = 0
83       scaling = 1e+10
84    [../]
85
86    [./conc_co59_cool]
87       order = FIRST
88       family = SCALAR
89       initial_condition = 0
90       scaling = 1e+10
91    [../]
92
93    [./conc_co60_oxide_active]
94       order = FIRST
95       family = LAGRANGE
96       initial_condition = 0
97       scaling = 1e+50
98    [../]
99
100   [./conc_co60_cool]
101      order = FIRST
102      family = SCALAR
103      initial_condition = 0
104      scaling = 1e+10
105   [../]
106[]
107
108################################
109########### AUXVARS ###########
110################################
111[AuxVariables]
112   [./thick_elemavg]
113      order = FIRST
```

```
114       family = SCALAR
115       initial_condition = 0
116    [../]
117    [./WA_elemavg]
118       order = FIRST
119       family = SCALAR
120       initial_condition = 0
121    [../]
122    [./density_elemavg]
123       order = FIRST
124       family = SCALAR
125       initial_condition = 0
126    [../]
127    [./dummy_scalar]
128       order = FIRST
129       family = SCALAR
130       initial_condition = 0
131    [../]
132    [./dummy_scalar_1]
133       order = FIRST
134       family = SCALAR
135       initial_condition = 1
136    [../]
137    [./aux_thick]
138       order = FIRST
139       family = LAGRANGE
140       initial_condition = 1e-08
141    [../]
142    [./aux_thick_act]
143       order = FIRST
144       family = LAGRANGE
145       initial_condition = 0
146    [../]
147    [./activity_oxide]
148       order = FIRST
149       family = LAGRANGE
150       initial_condition = 0
151    [../]
152    [./metal_density]
153       order = FIRST
154       family = LAGRANGE
155    [../]
156    [./cg_rate_ni]
```

```
157       order = FIRST
158       family = LAGRANGE
159       initial_condition = 0
160    [../]
161    [./cg_rate_const_ni]
162       order = FIRST
163       family = LAGRANGE
164    [../]
165    [./cg_rate_co]
166       order = FIRST
167       family = LAGRANGE
168       initial_condition = 0
169    [../]
170    [./cg_rate_const_co]
171       order = FIRST
172       family = LAGRANGE
173    [../]
174    [./oxide_density]
175       order = FIRST
176       family = LAGRANGE
177       initial_condition = 1200
178    [../]
179    [./reynolds]
180       order = FIRST
181       family = LAGRANGE
182       initial_condition = 0
183    [../]
184    [./density]
185       order = FIRST
186       family = LAGRANGE
187       initial_condition = 0
188    [../]
189    [./viscosity]
190       order = FIRST
191       family = LAGRANGE
192       initial_condition = 0
193    [../]
194    [./temp]
195       order = FIRST
196       family = LAGRANGE
197    [../]
198    [./nflux]
199       order = FIRST
```

```
200       family = LAGRANGE
201     [../]
202     [./cool_velocity]
203       order = FIRST
204       family = LAGRANGE
205     [../]
206     [./wetted_perimeter]
207       order = FIRST
208       family = LAGRANGE
209       initial_condition = 0
210     [../]
211     [./hydr_diameter]
212       order = FIRST
213       family = LAGRANGE
214     [../]
215     [./ECP]
216       order = FIRST
217       family = LAGRANGE
218     [../]
219     [./wet_area]
220       order = FIRST
221       family = LAGRANGE
222       initial_condition = 0
223     [../]
224     [./equiv_node_length]
225       order = FIRST
226       family = LAGRANGE
227     [../]
228     [./specific_area]
229       order = FIRST
230       family = LAGRANGE
231       initial_condition = 0
232     [../]
233     [./effective_wet_area]
234       order = FIRST
235       family = LAGRANGE
236     [../]
237     [./effective_volume]
238       order = FIRST
239       family = LAGRANGE
240     [../]
241     [./my_volume]
242       order = FIRST
```

```
243       family = LAGRANGE
244       initial_condition = 0
245    [../]
246    [./base_fe]
247       order = FIRST
248       family = LAGRANGE
249    [../]
250    [./base_ni]
251       order = FIRST
252       family = LAGRANGE
253    [../]
254    [./base_cr]
255       order = FIRST
256       family = LAGRANGE
257    [../]
258    [./base_zr]
259       order = FIRST
260       family = LAGRANGE
261    [../]
262    [./base_co]
263       order = FIRST
264       family = LAGRANGE
265    [../]
266    [./dummy]
267       order = FIRST
268       family = LAGRANGE
269       initial_condition = 0
270    [../]
271    [./molarmass_base]
272       order = FIRST
273       family = LAGRANGE
274       initial_condition = 0
275    [../]
276    [./total_oxide_conc]
277       order = FIRST
278       family = LAGRANGE
279       initial_condition = 0
280    [../]
281    [./surfconc_Ni2p_NiO]
282       order = FIRST
283       family = LAGRANGE
284       initial_condition = 0
285    [../]
```

```
286   [./preciprate_Ni2p_NiO]
287      order = FIRST
288      family = LAGRANGE
289      initial_condition = 0
290   [../]
291   [./dissolrate_Ni2p_NiO]
292      order = FIRST
293      family = LAGRANGE
294      initial_condition = 0
295   [../]
296   [./releaserate_Ni2p_NiO]
297      order = FIRST
298      family = LAGRANGE
299      initial_condition = 0
300   [../]
301   [./diffusivity_Ni2p]
302      order = FIRST
303      family = LAGRANGE
304      initial_condition = 0
305   [../]
306   [./kc_Ni2p]
307      order = FIRST
308      family = LAGRANGE
309      initial_condition = 0
310   [../]
311   [./schmidt_Ni2p]
312      order = FIRST
313      family = LAGRANGE
314      initial_condition = 0
315   [../]
316   [./sherwood_Ni2p]
317      order = FIRST
318      family = LAGRANGE
319      initial_condition = 0
320   [../]
321   [./dummy_flux_Ni2p]
322      order = FIRST
323      family = LAGRANGE
324      initial_condition = 0
325   [../]
326   [./wtpercent_elemavg_ni]
327      order = FIRST
328      family = SCALAR
```

