Master's thesis and Internship : Design and implementation of a modular modeling framework for the ANICCA nuclear fuel cycle simulation code

Auteur : Mouchamps, Antoine
Promoteur(s) : Ernst, Damien
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil en génie de l'énergie à finalité spécialisée en Energy Conversion
Année académique : 2025-2026
URI/URL : http://hdl.handle.net/2268.2/26132

LIÈGE université
School of Engineering

University of Liège
Faculty of applied sciences
Academic year 2025-2026

Design and implementation of a modular modeling framework for the ANICCA nuclear fuel cycle simulation code

---

Thesis presented to obtain the degree of:
Master in Energy Engineering, professional focus in Energy Conversion

Thesis supervisors:

Pablo Romojaro
Gert Van den Eynde
Hamid Aït Abderrahim
Damien Ernst

Antoine Mouchamps

Acknowledgments

First and foremost, I want to thank my promoter, Prof. Damien Ernst, whose guidance has been invaluable throughout the past two years of our research collaboration, up until this master thesis. Beyond his scientific mentorship, working with him allowed me to develop the research skills that proved to be essential throughout this thesis. I also want to thank him for introducing me to Prof. Hamid Aït Abderrahim, which ultimately made this internship possible. I also want to thank Prof. Hamid Aït Abderrahim for giving me the opportunity to work at SCK CEN and for his insights and guidance throughout the project.

Furthermore, I particularly want to thank Dr. Pablo Romojaro, who welcomed me at SCK CEN and who was always available to discuss and answer my questions, from the very beginning of the project to the very end. His openness and support eventually proved to be a key contribution to the quality of my work on ANICCA. I also want to thank Prof. Gert Van den Eynde for his many pieces of advice during our meetings and for his thorough reviews of my work, which greatly helped to improve the clarity and quality of this thesis. Lastly, I want to thank Michel Rigo for his review and his advice on the mathematical notation used throughout this thesis.

Finally, I want to thank my parents for their support throughout my internship and my girlfriend, for her presence in my life and for her constant reminders that there is also a life to enjoy outside my work.

Summary

The growing interest in advanced nuclear fuel cycles, driven by concerns over uranium availability, security of energy supply, and long-lived radioactive waste management, has created a strong demand for robust and flexible nuclear fuel cycle simulation tools. Such tools must accurately model the entire chain of material flows, from uranium mining to final waste disposal, while handling intricate recycling loops and composition-dependent constraints.

This thesis presents the design and implementation of a new modular modeling framework for the ANICCA nuclear fuel cycle simulation code, developed at the Belgian Nuclear Research Center (SCK CEN). The legacy version of ANICCA relies on an ad hoc solving algorithm that is too rigid for complex scenario studies. In addition, its lack of modularity and implementation structure prevents the development of new features or advanced studies such as optimization. The objective of this thesis is to address these limitations by defining a rigorous and practical mathematical and modeling framework for the new version of ANICCA.

The theoretical background supporting the new framework is first established. Relevant concepts from graph theory are then introduced. The nuclear physics background is also reviewed, covering radioactive decay, neutron-induced reactions, and their specificities. For both the decay and irradiation equation systems, the Chebyshev Rational Approximation Method (CRAM) is presented and used.

The simulation framework is then described. A scenario is represented as a directed multigraph, with facilities as nodes and material connections as directed edges. Three distinct abstraction layers structure the framework: the network layer, which manages the graph topology and material dispatch; the facility layer, which orchestrates internal inventories and process execution; and the process layer, which implements the physical transformations. The default mass dispatch algorithm currently implemented requires the graph to be acyclic. Recycling loops, which would break the acyclic assumption, are handled through a graph-tearing strategy that exploits the graph topology. The dispatch algorithm operates through a backward pass, which propagates demand upstream through the graph in reverse topological order, followed by a forward pass, which physically transfers packages (masses) between facilities in topological order.

Four cycle types are defined for facilities and their processes. The *fixed demand* type corresponds to facilities for which the mass of each input and output is known. This is typically the case for reactor fleets. The *on demand* type corresponds to facilities that scale their production for a given requested demand. This is typically the case for fuel fabrication facilities. Third, *variable demand* facilities only request available packages up to their processing capacity. For example, this cycle type can be used for spent fuel reprocessing facilities. Finally, *none* facilities do not request or produce any packages. This is typically useful for storage facilities without any processes.

The nuclear physics models implemented in ANICCA are then described. Nuclide composition is tracked through a mass-fraction vector augmented with a dummy nuclide that aggregates all untracked nuclides. Both radioactive decay and irradiation are implemented using CRAM, with LU factorization caching to minimize computation time. The currently implemented processes are the same as in the legacy ANICCA version: `Conversion`, `Enrichment` (with a ^{236}U equivalent correction variant), `Combine` (with Pu-equivalent and MOX-equivalent variants for composition-dependent fuel fabrication), `Reprocess`, `CombinePackages`, and `Irradiation`. The irradiation process simulates a fleet of reactors based on a historical opera-

tion timeline and pre-computed fuel libraries based on the ALEPH2 depletion code outputs.

Four case studies are then presented. The first study compares the radioactive decay of a spent UOX fuel sample in both the legacy ANICCA code and its new version, and ultimately identifies two errors in the legacy implementation. The second study compares the irradiation of a single fuel batch between both versions. The third case study then models a classical open-cycle PWR scenario. The fourth one finally presents a closed MOX recycling loop, showcasing the graph-tearing strategy, the `MoxEquivalentCombine` process, the handling of multiple fuels in reactor cores, and the fleet handling mechanism.

The thesis concludes with a discussion about the thesis achievements and some suggestions for future work. The most significant ones are the extension of capacity constraints to on demand processes, the resolution of the limitation preventing composition-dependent on demand processes from being supplied by on demand facilities, and the design of a new irradiation logic for partial fuel loading and unloading.

Contents

List of Figures **iv**

I Introduction 2

1 Context **2**

2 Thesis objectives **2**

3 Thesis structure **3**

II Theoretical background and state of the art 5

4 State of the art **5**

5 Graph theory **6**
5.1 Definitions 6

6 Nuclear physics **7**
6.1 General definitions 7
6.1.1 Nuclide classification 7
6.2 Radioactive decay 8
6.3 Irradiation 9
6.3.1 Neutron-nucleus interactions 9
6.3.2 Cross-sections and neutron flux 10
6.3.3 Induced fission and fission product yields 10
6.3.4 One-group quantities 10
6.3.5 Burnup 11
6.4 Generalized Bateman equations: formulation and solution 11
6.4.1 Formulation 12
6.4.2 Solution 13

III Simulation Framework 14

7 ANICCA **14**
7.1 Configuration file 15
7.2 Customization 15
7.3 Time discretization 15
7.4 Scenario representation 16

8 Facility **17**
8.1 Cycle type 17
8.2 State 17

9 Process **18**
9.1 General model 18
9.1.1 Processing time 18
9.2 Fixed Demand (FD) process 19

9.3 On Demand (OD) process . . . 19
9.3.1 Composition-Dependent On Demand (CDOD) process . . . 19
9.4 Variable Demand (VD) process . . . 20
9.5 Relation to the parent facility . . . 20
9.5.1 Get all required inputs and outputs . . . 20
9.5.2 Get all produced packages . . . 21
9.5.3 Recompute composition-dependent requested inputs . . . 22

10 Mass dispatch algorithm 22
10.1 Problem definition . . . 22
10.2 Solving approach . . . 23
10.2.1 Acyclic graph . . . 23
10.2.2 Cyclic graph . . . 24
10.3 Algorithm . . . 25
10.3.1 Backward pass . . . 25
10.3.2 Forward pass . . . 29
10.3.3 Final algorithm . . . 32

IV Nuclear physics modeling 33

11 Nuclide composition tracking 33

12 Nuclear data 33
12.1 Data filtering . . . 34
12.2 Data preprocessing . . . 34
12.2.1 Spontaneous fission handling . . . 34
12.2.2 Alpha decay . . . 35
12.2.3 Chain collapsing . . . 35

13 Radioactive decay 37
13.1 Formulation . . . 37
13.2 Decay matrix construction . . . 38
13.3 Integration in scenario simulation . . . 38
13.4 Decay engine . . . 39
13.4.1 LU caching . . . 39

14 Processes description 39
14.1 Conversion . . . 40
14.2 Enrichment . . . 40
14.2.1 U236EquivalentEnrichment . . . 41
14.3 Combine . . . 42
14.3.1 PuEquivalentCombine . . . 43
14.3.2 MoxEquivalentCombine . . . 44
14.4 Reprocess . . . 46
14.5 CombinePackages . . . 47
14.6 Irradiation . . . 47
14.6.1 Reactors fleet handling . . . 47
14.6.2 Reactor batch management . . . 48
14.6.3 Input data . . . 51
14.6.4 Data preprocessing . . . 52

14.6.5 Irradiation step 53
14.6.6 Process interface 55

V Case studies 56

15 Decay 56
15.1 Overproduction 57
15.2 Underproduction 58

16 Irradiation 59

17 Simple example 60

18 Recycling loop with MOX 60

VI Conclusion 64

VII Appendix 69

A Derivation of the state-transition split 69

B SCK CEN 70

List of Figures

I Introduction **2**

II Theoretical background and state of the art **5**

III Simulation Framework **14**
1 Simple scenario without any cycle. 24
2 Scenario with recycling loops. 25
3 Lifecycle of the rolling buffer update. 27

IV Nuclear physics modeling **33**
4 Simplified example of the collapse of the decay tree of a tracked nuclide denoted A. Each round node denotes a tracked nuclide, while each square represents an untracked nuclide. 36
5 Illustration of a 3-batch reactor core modeled in ANICCAv2. 49
6 Main mass transfer occurring at each time step in a given reactor batch. 50

V Case studies **56**
7 Relative difference in nuclide mass fractions between $v1$ and $v2$, both after a decay of 1 year and at the end of the simulation. 57
8 Evolution of ^{125}Sb and ^{125}Te masses over the first 10 year. 57
9 Evolution of the mass of ^{81}Kr in $v1$ and $v2$ over all time steps. 58
10 Relative difference in nuclide mass fractions between $v1$ and $v2$ after one irradiation step of 1 year. 59
11 Facility graph of the simple PWR scenario. 60
12 Evolution of each facility inventory over time, for the simple PWR scenario. . . 61
13 Facility graph of the MOX recycling loop scenario. 62
14 Evolution of each facility inventory over time, for the MOX recycling loop scenario. 63

VI Conclusion **64**

VII Appendix **69**

List of Algorithms

1 Backward pass algorithm. 26
2 Backward pass algorithm for computing the deferred demands at a future time t'. 28
3 Algorithm for the initialization of the buffer B. 28
4 Forward pass algorithm. 31
5 Mass dispatch algorithm for a single simulation time step. 32
6 Build the decay matrix $\mathbf{A}$. 38
7 Build the burnup matrix $\mathbf{A}$. 54

Part I
Introduction

1 Context

The growing need for low-carbon, large-scale, and dispatchable electricity production, coupled with the increasing importance of energy supply security, has recently renewed interest in nuclear energy in many countries [1]. While nuclear power offers clear advantages in terms of carbon intensity and reliability, traditional nuclear energy systems are predominantly based on open-fuel cycles, in which nuclear fuel is produced, used once, and ultimately disposed of as high-level waste. Such open-cycle approaches lead to inefficient uranium usage, larger amounts of long-lived radioactive waste and higher long-term radiotoxicity (with respect to other cycles), which collectively limit the scalability and sustainability of nuclear energy deployment, as well as its public acceptability [2].

Furthermore, current market projections forecast an increasing risk of uranium supply shortage in the coming decades, as many countries aim to commission new reactors [3, 4], which could lead to a rise in prices. Whereas the current strategy of open-fuel cycles relies heavily on low uranium prices, an increase in prices might further incentivize the deployment of advanced reactor designs, which often involve more complex fuel production chains and spent fuel handling. For all these reasons, there is a strong industrial, social, and scientific interest [5, 6] in the development of advanced Nuclear Fuel Cycles (NFCs) capable of involving multi-recycling of spent nuclear fuel, separation of highly radioactive elements (minor actinides) and potential transmutation into less radiotoxic ones, and the deployment of advanced fast-reactor technologies. In order to efficiently plan the deployment of such fleets, accurate modeling of the whole fuel cycle is necessary.

The transition from linear to circular nuclear economies introduces significant technical complexity. Planning the deployment of such heterogeneous reactor fleets requires a precise understanding of material flows, involving nuclide composition and facility capacities over several time steps, from months to centuries and beyond. Consequently, the development of robust and flexible NFC simulation tools is essential. These tools must model the entire value chain, from mining and enrichment to reprocessing and waste conditioning, while accounting for the dynamic behavior of material inventories. Moreover, non-trivially enforced radioprotection-related or operational constraints need to be accounted for, such as in the context of reprocessed uranium enrichment [7] or timing constraints related to plutonium use as fuel [8].

2 Thesis objectives

The Belgian Nuclear Research Center (SCK CEN) has, in the past decades, investigated advanced fuel cycle scenarios, notably in the context of the MYRRHA program. The primary goal of MYRRHA is to demonstrate the industrial feasibility of an Accelerator Driven Subcritical (ADS) reactor in order to transmute minor actinides, thereby reducing the radiotoxicity of nuclear waste.

In support of these studies, SCK CEN has developed the NFC simulation code ANICCA. However, the legacy version of the tool relies on an *ad hoc* solving algorithm that lacks the

flexibility required for complex or modular scenario studies. Furthermore, its rigid modeling assumptions limit its ability to simulate systems with intricate inter-component constraints. For instance, the current framework cannot dynamically restrict fuel fabrication based on the specific nuclide composition of incoming materials, which is a critical requirement for modeling Mixed Oxide (MOX) or ADS fuel cycles where nuclide-dependent reactivity limits apply. In addition, SCK CEN is currently developing a small modular reactor based on the Lead-cooled Fast Reactor (LFR) technology, fueled by MOX. In this context, ANICCA is planned to be used in order to simulate scenarios regarding the deployment of this LFR in the Belgian energy system, in order to analyze aspects such as fuel requirements, spent fuel handling and waste production.

This master's thesis aims to design and implement a new modeling paradigm to address these shortcomings and significantly enhance the capabilities and extensibility of ANICCA. The specific objectives are as follows:

- **Formal framework:** Define a mathematical and numerical representation of NFC scenarios using a graph-based approach, where nodes represent facilities (mining, fuel fabrication, ...) and edges material flows.
- **Solver:** Develop and implement a solving algorithm capable of propagating demand and material flows through complex network topologies, including those containing recycling loops. In contrast to the current version of ANICCA, this algorithm has to be agnostic to the specificities of each node.
- **Nuclear process modeling:** Describe and implement the core nuclear-related processes such as enrichment, reprocessing, and reactor fleet management, as well as radioactive decay and fuel irradiation, while maintaining a clear separation between the solver logic and physical process models.
- **Validation:** Demonstrate the new code version performance by comparing its results with those of the legacy ANICCA code on a selection of representative case studies.

3 Thesis structure

To achieve these objectives, the thesis is organized as follows:

- **Part I** introduces the research context, motivates the transition toward closed nuclear fuel cycles, and defines the specific goals of the ANICCA software redesign.
- **Part II** reviews the state of the art in NFC simulation tools and establishes the theoretical bases of the study, specifically focusing on graph theory, radioactive decay, and irradiation physics.
- **Part III** details the architectural design of the new simulation engine. It formalizes the NFC as a graph-based system with facilities and connections and describes the solving algorithm developed to propagate demand and material flows while remaining agnostic to individual node behavior.
- **Part IV** describes the implementation of specific nuclear processes within each facility. Most notably, it covers the modeling of enrichment, fuel fabrication and reprocessing, and reactor batch management. In this part, radioactive decay and fuel irradiation modeling

are also described.

- **Part V** presents the validation of the new framework through a comparison of its results with the legacy ANICCA code, and showcases the new capabilities of the new version on a selection of representative case studies.

- **Part VI** concludes the thesis by summarizing its main contributions. It also sheds light on possible future work, improvements, and planned features for the new version of ANICCA and its development.

Part II
Theoretical background and state of the art

In this part, the theoretical background on which the remainder of the thesis is based is presented.

First, Section 4 gives an overview of existing NFC simulation codes. Then, the basics of graph theory are presented in Section 5, as it is of significant importance for both nuclear physics modeling and the new modeling framework of the new version of ANICCA. Finally, Section 6 provides the necessary background in nuclear physics, covering topics from decay modeling to irradiation processes, while also introducing terminologies and concepts related to nuclear physics.

Note: from now on, the current legacy version of ANICCA is denoted as ANICCAv1, whereas the new version presented in this thesis is labeled ANICCAv2.

4 State of the art

Over the past years, several simulation codes for nuclear fuel cycle analysis have been developed by various research institutions and countries. Each of these tools was built independently, tailored to the specific scenarios, hypotheses, and computational environments of its home institution. ANICCA (v1), CLASS, COSI, DYMOND, FAMILY, NMB4.0, and TR_EVOL are among the simulation codes with the most active and recent research records [9].

Despite the variety in their modeling assumptions and features, all of these codes suffer from the same limitations: they are monolithic by nature. Physical models, scenario components, and solver logic are intertwined into a single, tightly coupled software. Therefore, individual components cannot be replaced or extended without modifying the whole software, making it impractical (if even possible) to modify or add components, experiment with alternative solving strategies, or substitute physics engines. Most software is also not distributed as open-source or modular codes, which makes them ill-suited for integration into broader computational pipelines, such as uncertainty quantification or optimization loops.

A newer generation of codes has recently emerged with modularity as explicit design goals. For instance, the Cyclus [10] NFC simulation code is written in C++, open-source, and built around a plugin architecture that enables full modularity of each component. It is therefore highly modular, and also capable of running on high-performance computing clusters for advanced simulations such as sensitivity analyses. However, its agent-based resource exchange mechanism, which is based on a market-clearing algorithm, as well as its high modularity, both introduce multiple layers of conceptual and implementation complexity as well as significant overhead.

ANICCAv1 [11] is no exception to the limitations of the traditional codes: its solver is ad hoc and monolithic, tightly coupled to the scenario representation and its facilities, and offers little room for extension or modification. In addition, multiple issues or modeling inconsistencies were identified in the original implementation, which proved to be difficult to address without

a full rewrite. This motivated the development of a new version of ANICCA, which introduces, among other things, a modular architecture, a customizable solver algorithm, improved scenario modeling capabilities, and a well-defined modeling framework, which is presented in this thesis.

5 Graph theory

In ANICCAv2, the NFC is modeled as a network where material flows between facilities, which act as nodes. To solve the mass balance across this network, the system can be defined as a graph. Additionally, decay chains that are simulated in ANICCAv2 are also commonly analyzed using the vocabulary of graph theory. The present section therefore serves as the theoretical basis for the algorithms and physical models detailed in the following chapters.

5.1 Definitions

The definitions in this section are based on the work of Grinberg [12].

Simple Graph

A *simple graph* G is defined by a pair $G = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is a finite set of nodes (no repeating elements), and $\mathcal{E}$ is a finite set of edges. An edge $E \in \mathcal{E}$ is defined as a pair $E = (u, v)$ where $u, v \in \mathcal{N}$ are the two nodes linked by the edge. In a simple graph, u and v are interchangeable, meaning that if $(u, v) \in \mathcal{E}$, then $(v, u) \in \mathcal{E}$.

Multigraph

A *multigraph* is a generalization of a graph that allows for multiple edges between the same pair of nodes.

Directed Graph

A *directed graph* is a graph where the edges have a distinct orientation. In contrast with simple graph edges, a directed edge $E = (u, v)$ is an ordered pair where u is the source node and v is the target node, representing a unidirectional flow of material or information.

Path

A *path* in a directed graph is a sequence of nodes $(v_0, v_1, \ldots, v_k)$ such that for each $0 \leq i < k$, there exists a directed edge $(v_i, v_{i+1}) \in \mathcal{E}$.

In a directed graph, relationships between nodes are defined by the flow direction. If an edge (u, v) exists, u is a *parent* of v, and v is a *child/daughter* of u. A node u is an *ancestor* of v if there exists a directed path from u to v. Conversely, if such a path exists, v is a *descendant* of u.

Cycle

A *cycle* is a path where the first and last nodes are identical ($v_0 = v_k$). A graph containing at least one cycle is called a *cyclic graph*.

Directed Acyclic Graph

A *Directed Acyclic Graph* (DAG) is a directed graph with no directed cycles. It consists of nodes and edges such that following the edge directions will never result in returning to a previously visited node.

A directed graph is a DAG if and only if it admits a *topological sort* [13]. A topological sort is a linear ordering of nodes such that for every directed edge (u, v), node u comes before v in the ordering.

Topological sorting is typically performed using a depth-first search exploration of the graph [13].

6 Nuclear physics

This section establishes the theoretical bases of nuclear physics and simulation. Its purpose is to introduce the terminology and physical mechanisms that are referenced in the modeling and implementation parts of this work.

Section 6.1 first introduces the notation used to identify nuclides. Section 6.2 then describes radioactive decay and related parameters. Section 6.3 defines neutron-induced reactions, cross-sections, and the one-group formalism used to condense energy-dependent data into a single effective value. Finally, Section 6.4 presents the mathematical framework that unifies decay and irradiation into a single matrix equation.

6.1 General definitions

A nuclide designates a chemical element with both a specific number of protons Z (the atomic number) and a mass number $A = Z + N$, where N is the number of neutrons. Nuclides sharing the same Z (same element) but differing in A are referred to as *isotopes*. When referring to a specific nuclide, the notation

$$^{A}_{Z}X \tag{1}$$

is used, where X designates the specific element, Z its atomic number, and A its mass number. For example, uranium 235, an isotope of uranium, is designated by $^{235}_{92}$U or ^{235}U for short.

In numerical applications related to nuclear research, it is common to represent a nuclide with a unique integer identifier. In this thesis and in ANICCAv2, the six-digit $ZZAAAS$ identifier is used. This label is obtained using

$$\text{code} = Z \times 10^4 + A \times 10 + S, \tag{2}$$

where $S \in \{0, 1, 2, \ldots\}$ denotes the isomeric state (state in which the nucleus occupies excited energy levels), with $S = 0$ corresponding to the ground state. For example, the ^{235}U nuclide is represented using the label 922350.

6.1.1 Nuclide classification

In NFC analyses, it is common to group certain sets of elements together in specific groups. The following list explains the ones used in this thesis:

1. *Heavy Metals* (HM) is most often used in order to distinguish the actual nuclear fuel mass (e.g., uranium) from the total mass of fuel assemblies (oxygen originating from the

uranium being present in the form of oxide, cladding, ...) when talking about the amount of fuel loaded in reactor cores.

2. *Minor Actinides* (MA) designates all chemical elements belonging to the actinide group, except uranium (U) and plutonium (Pu). The most significant elements of this list are neptunium, americium, curium, and californium.

3. *Major Actinides* (MJA) designates uranium and plutonium.

4. *Fission Products* (FP) designates all elements produced by the fission of heavier elements.

6.2 Radioactive decay

Radioactive decay is a stochastic process by which an unstable nuclide spontaneously transforms into a different nuclide, emitting energy as particles or gamma rays. The rate at which a population of radioactive atoms of a given nuclide decay is characterized by the decay constant λ [s^{-1}], which is a property of each nuclide. The evolution of the number of radioactive atoms in a sample is described by the first-order differential Bateman equation [14]:

$$\frac{\mathrm{d}n(t)}{\mathrm{d}t} = -\lambda\, n(t), \tag{3}$$

where $n(t)$ denotes the population of a given nuclide, i.e., a number of atoms. The decay constant is equivalently expressed through the half-life $t_{1/2} = \ln 2/\lambda$, the expected time for half of the atoms in a sample to decay. Half-lives of nuclides relevant to nuclear fuel cycles span many orders of magnitude, from microseconds for highly unstable nuclides to billions of years for others.

A nuclide may decay through different modes. The most common ones include α decay (emission of an α particle, which is a ${}^{4}_{2}\mathrm{He}^{2+}$ ion), β^- decay (emission of an electron), β^+ decay (emission of a positron), γ decay (emission of a gamma ray) and Spontaneous Fission (SF). Each decay mode (except SF) corresponds to a specific change in A or Z of the nuclide. For instance, alpha decay will result in a change in mass number and atomic number of respectively $A-4$ and $Z-2$. By manipulating the $ZZAAAS$ codes, one can therefore map each decay mode to the nuclide produced after decay.

When multiple decay modes are possible for a given nuclide, the fraction of disintegrations leading from a parent nuclide j to its daughter i is described by the *branching ratio* $b_{j\to i}$. If all decay modes are accounted for, the sum of the branching ratios is equal to one:

$$\sum_i b_{j\to i} = 1. \tag{4}$$

Since each daughter may itself be radioactive and prone to decay, a single initial nuclide can give rise to what is called a *decay chain*. The population balance for nuclide i accumulating from all *precursors* j while simultaneously depleting through its own decay is

$$\frac{\mathrm{d}n_i(t)}{\mathrm{d}t} = \sum_{j\neq i} b_{j\to i}\,\lambda_j\, n_j(t) - \lambda_i\, n_i(t). \tag{5}$$

Spontaneous fission is a decay mode in which a heavy nuclide splits into lighter nuclides without any external neutron interaction. It is treated as one of the available decay branches

of the parent nuclide, with an associated branching fraction $b_{j\to\mathrm{SF}}$. Unlike single-daughter decay modes, a single fission event produces a statistical distribution of fission products. The expected number of atoms of nuclide i produced per fission event of parent j is described by the *fission product yield* $y_{j\to i}$. The effective production rate of species i from spontaneous fission of parent j therefore contributes an additional term $y_{j\to i}\, b_{j\to\mathrm{SF}}\, \lambda_j\, n_j(t)$ to the right-hand side of Equation 5.

