A Report

ON

Development of Input Processing Module for NuPAC

BY

Name of Student	ID No.
Ivanshu Kaushik	2018A8PS0520P

AT

Atomic Energy Regulatory Board, Mumbai

A Practice School Station of

Birla Institute of Technology & Science, Pilani

June, 2020

A Report

ON

Development of Input Processing Module for NuPAC

BY

Name of Student	ID Nos	Discipline
Ivanshu Kaushik	2018A8PS0520P	ENI

Prepared in Partial fulfilment of the

Practice School – 1 Course No.

BITS C221

AT

Atomic Energy Regulatory Board, Mumbai

A Practice School Station of

Birla Institute of Technology & Science, Pilani

June, 2020

ACKNOWLEDGEMENT

I deem it a pleasure to acknowledge my sense of gratitude to my project guide Prof Prashant K Wali and Prof P Srinivasan under whom I have carried out the project work. Their incisive and objective guidance and timely advice encouraged me with constant flow of energy to continue the work.

I wish to reciprocate in full measure the kindness shown by Mr Aniket Gupta (AERB, Mumbai) who inspired me with his valuable suggestions in doing the project work

I shall remain grateful to Prof Souvik Bhattacharyya, Vice Chancellor, BITS Pilani, for providing me an opportunity to do the project work with utmost concentration and dedication

Finally, I must say that no height is ever achieved without some sacrifices made at some end and it is here I owe my special debt to my parents and friends for showing their generous love and care throughout the entire period of time.

Date: 22.06.2020

Name: Ivanshu Kaushik

ID No.: 2018A8PS0520P

Birla Institute of Technology and Science, Pilani

(Rajasthan)

Practice School Division

Station: Atomic Energy Regulatory Board **Centre:** Mumbai

Duration: 18th May to 27th June 2010 **Date of Start:** 18th May 2020

Date of Submission: 25th June 2020

Title of Project: Development of Input Processing Module for NuPAC

ID No: 2018A8PS0520P

Name: Ivanshu Kaushik

Discipline of Student: Electronics and Instrumentation

Name of Expert: Mr. Aniket Gupta

Designation of the expert: Scientist at AERB

Name of the PS 1 faculty: Prof. Prashant K Wali

Key Words: NuPAC, Safety, PWR, PHWR, RELAP5 etc

Project Areas: Python, Programming, Computer Science

Abstract: Input Processing Module is an attempt to develop a module which can take inputs, and check them according to a rule file and give corresponding results.

Signature of Student Signature of PS-1 Faculty

Date: 25/06/2020 Date:

Contents

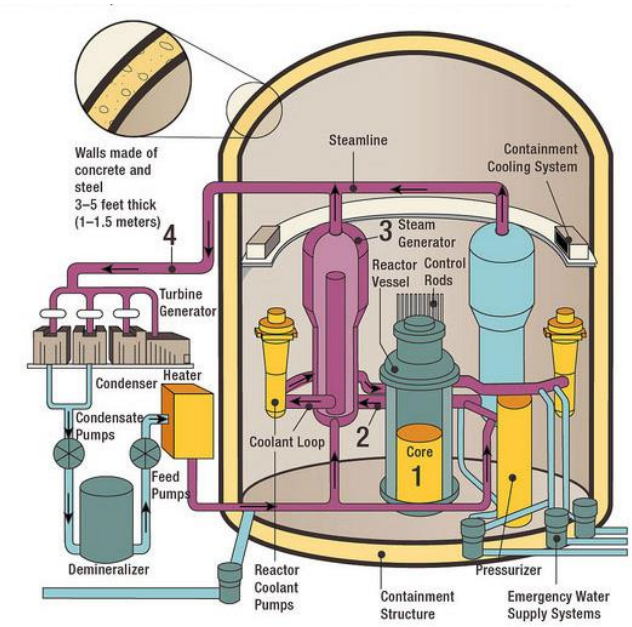
INTRODUCTION

As a part of the safety analysis code development programme, 1-D transient thermal hydraulic code, Nuclear Plant Analyser Code (NuPAC) is being developed. Transient thermal-hydraulic, two- phase phenomena were calculated from formulations of one-dimensional, homogeneous, equilibrium conservation equations for mass, momentum and energy.