```
329       initial_condition = 0
330     [../]
331     [./diffusivity_elemavg_ni]
332       order = FIRST
333       family = SCALAR
334       initial_condition = 0
335     [../]
336     [./base_ni_diffusivity]
337       order = FIRST
338       family = LAGRANGE
339       initial_condition = 0
340     [../]
341     [./total_precip_rate_ni]
342       order = FIRST
343       family = LAGRANGE
344       initial_condition = 0
345     [../]
346     [./total_dissol_rate_ni]
347       order = FIRST
348       family = LAGRANGE
349       initial_condition = 0
350     [../]
351     [./e0_ni]
352       order = FIRST
353       family = LAGRANGE
354     [../]
355     [./d0_ni]
356       order = FIRST
357       family = LAGRANGE
358     [../]
359     [./ncapture_ni58]
360       order = FIRST
361       family = LAGRANGE
362       initial_condition = 0
363     [../]
364     [./oxide_active_co58]
365       order = FIRST
366       family = LAGRANGE
367       initial_condition = 0
368     [../]
369     [./surfconc_Co2p_CoO]
370       order = FIRST
371       family = LAGRANGE
```

```
372       initial_condition = 0
373    [../]
374    [./preciprate_Co2p_CoO]
375       order = FIRST
376       family = LAGRANGE
377       initial_condition = 0
378    [../]
379    [./dissolrate_Co2p_CoO]
380       order = FIRST
381       family = LAGRANGE
382       initial_condition = 0
383    [../]
384    [./releaserate_Co2p_CoO]
385       order = FIRST
386       family = LAGRANGE
387       initial_condition = 0
388    [../]
389    [./diffusivity_Co2p]
390       order = FIRST
391       family = LAGRANGE
392       initial_condition = 0
393    [../]
394    [./kc_Co2p]
395       order = FIRST
396       family = LAGRANGE
397       initial_condition = 0
398    [../]
399    [./schmidt_Co2p]
400       order = FIRST
401       family = LAGRANGE
402       initial_condition = 0
403    [../]
404    [./sherwood_Co2p]
405       order = FIRST
406       family = LAGRANGE
407       initial_condition = 0
408    [../]
409    [./dummy_flux_Co2p]
410       order = FIRST
411       family = LAGRANGE
412       initial_condition = 0
413    [../]
414    [./wtpercent_elemavg_co]
```

```
415       order = FIRST
416       family = SCALAR
417       initial_condition = 0
418    [../]
419    [./diffusivity_elemavg_co]
420       order = FIRST
421       family = SCALAR
422       initial_condition = 0
423    [../]
424    [./base_co_diffusivity]
425       order = FIRST
426       family = LAGRANGE
427       initial_condition = 0
428    [../]
429    [./total_precip_rate_co]
430       order = FIRST
431       family = LAGRANGE
432       initial_condition = 0
433    [../]
434    [./total_dissol_rate_co]
435       order = FIRST
436       family = LAGRANGE
437       initial_condition = 0
438    [../]
439    [./e0_co]
440       order = FIRST
441       family = LAGRANGE
442    [../]
443    [./d0_co]
444       order = FIRST
445       family = LAGRANGE
446    [../]
447    [./ncapture_co58]
448       order = FIRST
449       family = LAGRANGE
450       initial_condition = 0
451    [../]
452    [./oxide_active_co59]
453       order = FIRST
454       family = LAGRANGE
455       initial_condition = 0
456    [../]
457    [./ncapture_co59]
```

```
458       order = FIRST
459       family = LAGRANGE
460       initial_condition = 0
461    [../]
462    [./oxide_active_co60]
463       order = FIRST
464       family = LAGRANGE
465       initial_condition = 0
466    [../]
467 []
468
469 ################################
470 ########### KERNELS ############
471 ################################
472 [NodalKernels]
473    [./td_conc_NiO_ox_nodal]
474       type = TimeDerivativeNodalKernel
475       variable = conc_NiO_ox_nodal
476    [../]
477
478    [./corr_NiO]
479       type = NodalCorrosionGrowth
480       variable = conc_NiO_ox_nodal
481       oxide_percentage = 1.0
482       rate_constant = cg_rate_const_ni
483       unit_conversion = 0.0001
484       wetted_area = wet_area
485       MM_oxide = 0.0746928
486       equiv_vol = my_volume
487       loop_volume = pp_volume_tot
488       base_material = base_ni
489    [../]
490    [./dp_Ni2p_NiO]
491       type = NodalDissolPrecip
492       variable = conc_NiO_ox_nodal
493       oxide_conc = conc_NiO_ox_nodal
494       release_rate = releaserate_Ni2p_NiO
495       bulk_conc = conc_ni_cool
496    [../]
497    [./mr_NiO_ox]
498       type = NodalMetalRelease
499       variable = conc_NiO_ox_nodal
500       diffusivity = base_ni_diffusivity
```

```
    wt_percent = base_ni
    thickness = aux_thick
    wetted_area = wet_area
    equiv_vol = my_volume
    loop_volume = pp_volume_tot
    alloy_density = metal_density
    oxide_conc = conc_NiO_ox_nodal
    MM_elem = 0.0586934
    scaling = 1e-12
  [../]