6.3 Irradiation

In the context of nuclear reactors, *irradiation* refers to the exposure of nuclear fuel to an intense neutron flux. As neutrons interact with fuel nuclides, they trigger a cascade of nuclear reactions. The most important one is the fission of *fissile* nuclides such as ^{235}U or ^{239}Pu. Each fission is caused by the absorption of one neutron, and releases a varying number of neutrons which can in turn induce further fissions. The *reactivity* of a fuel quantifies whether this chain reaction is self-sustaining: it reflects the net balance between neutron production through fission and neutron losses through absorption. A reactor is *critical* when these are exactly balanced and the neutron population remains steady. As fissile nuclides are consumed during irradiation, the fuel reactivity decreases, and needs to be compensated by other mechanisms in order to maintain a stable chain reaction. In addition to fission, neutrons can be absorbed without causing fission, thereby *transmuting* nuclides into different isotopes of the same element. The combined effect of all these reactions continuously alters the nuclide composition of the fuel over its irradiation lifetime, in a process called *depletion*. This section builds a description of this irradiation process step by step. It should be noted that other neutron-nucleus interactions do occur in practice (notably, scattering), but these are not relevant when focusing on the transmutation of nuclides due to neutrons and will therefore not be considered.

First, Section 6.3.1 presents different types of neutron-nucleus interactions. Then, Section 6.3.2 introduces the concepts of cross-sections and neutron flux, while Section 6.3.3 introduces fission yields. Finally, Section 6.3.4 explains how the energy dependence of cross-sections and fission yields can be collapsed into effective one-group values, and Section 6.3.5 formally defines burnup as the cumulative measure of irradiation.

6.3.1 Neutron-nucleus interactions

When a neutron interacts with a nucleus, the outcome depends on the incoming neutron energy and on the target nuclide. Several distinct reaction channels are possible, each producing a different set of product nuclides and secondary particles. The eight channels relevant to ANICCAv2 and their notation are:

- Radiative capture (n,γ): the neutron is absorbed and a gamma ray is emitted, producing a nuclide with $A+1$.
- Neutron emission $(n,2n)$, $(n,3n)$, $(n,4n)$: the neutron is absorbed and two, three, or four neutrons are emitted, producing nuclides with $A-1$, $A-2$, or $A-3$ respectively.
- Induced fission (n,f): the neutron is absorbed and the nucleus splits into fission products with the release of additional neutrons.
- Alpha emission (n,α): the neutron is absorbed and an α particle is emitted, reducing Z by 2 and A by 4.
- Proton emission (n,p): the neutron is absorbed and a proton is emitted, reducing Z by

1.

- Isomeric capture (n, γ^*): like radiative capture, but the product nucleus is left in a metastable state at a higher energy level rather than at the ground state.

Among these channels, induced fission occupies a special place. Just like SF, fission yields a statistical distribution of fission products (and releases additional neutrons that sustain the chain reaction) and therefore does not produce a single well-defined daughter nuclide.

6.3.2 Cross-sections and neutron flux

Having listed the possible interaction types, one now needs a quantitative measure of how often they occur. The likelihood that a neutron of energy E triggers reaction channel c on nuclide i is characterized by the microscopic cross-section $\sigma_{i,c}(E)$ [barn, $1\,\mathrm{b} = 10^{-24}\,\mathrm{cm}^2$]. A larger cross-section means that the reaction is more probable. Cross-sections are specific to both the nuclide and the channel and depend on the incoming neutron energy E [eV].

The neutron field in a medium is characterized by the energy-dependent neutron flux $\varphi(E)$ [$\mathrm{n\,cm^{-2}\,s^{-1}\,eV^{-1}}$], which quantifies the flux of neutrons at a given energy level E. The total neutron flux Φ [$\mathrm{n\,cm^{-2}\,s^{-1}}$] can then be computed by integrating $\varphi(E)$ over the whole energy spectrum:

$$\Phi = \int_0^\infty \varphi(E)\,\mathrm{d}E. \tag{6}$$

6.3.3 Induced fission and fission product yields

As mentioned previously, when a heavy nuclide absorbs a neutron, the resulting nucleus can split into (usually) two lighter *fission products* and releases a given number of neutrons. The nature of the fission products follows a statistical distribution that depends on both the fissile parent and the incoming neutron energy. The *fission product yield* $y_{j\to i}(E)$ is defined as the expected number of atoms of nuclide i produced per fission event of parent j, for a neutron energy E.

This is directly analogous to the spontaneous fission case introduced in Section 6.2, with the difference that here the process is triggered by neutron absorption rather than occurring spontaneously.

6.3.4 One-group quantities

Both the cross-sections and fission product yields introduced above are functions of the incoming neutron energy E. In full neutron transport calculations, this dependence is tracked using dozens of energy groups, splitting the continuous spectrum in discrete bins. In addition, the energy spectrum also depends on the location in the reactor core (and on the spatial distribution of its materials), and therefore has a spatial dimension. In fuel cycle simulators, however, this level of precision is both unnecessary and computationally expensive. The standard approach is therefore to collapse the energy (and spatial) dimensions into a single *one-group* (or spectrum-averaged) value by weighting it with a neutron spectrum representative of the system. Because the neutron spectrum heavily depends on location and materials present, it is system-dependent.

For instance, the one-group cross-section of a nuclide i can be defined as

$$\bar{\sigma}_{i,c} = \frac{\int_0^\infty \sigma_{i,c}(E)\,\varphi(E)\,\mathrm{d}E}{\int_0^\infty \varphi(E)\,\mathrm{d}E} \tag{7}$$

where c corresponds to the reaction channel considered. Let us note that since only one-group quantities will be considered in this thesis, the following integrals will always be implicitly calculated for the whole energy spectrum, i.e., $[0,\infty)$. The one-group cross-section for the production of nuclides can be defined as

$$\bar{\sigma}_{j\to i,c} = \frac{\int \gamma_{j,c\to i}(E)\,\sigma_{j,c}(E)\,\varphi(E)\,\mathrm{d}E}{\int \varphi(E)\,\mathrm{d}E}, \tag{8}$$

where $\gamma_{j,c\to i}$ is a non-zero yield that designates either the transmutation of nuclide j into i by the channel c in case c is not a fission event, or the fission yield $y_{j\to i}$ otherwise. Fission yields being energy-dependent as well, one-group fission yields can be defined as

$$\bar{y}_{j\to i} = \frac{\int y_{j\to i}(E)\,\sigma_{j,f}(E)\,\varphi(E)\,\mathrm{d}E}{\int \sigma_{j,f}(E)\,\varphi(E)\,\mathrm{d}E}. \tag{9}$$

Starting from energy-dependent cross-sections and fission yield data available in evaluated nuclear data libraries (such as the ENDF/B data library [15]), neutron transport codes can compute the neutron spectrum and derive the problem-specific one-group cross-sections and effective fission yields. These quantities can then be stored in fuel libraries. Other software, such as NFC simulation codes, can then use and process these fuel libraries in order to compute fuel depletion at a fraction of the time taken by the transport codes, albeit at relatively lower accuracy.

6.3.5 Burnup

The cumulative effect of all neutron-induced reactions on the fuel composition is quantified by a single variable: *burnup*. Burnup measures the total thermal energy released per unit of HM mass and is usually expressed in [GWd/tHM]. Burnup is used to keep track of how the fuel properties evolve during irradiation: as the fissile inventory is consumed and fission products accumulate, the neutron spectrum shifts and all one-group quantities change accordingly. Fuel libraries can therefore store irradiation data as a sequence of burnup steps, and NFC simulation codes can advance through these steps to track the evolving nuclide composition of the fuel.

6.4 Generalized Bateman equations: formulation and solution

This section presents how the decay chains introduced in Section 6.2 and the neutron-induced reactions described in Section 6.3 can be unified into a single system of first-order linear differential equations by introducing effective rate coefficients that incorporate both contributions.

Section 6.4.1 presents the generalized Bateman equations formulation and the structure and properties of the so-called transition matrix $\mathbf{A}$, while Section 6.4.2 presents the methodology employed to solve the system, using the Chebyshev Rational Approximation Method (CRAM).

6.4.1 Formulation

Nuclear fission can be modeled using an equation similar to Equation 5 and by employing effective coefficients for the production and consumption terms in the following equation:

$$\frac{\mathrm{d}n_i(t)}{\mathrm{d}t} = \sum_j \lambda_{j\to i}^{eff}\, n_j(t) - \lambda_i^{eff}\, n_i(t). \tag{10}$$

Considering that neutron interactions are the dominant effect for particle-nuclide interactions, the effective coefficients λ_i^{eff} and $\lambda_{j\to i}^{eff}$ are given by

$$\lambda_i^{eff} = \lambda_i + \Phi \frac{\int \sigma_{abs,i}(E)\,\varphi(E)\,\mathrm{d}E}{\int \varphi(E)\,\mathrm{d}E} \tag{11}$$

$$= \lambda_i + \Phi \cdot \bar{\sigma}_{abs,i} \tag{12}$$

$$\lambda_{j\to i}^{eff} = b_{j\to i}\lambda_j + \Phi \sum_c \frac{\int \gamma_{j,c\to i}(E)\,\sigma_{j,c}(E)\,\varphi(E)\,\mathrm{d}E}{\int \varphi(E)\,\mathrm{d}E} \tag{13}$$

$$= b_{j\to i}\lambda_j + \Phi \sum_c \bar{\sigma}_{j\to i,c}. \tag{14}$$

In Equation 11, σ_{abs} represents the total absorption cross-section, given by

$$\sigma_{abs} = \sum_c \sigma_c. \tag{15}$$

Using this formulation, pure decay can be recovered by setting $\Phi = 0$, and pure neutron-driven transmutation in the absence of decay can be recovered by setting all $\lambda_i = 0$.

For a system of k tracked nuclides, Equation 10 can be written in the compact matrix form

$$\frac{\mathrm{d}\mathbf{n}(t)}{\mathrm{d}t} = \mathbf{A}\,\mathbf{n}(t), \tag{16}$$

where $\mathbf{n}(t) \in \mathbb{R}^k$ is the nuclide abundance vector (counting the number of atoms) and $\mathbf{A} \in \mathbb{R}^{k\times k}$ is the transition matrix. Its elements are defined as

$$A_{ij} = \begin{cases} -\lambda_i^{eff} & \text{if } i = j, \\ \lambda_{j\to i}^{eff} & \text{if } i \neq j, \end{cases} \tag{17}$$

so that diagonal elements represent the total removal rate of each nuclide and off-diagonal elements represent production paths. It should be noted that the λ_i^{eff} and $\lambda_{j\to i}^{eff}$ coefficients depend on time through the burnup of the fuel: as nuclides fission (and decay) and neutrons interact with the fuel materials, its composition drifts and so does the neutron flux and its energy spectrum, thereby making the coefficients evolve as well.

Regarding the properties of $\mathbf{A}$, it is sparse (since non-diagonal elements are limited to decay chains, neutron interactions, and fission events) and can be of large size (hundreds or thousands) depending on the number of tracked nuclides. Another important characteristic of $\mathbf{A}$ is its *stiffness*. The elements of the matrix can span dozens of orders of magnitude (because of half-lives ranging from microseconds to billions of years), which makes computations prone to catastrophic cancellation and numerical instability [16].

6.4.2 Solution

In the general case, different approaches can be used to solve Equation 16, whose *effectiveness* (with respect to accuracy, stability, or computation time) heavily depends on the properties of $\mathbf{A}$ [16]. In the case of decay/irradiation systems, one of the most used and effective approaches is to express the solution of Equation 16 as the matrix exponential and to compute an approximation of it.

The solution to Equation 16 is given by the matrix exponential [17]:

$$\mathbf{n}(t) = e^{\mathbf{A}t}\mathbf{n}_0 \tag{18}$$

where $e^{\mathbf{A}t}$ can be formally defined by the Taylor expansion

$$e^{\mathbf{A}t} = \sum_{k=0}^{\infty} \frac{(\mathbf{A}t)^k}{k!} \quad , \mathbf{A}^0 = \mathbf{I}. \tag{19}$$

In order to compute an approximation of $e^{\mathbf{A}t}$, modern nuclear codes frequently employ the Chebyshev Rational Approximation Method (CRAM) [18]. Because the eigenvalues of $\mathbf{A}$ are located on the left side of the complex plane, this method approximates the exponential function on the negative real axis, using a rational function.

Different rational approximations can be defined. Notably, the Partial Fraction Decomposition (PFD) [18] and the Incomplete Partial Fraction (IPF) [19] forms will be used throughout this thesis. The PFD form is given by Equation 20

$$\mathbf{n}_{PFD} = \alpha_0\mathbf{n}_0 + 2\Re\left[\sum_{j=1}^{k/2} \alpha_j(\mathbf{A}t - \theta_j\mathbf{I})^{-1}\mathbf{n}_0\right], \tag{20}$$

whereas the IPF form is an iterative formulation given by Equation 21.

$$\mathbf{n}_{IPF} = \alpha_0 \cdot \mathbf{y}_{k/2}, \tag{21}$$

$$\mathbf{y}_l = \begin{cases} 2\,\Re\left[\tilde{\alpha}_l(\mathbf{A}t - \theta_l\mathbf{I})^{-1}\mathbf{y}_{l-1}\right] + \mathbf{y}_{l-1} & \forall l \in \{1, 2, \ldots, k/2\} \\ \mathbf{n}_0 & \text{if } l = 0 \end{cases}. \tag{22}$$

Note: the algorithm given in [19] for the IPF implementation proved to be erroneous. The IPF formulation algorithm presented in the original paper [20] was therefore used.

In Equation 20 and Equation 21, k is the order of the approximation, and α_j and θ_j are fixed complex parameters that depend on the order of the method. In both formulations, computing Equation 18 requires solving $k/2$ linear systems. In the case of the PFD form, the systems are independent which means that they can be solved in parallel, accelerating the computations. However, one can show that the IPF formulation is numerically more stable and less prone to catastrophic cancellation [20, 19], at the price of being non-parallelizable.

The accuracy and required order of the approximations are heavily dependent on the nature of the problem. In pure radioactive decay scenarios, a high approximation order up to $k = 48$ is typically required [19]. Conversely, for irradiation simulations, an order of $k = 16$ is generally sufficient to achieve acceptable accuracy [18].

Part III
Simulation Framework

The new version of the ANICCA simulation code, ANICCAv2, is designed to simulate complex nuclear fuel cycles by decoupling the computation of material flows from the underlying physical transformations. To achieve this, the framework is structured into three distinct layers of abstraction:

1. **The network layer:** The fuel cycle is represented as a directed multigraph where nodes represent nuclear facilities and edges represent the flow of information (demand) and matter (mass packages). The mass dispatch algorithm orchestrates the facilities by dispatching masses between facilities through their edges.

2. **The facility layer:** Acting as the primary interface between the dispatcher and the physical processes, the `Facility` class manages internal inventories and state transitions. It serves as a container for physical processes, isolating the solver from the specificities of each individual process and its related physics.

3. **The process layer:** Processes are the main working components of the simulation that take packages of mass in, process them over time, and output new packages. The `Process` class contains the specific transformations such as irradiation, enrichment, ... and is completely isolated from the mass-dispatch algorithm.

By separating the solver logic from the process definitions, ANICCAv2 allows for modular construction of NFC scenarios, as well as modular upgrades of the code. Possible extensions range from integration in optimization pipelines, customized dispatch algorithms such as reinforcement-learning-based ones, addition of new processes, process-specific constraints on material flows, ...

This part focuses on the first two layers (network and facility) and is organized into four primary sections. First, Section 7 introduces the ANICCAv2 simulation code and its main modeling approach. Next, Section 8 describes the internal logic of the `Facility` class, which serves as the interface between the global solver and the local physical transformations. Building upon this framework, Section 9 presents the mathematical model of each type of process that can be implemented, as well as their hypotheses and limitations. With the main components defined, Section 10 details the currently implemented dispatch algorithm responsible for the movement of masses between facilities and the demand propagation.

7 ANICCA

ANICCAv2 is a simulation code that allows running simulations of NFC scenarios. A scenario represents a specific nuclear fuel cycle configuration, with specific facilities such as mines, enrichment plants, and reactors and the evolution of the mass transfers between them over time. Each simulation relies on a dedicated configuration file that prescribes all the details of a scenario.

Section 7.1 explains the main inputs of an ANICCAv2 simulation. Then, Section 7.2 explains how ANICCAv2 can be customized using user-defined custom classes. Section 7.3 explains how

ANICCAv2 can handle different time scales (months, years, centuries, and beyond) in a single simulation. Finally, Section 7.4 defines the formal representation of a nuclear fuel cycle as a directed multigraph.

7.1 Configuration file

In ANICCAv2, scenarios are described via a `.yaml` configuration file. This file contains four main sections: `scenario`, `components`, `connections`, and `viewers`. The specific syntax of the scenario description is out of the scope of this thesis.

The `scenario` section contains the global parametrizations of the scenario; most notably, its starting date, the sequence of time steps, and the `nuclear_data` subsection. In this subsection, physical parameters related to nuclear physics are prescribed. These comprise whether to model radioactive decay or spontaneous fission, the list of fuel libraries required in the scenario, and the configuration of the list of nuclides to keep track of. Finally, the customization section explained in Section 7.2 is written in this section.

The `components` section describes all the facilities to model in the scenario and all of their parameters. Each facility contains the processes that will request and process masses at different times in the simulation. The `connections` section lists all the connections between each facility (source, target, and type of mass that is transported). Finally, the `viewers` section contains all the data that the user wishes to retrieve from the simulation. Notably, this can be the evolution of facility inventories (by type of mass or by nuclide composition), the inventory of a facility at a given time, the evolution of the flow in connections, ... The format of these outputs can be `.json` or `.xlsx` Excel files, as well as plots.

7.2 Customization

Various features of ANICCAv2 can be customized via a user-defined custom class. For instance, the dispatch algorithm presented later on in this thesis is an implementation of the `MassDispatcher` class that can be overwritten by the user using a custom-made implementation of the same abstract class. In the remainder of the thesis, every feature that can be customized will be explicitly mentioned. Other important examples of customizable classes that will be explained later on include the `DecayEngine` used to handle radioactive decay and the `IrradiationEngine` used to handle fuel irradiation.

7.3 Time discretization

ANICCAv2 is a discrete-time simulator. Not only can it simulate time steps of various durations (1 step corresponds to x months or years[1]), but it can also simulate multiple distinct durations in a given scenario. For instance, the user might wish to simulate 300 years with a yearly time step and then to simulate the decay of obtained materials after hundreds, thousands, or millions of years.

The exact sequence of time steps to follow is listed in the configuration file. Before the simulation begins, these are preprocessed. For modularity and clarity reasons in the code, it was decided that the main dispatch algorithm should work in *unitless* time steps, and therefore be agnostic of the duration of each time step. During the configuration preprocessing, each

[1]For NFC simulations, monthly time steps are sufficient to accurately represent all relevant nuclear processes. Finer discretizations (e.g., weekly or daily) therefore provide little practical benefits. ANICCAv2 consequently adopts one month as its smallest acceptable time step.

duration (in the parametrization of the facilities) is therefore converted into a *number of time steps* units. Each duration that lands between two simulation time steps is rounded up. For this system to work, it is required that all active facilities span durations comprised of a single common time step duration. For example, it means that a facility cannot activate when steps of one month are used, and then continue to be active while the steps transition to durations of one year.

The only phenomena that require the knowledge of a time step duration are radioactive decay and irradiation in reactors. The former is handled independently from the dispatch algorithm and from the process system and therefore has access to the step durations. For the latter, since it is process-specific information, it is also handled independently from the dispatch algorithm.

Conversely, from the dispatcher perspective, all scenarios begin at $t = 0$ (no matter the initial date), and each transition from one time step to the next is done by incrementing t by 1. In this part of the thesis, transitions from one time step to the next can therefore be designated by $t \to t + 1$, regardless of the actual step duration.

7.4 Scenario representation

In order to determine the mass flows between the different facilities at each time step, a scenario is first represented internally as a directed multigraph, denoted by $\mathcal{G} = (\mathcal{F}, \mathcal{C})$, with both $\mathcal{F}$ and $\mathcal{C}$ being fixed sets created from the scenario configuration file. In ANICCAv2, the graph is implemented and manipulated using the NetworkX Python package [21].

- $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ is the set of nodes, with each node representing a facility.
- $\mathcal{C}$ is the set of directed edges, with each edge representing a connection.

The fundamental unit of mass transfer in ANICCAv2 is called a *package*. A package p is defined by its total mass m, its normalized (by mass) nuclide composition vector $\mathbf{w}$, and an arrival time $t^{arrival}$:

$$p = (m, \mathbf{w}, t^{arrival}). \tag{23}$$

The time $t^{arrival}$ corresponds to the time at which a package arrived at a given facility. As explained in the previous section, the mass dispatch algorithm operates in unitless time steps. $t^{arrival} \in \mathbb{N}$ is therefore internally stored as a number of time steps, after conversion. It is important to note that the masses in ANICCAv2 are measured in mass of HM, which for instance excludes the mass of oxygen when talking about the mass of UOX fuel (which is composed of UO_2 pellets). Therefore, the mass of each package is measured in *tons of HM*.

A connection $c \in \mathcal{C}$ is a directed edge that allows the movement of a single package type from one facility to another. Each connection is defined by the tuple:

$$c = (f^{src}, f^{trg}, p[t], \tau, m^{req}[t]) \tag{24}$$

where f^{src} is the source facility, f^{trg} is the target facility, $p[t]$ is the package transferred through this connection at a given time step, τ represents the specific package type the connection is allowed to carry, and $m^{req}[t]$ indicates the mass of the package that is requested to go through this connection at a given time step.

In the next sections, the following notation convention will be adopted. x_y designates *the "x" attribute of the "y" object*. For instance, m_p designates the mass of package p. Similarly, m_{p_τ}

designates the mass of the package p of type τ, f_c^{trg} designates the target facility of connection c, and so on.

8 Facility

A facility serves two main roles. First, it handles the dispatching of the packages in its inventory to processes requiring them. It also orchestrates the actual execution of processes. Second, it interfaces with the solver and the connections attached to it. It should be noted that a facility is characterized by a single inventory, shared by all its processes.

This section details the main characteristics of facilities. First, the different types of facilities available in ANICCAv2 are described in Section 8.1. Then, Section 8.2 describes how processes are orchestrated and run using states.

8.1 Cycle type

Each facility is characterized by a cycle type that all processes of the facility inherit. Since the cycle type plays an important role in the structure of $\mathcal{G}$ and in the dispatch algorithm, it is fixed for the whole duration of a simulation.

The different cycle types currently implemented are the following:

1. `fixed_demand` (FD): at each time step, the mass of every package requested and pushed by the facility is known. This is typically the case for reactor fleets whose electricity production (and therefore fuel consumption and spent fuel production) is known and read from a history file.

2. `on_demand` (OD): the facility scales its production to a requested demand (computed by the mass dispatcher), thereby dictating the mass requested of all of its input packages. This is typically the case for fuel fabrication facilities that need to deliver fuel to downstream reactors.

3. `variable_demand` (VD): the facility does not enforce a specific package intake but requests only what is available, limited by the underlying processes capacity. For example, this cycle type can be used for reprocessing facilities that process spent fuel packages which are sent to it.

4. `none` (N): the facility does not actively produce any package nor actively request any. This is the default cycle type, used for instance for storage facilities which do not run any processes.

Let $\mathcal{F}_x \subseteq \mathcal{F}, x \in \{FD,\ OD,\ VD,\ N\}$, be the set of facilities of a specific cycle type.

8.2 State

Each facility f keeps track of n states $s_i,\ i \in [1, n]$ of a given duration, followed in a specific order. It is the state itself that contains the processes to run at each time step. This allows, for instance, for the *activation* of a facility at a given time (when a state with processes is active) and its deactivation after the working state ends (followed by no state or a state without processes).

States are characterized by a residence time $t_{s,res,\tau}$ that gives the minimal time that a package of type τ has to stay in the facility inventory before moving out. By default, it was arbitrarily chosen that the residence time of a package type is infinite, which means that packages of the given type will never go out of their current facility.

9 Process

Processes are responsible for transforming input packages into output packages and interacting with their facility inventory.

At the level of the facility, processes are handled as black-box objects. However, depending on the cycle type, different process sub-methods exist in order to change or add capabilities to processes.

First, the general model of a process is described in Section 9.1. Sections 9.2 to 9.4 detail the specific modeling of FD, OD, and VD processes. Cycle type N having no particular logic associated with it, there is no section dedicated to it. Finally, Section 9.5 explains how to link all processes of a parent facility in a common interface, defined at the level of the facility.

9.1 General model

A process is characterized by a black-box function that receives packages and outputs processed packages. Let $\mathcal{T}$ be the set of defined package types τ. Let $\mathcal{T}_{P,in}$, $\mathcal{T}_{P,out} \subseteq \mathcal{T}$ be the sets of, respectively, input and output packages of the process designated by the index $_P$.

The transformation at timestep t is defined as

$$\mathcal{P}_{P,out}[t] = \phi_P(t, \mathcal{P}_{P,in}[t], m_{P,out}[t]) \tag{25}$$

where $\mathcal{P}_{P,in}[t]$ and $\mathcal{P}_{P,out}[t]$ are, respectively, the input and output packages of the process and are defined as follows:

$$\mathcal{P}_{P,in}[t] = \{p_\tau \mid \forall \tau \in \mathcal{T}_{P,in}\} \tag{26}$$

$$\mathcal{P}_{P,out}[t] = [p_1, p_2, \dots] \quad \text{with} \quad \tau_{p_i} \in \mathcal{T}_{P,out} \tag{27}$$

and $m_{P,out}$ is the output mass of the package type driving the demand of P among all of its output packages in $\mathcal{P}_{P,out}[t]$. $\mathcal{P}_{P,in}[t]$ fixes the masses of $\mathcal{P}_{P,out}[t]$ by mass balance. Nevertheless, $m_{P,out}[t]$ is still required in specific situations such as when a process acts as a source and therefore has no required input packages.