NuPAC code is based on finite volume method and employs a semi-implicit numerical scheme for solution of governing equations. The NuPAC code has the capability to simulate closed loop, open loop and parallel flow systems. It is an indigenous replacement for the earlier used code RELAP5 (Reactor Excursion and Leak Analysis Program) which was developed at Idaho National University, United States of America.

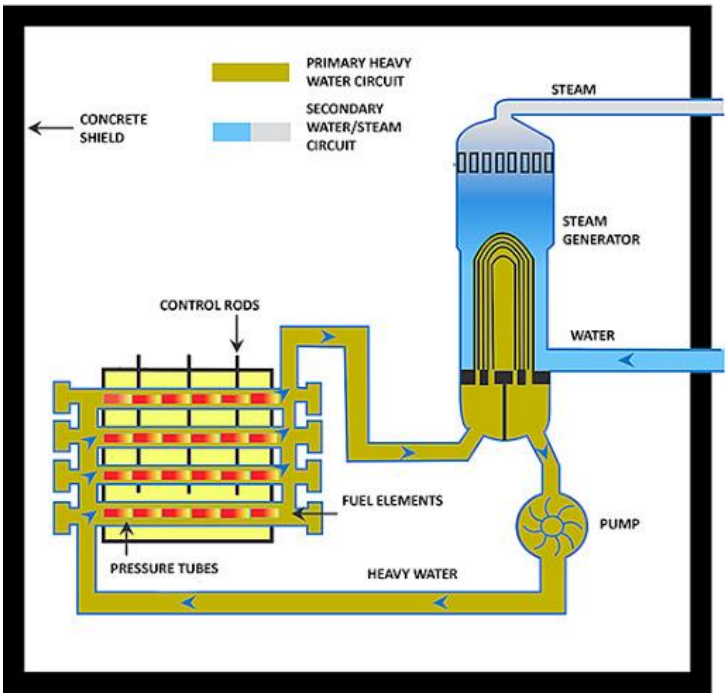
RELAP5 (Figure 1.1)

RELAP5 was developed in early 1980's in USA to meet the United States reactors safety needs. RELAP5 code is primarily for the PWR (Pressurised Water Reactor) type Nuclear Reactors. RELAP5-3D is a transient, two-fluid model for flow of a two-phase vapor/gas-liquid mixture that can contain non-condensable components in the vapor/gas phase and/or a soluble component in the liquid phase.



A typical PWR type Nuclear Reactor (Figure 1.2)

However, the majority of Indian reactors are PHWR (Pressurised Heavy Water) type nuclear reactors.



A typical PHWR type Nuclear Reactor (Figure 1.3)

So, the RELAP5 code is not directly applicable to meet the safety demands of our Indian Nuclear Reactors. The RELAP5 code used to be customised for use in PHWR type Nuclear reactors but this process was very long and inefficient. So AERB, under the banner of DAE (Department of Atomic Energy) decided to develop our own Nuclear safety simulation code called the NuPAC.

About Atomic Energy Regulatory Board (AERB)

The Atomic Energy Regulatory Board (AERB) was constituted on 15 November 1983 by the President of India exercising the powers conferred by Section 27 of the Atomic Energy Act, 1962 (33 of 1962) to carry out certain regulatory and safety functions under the Act. The regulatory authority of AERB is derived from the rules and notifications promulgated under the Atomic Energy Act, 1962 and the Environmental (Protection) Act, 1986. The headquarters is in Mumbai.

The mission of the Board is to ensure that the use of ionising radiation and nuclear energy in India does not cause undue risk to health and the environment.