  [./td_conc_co58_oxide_active]
    type = TimeDerivativeNodalKernel
    variable = conc_co58_oxide_active
  [../]

  [./precip_co58_act_ox_ni58]
    type = PrecipRate
    variable = conc_co58_oxide_active
    precip_rate = total_precip_rate_ni
    elem_conc = conc_ni_cool
    iso_conc = conc_co58_cool
    lambda = 1.08e-07
    equiv_vol = my_volume
    loop_volume = pp_volume_tot
  [../]

  [./td_conc_CoO_ox_nodal]
    type = TimeDerivativeNodalKernel
    variable = conc_CoO_ox_nodal
  [../]

  [./corr_CoO]
    type = NodalCorrosionGrowth
    variable = conc_CoO_ox_nodal
    oxide_percentage = 1.0
    rate_constant = cg_rate_const_co
    unit_conversion = 0.0001
    wetted_area = wet_area
    MM_oxide = 0.0749326
    equiv_vol = my_volume
    loop_volume = pp_volume_tot
    base_material = base_co
```

```
  [../]
  [./dp_Co2p_CoO]
    type = NodalDissolPrecip
    variable = conc_CoO_ox_nodal
    oxide_conc = conc_CoO_ox_nodal
    release_rate = releaserate_Co2p_CoO
    bulk_conc = conc_co_cool
  [../]
  [./mr_CoO_ox]
    type = NodalMetalRelease
    variable = conc_CoO_ox_nodal
    diffusivity = base_co_diffusivity
    wt_percent = base_co
    thickness = aux_thick
    wetted_area = wet_area
    equiv_vol = my_volume
    loop_volume = pp_volume_tot
    alloy_density = metal_density
    oxide_conc = conc_CoO_ox_nodal
    MM_elem = 0.058933
    scaling = 1e-12
  [../]

  [./td_conc_co59_oxide_active]
    type = TimeDerivativeNodalKernel
    variable = conc_co59_oxide_active
  [../]

  [./precip_co59_act_ox_co58]
    type = PrecipRate
    variable = conc_co59_oxide_active
    precip_rate = total_precip_rate_co
    elem_conc = conc_co_cool
    iso_conc = conc_co59_cool
    lambda = 0
    equiv_vol = my_volume
    loop_volume = pp_volume_tot
  [../]

  [./td_conc_co60_oxide_active]
    type = TimeDerivativeNodalKernel
    variable = conc_co60_oxide_active
  [../]
```

```
587
588    [./precip_co60_act_ox_co59]
589       type = PrecipRate
590       variable = conc_co60_oxide_active
591       precip_rate = total_precip_rate_co
592       elem_conc = conc_co_cool
593       iso_conc = conc_co60_cool
594       lambda = 4.17e-09
595       equiv_vol = my_volume
596       loop_volume = pp_volume_tot
597    [../]
598 []
599
600 [ScalarKernels]
601
602    [./td_conc_ni_cool]
603       type = ODETimeDerivative
604       variable = conc_ni_cool
605    [../]
606
607    [./mr_ni_cool_bulk]
608       type = MetalReleaseODE
609       variable = conc_ni_cool
610       diffusivity = diffusivity_elemavg_ni
611       wt_percent = wtpercent_elemavg_ni
612       thickness = thick_elemavg
613       wetted_area = WA_elemavg
614       alloy_density = density_elemavg
615       loop_volume = pp_volume_tot
616       MM_elem = 0.0586934
617       scaling = 1e-12
618    [../]
619    [./dp_Ni2p_NiO_cool_bulk]
620       type = ImplicitODEVariableRate
621       variable = conc_ni_cool
622       pp_rate = pp_releaserate_Ni2p_NiO
623       loop_volume = pp_volume_tot
624    [../]
625
626    [./td_conc_ni58_cool]
627       type = ODETimeDerivative
628       variable = conc_ni58_cool
629    [../]
```

```
630
631   [./conc_ni58_cool_bulk_nonact]
632      type = ImplicitODENonActive
633      variable = conc_ni58_cool
634      elem_conc = conc_ni_cool
635      natural_abundance = 0.68077
636      ncapture_pp = ncapture_ni58_pp
637      dissolrate = pp_total_dissol_rate_ni
638      preciprate = pp_total_precip_rate_ni
639      loop_volume = pp_volume_tot
640   [../]
641   [./conc_co58_cool_bulk]
642      type = ImplicitODEActive
643      variable = conc_co58_cool
644      elem_conc = conc_ni_cool
645      nonactive_conc = conc_ni58_cool
646      lambda = 1.08e-07
647      ncapture_pp = ncapture_ni58_pp
648      preciprate = pp_total_precip_rate_ni
649      loop_volume = pp_volume_tot
650   [../]
651
652   [./td_conc_co_cool]
653      type = ODETimeDerivative
654      variable = conc_co_cool
655   [../]
656
657   [./mr_co_cool_bulk]
658      type = MetalReleaseODE
659      variable = conc_co_cool
660      diffusivity = diffusivity_elemavg_co
661      wt_percent = wtpercent_elemavg_co
662      thickness = thick_elemavg
663      wetted_area = WA_elemavg
664      alloy_density = density_elemavg
665      loop_volume = pp_volume_tot
666      MM_elem = 0.058933
667      scaling = 1e-12
668   [../]
669   [./dp_Co2p_CoO_cool_bulk]
670      type = ImplicitODEVariableRate
671      variable = conc_co_cool
672      pp_rate = pp_releaserate_Co2p_CoO
```