In Equation 26, the set notation implies that each τ maps to exactly one p. In contrast, Equation 27 defines $\mathcal{P}_{P,out}[t]$ as a list of packages that can share the same τ. This distinction is specifically useful for reactors that can output multiple spent fuel packages at a given time step.

9.1.1 Processing time

Equation 25 assumes that input packages are instantaneously converted to output packages. However, for processing tasks such as fuel fabrication or reprocessing, a non-negligible processing time is required. This is modeled by adding a new parameter to the process such that Equation 25 becomes

$$\mathcal{P}_{P,out}[t + t_{P,prcs}] = \phi_P(t, \mathcal{P}_{P,in}[t], m_{P,out}[t]) \tag{28}$$

where $t_{P,prcs} \in \mathbb{N}$ denotes the processing time of the process (converted in a number of time steps) that delays the production of $\mathcal{P}_{P,out}$ from t to $t + t_{P,prcs}$.

9.2 Fixed Demand (FD) process

The masses requested by FD processes are known for each t. Therefore, FD processes possess a dedicated method that gives the requested quantity of each package type at a given time t. This method has the following form:

$$\mathcal{M}_{P,in}[t] = \psi_P(t) \tag{29}$$

where $\mathcal{M}_{P,in}[t] = \{m_{p_\tau} \mid \tau \in \mathcal{T}_{P,in}\}$, with the notation m_p designating the mass of a given package p. As it will play an important role later, it should be noted that Equation 29 is state-independent since it only depends on t and not on any facility inventory.

9.3 On Demand (OD) process

Since OD processes scale their production to a given requested demand, they have to be able to compute the requested masses of each of their input packages, based on a specified demand. This method has the following form:

$$\mathcal{M}_{P,in}[t] = \theta_P(m_{P,out}[t]) \tag{30}$$

with $\mathcal{M}_{P,in}[t]$ as in Equation 29 and $m_{P,out}[t]$ as in Equation 25 and which is the driving demand of the process. Although an OD process can have more than one output package type, only a single one can actively drive the demand, in order to avoid incompatible scenarios in which two requested demands cannot be exactly fulfilled at the same time.

Because OD processes are the only processes capable of actively propagating package demand upstream (through θ_P), OD processes have to be directly upstream of FD processes. As for Equation 29, Equation 30 is state-independent.

9.3.1 Composition-Dependent On Demand (CDOD) process

Equation 30 supposes that the required mass is independent from the actual incoming package composition. However, in some situations, this is not the case. For example, UOX fuel produced from reprocessed uranium has to increase the ^{235}U content in order to compensate for the presence of ^{236}U in the input stream, which is a neutron absorber. In order to deal with such processes, the following function, similar to Equation 30 but with an additional dependency, can also be written:

$$\mathcal{M}^*_{P,in}[t] = \theta^*_P(m_{P,out}[t], I_{upstrm}[t]) \tag{31}$$

where $I_{upstrm}[t]$ is defined as

$$I_{upstrm}[t] = \{(\tau, I_f[t]) \mid \forall \tau \in \mathcal{T}_{P,in}\}, \tag{32}$$

linking each package type requested by the process to the inventory of the facility supplying this package. The dependency on inventory in Equation 31 makes such a process state-dependent. This will have implications in the possible combinations of processes that will be discussed in the section related to the mass dispatch algorithm in Section 10.3.1. In the following sections, these processes will be referred to as Composition-Dependent On Demand (CDOD) processes.

9.4 Variable Demand (VD) process

VD processes do not enforce a specific request and only request packages that are available, up to their processing capacity. Therefore, each VD process possesses a capacity $c_{P,max}$, giving an upper bound on the mass of each input package that can be processed through $\phi_P(\cdot)$ in a single time step.

Whereas a capacity constraint could also be implemented to OD processes, it would require planning production in advance in the case that $c_{P,max}$ is not sufficient to fulfill $m_{P,out}$ in a single time step. Although this functionality is planned for future versions of the code, it is not currently supported.

9.5 Relation to the parent facility

As explained earlier, multiple processes can exist concurrently in a given facility. In addition, the solver will solely interact with endpoints on facilities. The following functions, therefore, serve as bridges between the solver and the processes.

9.5.1 Get all required inputs and outputs

Let us first define the set of masses requested to be produced at a facility f by

$$m_{f,out}[t] = \{m_c^{req} \mid \forall c \in \mathcal{C}_f^{out}\}, \tag{33}$$

which represents all the masses requested by the outbound connections of f. Let us note that multiple connections can request the same package type. In such a case, their masses are simply added to one another. The set $m_{f,out}[t]$ then essentially represents a mapping from package types τ to their respective requested masses m. The package type corresponding to a requested mass m will therefore be denoted as τ_m.

At this step, from Equation 33, the total required masses to be outputted $\mathcal{M}_{f,out}[t]$ is simply

$$\mathcal{M}_{f,out}[t] = m_{f,out}[t] \tag{34}$$

The total required masses of a facility $\mathcal{M}_{f,in}[t]$ can be expressed as

$$\mathcal{M}_{f,in}[t] = \{\mathcal{M}_{P,in}[t] \mid \forall P \in \mathcal{P}_f\} \tag{35}$$

$$= \left\{ \begin{cases} \psi_P(t) & \text{if } f \in \mathcal{F}_{FD} \\ \{\theta_P(m),\ m \in m_{f,out}[t] : \tau_m \in \mathcal{T}_{P,out}\} & \text{if } P \in \mathcal{P}_{OD} \\ \emptyset & \text{if } P \in \mathcal{P}_{CDOD} \\ \emptyset & \text{if } f \in \mathcal{F}_{VD} \cup \mathcal{F}_N \end{cases} \middle| \ \forall P \in \mathcal{P}_f \right\}. \tag{36}$$

The computation of the tuple $(\mathcal{M}_{f,in}[t], \mathcal{M}_{f,out}[t])$ is denoted by

$$(\mathcal{M}_{f,in}[t], \mathcal{M}_{f,out}[t]) = \Omega_f(t, m_{f,out}) \tag{37}$$

By returning an empty set $\emptyset$, the formulation above excludes CDOD processes ($\theta_P^*(\cdot)$) by design, because these will be handled differently by the solver. VD processes return an empty set as well even though they receive packages from upstream facilities, because $\mathcal{M}_{P,in}[t]$ represents the set of requests that *has* to be filled. Since the demand of VD processes is not a *strict* demand, it is also handled differently by the solver. N processes return an empty set as well since they

do not have any requested inputs. As explained previously, in Equation 36, the case where multiple processes of a facility can fulfill the demand $m \in m_{f,out}$ is ambiguous. There should also always be a single package type driving the demand of a process. Both cases are, therefore, forbidden. Regarding the latter case, an example would be when a uranium enrichment facility would receive requests for both the depleted and enriched uranium produced.

In addition, Equation 37 only allows us to capture instantaneous demand and ignore the presence of any processing time on the processes of f. Indeed, $\mathcal{M}_{P,in}[t]$ as defined previously only gives the masses required to compute a given quantity of output and ignores any potential timing involved. Therefore, Ω'_f is also defined in order to override the required in/outputs for deferred demands:

$$(\mathcal{M}'_{f,in}[t], \mathcal{M}'_{f,out}[t]) = \Omega'_f(t, B, (\mathcal{M}_{f,in}[t], \mathcal{M}_{f,out}[t])) \tag{38}$$

with

$$\mathcal{M}'_{f,in}[t] = \left\{ \begin{cases} \mathcal{M}_{P,in}[t] & \text{if } t_{P,prcs} = 0 \\ B[t + t_{P,prcs}]_{f,P,in} & \text{if } t_{P,prcs} \neq 0 \end{cases} \middle| \; \forall P \in \mathcal{P}_f \right\} \tag{39}$$

and

$$\mathcal{M}'_{f,out}[t] = \left\{ \begin{cases} m_{P,out}[t] & \text{if } t_{P,prcs} = 0 \\ B[t + t_{P,prcs}]_{f,P,\tau,out},\ \forall \tau \in \mathcal{T}_{P,out} & \text{if } t_{P,prcs} \neq 0 \end{cases} \middle| \; \forall P \in \mathcal{P}_f \right\}. \tag{40}$$

In Equation 40, $m_{P,out}[t]$ is defined using the following equation:

$$m_{P,out}[t] = m \in \mathcal{M}_{f,out}[t] : \tau_m \in \mathcal{T}_{P,out}. \tag{41}$$

Again, it should be noted that per process, $m_{P,out}$ is always unique, since each process has a distinct package type driving its own demand.

For any process with a processing time $t_{P,prcs} \neq 0$, Equation 38 returns the demand required at time t in order to fulfill a demand at time $t + t_{P,prcs}$. This information comes from a buffer B, the determination of which will be explained in the solving algorithm section. For other processes, Equation 38 naturally returns the same as what Equation 37 would have returned.

9.5.2 Get all produced packages

$\mathcal{P}_{f,out}[t]$, using Equation 43, gives all the packages produced by a facility. For processes with $t_{P,prcs} \neq 0$, packages are consumed at t, but the corresponding output packages are produced at $t + t_{P,prcs}$.

$$\mathcal{P}_{f,out}[t] = \{\mathcal{P}_{P,out}[t] \mid \forall P \in \mathcal{P}_f\} \tag{42}$$

$$= \{\phi_P(t, \mathcal{P}_{P,in}[t], m_{P,out}[t]) \mid \forall P \in \mathcal{P}_f\} \tag{43}$$

For each process, $\mathcal{P}_{P,in}[t]$ is built by extracting $\mathcal{M}_{P,in}[t]$ from the facility inventory $I_f[t]$. It should be noted that the current behavior is to treat inventories as First-In-First-Out (FIFO) queues. This behavior is planned to be customizable in future upgrades of the code, for example with the possibility of extracting packages based on the specific composition of each package. $\mathcal{M}_{P,in}[t]$ directly comes from $\mathcal{M}_{f,in}[t]$. For VD processes, the mass to be extracted is given by

$$\mathcal{M}_{P_{VD},in}[t] = \{\min(c_{P,max},\ m_{\tau,tot,I_f}[t]) \mid \forall \tau \in \mathcal{T}_{P,in}\} \tag{44}$$

where $m_{\tau,tot,I_f}[t]$ is the total mass available in the facility inventory. Regarding $m_{P,out}[t]$, it is the same as defined in Equation 41.

Since Equation 43 is computed using $\mathcal{M}_{f,in}[t]$ and $\mathcal{M}_{f,out}[t]$, it can eventually be rewritten as

$$\mathcal{P}_{f,out}[t] = \Phi_f(t, \mathcal{M}_{f,in}[t], \mathcal{M}_{f,out}[t]). \tag{45}$$

9.5.3 Recompute composition-dependent requested inputs

As explained in the previous section, CDOD process demand propagation is skipped in Equation 36. Therefore, at some point, the required inputs of these processes will need to be computed, and the subsequent set of requested masses $\mathcal{M}_{f,in}[t]$ updated by these values.

For this, let us define $\mathcal{M}^*_{f,in}[t]$ as the requested masses of CDOD processes of the given facility. It can be expressed as

$$\mathcal{M}^*_{f,in}[t] = \{\mathcal{M}^*_{P,in}[t] \mid \forall P \in (\mathcal{P}_f \cap \mathcal{P}_{CDOD})\} \tag{46}$$

$$= \{\theta^*_P(m_{P,out}[t], I_{f'}[t]) \mid \forall P \in (\mathcal{P}_f \cap \mathcal{P}_{CDOD})\} \tag{47}$$

with $m_{P,out}[t]$ being the same as in Equation 41. Eventually, the computation of $\mathcal{M}^*_{f,in}[t]$ is written as

$$\mathcal{M}^*_{f,in}[t] = \theta^*_f(\tau, \mathcal{M}_{f,out}[t], I_{f'}[t]). \tag{48}$$

10 Mass dispatch algorithm

This section presents the algorithm developed to compute the mass transfers between all facilities at each time step. As mentioned in Section 7.2, the mass transfer is handled by an implementation of the `MassDispatcher` class. It should be noted that each algorithm may require specific properties on the scenario graph $\mathcal{G}$. In this section, the default mass dispatch algorithm currently implemented in ANICCAv2 is presented.

First, the mass dispatch problem is formally defined in Section 10.1. The choice of a sequential modular approach over an equation-oriented one for the dispatch algorithm is justified in Section 10.2, followed by a presentation of the general solving strategy. Finally, Section 10.3 gives a detailed description of the algorithm implemented.

10.1 Problem definition

The state of the nuclear fuel cycle at any discrete time t is characterized by the distribution of materials across all facilities and connections. A fuel cycle graph $\mathcal{G}$ is considered *solved* for a given time step once two conditions are met:

1. The mass and nuclide composition of every package $p_c[t]$ traversing the set of connections $\mathcal{C}$ are determined.

2. The internal inventories $I_f[t]$ of all facilities $f \in \mathcal{F}$ are updated and known.

This problem is defined as the *mass dispatch problem*. It requires determining the movement of material (where and how much) based on the demands and constraints of the individual facilities. The objective is to solve the system of equations representing the state transition at each time step, as formulated in Equation 49:

$$\text{Find: } \begin{cases} \{p_c[t] & | \, \forall c \in \mathcal{C}\} \\ \{I_f[t] = [p_{f,n}] & | \, \forall f \in \mathcal{F}, n \in \mathbb{N}\} \end{cases} \tag{49a}$$

$$\text{Given: } \begin{cases} \{I_f[t-1] & | \, \forall f \in \mathcal{F}\} \\ \{\mathcal{M}_{f,in}[t] & | \, \forall f \in \mathcal{F}_{FD}, \, \forall t\} \end{cases}. \tag{49b}$$

10.2 Solving approach

Systems such as the one described above, where a process network is represented as a graph of interconnected units exchanging material streams, have been widely studied in the context of chemical process simulation [22]. Two solving strategies from this field can be adopted: the Equation-Oriented (EO) approach and the Sequential Modular (SM) approach.

An EO approach [22] treats the entire system as a single system of equations, requiring each facility behavior to be written as explicit equations that can be assembled into a global system. This is only possible for *white-box* facilities whose internal logic is fully known and kept track of by the dispatcher. In ANICCAv2, however, facilities are intentionally treated as *black-box* components: each exposes only an input/output interface (as seen in the previous section). These modeling choices are therefore incompatible with an EO approach.

Conversely, ANICCAv2 implements a Sequential Modular (SM) approach [22], in which each facility is treated as a black-box object and solved individually in a defined order, with outputs of one facility being passed as inputs to downstream facilities. This architecture is particularly suited for modeling local facility logic, such as a facility recomputing its input requirements based on the input stream composition or composition-related constraints, which is more intuitively handled through sequential graph traversal than via global formulations. In chemical processes (and advanced NFC scenarios), recycling loops are common, preventing the SM approach from iterating over the facilities in order. In conventional SM solvers, loops are typically handled through stream *tearing* followed by iterative convergence: an initial guess for a given stream (the *tear stream/edge*) is chosen and propagated forward, compared to the new value computed at the end of the (now broken) loop, and the procedure is repeated until convergence.

To avoid such iterative procedures, the solving algorithm presented in this section requires the scenario graph $\mathcal{G}$ to be acyclic, making $\mathcal{G}$ a DAG. However, real-world fuel cycles often involve recycling loops, e.g., when spent fuel is reprocessed and the reprocessed uranium and plutonium are reused. In such cases, the scenario graph is not acyclic. Still, one can take advantage of the specific structure of most NFC scenarios in order to tear each cycle and get back to an acyclic graph. This way, no iterative procedure is needed, and the mass dispatch problem can be solved using a two-pass approach. To understand how the scenario graph is solved at each time step and how the tearing process is handled, let us analyze the problem in two different cases, depending on the (a)cyclic property of $\mathcal{G}$.

10.2.1 Acyclic graph

In general, the output of a facility can only be computed once all its input connections are known. Solving the graph by propagation of the facilities therefore implies a specific order: each facility can be solved because all its parents are solved. This is typically the order given by a topological sort, which can only exist in DAGs [13]. In such a case, the facilities of the

graph can be sorted in topological order. The algorithm used in NetworkX package can be found in [23], as per their documentation.

Figure 1 illustrates a simple scenario comprising an acyclic graph, with the facilities (blue nodes) sorted in topological order from left to right.

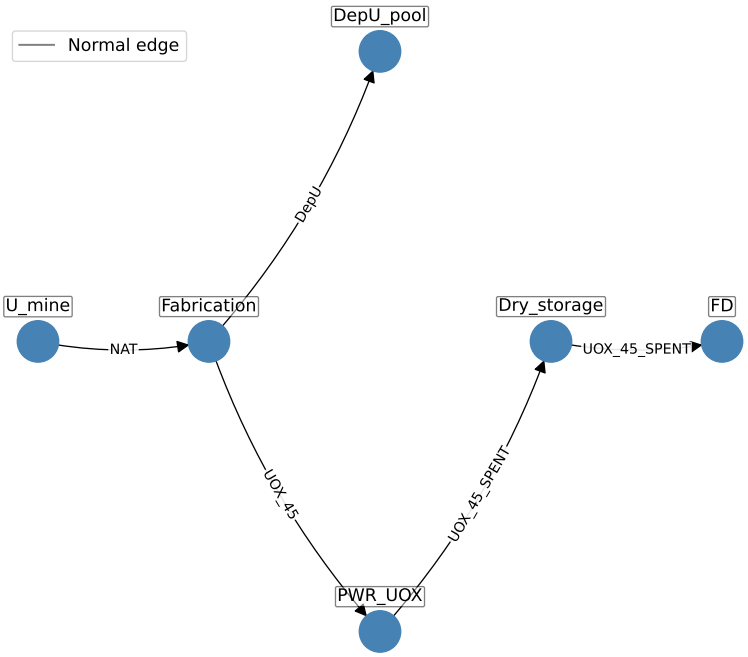


Figure 1: Simple scenario without any cycle.

Once the sorting order has been determined, the graph can be solved using a pull/push strategy.

In the *pull* phase (later referred to as *backward pass*), the demands (requested masses) of each facility are propagated to upstream facilities. This upstream propagation is equivalent to looking at the "reverse graph" (same topology but with all edges inverted). Traversing the reverse graph in topological order is equivalent to traversing the actual graph in inverse topological order.

In the *push* phase (*forward pass*), packages are actually moved, and the exact nuclide compositions and masses are determined in each connection, propagating downstream following the topological order.

10.2.2 Cyclic graph

The fundamental issue of cycles is that they make, for a given facility of the cycle, the input dependent on the output. However, in the case where an FD facility is present in the loop, this no longer holds since the inputs of a FD facility are known *a priori*. Therefore, all cycles containing an FD facility can be *torn* in order to get an acyclic graph back. In addition, the output of an FD facility being directly dependent on its known input, it is independent of downstream demand, whereas the demand on the input side has to be propagated upstream through the backward pass. It is therefore more convenient to tear the output connection of each FD facility not to interfere with the backward pass. The then called *tear edge* will still need to be kept track of for mass transfers during the forward pass.

Another type of cycle can also be torn, in the case of a cycle involving two facilities. In the

case where the output connection of one of the facilities is neither consumed by nor sent out of the other facility, it can safely be torn. Lastly, since a facility of type *none*, acting as a storage, does not propagate demand upstream, its input connection(s) can also be torn in case it is part of a cycle.

Figure 2 illustrates a more complex scenario comprising cycles. It shows the tear edges in red, showcasing the tearing algorithm yielding an acyclic graph (when discarding the red edges). Both `UOX_45_SPENT` edges are torn in order to break a recycling loop, whereas `DepU` is torn to avoid a two-facility loop with unused package.

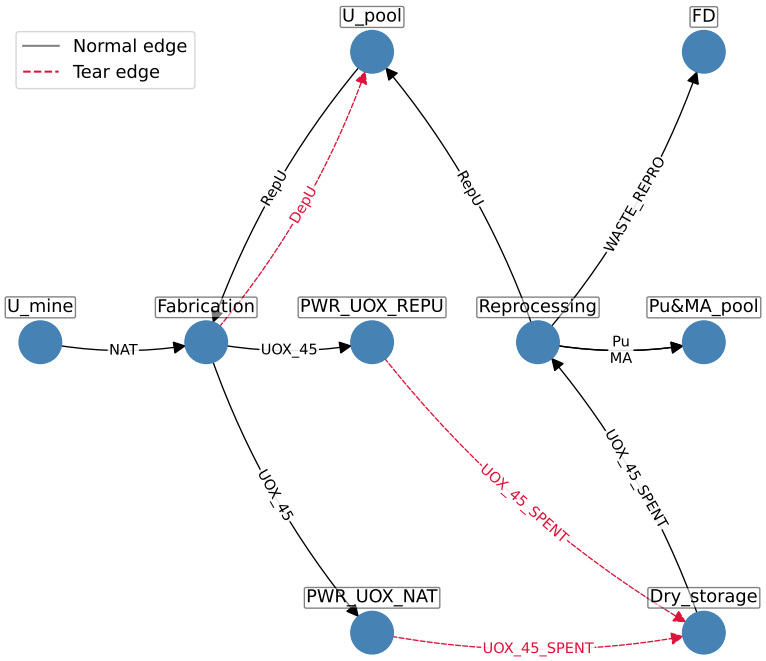


Figure 2: Scenario with recycling loops.

10.3 Algorithm

This section explains all the algorithmic building blocks required in order to obtain the final solving algorithm.

First, the algorithms related to the backward and forward passes are presented respectively in Section 10.3.1 and Section 10.3.2. Then, Section 10.3.3 presents the complete mass dispatch algorithm for a given time step.

10.3.1 Backward pass

The backward pass consists of looping over the inverse topological order of facilities and propagating all required package production upstream of each facility.

First, all the pending demands on the outbound connections of f are collected in $m_{f,out}$ using the requested mass attribute m_c^{req} of each connection c. Then, the inputs required to produce a given quantity $m_{f,out}$ are given either by $\Omega_f(\cdot)$ in the case where all processing times of the facility processes are 0, or by $\Omega'_f(\cdot)$ otherwise. Finally, the masses are propagated upstream to the inbound connections through the use of m^{req}.

In the case where multiple connections are able to supply $m_{\tau,in}$, a splitting strategy `Split` is applied to share the load on all fitting connections. By default, $m_{\tau,in}$ is shared equally. It

should however be noted that as stated in Section 9.3.1, there can only be one source facility for each package type requested by CDOD facilities. More splitting strategies are planned to be added in the future. Moreover, a user can provide a custom `Split` implementation for a scenario, using the customization section of the scenario configuration file, as explained in Section 7.2.

The exact algorithm for the backward pass is given in Algorithm 1.

Algorithm 1: Backward pass algorithm.

Function `backward_pass`(t)
  foreach $f \in$ `ReversedSort`$(\mathcal{F})$ **do**
    $m_{f,out,\tau} \leftarrow 0, \quad \forall \tau \in \mathcal{T}$
    foreach $c \in \mathcal{C}_f^{out}$ **do**
      m_{f,out,τ_c} += m_c^{req}
    end
    $(\mathcal{M}_{f,in}, \mathcal{M}_{f,out}) \leftarrow \Omega_f(t, m_{f,out})$
    if $\exists t_{P,prcs} \in P_f : t_{P,prcs} \neq 0$ **then**
      $(\mathcal{M}_{f,in}, \mathcal{M}_{f,out}) \leftarrow \Omega'_f(t, B, (\mathcal{M}_{f,in}, \mathcal{M}_{f,out}))$
    end
    foreach $(P,\, \mathcal{M}_{P,in}) \in \mathcal{M}_{f,in}$ **do**
      foreach $(\tau, m_{\tau,in}) \in \mathcal{M}_{P,in}$ **do**
        $\mathcal{C}_{match} \leftarrow \{c \in \mathcal{C}_f^{in} \mid \tau_c = \tau\}$
        $mass\_allocation \leftarrow$ `Split`$(m_{\tau,in}, \mathcal{C}_{\text{match}})$
        foreach $(c, \delta m) \in mass\_allocation$ **do**
          m_c^{req} += δm
        end
      end
    end
  end
end

As already mentioned in Section 9.5, Ω'_f requires to know the future demand of a process at time $t + t_{P,prcs}$ in order to request the corresponding required input mass at t. This is done through the use of a buffer B that stores both the required inputs and output masses of all facilities at future times.

In a simulation with different processing times, at a time step t, the buffer only has to keep track of the deferred demands of times $t' \in \left(t,\, t_{prcs}^{max}\right]$, with

$$t_{prcs}^{max} = \max_{p \in P_f,\, f \in \mathcal{F}} (t_{P,prcs}). \tag{50}$$

In practice, B is implemented as a *rolling buffer*. At every time step, all deferred demands are first computed at time $t + t_{prcs}^{max}$ and stored in B. Then, the values stored in B for time t are discarded since they are no longer useful. Finally, the backward pass is computed using the information provided in the buffer. These steps are illustrated in Figure 3.

This implementation makes it possible, at each time step, to compute deferred demands at a single future time step, even in the presence of multiple processing times. Indeed, deferred demands at a time $t + t_{prcs}$, $t_{prcs} \in (0,\, t_{prcs}^{max})$ were already computed at time $t - t_{prcs}^{max} + t_{prcs}$.