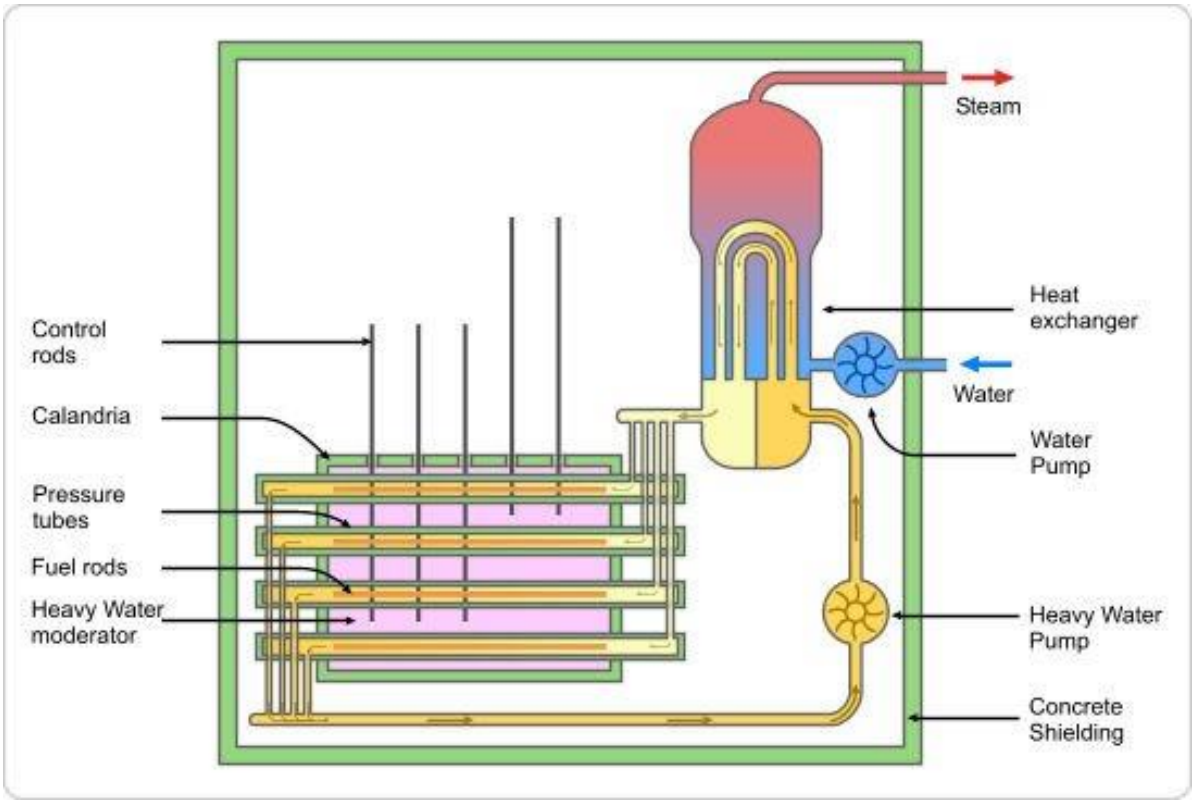
AERB secretariat has Nine technical divisions & two supporting divisions. The heads and directors of divisions constitute the Executive Committee which meets periodically with Chairman, AERB and Executive Director, AERB to take decisions on important policy matters related to the management of the Secretariat of the Board.

The different divisions of AERB are:

1. OPSD: Operating Plant Safety Division
2. NPSD: Nuclear Projects Safety Division
3. RSD: Radiological Safety Division
4. NSAD: Nuclear Safety Analysis Division
5. R&DD: Resources & Documentation Division
6. DRI: Directorate of Regulatory Inspection
7. DRA&C: Directorate of Regulatory Affairs & Communications
8. DRP&E: Directorate of Radiation Protection & Environment
9. SRI: Safety Research Institute, Kalpakkam

About PHWR type Nuclear Reactors

A **pressurized heavy water reactor** (PHWR) is a nuclear power reactor, commonly using unenriched natural uranium as its fuel, that uses heavy water (deuterium oxide D_2O) as its coolant and moderator. The heavy water coolant is kept under pressure, allowing it to be heated to higher temperatures without boiling, much as in a typical pressurized water reactor. While heavy water is significantly more expensive than ordinary light water, it yields greatly enhanced neutron economy, allowing the reactor to operate without fuel enrichment facilities (mitigating the additional capital cost of the heavy water) and generally enhancing the ability of the reactor to efficiently make use of alternate fuel cycles.