```
673       loop_volume = pp_volume_tot
674     [../]
675
676     [./td_conc_co58_cool]
677       type = ODETimeDerivative
678       variable = conc_co58_cool
679     [../]
680
681     [./conc_co58_cool_bulk_nonact]
682       type = ImplicitODENonActive
683       variable = conc_co58_cool
684       elem_conc = conc_co_cool
685       natural_abundance = 1e-05
686       ncapture_pp = ncapture_co58_pp
687       dissolrate = pp_total_dissol_rate_co
688       preciprate = pp_total_precip_rate_co
689       loop_volume = pp_volume_tot
690     [../]
691
692     [./td_conc_co59_cool]
693       type = ODETimeDerivative
694       variable = conc_co59_cool
695     [../]
696
697     [./conc_co59_cool_bulk]
698       type = ImplicitODEActive
699       variable = conc_co59_cool
700       elem_conc = conc_co_cool
701       nonactive_conc = conc_co58_cool
702       lambda = 0
703       ncapture_pp = ncapture_co58_pp
704       preciprate = pp_total_precip_rate_co
705       loop_volume = pp_volume_tot
706     [../]
707     [./conc_co59_cool_bulk_nonact]
708       type = ImplicitODENonActive
709       variable = conc_co59_cool
710       elem_conc = conc_co_cool
711       natural_abundance = 0.9999999
712       ncapture_pp = ncapture_co59_pp
713       dissolrate = pp_total_dissol_rate_co
714       preciprate = pp_total_precip_rate_co
715       loop_volume = pp_volume_tot
```

```
   [../]

   [./td_conc_co60_cool]
      type = ODETimeDerivative
      variable = conc_co60_cool
   [../]

   [./conc_co60_cool_bulk]
      type = ImplicitODEActive
      variable = conc_co60_cool
      elem_conc = conc_co_cool
      nonactive_conc = conc_co59_cool
      lambda = 4.17e-09
      ncapture_pp = ncapture_co59_pp
      preciprate = pp_total_precip_rate_co
      loop_volume = pp_volume_tot
   [../]
[]

################################
######### AUXKERNELS ##########
################################
[AuxKernels]
   [./total_oxide_conc_aux]
      type = SumVariableValues
      variable = total_oxide_conc
      vars_to_sum = 'conc_NiO_ox_nodal conc_CoO_ox_nodal '
      execute_on = 'initial timestep_end'
   [../]
   [./total_oxide_act_aux]
      type = SumVariableValues
      variable = activity_oxide
      vars_to_sum = 'oxide_active_co58 oxide_active_co59 oxide_active_co60 '
      execute_on = 'initial timestep_end'
   [../]
   [./total_dissol_rate_ni_aux]
      type = SumVariableValues
      variable = total_dissol_rate_ni
      vars_to_sum = 'dissolrate_Co2p_CoO '
      execute_on = timestep_begin
   [../]
   [./total_precip_rate_ni_aux]
      type = SumVariableValues
```

```
    variable = total_precip_rate_ni
    vars_to_sum = 'preciprate_Co2p_CoO '
    execute_on = timestep_begin
  [../]
  [./total_dissol_rate_co_aux]
    type = SumVariableValues
    variable = total_dissol_rate_co
    vars_to_sum = 'dissolrate_Co2p_CoO '
    execute_on = timestep_begin
  [../]
  [./total_precip_rate_co_aux]
    type = SumVariableValues
    variable = total_precip_rate_co
    vars_to_sum = 'preciprate_Co2p_CoO '
    execute_on = timestep_begin
  [../]
  [./thickness_aux]
    type = OxideGrowthNodal
    variable = aux_thick
    oxide_conc = total_oxide_conc
    wetted_area = wet_area
    equiv_vol = my_volume
    density_oxide = 5368.0
    molar_mass = 0.234379
    execute_on = 'initial timestep_end'
  [../]
  [./activity_aux_co58]
    type = ActivityOxideCalc
    variable = oxide_active_co58
    oxide_conc = conc_co58_oxide_active
    wetted_area = wet_area
    decay_parameter = 1.08e-07
    equiv_vol = my_volume
    execute_on = timestep_end
  [../]
  [./activity_aux_co59]
    type = ActivityOxideCalc
    variable = oxide_active_co59
    oxide_conc = conc_co59_oxide_active
    wetted_area = wet_area
    decay_parameter = 0
    equiv_vol = my_volume
    execute_on = timestep_end
```

```
802   [../]
803   [./activity_aux_co60]
804     type = ActivityOxideCalc
805     variable = oxide_active_co60
806     oxide_conc = conc_co60_oxide_active
807     wetted_area = wet_area
808     decay_parameter = 4.17e-09
809     equiv_vol = my_volume
810     execute_on = timestep_end
811   [../]
812   [./molarmass_basemat]
813     type = MolarMassAux
814     variable = molarmass_base
815     fe = base_fe
816     ni = base_ni
817     zr = base_zr
818     co = base_co
819     cr = base_cr
820     execute_on = initial
821   [../]
822   [./n_corrgrowth_ni]
823     type = CorrosionGrowthAux
824     variable = cg_rate_ni
825     rate_constant = cg_rate_const_ni
826     unit_conversion = 1e-07
827   [../]
828   [./n_corrgrowth_co]
829     type = CorrosionGrowthAux
830     variable = cg_rate_co
831     rate_constant = cg_rate_const_co
832     unit_conversion = 1e-07
833   [../]
834   [./effective_wetarea]
835     type = EffectiveWetArea
836     variable = wet_area
837     wetted_area = effective_wet_area
838     total_loop_length = 78
839     eq_length = equiv_node_length
840     nx = 78
841     execute_on = 'initial timestep_begin'
842   [../]
843   [./effective_vol]
844     type = EffectiveWetArea
```