In order to compute the deferred demands at $t' = t + t_{prcs}^{max}$, denoted $B[t']$, an algorithm very

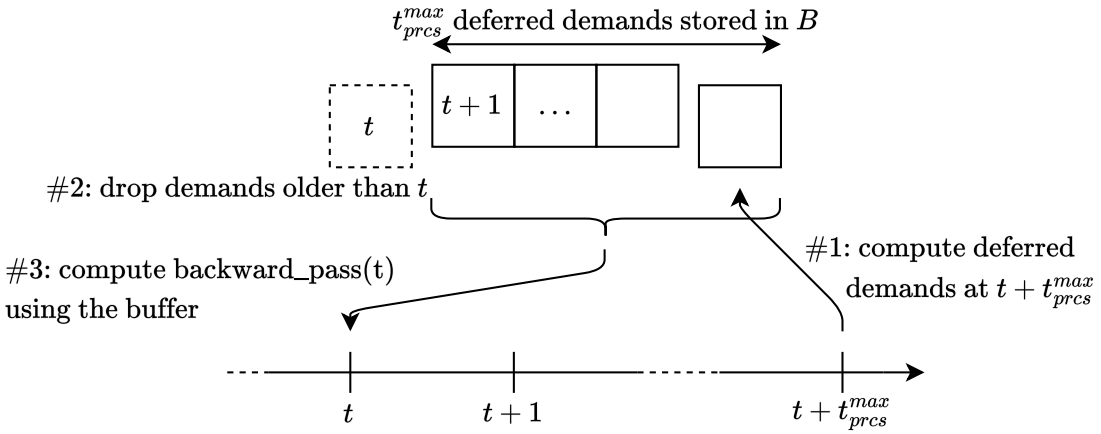


Figure 3: Lifecycle of the rolling buffer update.

similar to the backward pass is used. Indeed, the demands computed consist of propagating demands in the future (such as fuel loading required in the future) in order to predict the required masses, which is essentially what the backward pass performs, albeit at the present time. However, there are a few subtle differences between the two passes.

1. Most notably, since the goal is to predict the current demand (relative to t'), $\Omega_f(\cdot)$ is always used regardless of the presence of a processing time in f processes.

2. Then, since the pass has to be performed in the future, the facilities' states also have to be projected in what they will be at time t'. Therefore, each facility state s_f has to be stored, updated to the future state $s[t']$ and restored before the actual backward pass.

3. Finally, since this pass happens in the future, it should not modify the actual m^{req} values of the connections. A temporary set of connections is therefore used.

The first difference explained above answers a subtle question that might arise regarding the buffer: *can we store all required inputs/outputs in the buffer, and paste the values of B_t at time t instead of discarding them and recomputing the backward pass?* By construction, the actual backward pass computes all required masses requested at time t by looking at the future demand. However, the buffered deferred demand in B_t was computed at $t - t^{max}_{prcs}$, without knowledge of its (relative to $t - t^{max}_{prcs}$) future demand. B_t therefore can not reflect the actual demands at t, but only the demand that needed to be fulfilled in advance at previous time steps.

The algorithm for this backward pass run in the future is given in Algorithm 2.

Algorithm 2: Backward pass algorithm for computing the deferred demands at a future time t'.

Function `backward_pass_future(`t'`)`
  $m^{req}_{c,temp} \leftarrow 0, \quad \forall c \in \mathcal{C}$
  $B[t'] \leftarrow \{\}$
  foreach $f \in$ `ReversedSort(`$\mathcal{F}$`)` **do**
    $has\_deferred\_P\_at\_time\_t \leftarrow \exists\, t_{P,prcs} \in P_f : t_{P,prcs} \neq 0$
    $s_{old} \leftarrow s_f$
    $s_f \leftarrow s[t']$
    $m_{f,out,\tau} \leftarrow 0, \quad \forall \tau \in \mathcal{T}$
    foreach $c \in \mathcal{C}^{out}_f$ **do**
      m_{f,out,τ_c} += $m^{req}_{c,temp}$
    end
    $(\mathcal{M}_{f,in},\ \mathcal{M}_{f,out}) \leftarrow \Omega_f(t', m_{f,out})$
    if $has\_deferred\_P\_at\_time\_t$ **then**
      $B_{P,in},\ B_{P,\tau,out} \leftarrow \mathcal{M}_{f,in},\ \mathcal{M}_{f,out}$
    end
    $s_f \leftarrow s_{old}$
    foreach $(P,\ \mathcal{M}_{P,in}) \in \mathcal{M}_{f,in}$ **do**
      foreach $(\tau, m_{\tau,in}) \in \mathcal{M}_{P,in}$ **do**
        $\mathcal{C}_{match} \leftarrow \{c \in \mathcal{C}^{in}_f \mid \tau_c = \tau\}$
        $mass\_allocation \leftarrow$ `Split(`$m_{\tau,in}, \mathcal{C}_{\text{match}}$`)`
        foreach $(c, \delta m) \in mass\_allocation$ **do**
          $m^{req}_{c,temp}$ += δm
        end
      end
    end
    $B[t']_f \leftarrow [B_{P,in},\ B_{P,\tau,out}]$
  end
  return $B[t']$
end

Initialization of B

The algorithms presented in Algorithm 1 and Algorithm 2 assume that the buffer is already populated, and describe the evolution from a time step to the next. However, at $t = 0$, the buffer B is empty and needs to be populated $\forall t' \in [0,\ t^{max}_{prcs})$.

This is simply done by running Algorithm 2 for all future times in this interval, as shown in Algorithm 3.

Algorithm 3: Algorithm for the initialization of the buffer B.

Function `pre_fill()`
  $B \leftarrow \{\}$
  foreach $t' \in [0,\ t^{max}_{prcs})$ **do**
    $B[t']$ ← `backward_pass_future(t')`
  end
  return B
end

III Simulation Framework

Limitations of CDOD processes

It should be noted that performing the backward pass in a future time such as by using Algorithm 2 is only possible thanks to the *state-independent* property of $\mathcal{M}_{P,in}[t]$ in Equation 29 and Equation 30 (and by extension of $\Omega_f(\cdot)$) that was briefly mentioned in Section 9.3. Indeed, this is because $\Omega_f(\cdot)$ does not depend on the facilities inventories that it can be evaluated at any future time step. CDOD processes, however, break this state-independent property since their demand computation depends on upstream inventories, which can not be known ahead of time. Practically, it means that these processes can not be supplied by other OD processes upstream of them. CDOD processes are typically used for fuel fabrication facilities using reprocessed materials as input. It means that with the current version of ANICCAv2, it is not possible to feed these facilities with OD reprocessing facilities but only using VD ones[2]. Possible ways to alleviate this issue could be investigated as future work.

10.3.2 Forward pass

The forward pass is responsible for actually moving packages from facility to facility. For this reason, it needs to iterate over facilities in topological order, so that all facilities sending packages to a given facility f have run before f is actually executed.

First, let us define $\varphi_f(\tau, t)$ as the mass of packages of type τ available in f at a time t, given by

$$\varphi_f(\tau, t) = \sum_{p_\tau \in I_f[t]} \left(m_{p_\tau} \text{ if } (t - t_p^{arrival}) > t_{s,res,\tau} \right) \tag{51}$$

where s is the current active state of f. Let us also define $\chi_f(f', \tau, m, t)$ as the mass of package of type τ to extract from f and sent to f'. $\chi_f(f', \tau, m, t)$ is expressed using Equation 51 as

$$\chi_f(f', \tau, m, t) = \begin{cases} m & \text{if } f' \in \mathcal{F}_{FD} \\ m & \text{if } f' \in \mathcal{F}_{OD} \\ \min\left(\varphi_f(\tau, t),\ \sum_{P \in \mathcal{P}_f} (c_{P,max} \text{ if } (\tau \in \mathcal{T}_{P,in}))\right) & \text{if } f' \in \mathcal{F}_{VD} \\ \varphi_f(\tau, t) & \text{otherwise} \end{cases} . \tag{52}$$

In Equation 52, the expression reflects the properties of each cycle type. In the case of FD and OD processes, it is assumed that $\varphi_f(\tau, t) \geq m$, since f is *forced* to supply m to f'. In the case of VD, since there is no requested mass, only the amount that can be fully processed is sent (i.e., the minimum between capacity and available mass). Note: this logic does not take into account the current inventory of f' nor the potential existence of other sources of package τ competing with f' capacity[3]. For other cycle types (N), whatever is available is sent since no specific request exists. In future upgrades of the code, it is planned to add customizable conditions to check if each individual package can be sent to f', for example based on their composition.

For each facility f, one first has to get all the packages produced from all inbound connections. The required mass to extract is first computed, and if it is > 0, a corresponding quantity of the given package type is extracted from the source facility and added to f inventory.

[2]This feature was not implemented in ANICCAv1 either. However, ANICCAv2 *can* actually be improved in order to support this use-case.

[3]This would typically be a problem when multiple reactors would want to send their spent fuel to a reprocessing plant. However, the issue can be easily avoided by adding an interim storage collecting all spent fuel between the reactors and the reprocessing plant. Then, there would be a single connection leading to the reprocessing plant.

This computation is based on m^{req} of the inbound connections that was set in the backward pass. However, it should be remembered that the backward pass could not track the requested masses of CDOD processes because the inbound inventories were not yet known. In the forward pass, however, the inventory of each source facility is fully determined since the facilities are processed in topological order. The facility therefore has to (and is able to) update the requested mass of each of its inbound connection in case it hosts CDOD processes. This update can be performed using Equation 48 defined in Section 9.5.3, and has to take place right before computing the packages to send. As mentioned in Section 9.5.3, the set of requested masses also has to be updated using the requested masses of CDOD processes. This update is straightforward, since the set of requested masses is a *per-process* set that was initialized with empty sets for CDOD processes (see Equation 36).

Once all incoming packages have been transferred, the facility can run all of its processes. The obtained packages are then added to the facility inventory.

If any tear edge exists and has f as a source node, because of how the tear edges were selected, there is never any required mass and the target facility can neither be FD nor OD. The mass to send to the connection is therefore given using $\varphi_f(\cdot)$ instead of $\chi_f(\cdot)$.

The algorithm explained above is formalized in Algorithm 4.

Algorithm 4: Forward pass algorithm.

Function `forward_pass`(t)
  foreach $f \in \texttt{Sort}(\mathcal{F})$ **do**
    if $\exists P \in P_f : P \in \mathcal{P}_{CDOD}$ **then**
      $I_{upstream} \leftarrow \{\}$
      foreach $c \in \mathcal{C}_f^{in}$ **do**
        $I_{upstream}[\tau_c] \leftarrow I_{f_c^{src}}$
      end
      $\mathcal{M}_{f,in}^* \leftarrow \theta_f^*(\tau, \mathcal{M}_{f,out}, I_{upstream})$
      foreach $(P,\, \mathcal{M}_{P,in}^*) \in \mathcal{M}_{f,in}^*$ **do**
        $\mathcal{M}_{f,in} \leftarrow \mathcal{M}_{f,in} \cup \mathcal{M}_{P,in}^*$
        foreach $(\tau, m_{\tau,in}) \in \mathcal{M}_{P,in}^*$ **do**
          $c_{match} \leftarrow \{c \in \mathcal{C}_f^{in} \mid \tau_c = \tau\}$
          $m_c^{req} \leftarrow m_{\tau,in}$
        end
      end
    end
    foreach $c \in \mathcal{C}_f^{in}$ **do**
      $m_{p_\tau} \leftarrow \chi_{f_c^{src}}(f,\, \tau_c,\, m_c^{req},\, t)$
      if $m_{p_\tau} > 0$ **then**
        $p_c \leftarrow I_{f_c^{src}} - \{m_{p_\tau}\}$
        $I_f \leftarrow I_f + \{p_c\}$
      end
    end
    $\mathcal{P}_{f,out} \leftarrow \Phi_f(t, \mathcal{M}_{f,in}, \mathcal{M}_{f,out})$
    $I_f \leftarrow I_f + \{\mathcal{P}_{f,out}\}$
    foreach $c,\ f_c^{trg} \in tear\_edges$ **do**
      if $f_c^{src} = f$ **then**
        $m_{p_\tau} \leftarrow \varphi_f(\tau_c,\, t)$
        if $m_{p_\tau} > 0$ **then**
          $p_c \leftarrow I_f - \{m_{p_\tau}\}$
          $I_{f_c^{trg}} \leftarrow I_{f_c^{trg}} + \{p_c\}$
        end
      end
    end
  end
end

The $I_f - \{\cdot\}$ notation means that a given quantity is retrieved from the inventory, while the notation $I_f + \{\cdot\}$ means that a given package is added to the inventory. As mentioned previously, the current package removal ordering is FIFO.

It should be noted that the dispatch of packages from inbound connections to the considered facility f is currently done in a naive way, simply by iterating over the connections and sending packages solely based on the cycle type of f. More sophisticated dispatching techniques are planned in future improvements of the code, such as the possibility of specifying priorities on the connections, or specific selection of packages based on their exact composition.

10.3.3 Final algorithm

With the main steps of the dispatch algorithm formalized, the complete algorithm to solve one simulation time step can be written. It is shown in Algorithm 5, where `backward_pass_future()`, `backward_pass()` and `forward_pass()` respectively refer to Algorithm 2, Algorithm 1, and Algorithm 4.

Algorithm 5: Mass dispatch algorithm for a single simulation time step.

```
1  Function solve_step(t)
2      foreach f ∈ F do
3          s_f ← s[t]
4      end
5      if t_prcs^max > 0 then
6          t' ← t + t_prcs^max
7          B[t'] ← backward_pass_future(t')
8          pop(B, t)
9      end
10     backward_pass(t)
11     forward_pass(t)
12 end
```

Let us note that a call to `pre_fill()` (Algorithm 3) has to be made before the first time step is resolved using Algorithm 5.

Part IV
Nuclear physics modeling

This part details the physical models implemented in ANICCAv2 to simulate the evolution of material compositions within the nuclear fuel cycle. While the core solver manages the macroscopic movement of mass between facilities, the different models handle the actual composition evolution and package transformations. These encompass the nuclear data collection and nuclide tracking, radioactive decay, and the individual processes running in each facility.

This part is therefore structured into four sections. First, Section 11 defines how each package nuclide composition is represented. This is followed by Section 12, which explains the collection of nuclear data and the pre-processing required for compatibility with the set of tracked nuclides in each scenario. Section 13 then describes how radioactive decay is implemented using the Chebyshev Rational Approximation Method (CRAM). Finally, Section 14 provides a comprehensive description of the currently implemented processes.

11 Nuclide composition tracking

As established in the previous sections, each material package p is characterized by a total mass and a nuclide composition vector $\mathbf{w}_p$. For a given package, each element $w_{p,i} \in \mathbf{w}_p$ represents the mass fraction of the nuclide $i \in \mathcal{I}$ relative to the total mass of the package, where $\mathcal{I}$ represents the set of all tracked nuclides in the given scenario.

A nuclide i is defined as *tracked* if it is a member of the set $\mathcal{I}$ and can thus be explicitly assigned a mass fraction. Since the tracking set $\mathcal{I}$ is fixed for a given scenario, a mechanism is required to account for the transformation of a nuclide i into a descendant $i' \notin \mathcal{I}$. In order to respect mass conservation, all *untracked* nuclides are agglomerated into a single dummy nuclide. This virtual nuclide category is assigned a special identifier ι (e.g., a ZZAAAS code of 0) that does not correspond to any physical nuclide. By assigning specific nuclear properties to this dummy nuclide, untracked nuclides can actually be manipulated as a part of $\mathbf{w}_p$ like any other nuclide. Consequently, ι represents the cumulative mass of all nuclides in p that fall outside the scenario-specific tracking set. For every package p, the sum of mass fractions must satisfy the following identity:

$$\sum_{i \in \mathcal{I} \cup \{\iota\}} w_{p,i} = 1 \tag{53}$$

where $\mathcal{I} \cup \{\iota\}$ denotes the set of all tracked nuclides *plus* the untracked mass. Based on this representation, the total mass of a given nuclide $m_{p,i}, i \in \mathcal{I} \cup \{\iota\}$ within a package p of total mass m_p is computed as:

$$m_{p,i} = w_{p,i} \cdot m_p. \tag{54}$$

12 Nuclear data

In order to model decay, irradiation, or extract properties such as radiotoxicity, each nuclide tracked by ANICCAv2 needs to be associated with a set of accurate nuclide physical data and properties. ANICCAv2 uses the same library as ANICCAv1, the JEFF-3.1 nuclear data library [24].

From this database, data such as atomic masses, decay constants λ_i, branching ratios $b_{i\to j}$, and spontaneous branching ratio b_i^{SF} are extracted. In addition, fission product yields $y_{i\to j}$ are extracted in order to model Spontaneous Fission (SF)[4]. Since the dummy nuclide represents the mass of the nuclides that are not tracked, this nuclide is considered stable, and its decay constant is set to 0.

In the following subsections, Section 12.1 explains how the set of tracked nuclides can be set for each scenario. Then, Section 12.2 explains the specific preprocessing of the nuclear data that is necessary to make it compatible with the specific set of tracked nuclides.

12.1 Data filtering

The JEFF-3.1 database contains specifications for thousands of nuclides. Keeping track of all nuclides would hinder the performances of ANICCAv2. Moreover, most nuclides are short-lived and therefore have a negligible impact on the macroscopic mass flows of an NFC analysis. Therefore, ANICCAv2 allows for selective nuclide tracking. In the scenario configuration, the user can provide an explicit list of nuclides to keep track of. Several lists are provided by default in ANICCAv2. The main ones are `origen` [25] (1221 nuclides) and two lists created and validated by SCK CEN for their own research: `nuclides_lfr` (152 nuclides) [26] and `nuclides_ml` (148 nuclides) [27]. Two additional lists `piet` (81 nuclides) [28] and `piet_hm` (34 nuclides) are also provided for development purposes.

One cannot simply discard the untracked nuclides from the nuclear data obtained from the JEFF-3.1 database, since the daughters of untracked nuclides can still contain tracked nuclides. The nuclear data, therefore, needs to be pre-processed in order to, on one hand, make it self-consistent by only referring to tracked nuclides (and the dummy nuclide) and, on the other hand, maintain physical consistency. In practice, this is done by collapsing the decay chains of tracked nuclides and by maintaining an *effective daughters* list of each untracked nuclide.

12.2 Data preprocessing

Before the simulation begins, the raw JEFF-3.1 data undergoes three successive preprocessing steps to produce a consistent transition map for the tracked nuclide set $\mathcal{I}$. First, the daughters of tracked nuclides capable of undergoing SF are updated using available fission product yields. Second, the decay chains of alpha-decaying nuclides are updated in order to keep track of the ^{4}He nucleus emitted during alpha decay in the daughters list. Third, the resulting decay trees are collapsed so that every tracked nuclide maps exclusively to other tracked nuclides or to the dummy sink ι.

12.2.1 Spontaneous fission handling

In the JEFF-3.1 decay data, spontaneous fission is recorded as a decay mode with branching ratio b_i^{SF} but with no associated daughter nuclide, since a single fission event produces more than a single fragment. Consequently, the raw daughters of an SF-capable parent i are incomplete: their branching ratios sum to $1 - b_i^{SF}$ rather than 1.

To account for the SF products prior to chain collapsing, ANICCAv2 augments the daughters of each SF-capable nuclide using fission product yields loaded from a second JEFF-3.1

[4]The fission product yields used for induced fission during fuel irradiation will be extracted from another library.

data file. For each fission product j of parent i, an effective branching ratio

$$b^{SF}_{i\to j} = b^{SF}_i \cdot y_{i\to j} \tag{55}$$

is added to the daughters of i, where $y_{i\to j}$ is the fission yield. If a fission product is not present in the decay data, it is simply discarded.

After augmentation, the branching ratios of an SF-capable parent sum to

$$(1 - b^{SF}_i) + b^{SF}_i \sum_j y_{i\to j} > 1. \tag{56}$$

This is due to the fact that fission produces (in general) two new nuclides from a single parent, which means that the number of atoms increases on average for each nuclide capable of SF.

12.2.2 Alpha decay

The ^{4}He nuclei emitted during alpha decay can accumulate in the matrices of spent nuclear fuels during storage. Since the potential impacts of ^{4}He on the storage integrity are extensively studied [29, 30], one might be interested in keeping track of the amount of helium accumulating in the different spent fuel packages. For this purpose, the daughters of each nuclide capable of alpha decay are updated in order to account for the ^{4}He production, using the branching ratio associated with this decay mode.

In addition to the sole benefit of being able to track ^{4}He buildup, accounting for it in the mass balance also eliminates the mass conservation error due to ignoring the contribution of ^{4}He in the mass of a package. Taking the decay of ^{242}Cm as an example, which decays by alpha decay into ^{238}Pu with a half-life of 163 days, the mass of the alpha particle relative to the nuclide mass is computed as

$$\Delta m \approx \frac{4}{238 + 4} = 1.65\,\%. \tag{57}$$

Without keeping track of ^{4}He or without any other compensation, this mass would be wrongly attributed to the daughter nuclide (^{238}Pu in the example). This error is avoided by explicitly tracking the alpha particle of alpha decay.

12.2.3 Chain collapsing

For any nuclide $i \in \mathcal{I}$ that decays into a daughter $j \notin \mathcal{I}$, the decay chain of j is followed and collapsed until it reaches a tracked nuclide. Practically, this is done by a recursive tree-traversal algorithm applied to every tracked nuclide after updating their daughters list for SF.

For each nuclide $i \in \mathcal{I}$, the algorithm explores its decay tree, i.e., the entirety of all the decay chains of the daughters of i. For any branch originating from a parent nuclide u, the recursion follows the branching ratio $b_{u\to v}$ and behaves according to the status of the daughter v:

1. If $v \in \mathcal{I}$, the branch is terminated, and v is recorded as an *effective daughter* of the original parent i. A contribution for the associated effective branching ratio $b'_{i\to v}$ is added to the existing branching ratio, if any, with

$$b'_{i\to v} = \prod_{(u',v')\in pth} b_{u'\to v'} \tag{58}$$

where $pth = ((i, u_1), (u_1, v_1), (u_2, v_2), \ldots, (u_n, v))$ denotes the followed path leading from i to v.

2. If $v \notin \mathcal{I}$ and v is unstable, the algorithm continues the recursion by exploring the daughters of v.

3. If $v \notin \mathcal{I}$ and v has no daughters, the branch terminates at the dummy nuclide ι with the associated effective branching ratio $b'_{i\to\iota}$ added to the existing one, if any.

At the end of the algorithm, each nuclide $i \in \mathcal{I}$ possesses an effective daughter list mapping each tracked daughter j to an effective branching ratio $b^{eff}_{i\to j}$. The effective branching ratio $b^{eff}_{i\to j}$ is then defined as

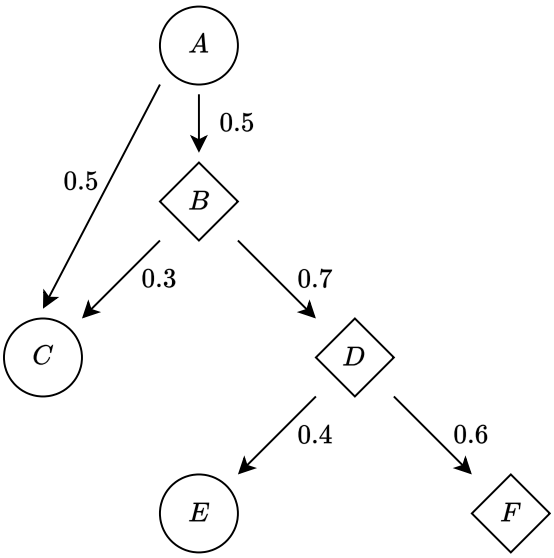
$$b^{eff}_{i\to j} = \sum_{pth\in\mathcal{P}th_{i\to j}} b'_{pth,i\to j} \tag{59}$$

$$= \sum_{pth\in\mathcal{P}th_{i\to j}} \prod_{(u,v)\in pth} b_{u\to v} \tag{60}$$

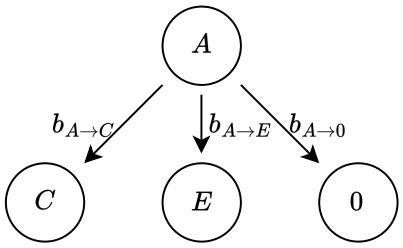
where $\mathcal{P}th$ represents the set of all the paths that eventually lead from i to j in the decay tree traversal of i.

By resolving the entire tree for each nuclide, ANICCAv2 generates a simplified decay tree where every tracked nuclide maps exclusively to other tracked nuclides or to the dummy sink ι. For non-SF nuclides, this ensures the condition $\sum_{k\in\mathcal{I}\cup\{\iota\}} b^{eff}_{i\to k} = 1$; for SF-capable nuclides, the sum is > 1 as discussed above. In both cases, the mass of untracked intermediates is effectively *lumped* into their tracked descendants.

A simplified example of the so-called chain collapsing algorithm is illustrated in Figure 4 in the case of a non-SF-capable nuclide for the sake of simplicity.



(a) Initial decay tree of A and its descendants.



$b_{A\to C} = 0.5 + 0.3 \cdot 0.5$
$b_{A\to E} = 0.5 \cdot 0.7 \cdot 0.4$
$b_{A\to 0} = 0.5 \cdot 0.7 \cdot 0.6$

(b) *Collapsed* decay tree of A after preprocessing, leading to its effective descendants with effective branching ratios, so as to discard all untracked nuclides.

Figure 4: Simplified example of the collapse of the decay tree of a tracked nuclide denoted A. Each round node denotes a tracked nuclide, while each square represents an untracked nuclide.

One should pay close attention to the choice of the tracked nuclides list. As Figure 4 illustrates, all the collapsed intermediary nuclides and their decay constants are discarded so that only the decay constant of the parent nuclide is taken into account. In case the decay constant of the considered nuclide i is not significantly larger than one of its untracked descendants u, i.e. if $\lambda_i \not\gg \lambda_u$, then a large error is introduced by not tracking u and collapsing its decay chain.

Untracked nuclide map

During irradiation, untracked nuclides could be produced as fission products. Therefore, one has to find a way to map each untracked nuclide produced to tracked ones. The way chosen in ANICCAv2 is to instantaneously decay them into their *effective daughters* list. The mapping between untracked nuclides and their effective daughters can be built using the exact same algorithm as the one described above.

13 Radioactive decay

In ANICCAv2, radioactive decay is handled by a dedicated `DecayEngine` class. In its current version, only a CRAM-based engine is implemented with three distinct variants, based on the actual CRAM formula and order implemented.