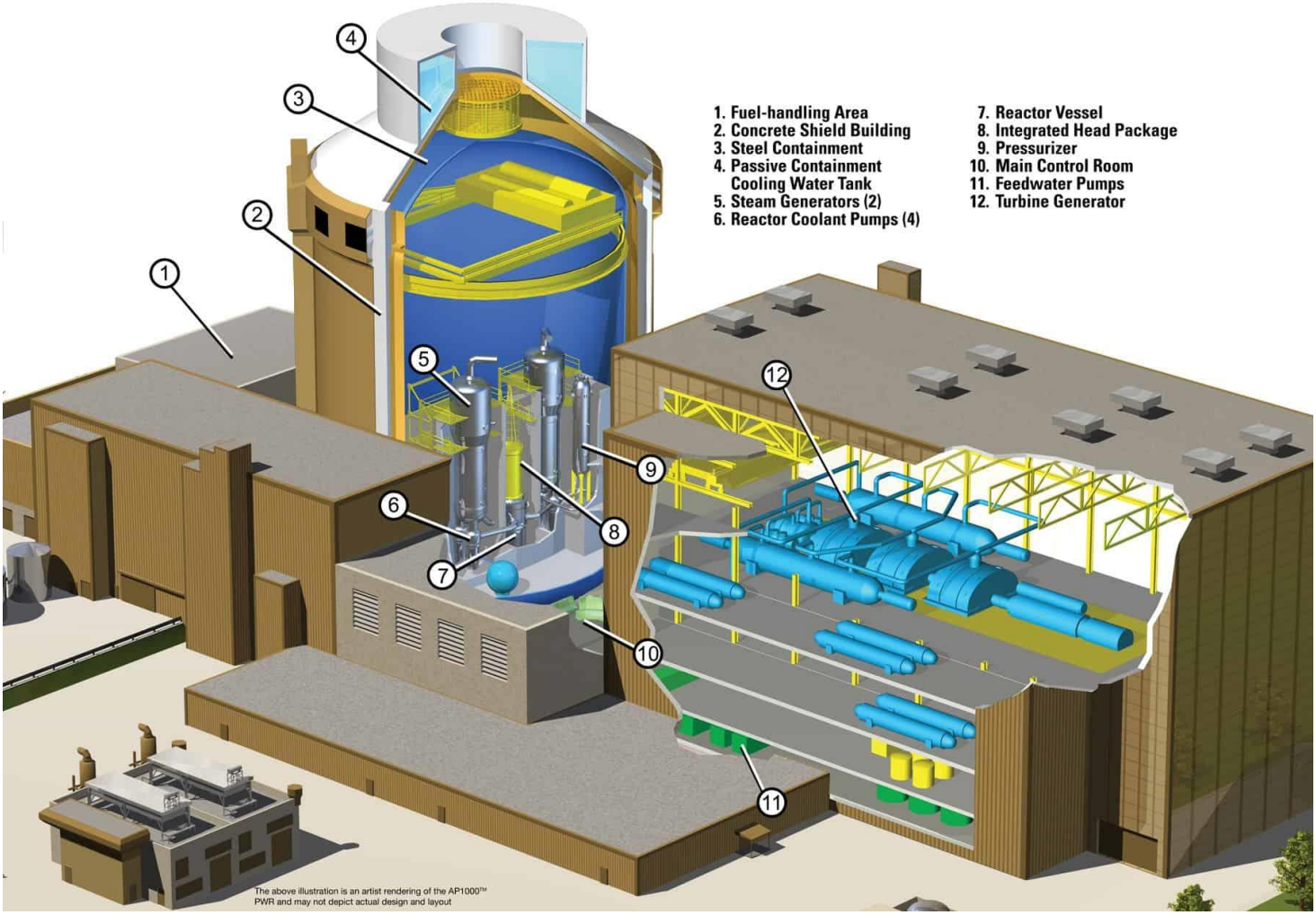


A PHWR type Nuclear Reactor (Figure 4.1)