```
845       variable = my_volume
846       wetted_area = effective_volume
847       total_loop_length = 78
848       eq_length = equiv_node_length
849       nx = 78
850       execute_on = 'initial timestep_begin'
851     [../]
852     [./visc]
853       type = Reynolds
854       variable = viscosity
855       property = viscosity
856       execute_on = timestep_end
857     [../]
858     [./reyn]
859       type = Reynolds
860       variable = reynolds
861       property = reynolds
862       execute_on = timestep_end
863     [../]
864     [./dens]
865       type = Reynolds
866       variable = density
867       property = density
868       execute_on = initial
869     [../]
870     [./ni_ox_diff]
871       type = Reynolds
872       variable = base_ni_diffusivity
873       property = oxide_diffusivity
874       E0_elem = e0_ni
875       D0_elem = d0_ni
876       execute_on = initial
877     [../]
878     [./surface_conc_Ni2p_NiO_aux]
879       type = EchemSurfaceConcentration
880       variable = surfconc_Ni2p_NiO
881       x = 2
882       b = 1
883       z = 0
884       delta_H = -100130.0
885       delta_S = -96.98
886       execute_on = 'initial'
887     [../]
```

```
  [./flux_aux_Ni2p]
    type = SpeciesDiffusivity
    variable = dummy_flux_Ni2p
    kc_species = kc_Ni2p
    species_diffusion_prefactor = 1.81e-07
    surface_concentration_species = surfconc_Ni2p_NiO
    bulk_conc = conc_ni_cool
    diff_coupledvar = diffusivity_Ni2p
    get_flux = true
    ox_diffusivity = base_ni_diffusivity
  [../]
  [./kc_Ni2p_aux]
    type = MassTransferCoefficientAux
    variable = kc_Ni2p
    surface_concentration_species = surfconc_Ni2p_NiO
    bulk_conc = conc_ni_cool
    species_flux = dummy_flux_Ni2p
    execute_on = timestep_begin
  [../]
  [./diff_Ni2p_aux]
    type = SpeciesDiffusivity
    variable = diffusivity_Ni2p
    kc_species = kc_Ni2p
    species_diffusion_prefactor = 1.81e-07
    surface_concentration_species = dummy
    bulk_conc = dummy_scalar
    diff_coupledvar = dummy
    ox_diffusivity = base_ni_diffusivity
  [../]
  [./schmidt_Ni2p_aux]
    type = SchmidtNumber
    variable = schmidt_Ni2p
    species_diffusivity = diffusivity_Ni2p
  [../]
  [./sherwood_Ni2p_aux]
    type = Sherwood
    variable = sherwood_Ni2p
    schmidt_number = schmidt_Ni2p
  [../]
  [./releasedepositionrate_Ni2p_NiO]
    type = ReleaseDepositionRate
    variable = releaserate_Ni2p_NiO
    bulk_conc = conc_ni_cool
```

```
931       surface_concentration_species = surfconc_Ni2p_NiO
932       species_diffusivity = diffusivity_Ni2p
933       sherwood_number = sherwood_Ni2p
934       wetted_perimeter = pp_wa_avg
935       wetted_area = wet_area
936       base_matprop = base_ni
937       loop_volume = pp_volume_tot
938       equiv_vol = my_volume
939       hydraulic_diameter = hydr_diameter
940       execute_on = 'initial timestep_begin'
941    [../]
942    [./preciprate_Ni2p_NiO_aux]
943       type = DissolutionPrecipitationRate
944       variable = preciprate_Ni2p_NiO
945       precipitate = true
946       bulk_conc = conc_ni_cool
947       surface_concentration_species = surfconc_Ni2p_NiO
948       releasedepositionrate_species = releaserate_Ni2p_NiO
949       execute_on = timestep_begin
950    [../]
951    [./dissolrate_Ni2p_NiO_aux]
952       type = DissolutionPrecipitationRate
953       variable = dissolrate_Ni2p_NiO
954       precipitate = false
955       bulk_conc = conc_ni_cool
956       surface_concentration_species = surfconc_Ni2p_NiO
957       releasedepositionrate_species = releaserate_Ni2p_NiO
958       execute_on = timestep_begin
959    [../]
960    [./ncapture_ni58_aux]
961       type = NeutronCapture
962       variable = ncapture_ni58
963       neutron_flux = nflux
964       bulk_conc = dummy_scalar_1
965       capture_xs = 4.6e-28
966    [../]
967    [./co_ox_diff]
968       type = Reynolds
969       variable = base_co_diffusivity
970       property = oxide_diffusivity
971       E0_elem = e0_co
972       D0_elem = d0_co
973       execute_on = initial
```

```
  [../]
  [./surface_conc_Co2p_CoO_aux]
    type = EchemSurfaceConcentration
    variable = surfconc_Co2p_CoO
    x = 2
    b = 1
    z = 0
    delta_H = -106090.0
    delta_S = -96.06
    execute_on = 'initial'
  [../]
  [./flux_aux_Co2p]
    type = SpeciesDiffusivity
    variable = dummy_flux_Co2p
    kc_species = kc_Co2p
    species_diffusion_prefactor = 1.81e-07
    surface_concentration_species = surfconc_Co2p_CoO
    bulk_conc = conc_co_cool
    diff_coupledvar = diffusivity_Co2p
    get_flux = true
    ox_diffusivity = base_co_diffusivity
  [../]
  [./kc_Co2p_aux]
    type = MassTransferCoefficientAux
    variable = kc_Co2p
    surface_concentration_species = surfconc_Co2p_CoO
    bulk_conc = conc_co_cool
    species_flux = dummy_flux_Co2p
    execute_on = timestep_begin
  [../]
  [./diff_Co2p_aux]
    type = SpeciesDiffusivity
    variable = diffusivity_Co2p
    kc_species = kc_Co2p
    species_diffusion_prefactor = 1.81e-07
    surface_concentration_species = dummy
    bulk_conc = dummy_scalar
    diff_coupledvar = dummy
    ox_diffusivity = base_co_diffusivity
  [../]
  [./schmidt_Co2p_aux]
    type = SchmidtNumber
    variable = schmidt_Co2p
```