First, Section 13.1 recalls the theoretical modeling of radioactive decay, followed by an explanation of how the decay matrix is built in ANICCAv2 in Section 13.2. Then, Section 13.3 explains how decay is implemented in the lifecycle of an ANICCAv2 simulation. Finally, Section 13.4 details the implementation of the decay engine in ANICCAv2, which is based on CRAM.

13.1 Formulation

As a reminder, the decay problem can be formulated in matrix form using

$$\frac{\mathrm{d}\mathbf{n}(t)}{\mathrm{d}t} = \mathbf{A}\mathbf{n}(t), \tag{61}$$

whose solution is given by

$$\mathbf{n}(t) = e^{\mathbf{A}t}\mathbf{n}_0 \tag{62}$$

or, in discrete times,

$$\mathbf{n}[t + \Delta t] = e^{\mathbf{A}\Delta t}\mathbf{n}[t] \tag{63}$$

in which $\mathbf{n}[t]$ is the abundance vector at time t and $\mathbf{A}$ is the decay matrix. Before using CRAM in order to approximate $e^{\mathbf{A}\Delta t}$, one has to compute the decay matrix $\mathbf{A}$. It should be noted that $\mathbf{n}[t]$ represents the abundance vector, i.e., a number of atoms. However, the nuclide composition vector $\mathbf{w}_p$ uses mass fractions. One should therefore also convert the mass fractions into actual atom counts before using Equation 63.

The abundance vector $\mathbf{n}[t]$ can be computed from $\mathbf{w}_p$ by

$$n_i[t] = \begin{cases} m_p \cdot w_{p,i}[t]/A_i & \text{if } i \in \mathcal{I} \\ 0 & \text{if } i = \iota \end{cases} \tag{64}$$

with A_i being the atomic mass (tons/atom) of nuclide i. The dummy nuclide content can be set to 0 since it is a non-physical nuclide and, therefore, does not intervene in the decay process.

Then, the new composition is computed using Equation 63. By assuming that no mass is lost in the decay process, the total mass of the package m_p can be considered conserved before and after the decay. From there, the new composition of tracked nuclides $\mathbf{w}_p[t+1]$ can be recovered as

$$w_{p,i}[t+1] = n_i[t+1] \cdot A_i / m_p \quad \text{if } i \in \mathcal{I} \tag{65}$$

whereas the new fraction of untracked nuclides can simply be computed as

$$w_{p,\iota}[t+1] = 1 - \sum_{j \in \mathcal{I}} w_{p,j}[t+1] \tag{66}$$

Finally, one can note that Equation 63 is linear with respect to $\mathbf{n}$, and one can therefore use a normalized version of $\mathbf{n}$, $\mathbf{n}' = \mathbf{n}/m_p$. This way, one avoids the respective multiplication and division by m_p in Equation 64 and Equation 65.

13.2 Decay matrix construction

Building the decay matrix using the preprocessed nuclear data is straightforward.

Each element of the decay matrix $A_{i,j}$ yields the rate of transformation of nuclide j into i. Therefore, a simple pass over the tracked nuclides and their effective daughters is enough to fully build $\mathbf{A}$. This is illustrated in Algorithm 6, where i_x and $daughters(x)$ represent, respectively, the index (i.e. the place in the composition vector $\mathbf{n}$) and the effective daughters of nuclide x.

Algorithm 6: Build the decay matrix $\mathbf{A}$.

Function `build_decay_matrix()`
  $A_{i,j} \leftarrow 0, \quad \forall i,j \in [0, |\mathcal{I} \cup \{\iota\}|) \times [0, |\mathcal{I} \cup \{\iota\}|)$
  foreach $ncld \in \mathcal{I} \cup \{\iota\}$ **do**
    $A_{i_{ncld}, i_{ncld}} \mathrel{+}= -\lambda_{ncld}$
    foreach $dgth \in daughters(ncld)$ **do**
      $A_{i_{dgth}, i_{ncld}} \mathrel{+}= \lambda_{ncld} \cdot b^{eff}_{ncld \to dgth}$
    end
  end
  return A
end

13.3 Integration in scenario simulation

Decay is treated as a per-package operation within the simulation lifecycle. Each package, therefore, needs to be updated between each time step of length Δt, for its decay during a corresponding Δt time.

As packages only reside in facilities inventories between each time step, the solver applies decay to all of them by iterating over facilities between each `solver_step()` call. Notably, the packages that are in production (for processes with a processing time) are also decayed. However, packages representing fuel inside a reactor core for irradiation processes are not decayed in this way, since the decay is modeled alongside the irradiation process itself.

13.4 Decay engine

Decay is applied using an implementation of the `DecayEngine` class. By default, ANICCAv2 provides `CRAMDecayEngine` which uses a CRAM implementation in order to solve Equation 63. Specifically, three subclasses are available, corresponding to three implementations of CRAM:

1. `PFD16DecayEngine` (PFD of order $k = 16$);
2. `IPF16DecayEngine` (IPF of order $k = 16$);
3. `IPF48DecayEngine` (IPF of order $k = 48$).

By default, since it is the recommended implementation for solving decay problems (see Section 6.4.2), `IPF48DecayEngine` is used. It can, however, be overridden by any other class available in ANICCAv2 or by a custom `DecayEngine` implementation, as mentioned in Section 7.2. It should be noted that ANICCAv1 decay computation approach was based on the analytical solution of the Bateman system. However, as explained previously, the problem is stiff in the general case, which makes this approach unreliable and inaccurate.

The next subsubsection is dedicated to an implementation trick used in ANICCAv2 in order to speed up decay computations significantly, leveraging the property that the decay matrix is constant for a given scenario.

13.4.1 LU caching

The CRAM method essentially consists of solving $k/2$ linear systems of the form

$$\mathbf{n_j} = \alpha_j(\mathbf{A}\Delta t - \theta_j\mathbf{I})^{-1}\mathbf{n}_* \tag{67}$$

with α_j, θ_j as parameters of the specific implementation and order considered. Therefore, performing the full computation for each package at each time step, and especially for high orders, quickly becomes impractical when the number of packages in the simulation increases.

Fortunately, all the elements of $\mathbf{A}$ are constant (since they come from constant nuclear data), making the decay matrix constant. Additionally, the time interval Δt only encompasses a handful of values for a given simulation. Therefore, the number of different $(\mathbf{A}\Delta t - \theta_j\mathbf{I})$ matrices is limited, and hence one can cache their LU factorization. This caching has to be done at the first decay call of a different Δt and can be reused for all subsequent packages and Δt of the same length. This allows the solver to bypass the most computationally expensive part of the system solving, reducing the decay operation to a series of rapid substitutions, thereby dropping the computation cost per package.

14 Processes description

This section explains the implementation of all the processes currently implemented in ANICCAv2. The implementation follows the process interface and modeling approach described in Section 9. Specifically, for each process P, the relevant functions that are implemented are

- forward pass function $\phi_P(t, \mathcal{P}_{P,in}[t], m_{P,out}[t]) = \mathcal{P}_{P,out}[t + t_{P,prcs}]$,
- fixed inputs function (FD processes): $\psi_P(t) = \mathcal{M}_{P,in}[t]$,
- backward pass function (OD processes): $\theta_P(m_{P,out}[t]) = \mathcal{M}_{P,in}[t]$,

- corrected backward pass function (CDOD processes): $\theta^*_P(m_{P,out}[t], I_{f'}[t]) = \mathcal{M}^*_{P,in}[t]$,

along with the physical model underlying each transformation.

14.1 Conversion

The `Conversion` process models an infinite source that produces packages of fixed nuclide composition, without consuming any upstream feed. Typically, it is used to create natural uranium packages whose isotopic composition is known, and specified internally in ANICCAv2. It is used to represent primary supply facilities such as uranium mines and supports OD and FD cycle types. To each output type $\tau \in \mathcal{T}_{P,out}$ is allocated a configured mass fraction r_τ such that $\sum_\tau r_\tau = 1$.

Backward pass $\theta_P(m_{P,out}[t])$

Since the process has no material input ($\mathcal{T}_{P,in} = \{\}$), the backward pass returns no upstream request:

$$\mathcal{M}_{P,in}[t] = \{\}. \tag{68}$$

Forward pass $\phi_P(t, \mathcal{P}_{P,in}[t], m_{P,out}[t])$

The transformation produces one package per output type with the library-defined composition. In OD mode, the mass of each output package is

$$m_\tau = m_{P,out} \cdot r_\tau, \quad \forall \tau \in \mathcal{T}_{P,out}. \tag{69}$$

In FD mode, $m_{P,out}$ is replaced by the configured capacity $c_{P,max}$, producing $c_{P,max} \cdot r_\tau$ per output type at every time step.

14.2 Enrichment

The `Enrichment` process models uranium enrichment using a two-component mass balance. It accepts one feed stream and produces two output streams: an enriched product at target enrichment x_p (mass fraction of ^{235}U) and a depleted tail at depletion x_t. Therefore, $|\mathcal{T}_{P,in}| = 1$ and $|\mathcal{T}_{P,out}| = 2$. The value of x_p is automatically extracted from the fuel libraries, using the ^{235}U equivalent enrichment parameter of the first step of the matching fuel. The value of x_t is extracted from the configuration file. It supports OD and FD cycle types.

Backward pass $\theta_P(m_{P,out}[t])$

Given a required product mass $m_p = m_{P,out}[t]$, the feed quantity is computed by using the enrichment mass balance:

$$\mathcal{M}_{P,in}[t] = \left\{ m_p \cdot \frac{x_p - x_t}{x_f - x_t} \right\}, \tag{70}$$

where the quantity in $\mathcal{M}_{P,in}[t]$ corresponds to the only package type in $\mathcal{T}_{P,in}$ and x_f is the ^{235}U mass fraction of the feed, obtained from a nuclide library. This requires the feed composition to be known, which is the case when the feed originates from a library-defined package such as natural uranium.

Fixed inputs $\psi_P(t)$

In FD mode, the process requests exactly a configured capacity $c_{P,max}$ of feed per time step, independently of downstream demand.

Forward pass $\phi_P(t, \mathcal{P}_{P,in}[t], m_{P,out}[t])$

The product and tail masses are derived from mass and ^{235}U balances:

$$m_p = m_f \cdot \frac{x_f - x_t}{x_p - x_t} \quad (= m_{P,out}[t]), \tag{71}$$

$$m_t = m_f - m_p. \tag{72}$$

The nuclide composition of each output stream is constructed by fixing ^{235}U to the target enrichment and scaling all other nuclides proportionally:

$$w_{k,i} = \begin{cases} x_k & \text{if } i = {}^{235}\text{U} \\ w_{f,i} \cdot \dfrac{1 - x_k}{1 - x_f} & \text{otherwise} \end{cases} \tag{73}$$

where $k \in \{p, t\}$ and $i \in \mathcal{I}$. All nuclides other than ^{235}U are not explicitly tracked through the enrichment cascade. Their scaling assumes a simple linear redistribution so that $\sum w_i = 1$ in both the product and tail packages.

14.2.1 U236EquivalentEnrichment

When the feed originates from reprocessed uranium, it contains ^{236}U, a neutron absorber that degrades the effective reactivity of the material. To compensate, the ^{235}U target enrichment can be increased by a penalty term proportional to the ^{236}U content of the feed:

$$x_{p,eff} = x_p + \kappa \cdot x_f^{236}, \quad \kappa = 0.5, \tag{74}$$

where x_f^{236} is the ^{236}U mass fraction of the feed. The value $\kappa = 0.5$ is based on SCK CEN calculations. Reprocessed uranium in PWR reactors typically contains around 0.5 % of ^{236}U. For a burnup of 45 GWd/tHM, it was calculated that traditional UOX fuel fabricated using reprocessed uranium should receive an additional enrichment of 0.25 % in order to maintain the original reactivity level. This yields the factor[5] $\kappa = 0.25/0.5 = 0.5$.

The model further assumes that the fraction of ^{236}U is not increased by the enrichment process like it should be (because the atomic mass of ^{236}U is closer to the one of ^{235}U than of ^{238}U, it is actually enriched alongside ^{235}U in the enrichment cascade). Both the tail and product streams, therefore, conserve the feed fraction in x_f^{236}. Since the feed composition depends on the upstream reprocessing inventory, which is unknown at backward-pass time, `U236EquivalentEnrichment` operates as a CDOD process.

Backward pass $\theta_P(m_{P,out}[t])$

The process defers the input requirement computation to θ_P^*. θ_P therefore returns no upstream demand:

$$\mathcal{M}_{P,in}[t] = \{\}. \tag{75}$$

[5] ANICCAv1 computed the factor based on the enrichment level of a natural uranium-based fuel of 4.1 %. This yielded $\kappa = \frac{4.1+0.25}{4.1} = 1.06$. This is an erroneous interpretation, since one has to look at *how much additional enrichment is needed per unit of* ^{236}U, and not at the ratio between updated and initial fuel enrichments. The correct interpretation is therefore used in ANICCAv2.

Corrected backward pass $\theta_P^*(m_{P,out}[t], I_{upstrm}[t])$

Let us denote the upstream inventory holding the input package τ by $I_\tau[t] \in I_{upstrm}[t]$. The corrected backward pass θ_P^* iterates over the packages of $I_\tau[t]$ as a FIFO queue, computing the enrichment target (given by Equation 74) one package at a time and adding up their individual output contributions. Denoting the currently visited package by p_k, the output mass increment δm_k is given by

$$\delta m_k = m_{p_k} \cdot \frac{x_{f,p_k} - x_t}{x_{p,eff,p_k} - x_t}. \tag{76}$$

Starting from a demand $r = m_{P,out}[t]$, packages are consumed in FIFO order: if $\delta m_k \geq r$, a partial draw of $m_k \cdot r/\delta m_k$ is requested and the iteration terminates. Otherwise, the full package m_k is drawn and r is decremented by δm_k, until it reaches zero. Since the packages actually extracted later on to be processed will also be extracted in FIFO order, this allows us to exactly compute the required amount of inputs to send to this process.

Forward pass $\phi_P(t, \mathcal{P}_{P,in}[t], m_{P,out}[t])$

Product and tail masses follow from the same balance as in Equations 71–72, using $x_{p,eff}$ instead of x_p:

$$m_p = m_f \cdot \frac{x_f - x_t}{x_{p,eff} - x_t} \quad (= m_{P,out}[t]). \tag{77}$$

In the product composition, ^{235}U is set to $x_{p,eff}$, ^{236}U is held fixed at x_f^{236}, and all remaining nuclides are scaled to fill the remainder:

$$w_{p,i} = \begin{cases} x_{p,eff} & \text{if } i = {}^{235}\text{U} \\ x_f^{236} & \text{if } i = {}^{236}\text{U} \\ w_{f,i} \cdot \dfrac{1 - x_{p,eff} - x_f^{236}}{1 - x_f - x_f^{236}} & \text{otherwise.} \end{cases} \tag{78}$$

The tail composition is obtained analogously with x_t instead of $x_{p,eff}$.

14.3 Combine

The `Combine` process blends multiple input streams into a single output package. It is typically used for advanced fuel fabrication such as MOX or ADS fuel. It is currently only implemented in OD mode, although FD and VD modes could easily be implemented.

Backward pass $\theta_P(m_{P,out}[t])$

Each input package $k \in \mathcal{T}_{P,in}$ is requested in proportion to a configured mass fraction r_τ, with $\sum_\tau r_\tau = 1$:

$$\mathcal{M}_{P,in}[t] = \{m_{P,out} \cdot r_\tau \mid \forall \tau \in \mathcal{T}_{P,in}\}. \tag{79}$$

Forward pass $\phi_P(t, \mathcal{P}_{P,in}[t], m_{P,out}[t])$

All received input packages are blended into a single output package. The resulting package nuclide composition is therefore:

$$w_{out,i} = \frac{\sum_{p \in \mathcal{P}_{P,in}} m_p \, w_{p,i}}{\sum_{p \in \mathcal{P}_{P,in}} m_p} \quad , \forall i \in \mathcal{I} \tag{80}$$

and its mass is $\sum_{p \in \mathcal{P}_{P,in}} m_p$.

14.3.1 PuEquivalentCombine

The `PuEquivalentCombine` process blends exactly two input streams, a plutonium stream and another one. The proportion between the two feed streams is computed by matching the ^{239}Pu *equivalent enrichment* of the product to the fuel library target of the produced fuel ϵ_{Pu}^{lib}. This can be useful for the production of ADS fuel, which is typically composed of a mix of Pu and MA (of varying nuclide composition). Since stream compositions are unknown at backward-pass time, it operates as a CDOD process.

The reader should be aware that this process implementation is based on the implementation of ANICCAv1. However, the implementation is not fully documented or referenced [11]. It is therefore copied as-is in ANICCAv2, with improvements made only on the integration in ANICCAv2 and not in the modeling per se.

Concept of ^{239}Pu equivalence

The different isotopes of plutonium have widely different neutronic properties. For instance, in the low-energy spectrum, ^{239}Pu is fissile whereas ^{242}Pu is not. The concept of Pu equivalence was therefore developed over the years in order to characterize the *reactivity worth* of samples with varying composition of plutonium (or other nuclides) by comparing it to the one of pure ^{239}Pu.

For any nuclide i, one can define an approximation of its reactivity worth [31] by

$$\rho_i = \bar{\nu}_i \bar{\sigma}_{f,i} - \bar{\sigma}_{abs,i}, \tag{81}$$

where $\bar{\sigma}_{abs,i} = \sum_{c \neq (n,f)} \bar{\sigma}_{c,i}$ comprises all reaction channels except fission, and $\bar{\nu}$ is the average number of neutrons emitted per fission of one nucleus of nuclide i. The ^{239}Pu equivalent enrichment of a package with composition $\mathbf{w}_p$ can then be defined as

$$\epsilon_{Pu}(\mathbf{w}_p) = \sum_{i \in \mathcal{I}} w_{p,i} \cdot \frac{\rho_i - \rho_{238}}{\rho_{239} - \rho_{238}}, \tag{82}$$

where ρ_{238} and ρ_{239} denote the reactivity worth of ^{238}U and ^{239}Pu respectively, computed using Equation 81. ^{238}U and ^{239}Pu therefore serve as lower bound and upper bound respectively, since $\epsilon_{Pu} = 0$ for pure ^{238}U and $\epsilon_{Pu} = 1$ for pure ^{239}Pu. Note: the one-group cross-sections and the average number of neutrons emitted per fission event both used in Equation 82 are taken from the first burnup step of the fuel library associated with the output package, which also provides the library target ϵ_{Pu}^{lib}. The fuel libraries will be further discussed in the section related to the `Irradiation` process.

From there, the mixing strategy consists of finding the right mass fraction between the two input streams (denoted by stream 0 and stream 1) in order for the mixed fuel to have the same Pu equivalent enrichment as the target. This can be expressed as the following constraint:

$$\epsilon_{Pu}(\mathbf{w}_0) \cdot f_0 + \epsilon_{Pu}(\mathbf{w}_1) \cdot (1 - f_0) = \epsilon_{Pu}^{lib}, \tag{83}$$

where f_i represents the mass fraction of i in the output package. Together with the mass balance equation:

$$f_0 + f_1 = 1, \tag{84}$$

these two equations make it possible to compute f_0 and f_1 once the compositions of both input streams are known.

Backward pass $\theta_P(m_{P,out}[t])$

The process defers the input requirement computation to θ_P^*. θ_P therefore returns no upstream demand:

$$\mathcal{M}_{P,in}[t] = \{\}. \tag{85}$$

Corrected backward pass $\theta_P^*(m_{P,out}[t], I_{upstrm}[t])$

Solving the system given by Equation 83 and Equation 84 yields the following analytical solution:

$$f_0 = \frac{\epsilon_{Pu}^{lib} - \epsilon_{Pu}(\mathbf{w}_1)}{\epsilon_{Pu}(\mathbf{w}_0) - \epsilon_{Pu}(\mathbf{w}_1)}, \qquad f_1 = 1 - f_0. \tag{86}$$

However, this requires the knowledge of the final input package compositions, which will only be known once f_0 and f_1 are known. Indeed, by assuming that all packages available for stream 0 (or stream 1) have a different composition, the final input package composition will depend on the number of packages required in order to reach an input of $f_0 \cdot m_{P,out}[t]$ (or $f_1 \cdot m_{P,out}[t]$).

In order to solve this problem, both upstream inventories I_0 and I_1 are walked through simultaneously as FIFO queues, with two accumulating package variables keeping track of the already viewed packages in each inventory. The approach is similar to the one in Section 14.2.1, albeit with two inventories instead of one. At each step, the fractions f_k are recomputed from the currently visited package compositions using Equation 86, and the output mass that can be produced by these two packages is

$$q_{slice} = \min\left(r,\ \frac{a_0}{f_0},\ \frac{a_1}{f_1}\right), \tag{87}$$

where r is the remaining demand to fulfill (initially, $m_{P,out}[t]$) and a_k is the available mass of the current package in stream k. From it, the masses $q_{slice} \cdot f_k$ are accumulated for each stream, available masses a_k are decremented by $q_{slice} \cdot f_k$, and exhausted packages (package whose $a_k = 0$) advance to the next FIFO position in their respective inventory. The loop terminates when $r = 0$ or when one stream is exhausted (in the latter case, a shortage error will be raised since the required output mass $m_{P,out}[t]$ cannot be produced).

Forward pass $\phi_P(t, \mathcal{P}_{P,in}[t], m_{P,out}[t])$

All received input packages are blended into a single output package, identically to the base `Combine` forward pass.

14.3.2 MoxEquivalentCombine

The `MoxEquivalentCombine` process fabricates MOX fuel by blending a Pu stream with one of several candidate uranium streams, selecting the stream that burns the most Pu given the fissile Pu quality (reactivity worth). If the chosen uranium stream is insufficiently enriched, an *"inline"* enrichment sub-step is applied before blending. The process therefore has to operate as a CDOD process.

The same remark as in the previous section applies. The only document referenced for this process implementation in ANICCAv1 [11] is an "NEA benchmark on Transuranic (currently under study) led by the Expert Group in Advanced Nuclear Fuel Cycle Scenarios" [32]. Without

further information, the modeling approach was reproduced as-is, and is detailed in the following paragraphs. Although the modeling approach remains the same, the implementation is still improved. For instance, the exact composition of each input package is now taken into account, whereas ANICCAv1 based its computations on a single package only.

NEA equivalence equation

The minimum fissile Pu mass fraction p (in [%] of total Pu mass) required to achieve criticality in a MOX blend with a uranium ^{235}U content u (mass fraction [%]) is (supposedly) given by an NEA 2016 MOX benchmark quadratic equation:

$$a\,p^2 + b\,p + (c + u) = 0, \quad a = 0.00107, \quad b = 0.13113, \quad c = -10.77298. \tag{88}$$

The fissile Pu content of a Pu sample is defined as

$$x_{fiss} = \frac{x_{^{239}\mathrm{Pu}} + x_{^{241}\mathrm{Pu}}}{x_{Pu,total}}, \tag{89}$$

where $x_{Pu,total}$ is the sum over all Pu isotopes in the sample. Then, the outcome depends on the value of $p/100$ with respect to x_{fiss}.

- If $p/100 \leq x_{fiss}$, the fissile quality exceeds the minimum requirement, and the Pu fraction in the output blend is scaled down proportionally:

$$f_{Pu} = f_{Pu,max} \cdot \frac{p/100}{x_{fiss}}, \tag{90}$$

  where $f_{Pu,max} = 0.08$ is the maximum allowed Pu mass fraction, for which Equation 88 was calibrated.

- If $p/100 > x_{fiss}$, the fissile quality is insufficient. f_{Pu} is set to $f_{Pu,max}$, and the uranium feed requires an additional ^{235}U enrichment of

$$\delta_u = \frac{-a\,(100 x_{fiss})^2 - b\,(100 x_{fiss}) - c - u}{100}. \tag{91}$$

  to satisfy Equation 88 at $p = 100\,x_{fiss}$. In such a case, the uranium stream will therefore have to be enriched to $x_p = \frac{u+\delta_u}{100}$.

Backward pass $\theta_P(m_{P,out}[t])$

The process defers the input requirement computation to θ_P^*. θ_P therefore returns no upstream demand:

$$\mathcal{M}_{P,in}[t] = \{\}. \tag{92}$$

Corrected backward pass $\theta_P^*(m_{P,out}[t], I_{upstrm}[t])$

The computation is carried out in two phases. In the first phase, effective compositions of each stream are used to identify the best uranium candidate:

1. Compute the upper limit of the uranium mass requested using

$$m_{U,max} = (1 - f_{Pu,max}) m_{P,out}[t]. \tag{93}$$

2. For each candidate uranium stream, compute the ^{235}U content u of the package extracted if $m_{U,max}$ was requested, and solve Equation 88 to obtain f_{Pu} and, if applicable, δ_u.

3. Among streams requiring no extra enrichment, select the one with the largest f_{Pu} (maximizes Pu consumption). If all streams require enrichment, select the one with the smallest δ_u in order to minimize enrichment work.

In the second phase, the Pu stream and the selected uranium stream are simultaneously iterated upon using the same FIFO walk principle as in Section 14.3.1. The values recomputed for each package pair are x_{fiss}, u, and f_{Pu} via Equation 88. When enrichment is required, the raw uranium drawn from inventory is increased in order to account for the fraction that will be depleted and end up in the tail stream: for each unit of output, the raw uranium fraction is

$$f_{U,raw} = (1 - f_{Pu}) \cdot \frac{x_p - x_t}{x_f - x_t}, \tag{94}$$

where $x_p = x_f + \frac{\delta_u}{100}$. This ensures that $\mathcal{M}^*_{P,in}$ eventually covers both the uranium entering the blended MOX and the depleted tails.

Because the increase in raw uranium demand depends on δ_u, which is itself derived from the blended compositions after the walk, the FIFO walk is embedded in a convergence loop. The first iteration starts with $\delta_u = 0$ (no enrichment $\rightarrow$ no demand increase). At the end of each iteration, a new δ_u is derived from the resulting blended compositions via Equation 91. The convergence loop terminates early when $|\frac{\delta_u^{new}}{100} - \frac{\delta_u^{old}}{100}| < 10^{-6}$. Otherwise, because the composition of the selected uranium stream is usually quite homogeneous and the convergence is therefore expected to be both fast and stable, it is limited to ten iterations.