The major components of a PHWR type Nuclear Power Plant are:

1. Reactor Building
2. Cooling Tower
3. Turbine Building
4. Auxiliary Building
5. Spent Fuel Storage Pool
6. Control Room

4.1 Reactor Building



A Reactor Building (Figure 4.2)

A **reactor building** is a general term for a building that houses a reactor of some type. In particular, it often refers to a building containing a nuclear reactor. This can also be used to refer to pressure sealed buildings containing nuclear reactor, though it would be more correct to call them containment buildings**.**

A reactor building, in its most common usage, is a reinforced steel or lead structure enclosing a nuclear reactor. It is designed,

in any emergency, to contain the escape of radioactive steam or gas to a maximum pressure in the range of 275 to 550 kPa (40 to 80 psi). The containment is the fourth and final barrier to radioactive release (part of a nuclear reactor's defence in depth strategy), the first being the fuel ceramic itself, the second being the metal fuel cladding tubes, the third being the reactor vessel and coolant system.

4.2 Cooling Tower

A Cooling Tower (Figure 4.3)

A **cooling tower** is a heat rejection device that rejects waste heat to the atmosphere through the cooling of a water stream to a lower temperature. Cooling towers may either use the evaporation of water to remove process heat and cool the working fluid to near the wet-bulb air temperature.

The process begins with a reactor pressure vessel which houses fuel assemblies. These fuel assemblies cause fission chain

reactions, which release great amounts of heat. This process heats up water, which is transported to the heat exchanger. After different water is transformed into steam in a heat exchanger, it flows through a set of turbines that are attached to a steam generator, which creates energy. It then travels through a condenser, which converts it back to liquid form. The condenser acts as a reverse heat exchanger and cools the steam into water. The condenser takes in cold water and generates hot water. The cooling tower transforms the hot water back to cold water. Because there are three contained water cycles, (the primary which reacts directly with the fuel assemblies, secondary, which cools the primary, and tertiary, which cools the secondary) the water used in the condenser is not radioactive and can be released into the environment through cooling towers.

4.3 Turbine Building

The purpose of the steam turbine is to convert the heat contained in steam into mechanical energy. The engine house with the steam turbine is usually structurally separated from the main reactor building. It is aligned so as to prevent debris from the destruction of a turbine in operation from flying towards the reactor.

Turbine Building (Figure 4.4)

4.4 Auxiliary Building

A building at a nuclear power plant, which is frequently located adjacent to the reactor containment structure, and houses most of the auxiliary and safety systems associated with the reactor, such as radioactive waste systems, chemical and volume control systems, and emergency cooling water systems.

4.5 Spent Fuel Storage Pool

Spent fuel storage pool (Figure 4.5)

Spent fuel pools (SFP) are storage pools (or "ponds" - UK usage) for spent fuel from nuclear reactors. They are typically 40 or more feet (12 m) deep, with the bottom 14 feet (4.3 m) equipped with storage racks designed to hold fuel assemblies removed from reactors. A reactor's local pool is specially designed for the reactor in which the fuel was used and is situated at the reactor site. Such pools are used for immediate "cooling" of the fuel rods, which allows short-lived isotopes to decay and thus reduce the ionising radiation emanating from the rods. The water cools the fuel and provides radiological protection shielding from their radiation.

4.6 Control Room

A nuclear power plant control room (Figure 4.6)

Control Room in A PHWR Nuclear Power Plant Facility. Main Control Rooms. The Main Control Room in all countries is used as the main control centre in the plant. The Plant operating supervisor and senior operating personnel operate and monitor major plant equipment.

Software Requirements and Specifications

3.1 Industry Perspective

The idea behind this project is to develop a software code for input processing modules to simulate the thermal and hydraulic condition and postulated failures conditions using NuPAC.