```
      species_diffusivity = diffusivity_Co2p
    [../]
    [./sherwood_Co2p_aux]
      type = Sherwood
      variable = sherwood_Co2p
      schmidt_number = schmidt_Co2p
    [../]
    [./releasedepositionrate_Co2p_CoO]
      type = ReleaseDepositionRate
      variable = releaserate_Co2p_CoO
      bulk_conc = conc_co_cool
      surface_concentration_species = surfconc_Co2p_CoO
      species_diffusivity = diffusivity_Co2p
      sherwood_number = sherwood_Co2p
      wetted_perimeter = pp_wa_avg
      wetted_area = wet_area
      base_matprop = base_co
      loop_volume = pp_volume_tot
      equiv_vol = my_volume
      hydraulic_diameter = hydr_diameter
      execute_on = 'initial timestep_begin'
    [../]
    [./preciprate_Co2p_CoO_aux]
      type = DissolutionPrecipitationRate
      variable = preciprate_Co2p_CoO
      precipitate = true
      bulk_conc = conc_co_cool
      surface_concentration_species = surfconc_Co2p_CoO
      releasedepositionrate_species = releaserate_Co2p_CoO
      execute_on = timestep_begin
    [../]
    [./dissolrate_Co2p_CoO_aux]
      type = DissolutionPrecipitationRate
      variable = dissolrate_Co2p_CoO
      precipitate = false
      bulk_conc = conc_co_cool
      surface_concentration_species = surfconc_Co2p_CoO
      releasedepositionrate_species = releaserate_Co2p_CoO
      execute_on = timestep_begin
    [../]
    [./ncapture_co58_aux]
      type = NeutronCapture
      variable = ncapture_co58
```

```
      neutron_flux = nflux
      bulk_conc = dummy_scalar_1
      capture_xs = 1.9e-25
   [../]
   [./ncapture_co59_aux]
      type = NeutronCapture
      variable = ncapture_co59
      neutron_flux = nflux
      bulk_conc = dummy_scalar_1
      capture_xs = 2.07e-27
   [../]
[]

[AuxScalarKernels]

   [./WA_elemavg_aux]
      type = PostprocessorAux
      variable = WA_elemavg
      pp = pp_wetted_area_tot
      execute_on = 'initial timestep_begin'
   [../]
   [./thick_elemavg_aux]
      type = PostprocessorAux
      variable = thick_elemavg
      pp = avg_thickness
      execute_on = 'initial timestep_begin'
   [../]
   [./dens_elemavg_aux]
      type = PostprocessorAux
      variable = density_elemavg
      pp = avg_dens
      execute_on = 'initial timestep_begin'
   [../]
   [./wtpercent_elemavg_ni_aux]
      type = PostprocessorAux
      variable = wtpercent_elemavg_ni
      pp = pp_wtpercent_elemavg_ni
      execute_on = 'initial timestep_begin'
   [../]
   [./diffusivity_elemavg_ni_aux]
      type = PostprocessorAux
      variable = diffusivity_elemavg_ni
      pp = pp_diffusivity_elemavg_ni
```

```
      execute_on = 'initial timestep_begin'
   [../]
   [./wtpercent_elemavg_co_aux]
      type = PostprocessorAux
      variable = wtpercent_elemavg_co
      pp = pp_wtpercent_elemavg_co
      execute_on = 'initial timestep_begin'
   [../]
   [./diffusivity_elemavg_co_aux]
      type = PostprocessorAux
      variable = diffusivity_elemavg_co
      pp = pp_diffusivity_elemavg_co
      execute_on = 'initial timestep_begin'
   [../]
[]

################################
############## ICS ##############
################################
[ICs]
   [./metal_density_IC]
      type = FunctionIC
      variable = metal_density
      function = 'if(x<4.587, 6550, if(x>22.798&x<28.493, 8470, 8000))'
   [../]
   [./effective_wet_area_IC]
      type = FunctionIC
      variable = effective_wet_area
      function = 'if(x<0.855, 1400, if(x<3.416, 4200, if(x<4.27, 1400, if(x<4.587, 235.2, ↩
          if(x<8.137, 452.96, if(x<14.982, 50, if(x<22.797, 200, if(x<28.493, 4050, if(x↩
          <35.767, 200, if(x<43.178, 150, if(x<64.355, 100, 911.94))))))))))'
   [../]
   [./effective_volume_IC]
      type = FunctionIC
      variable = effective_volume
      function = 'if(x<0.855, 10.7, if(x<3.416, 31.74, if(x<4.27, 10.7, if(x<4.587, 11.05,↩
           if(x<8.137, 4.46, if(x<14.982, 15.83, if(x<22.797,18.07 , if(x<28.493, 151.4, ↩
          if(x<35.767, 17.99, if(x<43.178, 18.33, if(x<64.355, 46.47, 22.5))))))))))'
   [../]
   [./equiv_node_length_IC]
      type = FunctionIC
      variable = equiv_node_length
```