Forward pass $\phi_P(t, \mathcal{P}_{P,in}[t], m_{P,out}[t])$

If $\delta_u > 0$, a tail output stream must be declared, and $|\mathcal{T}_{out}| = 2$. The uranium feed package is then split into an enriched product at $x_p = x_f + \frac{\delta_u}{100}$ and a depleted tail at x_t, using the standard enrichment balance of Equation 71 and Equation 72. The enriched uranium is then blended with the Pu package to form the MOX output of mass $m_{P,out}[t]$.

14.4 Reprocess

The `Reprocess` process models the chemical separation of spent fuel into distinct output streams. It currently supports the VD type, processing only the maximum amount of packages available in upstream facilities, up to its capacity.

Separation rules are defined per output stream as a mapping from atomic number Z (or a named element group such as `minor_actinide`, covering

$$Z \in \{89, 90, 91, 93, 95, 96, 97, 98, 99, 100, 101, 102, 103\}, \tag{95}$$

`fission_product` covering $Z < 89$, ...) to a recovery fraction $\rho \in [0, 1]$. Each nuclide is assigned to at most one output stream using a first-match rule. A waste stream can be specified as well, if $\rho < 1$.

Forward pass $\phi_P(t, \mathcal{P}_{P,in}[t], m_{P,out}[t])$

For each nuclide i in the feed with atomic number Z_i, the recovered mass directed to output stream s is:

$$m_{s,i} = m_i \cdot \rho_s, \quad \forall Z_i \text{ matching the rules of stream } s. \tag{96}$$

Mass not captured by any stream and separation losses are directed to the designated waste stream.

14.5 CombinePackages

The `CombinePackages` process is a passive process that combines all packages of a given package type into a single package without any further transformation or logic.

This is useful in case a storage facility accumulates many packages. In such a case, decay will be applied to every package individually, which can quickly increase computation time. In case there is little value in accessing individual packages, the user might want to combine all of them in a single package that will then be decayed in a single step. This is typically the case for a depleted uranium storage pool or a storage facility serving as final disposal.

The process operates in VD mode with infinite capacity.

Forward pass $\phi_P(t, \mathcal{P}_{P,in}[t], m_{P,out}[t])$

As specified in Section 9.1, input packages of a given type are summed into a single package before being passed to ϕ_P. Therefore, this process can simply pass these packages through unchanged:

$$\mathcal{P}_{P,out}[t] = \mathcal{P}_{P,in}[t]. \tag{97}$$

14.6 Irradiation

The `Irradiation` process simulates the operation of a fleet of reactors of a given technology over a historical timeline. Because of how these processes interact with states and their parent facility, only one `Irradiation` process can be present per state in a given facility. It operates in FD mode only: both the required fuel input and spent fuel output at each time step are fully determined by the operating history, independently of upstream or downstream facilities. At each reload event, fresh fuel is loaded into the core, in-core fuel packages are irradiated, and spent fuel is discharged.

The batch-based approach adopted in ANICCAv2 is an upgrade with respect to the previous `Irradiation` implementation of ANICCAv1. Indeed, it allows keeping track of the irradiation level of the fuel at each time step, which is especially relevant at commissioning and decommissioning of reactor cores. For instance, in ANICCAv1, core mass was always constituted of fresh fuel, and only the extracted mass of fuel was irradiated (and replaced with fresh fuel). At decommissioning, the whole core was irradiated and then extracted as spent fuel. This approach does not reflect the actual evolution of fuel in reactor cores and was also replaced with a more accurate (although still very simplified) batch-based modeling.

The following subsections start by describing the reactors fleet handling in Section 14.6.1, and the batch-based approach is explained in Section 14.6.2. Then, the required nuclide data for the irradiation step are detailed in Section 14.6.3 and their preprocessing is detailed in Section 14.6.4. The numerical method for performing the depletion process is described in Section 14.6.5. Finally, the process interface functions $\psi_P(\cdot)$ and $\phi_P(\cdot)$ are described in Section 14.6.6.

14.6.1 Reactors fleet handling

A fleet of reactors is modeled by a single `Irradiation` process, that represents one or more reactors of a given technology and that all follow the same operating history. Simulating different technologies of reactors or reactors of the same technology but following different operating histories, therefore requires two distinct processes, and for implementation-related

reasons, two distinct parent facilities. The operating history of the fleet is provided as an Excel spreadsheet in which each row defines a reload event. Each row specifies the date of the event, the nominal electrical power per reactor unit P_{net} [GW_e], the load factor LF, the target discharge burnup BU [GWd/tHM], and commissioning and decommissioning of new reactor unit counts R_{in} and R_{out}. The number of reactors in operation in the fleet at a given time step is computed as

$$N_{reactors}[t] = \sum_{t'=0}^{t} R_{in}[t'] - R_{out}[t']. \tag{98}$$

From Equation 98, in classical scenarios, $N_{reactors}$ is expected to be 0 at the end of the reactor fleet lifetime. In addition, from the reload dates, a cycle duration t_{cycle} is also computed at each step.

The number of active reactor units $N_{reactors}$ evolves through commissioning ($R_{in} > 0$) and decommissioning ($R_{out} > 0$) events. Because they follow the same operation history, all the reactors in a given fleet share the same *core* object, whose size is proportional to the number of reactors in operation. When a reactor is commissioned, the core object is loaded with additional fresh fuel, distributed uniformly across all batches (the batch system is explained in the next subsection). When a unit is decommissioned, its core mass is drained proportionally from all positions and discharged as spent fuel packages.

This approach makes it possible to always require n_{steps} `irradiation_step()` calls per `Irradiation` process, regardless of the number of reactors currently operating in the fleet. This is physically consistent since the system of equations solved is linear with respect to the composition vector and because at the fleet level, one is only interested in the total fleet mass flow and never in individual reactors of the fleet. However, as explained in the beginning of this section, if one is interested in modeling multiple reactors with specific operating history, or different characteristics, multiple reactor facilities will be required, each with their own states and `Irradiation` processes.

14.6.2 Reactor batch management

Most nuclear reactors currently in operation worldwide are Pressurized Water Reactors (PWRs) operating in batch refueling mode. Typically, about one-third of the core is reloaded per cycle and the remaining two-thirds are shuffled in the core. The extracted fuel typically consists of the fuel assemblies with the highest accumulated burnup [33]. In order to approximate this reloading strategy, ANICCAv2 allows partitioning the core object of each reactor fleet into N_b *batch positions*. The size of each batch depends on the reactor core mass m_{core} and is given by $m_{batch} = \frac{m_{core}}{N_b}$. Each batch position holds fuel at a different stage of irradiation. Two options are then possible.

In *multi-batch mode* ($N_b = n_{steps}$), all N_b positions are irradiated simultaneously at each reload cycle. As will be explained later on in the section, the irradiation step is performed using data coming from a fuel library. Each library contains a sequence of n_{steps} burnup steps representing successive stages of fuel and irradiation conditions in the reactor. In multi-batch mode, each position k is irradiated using the k^{th} fuel library step. After irradiation, fuel is shifted by one position: the most irradiated batch (position $N_b - 1$) is discharged as spent fuel, each batch is moved by one position ($k \to k + 1$), and fresh fuel is loaded into position 0. A fuel assembly therefore spends N_b consecutive cycles in the core, accumulating one library step per cycle before discharge.

In *single-batch mode* ($N_b = 1$), all n_{steps} library steps are applied sequentially to the batch

within a single cycle. This enables the simulation of reactors whose full core is reloaded at each reloading cycle, such as ADS reactors.

The reload batch mass requested at each reload event is given by

$$m_{rld}[t] = \frac{P_{net}[t] \cdot N_{reactors}[t]}{\eta} \cdot LF[t] \cdot \frac{t_{cycle}[t]}{BU[t]} \tag{99}$$

where $\eta = P_{elec}/P_{th}$ is the plant thermal efficiency and P_{elec} and P_{th} are, respectively, the electrical and thermal rated powers of one reactor unit. It should be noted that P_{net} might differ from P_{elec}. Indeed, P_{net} is the rated power at a given year and might fluctuate, whereas P_{elec} is solely used to compute an assumed constant thermal efficiency of a reactor unit. Looking at Equation 99, one can notice that the reload mass is independent from the core mass, or by extension, from the batch mass. Therefore, although the reload mass would in principle approximately be equal to m_{batch} for $LF = 1$, it is actually not enforced, nor necessarily the case in practice. Indeed, the actual reload mass depends on burnup, load factor, actual power, ..., and depending on the combination of these values, $m_{rld}[t]$ can happen to be slightly larger than m_{batch}. To maintain physical consistency, the reload mass is therefore always capped to m_{batch} in order to fit one single batch.

In order to account for the evolution of $m_{rld}[t] \neq m_{batch}$, each batch slot is split into two regions, each one containing a single package: *charge* and *residual*. The representation of one reactor core using this partition is shown in Figure 5 in a case of a 3-batch reactor.

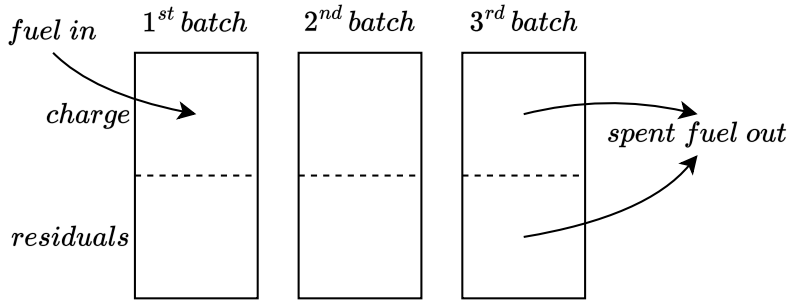


Figure 5: Illustration of a 3-batch reactor core modeled in ANICCAv2.

Charge is the package to which irradiation is applied. After irradiation, m_{rld} is unloaded from *charge* k and loaded in the next *charge* $k+1$, and the remaining mass $m_{batch} - m_{rld}$ is sent to *residual* k. The package in *residual* does not get irradiated and is unloaded in priority at the next reloading step.

More formally, the algorithm responsible for this *cascading* reload of moving a reload mass m_{rld} from one batch to the next is as follows:

1. the mass in the *charge* slot m_{chrg} is irradiated;
2. the mass going out of the *residual* slot $m_{res\rightarrow} = \min(m_{res}, m_{rld})$ is extracted;
3. the mass going out of the *charge* slot $m_{chrg\rightarrow} = m_{rld} - m_{res\rightarrow}$ is then extracted;
4. the remaining mass in *charge*, $m^{rem} = m_{chrg} - m_{chrg\rightarrow}$ is moved from *charge* to *res*;
5. *charge* slot being empty, m_{rld} is moved in from the previous batch.

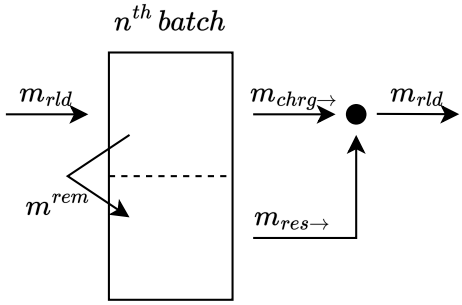


Figure 6: Main mass transfer occurring at each time step in a given reactor batch.

All the mass flows mentioned in the algorithm presented above are depicted in Figure 6.

It should be noted that using the load factor to vary the reloading batch mass is an approximation that originates from ANICCAv1. In practice, the mass of fuel batches is approximately constant across cycles, and it is the accumulated burnup that varies with the load factor. A more faithful representation could scale the total neutron flux value from the burnup library rather than adjusting the fuel mass. The refinement of the irradiation and reload logic is left as future work.

Multi-fuel core load

All the explanations given above implicitly assume a single fuel type being loaded in the core. However, especially for PWRs, two situations motivate multi-fuel load support: multiple fuels may be co-loaded simultaneously (e.g., a fraction of UOX fuel assemblies replaced by MOX), or the fuel composition may evolve across the reactor lifetime (e.g., switching to a higher-enrichment fuel at a later time). Both cases are supported natively: each *charge* and *residual* slot can store a set of packages of different types, which represent different loaded fuels. In steady-state, the fuel mass fractions in each slot are constant, equal to the mass fraction of incoming fuel. All fuel types present in the charge of a given position are irradiated together at each cycle using their respective fuel libraries, and the resulting spent fuel packages are also extracted separately per package type.

The issue of *transitioning* to a different fuel fraction (and therefore dynamic fuel composition) is addressed in the next paragraph.

State transitions

In order to have the most realistic fuel unloading profile after a state transition, one would want to somehow unload the fuel loaded with the previous composition first. For instance, if the reactor transitions from a 30 GWd/tHM burnup UOX fuel (UOX_30) to one with a burnup of 45 GWd/tHM (UOX_45), one would have to prioritize unloading the UOX_30 first, and let the UOX_45 progressively fully populate each core slot. The methodology to achieve this result is explained hereunder.

During the reload cascade following a state transition, the *charge* slots contain fuel loaded under the previous-state fractions $\mathbf{r}_{prev}$, while fresh fuel enters at the current-state fractions $\mathbf{r}_{cur}$. To ensure that the core composition converges toward $\mathbf{r}_{cur}$ cycle after cycle (by extracting the old fuel first), the charge extraction is performed so that the fuel remaining after each reload is always proportional to $\mathbf{r}_{cur}$. The extraction of $m_{chrg\rightarrow}$ is therefore split into two parts:

- a quantity x is extracted using mass fractions $\mathbf{r}_{prev}$, i.e., a mass $x \cdot r_{prev,\tau}$ is extracted for

each fuel type $\tau \in \mathcal{T}_{P_{prev},in}$;

- the remainder $m_{chrg\rightarrow} - x$ is extracted at fractions $\mathbf{r}_{cur}$, i.e., a mass $(m_{chrg\rightarrow} - x) \cdot r_{cur,k}$ is extracted for each fuel type $\tau \in \mathcal{T}_{P_{cur},in}$.

Note: $r_\tau = 0 \quad \forall \tau \notin \mathcal{T}_{P_i,in},\ i \in \{prev, cur\}$.

The split x is determined by imposing that the remaining charge mass of each type τ left after both extractions, $m^{rem}_{chrg,\tau}$, equals $r_{cur,\tau}(m_{chrg} - m_{chrg\rightarrow})$. Solving this constraint for any type τ with $r_{prev,\tau} \neq r_{cur,\tau}$ gives

$$x = \frac{m_{chrg,\tau} - r_{cur,\tau}\, m_{chrg}}{r_{prev,\tau} - r_{cur,\tau}}. \tag{100}$$

When $\mathbf{r}_{prev} = \mathbf{r}_{cur}$, the steady-state case is recovered. In such a case, $x = 0$ and the full extraction proceeds proportionally at $\mathbf{r}_{cur}$, as described earlier in this section.

Further information, such as the development leading to Equation 100 and additional insights and interpretations are given in Appendix A.

Regarding the *residual* slot, by design, because of its (in practice) small mass, it should be fully extracted from each slot at each reload event. However, if this would happen not to be the case, extracting masses from this slot should be done using a similar approach, using $m_{res\rightarrow}$ in place of $m_{chrg\rightarrow}$ and considering that m^{rem}_{res} should stay in its current slot.

14.6.3 Input data

The irradiation step is driven by pre-computed fuel libraries that encode neutron-induced reactions and fission product yields for a given fuel type and irradiation conditions (UOX, MOX, ADS fuel, burnup,...). Each library contains a sequence of n_{steps} burnup steps representing successive stages of fuel and irradiation conditions in the reactor. Let us denote $\mathcal{I}_{FL}$ and $\mathcal{K}_{FL}$ respectively the set of nuclides tracked in the fuel library and the set of fissile nuclides tracked in the fuel library. For each step, the library provides:

- a neutron flux Φ [n/cm^2/s], assumed constant over the step duration;
- a step duration Δt [days], converted internally to seconds;
- a burnup increment [GWd/tHM] (currently unused);
- effective one-group cross-sections $\bar{\sigma}_{c,i}$ [barn] for each tracked nuclide $i \in \mathcal{I}_{FL}$ and reaction channel c;
- one-group fission product yields $\bar{y}_{k\rightarrow j}$ [atoms/fission] for each fission product j produced by fissile parent $k \in \mathcal{K}_{FL}$;
- the average number of neutrons emitted per fission $\bar{\nu}_i$ for each fissile nuclide $i \in \mathcal{I}_{FL}$;
- a reactivity equivalence parameter: either a ^{239}Pu equivalent enrichment ϵ_{Pu} or, for uranium-based fuels, a ^{235}U equivalent enrichment ϵ_U.

Eight reaction channels $c \in \mathcal{RC}$ are encoded per nuclide: radiative capture (n,γ), neutron-ejection reactions $(n,2n)$, $(n,3n)$, $(n,4n)$, alpha emission (n,α), fission (n,f), proton emission (n,p), and isomeric capture (n,γ^*). As was explained in Section 6.3.1, each reaction except

fission has a defined transmutation product determined by its change in proton and mass numbers.

The libraries are built from the output files of the ALEPH2 depletion code developed by SCK CEN [34]. For each burnup sub-step and each fuel zone, ALEPH2 runs a neutron transport calculation to determine the local neutron energy spectrum $\varphi(E)$ and convolute it with tabulated nuclear data to produce effective one-group cross-sections and fission product yields representative of that specific irradiation condition. The library builder then reduces the per-sub-step, per-zone ALEPH2 outputs to the n_{steps} library entries by a flux-time-volume weighting, yielding the averaged quantities listed above. Regarding the fuel libraries created for ANICCAv1 (and reused in ANICCAv2), the ENDF/B-VIII.0 nuclear data library [15] was used as input of ALEPH2.

14.6.4 Data preprocessing

In the general case, the set of nuclides tracked in the fuel library $\mathcal{I}_{FL}$ and the set of fissile nuclides $\mathcal{K}_{FL}$ are distinct from the set of tracked nuclides in an ANICCAv2 scenario $\mathcal{I}$, and can contain nuclides that are not tracked in the given scenario. Therefore, the data present in the fuel library needs to be curated in order to account for nuclides that might be present in $\mathcal{I}_{FL}$ or $\mathcal{K}_{FL}$ but not in $\mathcal{I}$. This curation is performed using the untracked nuclide map mentioned in Section 12.2.3.

The preprocessing consists of iterating over each nuclide $i \in \mathcal{I}$ and building three quantities:

- total loss cross-section $\bar{\sigma}_{abs,i}, \forall i \in \mathcal{I}_{FL}$;
- transmutation-related daughters $daughters_{trans}(i), \forall i \in \mathcal{I}_{FL}$;
- fission-related daughters $daughters_{fission}(i), \forall i \in \mathcal{I}_{FL} \cap \mathcal{K}_{FL}$.

Each one of these quantities is explained in the three following paragraphs.

Total absorption cross-section.

The total neutron-induced absorption cross-section comprises all eight reaction channels:

$$\bar{\sigma}_{abs,i} = \sum_{c \in \mathcal{RC}} \bar{\sigma}_{c,i}. \tag{101}$$

Transmutation daughters ($daughters_{trans}(i)$).

For each channel c with cross-section $\bar{\sigma}_{c,i}$, the transmutation product $d_c(i)$ is determined from the channel change in proton and mass numbers. In case $d_c(i) \in \mathcal{I}$, the cross-section $\bar{\sigma}_{c,i}$ is simply associated with $d_c(i)$ in $daughters_{trans}(i)$. Otherwise, the untracked nuclide map of $d_c(i)$ is used, and $\bar{\sigma}_{c,i}$ is associated with each of $d_c(i)$ effective daughters, scaled by the effective branching factor $b^{eff}_{d_c(i)\to j}$, $\forall j \in daughters(d_c(i))$.

The (n, α) channel additionally contributes a production term for the α particle, which is added to $daughters_{trans}(i)$ in the same way.

In the following sections, the elements of $daughters_{trans}(i)$ will be referred to using the tuple $(dgth,\ \bar{\sigma}_{dgth})$.

Fission daughters ($daughters_{fission}(i)$).

For each fissile nuclide $i \in \mathcal{K}_{FL}$, the fission yield of product l is $\bar{y}_{i\to l}$. Let us denote the fission cross-section of i by $\bar{\sigma}_{f,i}$. Then, for all fission product l, a contribution is added to $daughters_{fission}(i)$, mapping l to $\bar{\sigma}_{f,i} \cdot \bar{y}_{i\to l}$ if $l \in \mathcal{I}$. Otherwise, a contribution is added for each of the effective daughters of l, where $\bar{\sigma}_{f,i} \cdot \bar{y}_{i\to l}$ is scaled by $b^{eff}_{l\to j}$, $\forall j \in daughters(l)$.

In the following sections, the elements of $daughters_{fission}(i)$ will be referred to using the tuple $(dgth,\ rate_{dgth})$.

14.6.5 Irradiation step

At each reload cycle, the irradiation engine updates the nuclide composition of the fuel in the *charge* position of each batch. As mentioned in Section 13.3, radioactive decay is not modeled separately but as part of the burnup matrix itself.

Formulation

The irradiation problem can be formulated using the exact same equations as for the decay problem. It therefore follows the same developments as in Section 13.1. The mass-fraction-to-atom-proportional conversion follows the same procedure as described in Section 13.1 as well.

Even though irradiation and decay both resolve to solve Equation 16, the structure of $\mathbf{A}$ is different in both cases. Specifically, let us recall that irradiation makes use of effective coefficients λ^{eff}_i and $\lambda^{eff}_{j\to i}$ (see Equation 10 to Equation 14).

Using the one-group cross-sections and fission yields from the fuel libraries, both coefficients can be eventually formulated as

$$\lambda^{eff}_i = \lambda_i + \Phi \cdot \bar{\sigma}_{abs,i} \tag{102}$$

$$\lambda^{eff}_{j\to i} = b_{j\to i}\lambda_j + \Phi \sum_{c\in\{\mathcal{RC}\setminus(n,f)\}} \bar{\sigma}_{j\to i,c} + \Phi \cdot \bar{\sigma}_{j,f} \cdot \bar{y}_{j\to i} \tag{103}$$

where the seven $\bar{\sigma}_{j\to i,c}$ under the summation symbol in Equation 103 are computed using the $daughters_{trans}(i)$ mapping, and the $\bar{\sigma}_{j,f} \cdot \bar{y}_{j\to i}$ term is stored in $daughters_{fission}(i)$.

The problem therefore comes down to solving Equation 63, where the difference with respect to the pure decay case lies in the structure of the coefficient matrix $\mathbf{A}$, which incorporates neutron-induced reaction channels in addition to radioactive decay.

Burnup matrix construction

The burnup matrix $\mathbf{A}$ is assembled using the pre-processed inputs of Section 14.6.4 and implementing the matrix coefficients given in Equation 102 and Equation 103. The algorithm is given by Algorithm 7.

Algorithm 7: Build the burnup matrix **A**.

Function `build_burnup_matrix()`
  $A_{i,j} \leftarrow 0, \quad \forall i, j \in [0, |\mathcal{I} \cup \{\iota\}|) \times [0, |\mathcal{I} \cup \{\iota\}|)$
  foreach $ncld \in \mathcal{I}$ **do**
    if *decay* **then**
      $A_{i_{ncld}, i_{ncld}} \mathrel{+}= -\lambda_{ncld}$
      foreach $dgth \in daughters(ncld)$ **do**
        $A_{i_{dgth}, i_{ncld}} \mathrel{+}= \lambda_{ncld} \cdot b^{eff}_{ncld \to dgth}$
      end
    end
    if $ncld \notin \mathcal{I}_{FL}$ **then**
      continue
    end
    $A_{i_{ncld}, i_{ncld}} \mathrel{+}= -\Phi \cdot \bar{\sigma}_{abs, i_{ncld}} \cdot 10^{-24}$
    foreach $(dgth,\ \bar{\sigma}_{dgth}) \in daughters_{trans}(i)$ **do**
      $A_{i_{dgth}, i_{ncld}} \mathrel{+}= \Phi \cdot \bar{\sigma}_{dgth} \cdot 10^{-24}$
    end
    foreach $(dgth,\ rate_{dgth}) \in daughters_{fission}(i)$ **do**
      $A_{i_{dgth}, i_{ncld}} \mathrel{+}= \Phi \cdot rate_{dgth} \cdot 10^{-24}$
    end
  end
  return A
end

Note: in the formulas used in Algorithm 7, the 10^{-24} factor converts cross-sections from barn to cm^2.

Irradiation engine

Irradiation is eventually computed using an implementation of the `IrradiationEngine` class. By default, ANICCAv2 provides `CRAMIrradiationEngine`, which uses a CRAM implementation in order to solve Equation 16. As for the decay engine, three subclasses are available, each corresponding to a different implementation of CRAM:

1. `PFD16IrradiationEngine` (PFD of order $k = 16$);
2. `IPF16IrradiationEngine` (IPF of order $k = 16$);
3. `IPF48IrradiationEngine` (IPF of order $k = 48$).

For burnup calculations, as mentioned previously, it can be shown that order $k = 16$ yields sufficient accuracy. The default class used in ANICCAv2 is the one based on the IPF implementation, `IPF16IrradiationEngine`, since it is more stable than the PFD16 implementation that was used in ANICCAv1. It can, however, be overridden by any other classes or by a custom `IrradiationEngine` class.

Within a given library step, the neutron flux Φ and all nuclide properties are held constant, making the coefficient matrix time-independent. Therefore, the same trick of caching the LU decomposition of the burnup matrix **A** can be applied, in the same way as in Section 13.4.1.

14.6.6 Process interface

Nuclear reactors can only be modeled using FD processes. Therefore, only the fixed inputs and forward pass functions are implemented.