The purpose of this project work is to provide and Input Processing Module for NuPAC which will process the inputs and validate them according to a rule file and then save the output in a way that it can be used further for stability testing, simulation and analysis of various parameters of Nuclear Power Plants. This will ensure the independence of our Nuclear Industry from using foreign technologies and relying on indigenous technologies instead. The Input Processing module makes it possible for the developers who are concerned about major safety parameters to not pay attention to inputting the files and thus ensuring an efficient and fast Safety Analysis in the Nuclear Power Plants to ensure that no accidents take place.

3.2 User Characteristics

The users who are making this program are trying to make use of the latest technologies so that with time this module doesn't

get outdated and the algorithm involves proficiency in the following areas:

- Python Programming Language and all the concepts of Object-oriented programming and especially abstraction methods
- List Methods and List Comprehensions, Anonymous functions, Dictionaries in Python and all the various methods involved with accessing dictionaries and using them. Loops including for loop, while loop, if loop, else loop etc.
- Different modules in Python including RegExp (re module) Itertools (itertools module) and islice method of the itertools module and use of List iterators and Iteration tools
- Use of PyCharm as IDE (Integrated Development Environment) as it provides a free development environment making it easier to code in Python
- Use of text editors like Sublime Text 3 as the input and output involve parsing, reading and writing to text files.

3.3 Functional Requirements

The functional requirement of this project is to develop a software code for input processing module to simulate the Thermal and Hydraulic condition of Reactor Core under Normal Operating Condition and Postulated Failure Conditions. The parameters for which this code will be tested are Temperature of Coolant, Temperature of Clad, Coolant Flow, Pressure of Coolant, Moderator Level, Neutron Flux, Power Level etc.

The developed code will be testable and user-friendly and secure for given operating parameters and to simulate the effects on reactor core.

3.4 Performance Requirements

The algorithm developed involves reading the input file, Parsing it and then turning the input into a Key-Value Pair. The input is nested so the Dictionary created will also be nested.

The input file is to be checked according to a set of rules which is also read as the rules file. If the input file matches all the rules and there is no error, then the input file is to be written into another file as a Key-Value Pair which can be inputted by the developers as a Module. The performance requirement is to meet the following requirements:

- Accuracy
- Consistency
- Time Response

Project Plan

The project plan can be divided into the following major aspects:

1. Reading the Rules File
2. Reading the Input File

5.1 Reading the Rules File

The Rules file needs to be read and stored in a manner in which we can later retain the information and use it for validating the input file. The Rules file maybe something like:

```
STRU CONTAINMENT  (O, BL, S, ,)
    STRU MATE     (O, BL, M,  ,)
        NAME    (C, C1, S, UNIQUE AMONG MATE,)
        ARR      (C, C1, S, <EQUIDIS | VOLUME | PROGRESS | FREE>)
        CP   (C, F0, S, >0)
        DL (C, F0, S, >0)
        EMI   (C, F0, S, >0)
        RHO   (C, F0, S, >0)
        MESH   (C, I0, S, >0)
        FAC     (C IF ARR == PROGRESS, F0, S, >0)
    END

    STRU COMP             (O, BL, M, ,)
        NAME        (C, C1, M, ,)
        PHAS        (C, C1, S,<FLUID | GAS>)
        MLMA        (C, F0, S, >0 ,)
        CDIF        (C IF PHAS == GAS, F0, S, >0,)
        CSTD        (C IF PHAS == GAS, F0, S, >0,)
        CDIS        (C IF PHAS == GAS, F0, S, >0,)
        CP       (C, F0, S, <FUNCTION | R0 VALUE>)
        DL       (C, F0, S, <FUNCTION | R0 VALUE>)
        ETA      (C, F0, S, <FUNCTION | R0 VALUE>)
    END

    STRU ZONE                 (0, BL, M,  )
        NAME          (C, C1, S, UNIQUE AMONG BLOCK ZONE )
        MODE         (C, C1, S, <NONEQUIL | EQUIL>)
        ROOM                 (O, I0, S, >0 )
        STRU GEOM              (C, BL, S, )
            VOLU           (C, F0, S, >0)
            ELEV          (C, F0, S, >SELE)
            SELE         (C, F0, S, )
            ASUM        (C, F0, S, >0)
        END
        STRU INIT               (C, BL, M,  )
            STRU COND            (C, BL. M,  )
                COMP          (C, C1, S, <PRES | SATUR | TEMP | AIR)
                VALU           (O, F0, S, >0)
```

```
                UNIT            (O, <C1 | C0 >, S, <BAR | C | '%'>)
            TYPE           (O, C1, S, <BAR | C | '%'>)
            END
        END
    END
    STRU CONN       (O, BL, M,  )
        NAME       (C, C1, S, UNIQUE AMONG BLOCK ZONE)
        FROM       (C, C1, S, VALUE SHOULD BE NAME OF ZONE)
        TO     (C, C1, S, VALUE SHOULD BE NAME OF ZONE)
        TYPE    (C, C1, S, <ATMOS_JU | RUPTURE>)
        LEN         (C, F1, S, >0)
        SECT         (C, F1, S, >0)
        ZETE         (C, F1, S, >0)
    END
END
```