```
    function = 'if(x<0.855, 0.854, if(x<3.416, 2.562, if(x<4.27, 0.854, if(x<4.587, 0.317, if(x<8.137, 3.55, if(x<14.982, 6.845, if(x<22.797, 7.815, if(x<28.493, 5.696, if(x<35.767, 7.274, if(x<43.178, 7.411, if(x<64.355, 21.177, 13.769))))))))))))'
[../]
[./hydr_diameter_IC]
    type = FunctionIC
    variable = hydr_diameter
    function = 'if(x<4.27, 0.004, if(x<4.587, 0.0111, if(x<8.137, 0.4, if(x<22.797, 0.7360, if(x<35.767,0.0169, if(x>35.767&x<64.355,0.7874, if(x<78.124,0.6985, 0.52)))))))'
[../]
[./ECP_IC]
    type = FunctionIC
    variable = ECP
    function = 'if(x<4.27, -0.01171*(x-4.27)-0.8, if(x<22.797,-0.8,if(x<28.493,0.0088*(x-28.493)-0.75,-0.75)))'
[../]
[./nflux_IC]
    type = FunctionIC
    variable = nflux
    function = 'if(x<4.59, cos((3.14/(2*2.3))*x-(3.14/2))*1E17, 0.0)'
[../]
[./temp_IC]
    type = FunctionIC
    variable = temp
    function = 'if(x<4.587,565+(25*sin(x*3.14159/4.587))+34*(x/4.587),if(x>22.798&x<28.493,(599-34*((x-22.8)/(28.492 - 22.8))),if(x>28.492,(599-34*((28.492-22.8)/(28.492 - 22.8))),565+(25*sin(4.587*3.14159/4.587))+34*(4.587/4.587))))'
[../]
[./cool_velocity_IC]
    type = FunctionIC
    variable = cool_velocity
    function = 'if(x<0.854, 7.93, if(x<3.416, 5.06, if(x<4.27, 5.32, if(x<8.137, 4., if(x<14.932, 15.98, if(x<22.797, 5.5, if(x<28.493, 5.25, if(x<35.767, 5.12, if(x<43.178, 12.4, if(x<64.355, 15.73, 5.))))))))))'
[../]
[./base_fe_IC]
    type = FunctionIC
    variable = base_fe
    function = 'if(x<4.587, 0.00205, if(x>22.798&x<28.493, 0.10, 0.709))'
[../]
[./base_ni_IC]
```

```
    type = FunctionIC
    variable = base_ni
    function = 'if(x<4.587, 0.0, if(x>22.798&x<28.493, 0.729, 0.10))'
[../]
[./base_cr_IC]
    type = FunctionIC
    variable = base_cr
    function = 'if(x<4.587, 0.001, if(x>22.798&x<28.493, 0.17, 0.19))'
[../]
[./base_zr_IC]
    type = FunctionIC
    variable = base_zr
    function = 'if(x<4.587, 0.99685, 0)'
[../]
[./base_co_IC]
    type = FunctionIC
    variable = base_co
    function = 'if(x<4.587, 0.0001, if(x>22.798&x<28.493, 0.001, 0.001))'
[../]
[./cg_rate_const_ni_IC]
    type = FunctionIC
    variable = cg_rate_const_ni
    function = 'if(x<4.587, 0.0, if(x>22.798&x<28.493, 1.0, 0.2))'
[../]
[./d0_ni_IC]
    type = FunctionIC
    variable = d0_ni
    function = 'if(x<4.587,1E-10,if(x>22.798&x<28.493,1.51e-4,6.1e-3))'
[../]
[./e0_ni_IC]
    type = FunctionIC
    variable = e0_ni
    function = 'if(x<4.587,1e-10,if(x>22.798&x<28.493,2.01,1.695))'
[../]
[./cg_rate_const_co_IC]
    type = FunctionIC
    variable = cg_rate_const_co
    function = 'if(x<4.587, 1e-6, if(x>22.798&x<28.493,1.16,1.16))'
[../]
[./d0_co_IC]
    type = FunctionIC
    variable = d0_co
    function = 'if(x<4.587,1E-10,if(x>22.798&x<28.493,1.51e-4,6.1e-3))'
```

```
   [../]
   [./e0_co_IC]
      type = FunctionIC
      variable = e0_co
      function = 'if(x<4.587,1e-10,if(x>22.798&x<28.493,2.01,1.695))'
   [../]
[]

################################
####### POSTPROCESSORS ########
################################
[Postprocessors]
   [./accumulated_activ]
      type = ElementIntegralVariablePostprocessor
      variable = activity_oxide
      execute_on = 'timestep_end'
   [../]
   [./avg_thickness]
      type = ElementIntegralVariablePostprocessor
      variable = aux_thick
      execute_on = 'initial timestep_begin'
   [../]
   [./dh_pp]
      type = ElementIntegralVariablePostprocessor
      variable = hydr_diameter
      execute_on = 'timestep_begin'
   [../]

   [./pp_wetted_area_tot]
      type = ElementIntegralVariablePostprocessor
      variable = wet_area
      execute_on = 'initial timestep_begin'
   [../]
   [./pp_wa_avg]
      type = ElementAverageValue
      variable = wet_area
      execute_on = 'initial timestep_begin'
   [../]
   [./pp_volume_tot]
      type = ElementIntegralVariablePostprocessor
      variable = my_volume
      execute_on = 'initial timestep_begin'
   [../]
```