Fixed inputs $\psi_P(t)$

The requesting of masses only happens on reload events, i.e. for a given row of the history file. When a reload event coincides with a simulation time step, the requested mass comprises two terms. If $R_{in}[t] > 0$, a commissioning full core request of

$$m_{comm,\tau} = R_{in}[t] \cdot m_{core} \cdot r_\tau, \quad \forall \tau \in \mathcal{T}_{P,in} \tag{104}$$

is issued.

Let us denote N_{after} the number of active reactors after taking $R_{in}[t]$ into account. If $N_{after} > 0$, a reload request of $m_{rld}[t] \cdot r_\tau$ is issued per fuel (or package) type τ, where $m_{rld}[t]$ is given by Equation 99 and is capped to $m_{batch} \cdot N_{after}$ to prevent exceeding the batch slot capacity.

Therefore, the total required mass at each time step t is given by

$$\mathcal{M}_{P,in}[t] = \begin{cases} \{(R_{in}[t] \cdot m_{core} + m_{rld}[t]) \cdot r_\tau \mid \forall \tau \in \mathcal{T}_{P,in}\} & \text{if } t \text{ is a reload event} \\ \emptyset & \text{otherwise} \end{cases}. \tag{105}$$

Forward pass $\phi_P(t, \mathcal{P}_{P,in}[t], m_{P,out}[t])$

The forward step essentially consists of refueling the reactors core and updating all of their batches. It takes fresh fuel packages in, places them in the core and moves the packages in the reload cascade, and outputs spent fuel packages. It is also responsible for commissioning new reactor cores (by uniformly distributing incoming fresh fuel to each batch), and decommissioning old ones (by extracting packages from each batch).

If no history row coincides with t, the process is idle and produces no output. The behavior described above is achieved using the following four consecutive steps:

1. The charge of each batch position k is first irradiated for every fuel type present. In multi-batch mode, library step k is applied to batch k. In single-batch mode, all n_{steps} steps are applied sequentially to the single batch.

2. Let us denote N_{before} as the number of active reactors before considering $R_{out}[t]$. If $R_{out}[t] > 0$, a fraction $R_{out}[t]/N_{before}$ is extracted from every batch position (both slots, proportionally to each fuel type) and added to the list of output packages, in order to simulate the decommission of $R_{out}[t]$ reactor cores.

3. In order to reload the core, batch positions are iterated in reverse order (from $k = N_b - 1$ down to $k = 0$). At each position, fuel is extracted via the algorithm presented in Section 14.6.2. The extracted mass from position $k < N_b - 1$ becomes the charge of position $k + 1$, and a part of the mass from position $N_b - 1$ is discharged as spent fuel. Fresh reload fuel is then loaded into the charge of position 0, now empty.

4. If $R_{in}[t] > 0$, $m_{core} \cdot R_{in}[t]$ of the input packages are distributed uniformly across all batch positions and merged into each charge slot, in order to represent the commissioning of $R_{in}[t]$ new reactor units.

Part V
Case studies

This part first compares ANICCAv1 and ANICCAv2 on two representative case studies, each targeting one of the core physical phenomena modeled in ANICCA. The comparison aims to validate the new implementation and to highlight the corrections and improvements introduced by ANICCAv2. Then, it also presents the capabilities of ANICCAv2 on two representative scenarios.

Four case studies are therefore analyzed. Section 15 focuses on the decay of a spent UOX fuel package. Section 16 then focuses on the irradiation step of a single fuel batch. Section 17 presents a classical PWR scenario with UOX fuel as a simple validation of the fuel cycle simulation. Finally, Section 18 addresses a closed recycling loop involving MOX fuel, showcasing the tearing logic, the MOX equivalence model, and the flexible reactor fleet handling introduced in ANICCAv2.

In the following sections, $v1$ and $v2$ will be used to refer to ANICCAv1 and ANICCAv2, respectively. Let us also note that each simulation is carried out using the `nuclides_ml` nuclide list.

15 Decay

The case study consists of decaying a sample of spent UOX fuel (initial ^{235}U enrichment of around 3.75 %, burnup of 45 GWd/tHM) over 40 time steps: 10 steps of 1 year, 10 of 100 year, 10 of 10^4 year, and 10 of 10^6 year. The same initial composition is used in both versions. The relative difference in mass fraction between $v1$ and $v2$ is computed for all tracked nuclides and plotted against the absolute mass fraction in $v1$ in Figure 7, at two different times: after the first decay step (1 year) and at the end of the simulation (around 10 000 000 year).

The results show that the vast majority of nuclides agree to within 1 %, confirming good overall agreement between both versions. The outliers, however, are not indicative of errors in $v2$ but reflect two distinct modeling errors in $v1$:

- an error in the decay chain construction, leading to apparent *overproduction* of specific nuclides in $v2$;
- an error in the time-step transition handling, leading to apparent *underproduction* of specific nuclides in $v2$.

Each defect is detailed in the following subsections. Note: although not *each and every* outlier has been thoroughly investigated, the presented modeling errors in $v1$ affect a wide range of nuclides and are severe enough to provide relatively more confidence in $v2$ results rather than in $v1$, especially without evidence of any $v2$ deficiency discovered during testing (as opposed to the multiple errors found in $v1$).

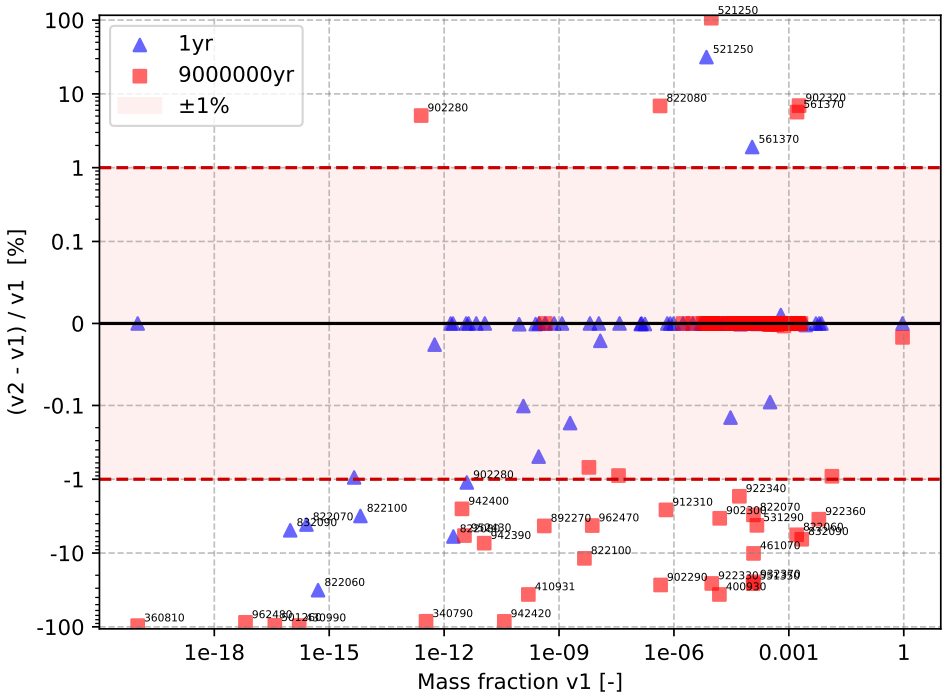


Figure 7: Relative difference in nuclide mass fractions between $v1$ and $v2$, both after a decay of 1 year and at the end of the simulation.

15.1 Overproduction

The most pronounced case of apparent overproduction in $v2$ is $^{125}_{52}$Te (nuclide 521250), a stable nuclide primarily produced by β^- decay of $^{125}_{51}$Sb, which has a half-life of 2.76 year.

Figure 8 shows the evolution of both nuclides over the first 10 decay steps of 1 year each.

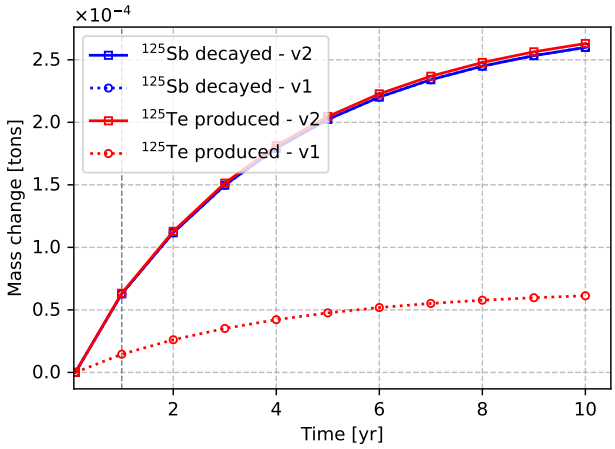


Figure 8: Evolution of ^{125}Sb and ^{125}Te masses over the first 10 year.

While ^{125}Sb disintegrates at identical rates in both versions, $v1$ does not translate this disintegration into the corresponding production of ^{125}Te. This is caused by an error in the decay chain construction of $v1$, which omits or misroutes specific branching paths, resulting in an apparent underproduction of certain stable daughters. This explains why ^{125}Te *seems* to be overproduced in $v2$, whereas it is actually wrongly underproduced in $v1$.

The most likely issue is that the different decay paths of ^{125}Sb are not accounted for properly.

Actually, ^{125}Sb decays into ^{125}Te with a branching ratio of 77.623 % and into ^{125m}Te with a ratio of 22.377 %. This metastable nuclide then quickly decays back to ^{125}Te. Looking at the ratio between $v1$ and $v2$ of the production of ^{125}Te by decay of ^{125}Sb after 10 years, and accounting for the contribution of the ^{125m}Te initially present in the fuel, one eventually gets

$$ratio = \frac{\Delta^{125}\mathrm{Te}_{v1}}{\Delta^{125}\mathrm{Te}_{v2}} = \frac{6.129\,03 \times 10^{-5} - 3.140\,126 \times 10^{-6}}{2.630\,07 \times 10^{-4} - 3.140\,126 \times 10^{-6}} \tag{106}$$

$$= 22.3768\,\%, \tag{107}$$

as if $v1$ were only accounting for the decay of ^{125}Sb into ^{125m}Te.

15.2 Underproduction

The most pronounced case of apparent underproduction in $v2$ is ^{81}Kr (nuclide 360810). Figure 9 shows its evolution over all forty time steps. Note: the masses were truncated at a minimum of 10^{-20} for clarity. Otherwise, one could observe both $v1$ and $v2$ masses reaching as low as 10^{-300} because of exponential decay never rounding quantities to 0 (except because of numerical rounding).

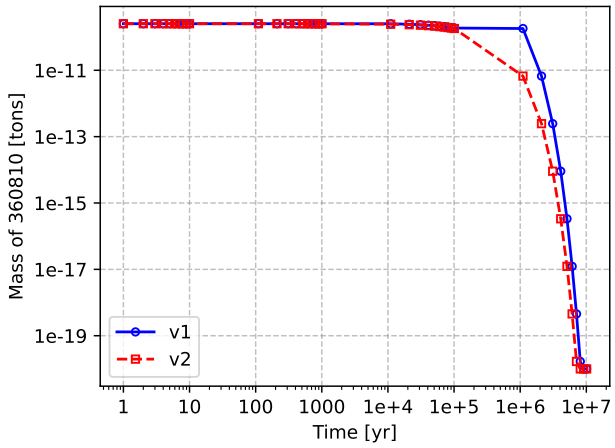


Figure 9: Evolution of the mass of ^{81}Kr in $v1$ and $v2$ over all time steps.

The divergence is most visible at the transition between the 10^4 year and 10^6 year step groups, where the mass in $v2$ already decreases, whereas the $v1$ one stays *"constant"* for an additional step. This is due to the fact that at each time-step transition, $v1$ systematically applies a decay step of the *previous* step duration rather than the new one, leaving the nuclide insufficiently decayed after the transition. Relative to $v2$, ^{81}Kr is consequently too abundant in $v1$, and its decay daughters are underproduced accordingly. The reason why the error *seems* to solely occur at the 10^4-10^6 transition is the long half-life of ^{81}Kr of around 210 000 year, which makes the population in ^{81}Kr constant (in practice) at small time horizons. The error is however general and affects every single nuclide at every single time step group transition.

This specific problem explains why it seems like there are more nuclides that appear to be under-produced in $v2$ after a long decay time (i.e., the existence of a large number of red squares in the negative half of Figure 7), since most of these nuclides have a long half-life and have therefore accumulated the time step error over multiple transitions. This is typically the case for the ^{81}Kr example, but also for ^{242}Pu (942420), ^{248}Cm (962480), ...

16 Irradiation

The case study consists of irradiating the same fresh UOX fuel sample as in Section 15, over a single burnup step of 1 year and comparing the spent fuel composition obtained in both $v1$ and $v2$. The relative difference in nuclide mass fraction between both versions is plotted against the absolute mass fraction in $v1$ in Figure 10.

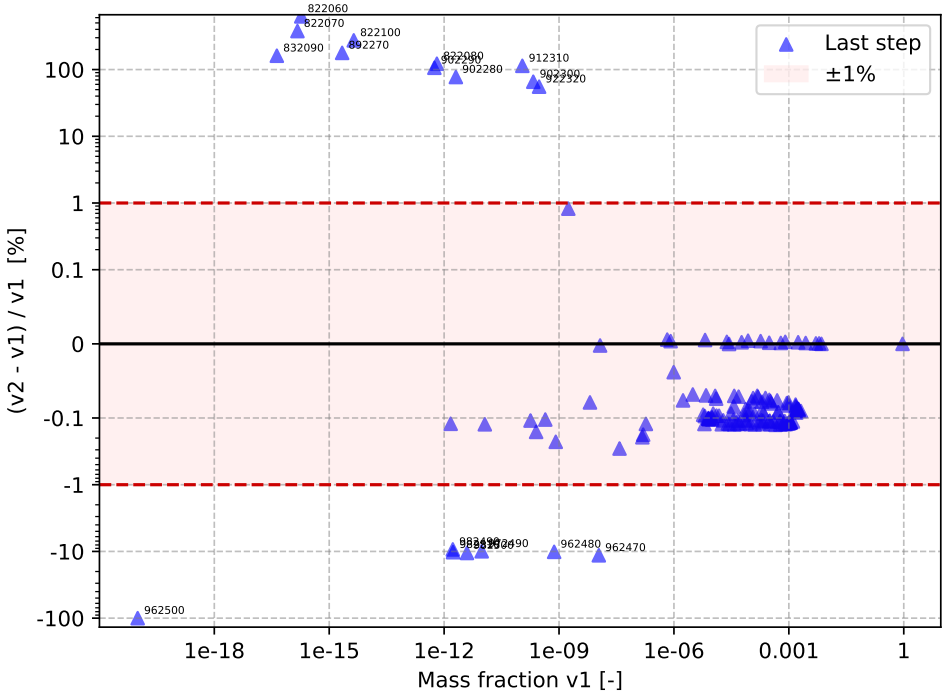


Figure 10: Relative difference in nuclide mass fractions between $v1$ and $v2$ after one irradiation step of 1 year.

The majority of nuclides agree to within ±1 %. A slight systematic underproduction of around 0.1 % is, however, visible in $v2$ for most nuclides. Two groups of outliers with more significant discrepancies can be identified:

- Overproduction in $v2$: a group of *comparatively lighter* (with respect to the other group) nuclides, composed of all lead isotopes ($^{206}_{82}$Pb, $^{207}_{82}$Pb, $^{208}_{82}$Pb, $^{210}_{82}$Pb), as well as specific isotopes of bismuth ($^{209}_{83}$Bi), actinium ($^{227}_{89}$Ac), thorium ($^{228}_{90}$Th, $^{229}_{90}$Th, $^{230}_{90}$Th), and protactinium ($^{231}_{91}$Pa), and one uranium isotope: ^{232}U.

- Underproduction in $v2$: a group of *comparatively heavier* (with respect to the first group) nuclides, all minor actinides, composed of very specific isotopes of curium ($^{247}_{96}$Cm, $^{248}_{96}$Cm, $^{250}_{96}$Cm), berkelium ($^{249}_{97}$Bk), and californium ($^{249}_{98}$Cf, $^{250}_{98}$Cf, $^{251}_{98}$Cf).

In both cases, the affected nuclides are among the rarest in the irradiated fuel, which indicates that the main fission events and their decay products are modeled consistently between both versions. It should nevertheless be noted that minor actinide discrepancies may still be practically important in NFC studies even at small mass fractions. The discrepancies also target specific groups of nuclides, pointing toward an actual modeling difference in the burnup matrix construction between $v1$ and $v2$ rather than purely numerical differences. The exact origin of these differences has, however, not been identified. Whether they are of practical significance and why only these specific nuclides appear differently in both versions is left for future investigation.

17 Simple example

This scenario models a single PWR operating on UOX fuel. Fresh fuel is supplied by an enrichment plant fed with natural uranium, and spent fuel is directed to a storage facility. The complete facility graph is shown in Figure 11.

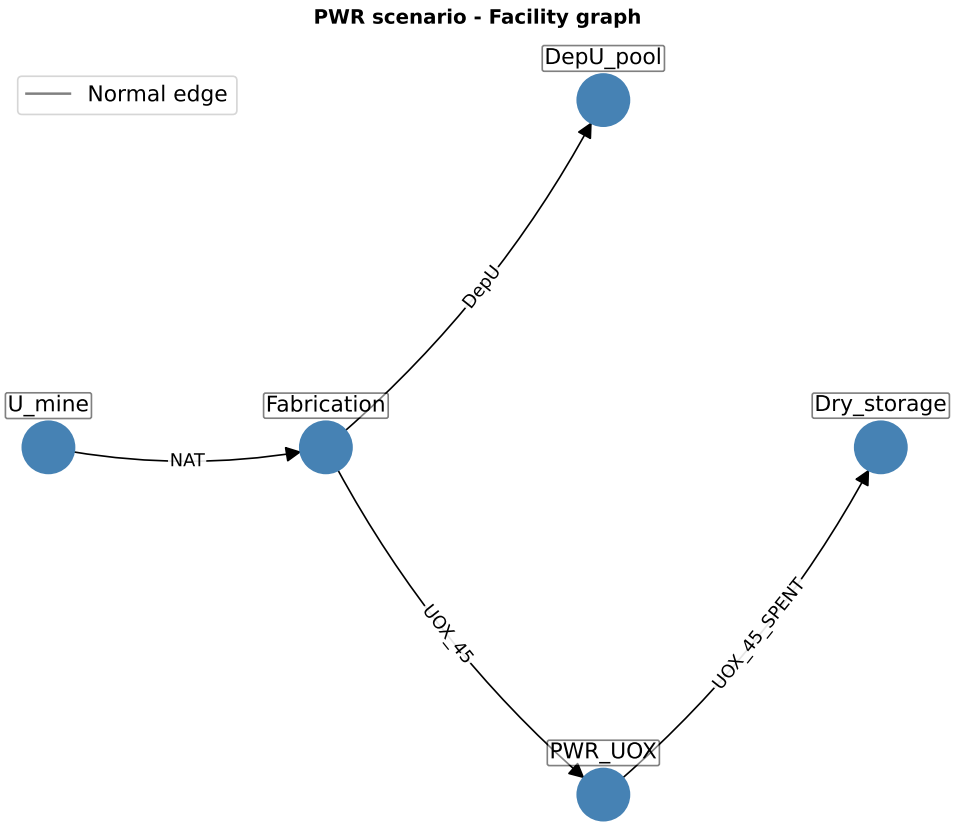


Figure 11: Facility graph of the simple PWR scenario.

The main parameters of the most important facilities are as follows:

- Fuel fabrication plant (`Fabrication`): produces low-enriched UOX fuel from natural uranium, with the same fuel characteristics as in the previous sections. A processing time of two years is used.

- Reactor (`PWR_UOX`): is commissioned at $t = 5$ year with a core mass of 75 tHM, operated in multi-batch mode. The reactor is decommissioned at $t = 45$ year. Spent fuel remains for five years in the `PWR_UOX` facility inventory before being transferred to the storage facility. This allows us to represent the spent fuel storage facility at the reactor site, which is abstracted in the `PWR_UOX` facility.

The resulting facility inventories over the simulation period are shown in Figure 12.

18 Recycling loop with MOX

This scenario extends the previous one to a fleet of two PWRs and introduces a recycling loop. Spent fuel is reprocessed to recover plutonium, which is then blended with depleted uranium to fabricate MOX fuel. The complete facility graph is shown in Figure 13.

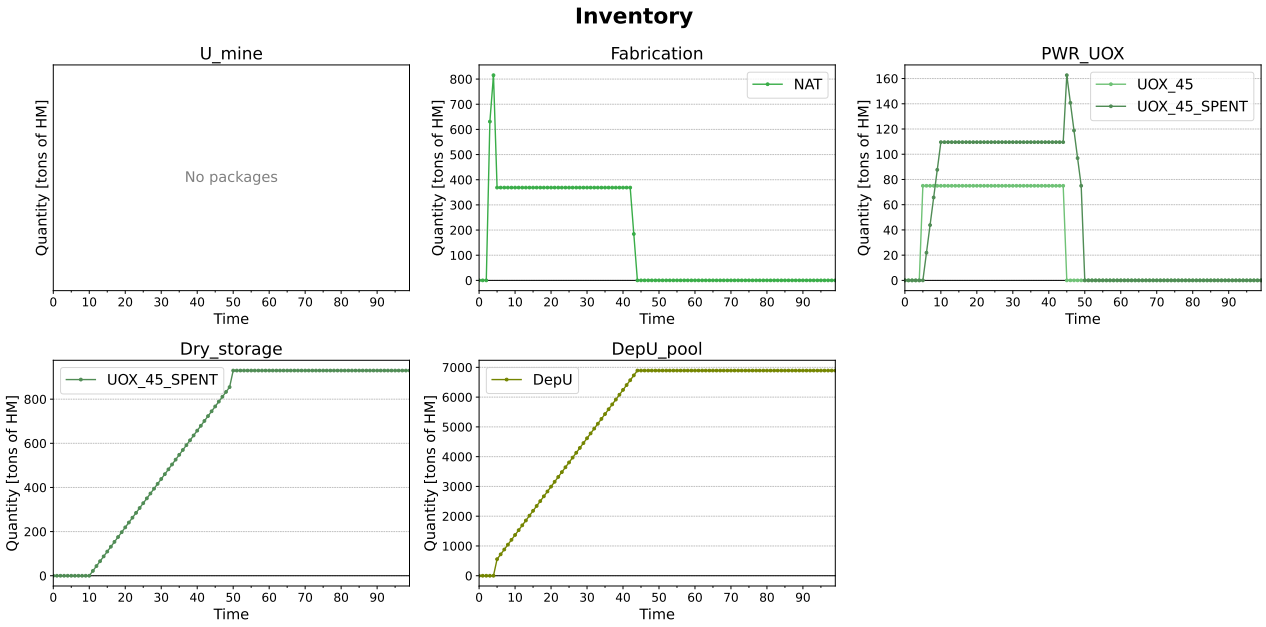


Figure 12: Evolution of each facility inventory over time, for the simple PWR scenario.

The main parameters of the most important facilities are as follows:

- Fuel fabrication plant (`Fabrication`): produces both low-enriched UOX fuel from natural uranium, with the same characteristics as in Section 17, and MOX fuel by blending plutonium recovered from reprocessing with depleted uranium. In both cases, a processing time of two years is used.

- Reactor fleet (`PWRs`): the first reactor is commissioned at $t = 5\,\text{year}$, the second at $t = 8\,\text{year}$. One reactor is decommissioned at $t = 45\,\text{year}$. At $t = 50\,\text{year}$, the remaining reactor begins co-loading MOX fuel at a fraction of 15 % of the core and is decommissioned at $t = 65\,\text{year}$. Each reactor has a core mass of 75 tHM and operates in multi-batch mode. Spent fuel remains for five years in the `PWRs` facility inventory before leaving the facility.

- Reprocessing plant (`Reprocessing`): processes spent fuel with a capacity of 40 tHM/year, separating it into plutonium, minor actinides (MA), reprocessed uranium (RepU), and a waste stream. A processing time of two years is used.

The resulting facility inventories are shown in Figure 14. In the `PWRs` inventory plot, the commissioning and decommissioning events are clearly visible. The transition to partial MOX loading at $t = 50\,\text{year}$ spans four reload cycles, as expected: three cycles are required to progressively replace the fuel composition in each batch slot, plus one additional cycle to drain the last *residual* slot (since $m_{rld}[t] \leq m_{batch},\ \forall t$).

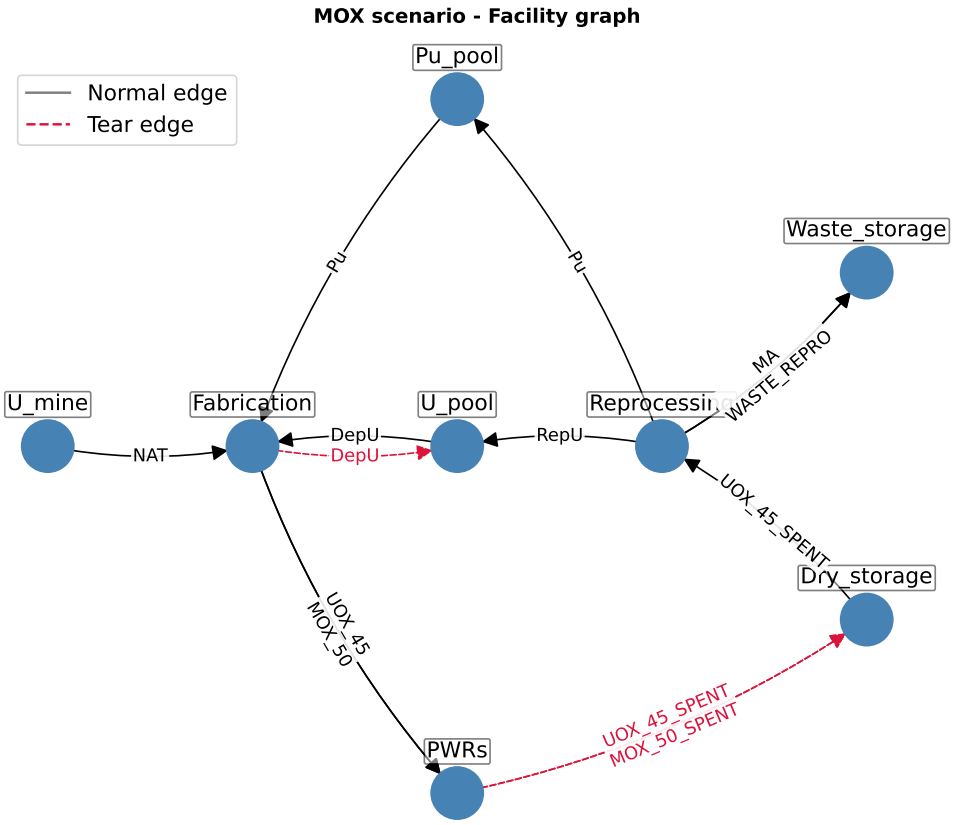


Figure 13: Facility graph of the MOX recycling loop scenario.