Sample Rules File (Figure 5.1)

So we need to read the rules file line by line and then ensure that the way in which it is stored retains every bit of information about the rules file. For this, we create a list called bi and then we append 1 to it if we encounter the keyword ‘STRU’, -1 if we encounter ‘END’ and in all other cases we append 0 to it. So we create another list called initial_rules and we add to it the rules file line by line.

```
with open('rules1.txt') as rule_file:
    for line in rule_file:
        if not line.isspace():
            initial_rules.append(line.strip())
```

Reading the Rules file line by line (Figure 5.2)

Now lets create the bi list in the manner mentioned above.

```
for rule in initial_rules:
    if stru in rule:
        bi.append(1)
    elif end in rule:
        bi.append(-1)
    else:
        bi.append(0)
```

Creating the bi list (Figure 5.3)

Now we create a list called sum_bi in which we add all the elements of the list bi until that corresponding element. This list

contains the information about the level of nesting with the value itself being the level. For e.g. a value of 2 in sum_bi denotes that this is second level of nesting and a value of 4 shows that this is fourth level

```
sum_bi = []

for i, b in enumerate(bi):
    sum_bi.append(cum_sum(i, bi[:i]))
```

Creation of the list called sum_bi (Figure 5.4)

During the creation of the sum_bi list we have used a function called cum_sum which performs the function of summing up the values of the list bi until the corresponding element. Next we make a new list which holds all the nested components individually and ensure that the information about nesting is not lost. For that we first create a list called level_two_index which holds the indexes of when nesting starts and ends. Then we add those values to another list called initial_rules_two.

```
lvl_two = []
lvl_two_index = []
sum_bi_two = []

for i, val in enumerate(initial_rules_two):
    if sum_bi[i] == 2 and sum_bi[i - 1] == 1:
        lvl_two_index.append(i)

lvl_two_index.append(len(initial_rules_two))

rules_three = []
for i in range(len(lvl_two_index) - 1):
    rules_three.append(initial_rules_two[lvl_two_index[i]:lvl_two_index[i +
1]])
    sum_bi_two.append(sum_bi[lvl_two_index[i]:lvl_two_index[i + 1]])
```

Creating sum_bin_two list (Figure 5.5)

We finally create the final Dictionary which holds the values of all the Rules files and call it rule

```
rule = {}
```

Creating the rule dictionary (Figure 5.6)

We need to populate this dictionary with key value pairs and ensure that the nesting components are clearly visible.

```
for i, val in enumerate(rules_three):
    if 3 in sum_bi_two[i]:
        key = val[0][1]
        for j, m in enumerate(sum_bi_two[i]):
            if m == 3 and sum_bi_two[i][j - 1] == 2:
                rule[lvl_one[0]][key][rules_three[i][j][1]] = {}
                lvl_three.append(rules_three[i][j][1])
```

Adding key-value pairs to the dictionary rule (Figure 5.7)