```
  [./dummy_pp]
    type = ScalarVariable
    variable = dummy_scalar
  [../]
  [./avg_ox_conc]
    type = ElementAverageValue
    variable = total_oxide_conc
  [../]
  [./pp_wtpercent_elemavg_ni]
    type = ElementIntegralVariablePostprocessor
    variable = base_ni
    execute_on = 'initial timestep_begin'
  [../]
  [./pp_diffusivity_elemavg_ni]
    type = ElementIntegralVariablePostprocessor
    variable = base_ni_diffusivity
    execute_on = 'initial timestep_begin'
  [../]
  [./pp_total_dissol_rate_ni]
    type = ElementIntegralVariablePostprocessor
    variable = total_dissol_rate_ni
    execute_on = 'initial timestep_begin'
  [../]
  [./pp_total_precip_rate_ni]
    type = ElementIntegralVariablePostprocessor
    variable = total_precip_rate_ni
    execute_on = 'initial timestep_begin'
  [../]
  [./ncapture_ni58_pp]
    type = ElementIntegralVariablePostprocessor
    variable = ncapture_ni58
    execute_on = 'initial timestep_begin'
  [../]
  [./diffusivity_Ni2p_pp]
    type = ElementIntegralVariablePostprocessor
    variable = diffusivity_Ni2p
    execute_on = 'initial timestep_begin'
  [../]
  [./sherwood_Ni2p_pp]
    type = ElementIntegralVariablePostprocessor
    variable = sherwood_Ni2p
    execute_on = 'initial timestep_begin'
  [../]
```

```
  [./surfconc_Ni2p_NiO_pp]
    type = ElementAverageValue
    variable = surfconc_Ni2p_NiO
  [../]
  [./pp_releaserate_Ni2p_NiO]
    type = ElementIntegralVariablePostprocessor
    variable = releaserate_Ni2p_NiO
    execute_on = 'initial timestep_begin'
  [../]
  [./pp_preciprate_Ni2p_NiO]
    type = ElementIntegralVariablePostprocessor
    variable = preciprate_Ni2p_NiO
    execute_on = 'initial timestep_begin'
  [../]
  [./pp_dissolrate_Ni2p_NiO]
    type = ElementIntegralVariablePostprocessor
    variable = dissolrate_Ni2p_NiO
    execute_on = 'initial timestep_begin'
  [../]
  [./surfconc_Ni2p_NiO_total_pp]
    type = ElementIntegralVariablePostprocessor
    variable = surfconc_Ni2p_NiO
    execute_on = 'initial timestep_begin'
  [../]
  [./pp_wtpercent_elemavg_co]
    type = ElementIntegralVariablePostprocessor
    variable = base_co
    execute_on = 'initial timestep_begin'
  [../]
  [./pp_diffusivity_elemavg_co]
    type = ElementIntegralVariablePostprocessor
    variable = base_co_diffusivity
    execute_on = 'initial timestep_begin'
  [../]
  [./pp_total_dissol_rate_co]
    type = ElementIntegralVariablePostprocessor
    variable = total_dissol_rate_co
    execute_on = 'initial timestep_begin'
  [../]
  [./pp_total_precip_rate_co]
    type = ElementIntegralVariablePostprocessor
    variable = total_precip_rate_co
    execute_on = 'initial timestep_begin'
```

```
  [../]
  [./ncapture_co58_pp]
    type = ElementIntegralVariablePostprocessor
    variable = ncapture_co58
    execute_on = 'initial timestep_begin'
  [../]
  [./ncapture_co59_pp]
    type = ElementIntegralVariablePostprocessor
    variable = ncapture_co59
    execute_on = 'initial timestep_begin'
  [../]
  [./diffusivity_Co2p_pp]
    type = ElementIntegralVariablePostprocessor
    variable = diffusivity_Co2p
    execute_on = 'initial timestep_begin'
  [../]
  [./sherwood_Co2p_pp]
    type = ElementIntegralVariablePostprocessor
    variable = sherwood_Co2p
    execute_on = 'initial timestep_begin'
  [../]
  [./surfconc_Co2p_CoO_pp]
    type = ElementAverageValue
    variable = surfconc_Co2p_CoO
  [../]
  [./pp_releaserate_Co2p_CoO]
    type = ElementIntegralVariablePostprocessor
    variable = releaserate_Co2p_CoO
    execute_on = 'initial timestep_begin'
  [../]
  [./pp_preciprate_Co2p_CoO]
    type = ElementIntegralVariablePostprocessor
    variable = preciprate_Co2p_CoO
    execute_on = 'initial timestep_begin'
  [../]
  [./pp_dissolrate_Co2p_CoO]
    type = ElementIntegralVariablePostprocessor
    variable = dissolrate_Co2p_CoO
    execute_on = 'initial timestep_begin'
  [../]
  [./surfconc_Co2p_CoO_total_pp]
    type = ElementIntegralVariablePostprocessor
    variable = surfconc_Co2p_CoO
```

```
      execute_on = 'initial timestep_begin'
   [../]
[]

################################
########## SETTINGS ############
################################
[Functions]
   [./dts]
      type = PiecewiseConstant
      x = '0 500 1000 10000 50000 605000 2400000 9984000 100000000'
      y = '0 10 100 500 1000 5000 10000 86400 604800'
      direction = right
   [../]
[]

[Executioner]
   type = Transient
   [./TimeIntegrator]
      type = ImplicitMidpoint
   [../]
   solve_type = PJFNK
   start_time = 0
   end_time = 31536000
   [./TimeStepper]
      type = FunctionDT
      function = dts
   [../]
   l_tol = 1.0e-5
   l_max_its = 15
   nl_max_its = 10
   nl_rel_tol = 1.0e-4
   nl_abs_tol = 1.0e-6
   petsc_options_iname = '-pc_type -pc_hypre_type'
   petsc_options_value = 'hypre boomeramg'
[]

[Outputs]
   file_base = results/scripted_nico_17/pH_6.5_nico
   exodus = true
   print_perf_log = true
   csv = true
[]
```

```
1432
1433 [Debug]
1434    show_parser = true
1435    show_actions = true
1436    show_var_residual_norms = true
1437 []
```

Listing C.1: Ouroboros input file generated from script