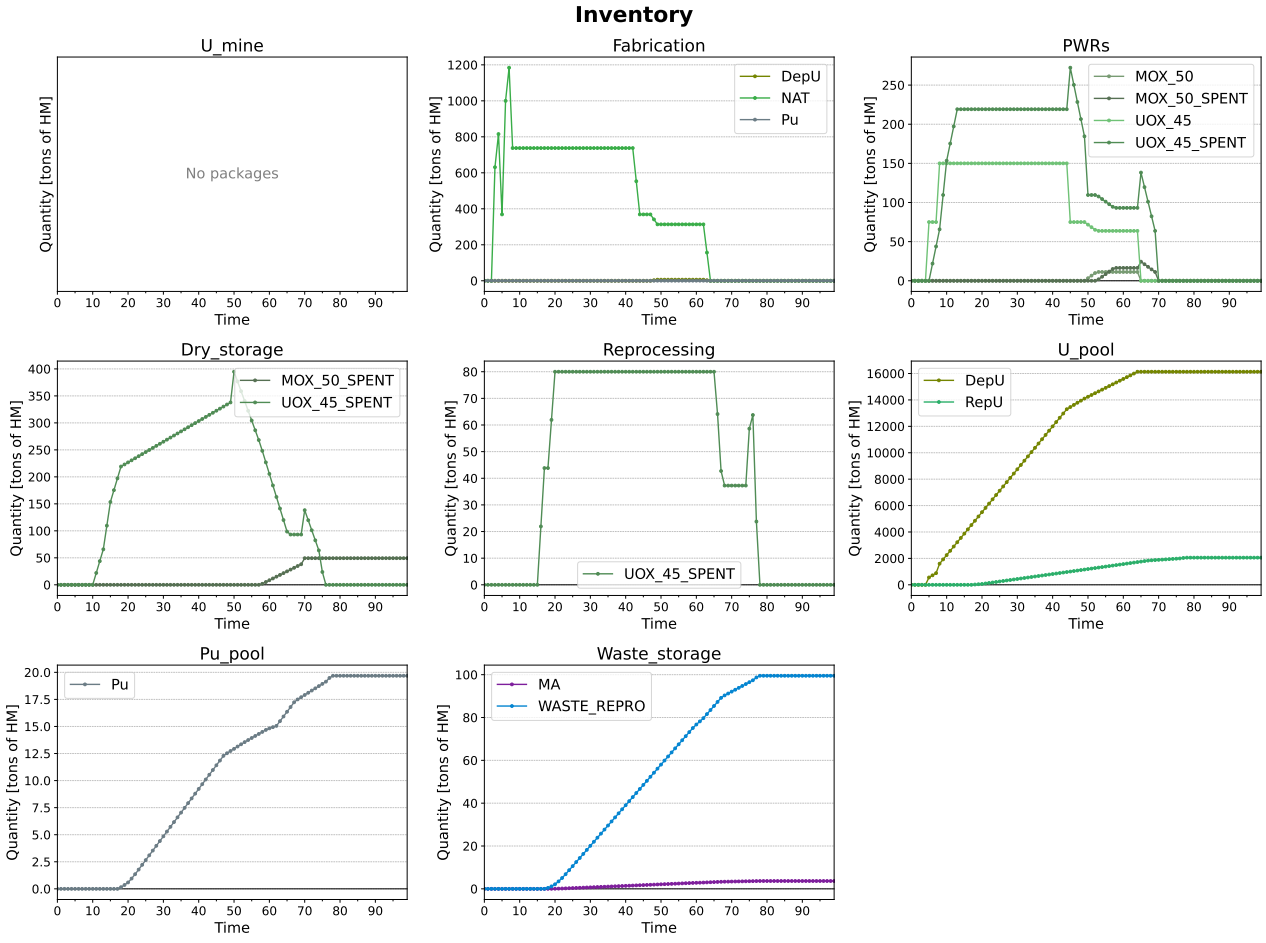


Figure 14: Evolution of each facility inventory over time, for the MOX recycling loop scenario.

Part VI
Conclusion

This thesis presents the design and implementation of ANICCAv2, a new modular framework for nuclear fuel cycle simulation. The core design choice is a three-layer architecture, network, facility, and process, that separates the concerns of graph topology and material dispatch, facility inventory management and process orchestration, and the actual execution of processes. The dispatch algorithm currently implemented follows a sequential modular approach inspired by chemical process simulation, operating through a backward pass that propagates demand upstream in reverse topological order, followed by a forward pass that physically transfers packages in topological order. Recycling loops are handled through a graph-tearing strategy. Four basic cycle types, fixed demand, on demand, variable demand, and none, make it possible to change the behavior of facilities and their processes. The nuclear physics engines implement radioactive decay and irradiation using CRAM, and the irradiation process is built around a multi-batch reactor core model driven by pre-computed fuel libraries. Four case studies validate the framework against the legacy ANICCAv1 code and demonstrate its application to representative open- and closed-cycle scenarios.

The main contributions of this thesis with respect to ANICCAv1 can be summarized as follows. First, the graph-based scenario representation provides a flexible and modular framework for describing nuclear fuel cycle configurations, with native support for recycling loops. The framework allows users to easily swap or customize any component, from processes to dispatch algorithms or decay/irradiation engines, and to easily improve or modify the code logic. Second, ANICCAv2 constitutes a complete rewrite of the code, with a significantly more structured and modular architecture than its predecessor, improving maintainability and laying the ground for future development. Third, the nuclear physics modeling has been improved in several respects: decay and irradiation are now both solved using CRAM with LU factorization caching, the irradiation process supports multi-batch core management and multi-fuel operation, and the equivalence models have been improved. Finally, the validation and rewrite work carried out in this thesis identified and corrected several modeling errors present in ANICCAv1. Overall, ANICCAv2 provides solid foundations for advanced fuel cycle studies that were out of reach with the legacy code.

Three directions for potential future work can be identified:

1. Process modeling: The irradiation reload batch logic should be improved for partial load factors. For instance, when $LF < 1$, the total flux could be scaled rather than the fuel mass so that the reload mass can never exceed the batch mass. Current processes could also be expanded to other cycle types, and new processes, such as vitrified nuclear waste production processes, could also be implemented.

2. Facility and mass dispatch algorithm: Currently, only variable demand facilities enforce a capacity limit; extending this to on demand processes and to facility inventories is a natural generalization. Another limitation is that composition-dependent on demand processes cannot be supplied by on demand facilities due to the structure of the two-pass mass dispatch algorithm that is currently implemented. Further flexibility could also be gained by supporting customizable inventory extraction orders beyond FIFO, including composition-based package selection and customizable dispatch conditions and priorities

on connections in the forward pass.

3. Additional features and code quality: The economic and radiotoxicity modules available in ANICCAv1 have not yet been ported to ANICCAv2 and therefore represent a natural extension. The fuel library generation originating from ANICCAv1 could also be reviewed, improved, and integrated into ANICCAv2. More generally, the code base, test suite, and scenario creation documentation can still be improved in order to make ANICCAv2 more accessible and well-documented.

Beyond these code-level improvements, the modular and flexible nature of ANICCAv2 opens the door to advanced studies such as optimization. Coupling the framework with optimization algorithms, including machine learning and AI-based approaches, for complex fuel cycle decision-making and planning was one of the main motivations behind the rewrite, and is only made possible by the robust and flexible implementation that ANICCAv2 now provides.

References

[1] International Energy Agency. *The path to a new era for nuclear energy*. Paris, France: IEA, 2025. URL: https://www.iea.org/reports/the-path-to-a-new-era-for-nuclear-energy.

[2] R. Taylor, W. Bodel, L. Stamford, and G. Butler. "A review of environmental and economic implications of closing the nuclear fuel cycle—Part one: wastes and environmental impacts". In: *Energies* 15.4 (2022), p. 1433. ISSN: 1996-1073. DOI: 10.3390/en15041433.

[3] S. Shannak, L. Cochrane, and D. Bobarykina. "Global uranium market dynamics: analysis and future implications". In: *International Journal of Sustainable Energy* 44.1 (2025). ISSN: 1478-646X. DOI: 10.1080/14786451.2025.2457376.

[4] T. Wang, Z. Wu, W. Xiong, X. Pan, X. Ye, and X. Liu. "The impact of uranium resource constraints on China's nuclear power development". In: *Energies* 18.6 (2025), p. 1507. ISSN: 1996-1073. DOI: 10.3390/en18061507.

[5] R. Taylor, W. Bodel, and G. Butler. "A review of environmental and economic implications of closing the nuclear fuel cycle—Part two: economic impacts". In: *Energies* 15.7 (2022), p. 2472. ISSN: 1996-1073. DOI: 10.3390/en15072472.

[6] W. I. Ko and F. Gao. "Economic analysis of different nuclear fuel cycle options". In: *Science and Technology of Nuclear Installations* 2012 (2012), pp. 1–10. ISSN: 1687-6083. DOI: 10.1155/2012/293467.

[7] G. D. DelCul, L. D. Trowbridge, J.-P. Renier, R. J. Ellis, K. A. Williams, B. B. Spencer, and E. D. Collins. *Analysis of the reuse of uranium recovered from the reprocessing of commercial LWR spent fuel*. Office of Scientific and Technical Information (OSTI), 2009. DOI: 10.2172/947143.

[8] T. Maldague, S. Pilate, and A. F. Renard. "Impact of plutonium and americium recycling in PWR on MOX fuel fabrication". In: *Actinide and Fission Product Partitioning and Transmutation*. Proceedings of the 3rd NEA International Information Exchange Meeting. Cadarache, France: OECD Nuclear Energy Agency, 1994. URL: https://oecd-nea.org/pt/docs/iem/cadarache94/session-4/Session4Maldague.pdf.

[9] Nuclear Energy Agency. *Nuclear Fuel Cycle Codes Catalogue — oecd-nea.org*. https://www.oecd-nea.org/jcms/pl_100633/nuclear-fuel-cycle-codes-catalogue. [Accessed 25-03-2026]. 2026.

[10] K. D. Huff, M. J. Gidden, R. W. Carlsen, R. R. Flanagan, M. B. McGarry, A. C. Opotowsky, E. A. Schneider, A. M. Scopatz, and P. P. Wilson. "Fundamental concepts in the Cyclus nuclear fuel cycle simulation framework". In: *Advances in Engineering Software* 94 (2016), pp. 46–59. ISSN: 0965-9978. DOI: 10.1016/j.advengsoft.2016.01.014.

[11] I. M. Rodríguez, A. Hernández-Solís, N. Messaoudi, and G. Van den Eynde. "The nuclear fuel cycle code ANICCA: Verification and a case study for the phase out of Belgian nuclear power with minor actinide transmutation". In: *Nuclear Engineering and Technology* 52.10 (2020), pp. 2274–2284. ISSN: 1738-5733. DOI: 10.1016/j.net.2020.04.004.

[12] D. Grinberg. *An introduction to graph theory*. 2023. DOI: 10.48550/ARXIV.2308.04512.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, 2022.

[14] H. Bateman. "The solution of a system of differential equations occurring in the theory of radio-active transformations". In: *Proceedings of the Cambridge Philosophical Society* 15 (1910), pp. 423–427.

[15] D. Brown, M. Chadwick, R. Capote, A. Kahler, A. Trkov, M. Herman, A. Sonzogni, Y. Danon, A. Carlson, M. Dunn, D. Smith, G. Hale, G. Arbanas, R. Arcilla, C. Bates, B. Beck, B. Becker, F. Brown, R. Casperson, J. Conlin, D. Cullen, M.-A. Descalle, R. Firestone, T. Gaines, K. Guber, A. Hawari, J. Holmes, T. Johnson, T. Kawano, B. Kiedrowski, A. Koning, S. Kopecky, L. Leal, J. Lestone, C. Lubitz, J. Márquez Damián, C. Mattoon, E. McCutchan, S. Mughabghab, P. Navratil, D. Neudecker, G. Nobre, G. Noguere, M. Paris, M. Pigni, A. Plompen, B. Pritychenko, V. Pronyaev, D. Roubtsov, D. Rochman, P. Romano, P. Schillebeeckx, S. Simakov, M. Sin, I. Sirakov, B. Sleaford, V. Sobes, E. Soukhovitskii, I. Stetcu, P. Talou, I. Thompson, S. van der Marck, L. Welser-Sherrill, D. Wiarda, M. White, J. Wormald, R. Wright, M. Zerkle, G. Žerovnik, and Y. Zhu. "ENDF/B-VIII.0: The 8th major release of the nuclear reaction data library with CIELO-project cross sections, new standards and thermal scattering data". In: *Nuclear Data Sheets* 148 (2018), pp. 1–142. ISSN: 0090-3752. DOI: 10.1016/j.nds.2018.02.001.

[16] C. Moler and C. Van Loan. "Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later". In: *SIAM Review* 45.1 (2003), pp. 3–49. ISSN: 1095-7200. DOI: 10.1137/s00361445024180.

[17] N. J. Higham. "Functions of matrices". In: Society for Industrial and Applied Mathematics, 2008, pp. 233–267. DOI: 10.1137/1.9780898717778.ch10.

[18] M. Pusa. "Rational approximations to the matrix exponential in burnup calculations". In: *Nuclear Science and Engineering* 169.2 (2011), pp. 155–167. ISSN: 1943-748X. DOI: 10.13182/nse10-81.

[19] M. Pusa. "Higher-order Chebyshev rational approximation method and application to burnup equations". In: *Nuclear Science and Engineering* 182.3 (2016), pp. 297–318. ISSN: 1943-748X. DOI: 10.13182/nse15-26.

[20] D. Calvetti, E. Gallopoulos, and L. Reichel. "Incomplete partial fractions for parallel evaluation of rational matrix functions". In: *Journal of Computational and Applied Mathematics* 59.3 (1995), pp. 349–380. ISSN: 0377-0427. DOI: 10.1016/0377-0427(94)00037-2.

[21] A. A. Hagberg, D. A. Schult, and P. J. Swart. "Exploring network structure, dynamics, and function using NetworkX". In: *Proceedings of the 7th Python in Science Conference*. Ed. by G. Varoquaux, T. Vaught, and J. Millman. Pasadena, CA USA, 2008, pp. 11–15.

[22] L. T. Biegler, I. E. Grossmann, and A. W. Westerberg. *Systematic methods for chemical process design*. United States: Prentice Hall, Old Tappan, NJ (United States), 1997. URL: https://www.osti.gov/biblio/293030.

[23] U. Manber. *Introduction to algorithms: a creative approach*. Addison-Wesley, 1989.

[24] M. Kellett, O. Bernillon, R. Mills, O. for Economic Co-Operation, and 7. .-. P. (Development Nuclear Energy Agency. *The Jeff-3.1/63.1.1 radioactive decay data and fission yields sub-libraries. Jeff report 20*. Organisation for Economic Co-Operation and Development - Nuclear Energy Agency, 2009.

[25] A. G. Croff. "ORIGEN2: A versatile computer code for calculating the nuclide compositions and characteristics of nuclear materials". In: *Nuclear Technology* 62.3 (1983), pp. 335–352. ISSN: 1943-7471. DOI: 10.13182/nt83-1.

[26] V. J. Casas-Molina, P. Romojaro, G. Nourry, G. V. d. Eynde, L. Fiorito, I. Merino-Rodríguez, T. Dhaene, and I. Couckuyt. "Dataset of observables for small modular lead-cooled fast reactor MOX spent nuclear fuel". In: *Data in Brief* 60 (2025), p. 111513. ISSN: 2352-3409. DOI: 10.1016/j.dib.2025.111513.

[27] V. J. Casas-Molina, A. Hernandez-Solis, P. Romojaro, I. Merino-Rodríguez, and N. Aguilera-Gómez. "Dataset of observables for UOX and MOX spent fuel extracted from Serpent2 fuel depletion calculations for PWRs". In: *Data in Brief* 49 (2023), p. 109412. ISSN: 2352-3409. DOI: 10.1016/j.dib.2023.109412.

[28] S. J. Piet. "Selection of isotopes and elements for fuel cycle analysis". In: Idaho National Laboratory (INL). United States, 2009. URL: https://www.osti.gov/biblio/952008.

[29] C. Ferry, J.-P. Piron, and A. Ambard. "Effect of helium on the microstructure of spent fuel in a repository: An operational approach". In: *Journal of Nuclear Materials* 407.2 (2010), pp. 100–109. ISSN: 0022-3115. DOI: 10.1016/j.jnucmat.2010.09.034.

[30] A. Valls, M. Grivé, O. Riba, M. Morales, and K. Spahiu. "Estimation of the long term helium production in high burn-up spent fuel due to alpha decay and consequences for the canister". In: *MRS Proceedings* 1665 (2014), pp. 297–302. ISSN: 1946-4274. DOI: 10.1557/opl.2014.658.

[31] K. O. Ott and R. C. Borg. "Derivation of consistent measures for the doubling of fast breeder reactor fuel". In: *Nuclear Science and Engineering* 62.2 (1977), pp. 243–261. ISSN: 1943-748X. DOI: 10.13182/nse77-a26960.

[32] Nuclear Energy Agency. *Working Party on Scientific Issues of Advanced Fuel Cycles (WPFC) — oecd-nea.org*. https://www.oecd-nea.org/jcms/c_12783/working-party-on-scientific-issues-of-advanced-fuel-cycles-wpfc. [Accessed 17-05-2026].

[33] International Atomic Energy Agency. *Reload design and core management in operating nuclear power plants: experiences and lessons learned*. 1898. Vienna, Austria: International Atomic Energy Agency, 2020. URL: https://www.iaea.org/publications/13585/reload-design-and-core-management-in-operating-nuclear-power-plants.

[34] A. Stankovskiy and G. Van den Eynde. "Advanced method for calculations of core burn-up, activation of structural materials, and spallation products accumulation in accelerator-driven systems". In: *Science and Technology of Nuclear Installations* 2012 (2012), pp. 1–12. ISSN: 1687-6083. DOI: 10.1155/2012/545103.

Part VII
Appendix

A Derivation of the state-transition split

This appendix derives the expression for the charge extraction split x introduced in Equation 100.

Consider a batch slot whose charge contains n fuel types. Let $m_{chrg,\tau}$ denote the mass of package type τ currently in the charge position, and $m_{chrg} = \sum_\tau m_{chrg,\tau}$ the total charge mass. A mass $m_{chrg\rightarrow}$ must be extracted from the slot during the reload cascade.

In the general case, following a state transition, the charge is a mixture of fuel loaded under the previous-state fractions $\mathbf{r}_{prev}$ and fuel that has already entered at the current-state fractions $\mathbf{r}_{cur}$. The extraction of $m_{chrg\rightarrow}$ is split into two sequential parts:

- a quantity x is extracted using mass fractions $\mathbf{r}_{prev}$, i.e., a mass $x \cdot r_{prev,\tau}$ is taken from each fuel type $\tau \in \mathcal{T}_{P_{prev},in}$;
- the remainder $m_{chrg\rightarrow} - x$ is extracted at fractions $\mathbf{r}_{cur}$, i.e., a mass $(m_{chrg\rightarrow} - x) \cdot r_{cur,\tau}$ is taken from type $\tau \in \mathcal{T}_{P_{cur},in}$.

The remaining charge mass of type τ after both extractions is therefore

$$m^{rem}_{chrg,\tau} = m_{chrg,\tau} - x\, r_{prev,\tau} - (m_{chrg\rightarrow} - x)\, r_{cur,\tau} \quad ,\forall\tau \tag{108}$$

and the total remaining charge mass is $m_{chrg} - m_{chrg\rightarrow}$.

The split x is chosen so that the remaining charge is proportional to $\mathbf{r}_{cur}$, i.e.,

$$m^{rem}_{chrg,\tau} = r_{cur,\tau}\,(m_{chrg} - m_{chrg\rightarrow}) \quad ,\forall\tau. \tag{109}$$

Substituting Equation 108 into Equation 109 gives

$$m_{chrg,\tau} - x\, r_{prev,\tau} - (m_{chrg\rightarrow} - x)\, r_{cur,\tau} = r_{cur,\tau}(m_{chrg} - m_{chrg\rightarrow}) \tag{110}$$

$$\Leftrightarrow m_{chrg,\tau} - x\, r_{prev,\tau} - m_{chrg\rightarrow}\, r_{cur,\tau} + x\, r_{cur,\tau} = r_{cur,\tau}\, m_{chrg} - r_{cur,\tau}\, m_{chrg\rightarrow} \tag{111}$$

$$\Leftrightarrow m_{chrg,\tau} - r_{cur,\tau}\, m_{chrg} = x\,(r_{prev,\tau} - r_{cur,\tau}). \tag{112}$$

Dividing by $(r_{prev,\tau} - r_{cur,\tau})$, which is non-zero by assumption, yields

$$x = \frac{m_{chrg,\tau} - r_{cur,\tau}\, m_{chrg}}{r_{prev,\tau} - r_{cur,\tau}}. \tag{100}$$

This expression is valid for any type τ with $r_{prev,\tau} \neq r_{cur,\tau}$. If $r_{prev,\tau} = r_{cur,\tau}$, x is set to 0 in order to get back to the steady-state behavior.

Interpretation

The charge after a state transition is a mixture of two components: a fraction α loaded at $\mathbf{r}_{prev}$ and a fraction $(1-\alpha)$ loaded at $\mathbf{r}_{cur}$, so that

$$m_{chrg,\tau} = \alpha\, r_{prev,\tau}\, m_{chrg} + (1-\alpha)\, r_{cur,\tau}\, m_{chrg}. \tag{113}$$

Substituting into Equation 100:

$$x = \frac{\alpha\,(r_{prev,\tau} - r_{cur,\tau})\,m_{chrg}}{r_{prev,\tau} - r_{cur,\tau}} = \alpha\, m_{chrg}. \tag{114}$$

Hence x is exactly the mass of old-state fuel still present in the slot: phase 1 flushes all remaining old-state content, after which the residual charge is purely at $\mathbf{r}_{cur}$, and phase 2 proceeds with proportional extraction, as in the steady-state case.

The following three edge cases can arise:

1. When $\mathbf{r}_{prev} = \mathbf{r}_{cur}$, all fractions are equal and the numerator of Equation 100 vanishes: $x = 0$. The full extraction proceeds at $\mathbf{r}_{cur}$, recovering the steady-state proportional reload.

2. When $\alpha\, m_{chrg} \geq m_{chrg\rightarrow}$, the old-state content exceeds the reload mass. The target composition therefore cannot be reached in a single cycle. In such a case, x is truncated to $m_{chrg\rightarrow}$ and the full extraction is performed at $\mathbf{r}_{prev}$.

3. If $x < 0$, this would indicate that the charge already contains *less* of the previous-state species than the target proportions require. x is therefore set to 0 and the extraction proceeds at $\mathbf{r}_{cur}$. However, one should note that this case should never arise in practice, since the extraction, by design, only extracts the old fuel content with the old fractions.

B SCK CEN

SCK CEN is Belgium's nuclear research center, founded in 1952 and located in Mol. As a public institution, its mission is defined through a management contract with the Belgian federal government, which provides funding and sets research priorities aligned with national objectives in the areas of nuclear safety, radioactive waste management, and the long-term role of nuclear energy in the Belgian and European energy mix.

SCK CEN research spans reactor physics, fuel cycle analysis, material science, and radioprotection. Its strategic positioning in the current European nuclear landscape is shaped by two main projects. The first one is the Multi-purpose hYbrid Research Reactor for High-tech Applications (MYRRHA), an ADS reactor designed to transmute long-lived radioactive waste. Although its timeline has been extended, it remains a unique infrastructure and scientific project with no equivalent in Europe. The other strategic focus that has been developing in recent years is LEANDREA, Belgium's first Lead-cooled Fast Small Modular Reactor (LFR-SMR) demonstrator, targeted for operation by 2034. LEANDREA is developed within the EAGLES consortium, which includes industrial and research partners such as Ansaldo Nucleare, ENEA, and RATEN. Beyond these projects, the BR2 research reactor, one of the most powerful in the world, supports both fundamental research and applied activities such as nuclide production for both medical and industrial applications.

Nuclear expertise is quite scarce, and sustaining this specialized knowledge is therefore a strategic challenge for SCK CEN. The center maintains strong partnerships with Belgian universities, notably with the University of Liège, Ghent University, and KU Leuven, through joint PhD programs and research collaborations. These partnerships serve a dual purpose: they supply SCK CEN with specialized researchers and engineering students while also contributing to the training of nuclear engineers in Belgium.

SCK CEN operates with a diversified funding model. Government allocation accounts for roughly 25% of the budget, while approximately another half is generated through SCK CEN's own income: reactor irradiation services, research contracts, and intellectual property valorization. Dedicated government contracts fund specific strategic projects such as MYRRHA and the SMR-LFR initiative, and European programs contribute to a smaller share. As a foundation of public utility under private law, SCK CEN has no shareholders and reinvests its revenues into research.