Thus we can populate the dictionary in this manner. We need to repeat this step for further nesting levels as this ensures that the corresponding information about the key-value pairs is not lost

```
for i, val in enumerate(rules_three):
    if 4 in sum_bi_two[i]:
        key = val[0][1]
        for j, m in enumerate(sum_bi_two[i]):
            if m == 3 and sum_bi_two[i][j - 1] == 2:
                key_one = rules_three[i][j][1]

            if m == 4 and sum_bi_two[i][j - 1] == 3:
                rule[lvl_one[0]][key][lvl_three[1]][rules_three[i][j][1]] =
{}
                lvl_four.append(rules_three[i][j][1])

for i in range(len(rule)):
    for j in range(len(rule[lvl_one[i]])):
        key = rules_three[j][0][1]
        for k in range(len(rules_three[j])):
            if k > 0 and sum_bi_two[j][k] == 3 and sum_bi_two[j][k - 1] ==
3:
                print(rules_three[j][k], k)
                rule[lvl_one[i]][key][lvl_three[0]][rules_three[j][k][0]] =
rules_three[j][k][1:]

            elif k > 0  and sum_bi_two[j][k] == 4 and sum_bi_two[j][k - 1]
== 4:

rule[lvl_one[i]][key][lvl_three[1]][lvl_four[0]][rules_three[j][k][0]] =
rules_three[j][k][1:]
```

Making the complete rule dictionary (Figure 5.8)

Once we are done with the steps mentioned, we finally obtain a dictionary which holds all the information in a key-value pair and it looks like:

{'CONTAINMENT': {'MATE': {'NAME': ['(C,', 'C1,', 'S,', 'UNIQUE', 'AMONG', 'MATE,)'], 'ARR': ['(C,', 'C1,', 'S,', '<EQUIDIS', '|',

'VOLUME', '|', 'PROGRESS', '|', 'FREE>)'], 'CP': ['(C,', 'F0,', 'S,', '>0)'], 'DL': ['(C,', 'F0,', 'S,', '>0)'], 'EMI': ['(C,', 'F0,', 'S,', '>0)'], 'RHO': ['(C,', 'F0,', 'S,', '>0)'], 'MESH': ['(C,', 'I0,', 'S,', '>0)'], 'FAC': ['(C', 'IF', 'ARR', '==', 'PROGRESS,', 'F0,', 'S,', '>0)'], 'END': []}, 'COMP': {'NAME': ['(C,', 'C1,', 'M,', ',)'], 'PHAS': ['(C,', 'C1,', 'S,<FLUID', '|', 'GAS>)'], 'MLMA': ['(C,', 'F0,', 'S,', '>0', ',)'], 'CDIF': ['(C', 'IF', 'PHAS', '==', 'GAS,', 'F0,', 'S,', '>0,)'], 'CSTD': ['(C', 'IF', 'PHAS', '==', 'GAS,', 'F0,', 'S,', '>0,)'], 'CDIS': ['(C', 'IF', 'PHAS', '==', 'GAS,', 'F0,', 'S,', '>0,)'], 'CP': ['(C,', 'F0,', 'S,', '<FUNCTION', '|', 'R0', 'VALUE>)'], 'DL': ['(C,', 'F0,', 'S,', '<FUNCTION', '|', 'R0', 'VALUE>)'], 'ETA': ['(C,', 'F0,', 'S,', '<FUNCTION', '|', 'R0', 'VALUE>)'], 'END': []}, 'ZONE': {'GEOM': {'VOLU': ['(C,', 'F0,', 'S,', '>0)'], 'ELEV': ['(C,', 'F0,', 'S,', '>SELE)'], 'SELE': ['(C,', 'F0,', 'S,', ')'], 'ASUM': ['(C,', 'F0,', 'S,', '>0)']}, 'INIT': {'COND': {'COMP': ['(C,', 'C1,', 'S,', '<PRES', '|', 'SATUR', '|', 'TEMP', '|', 'AIR)'], 'VALU': ['(O,', 'F0,', 'S,', '>0)'], 'UNIT': ['(O,', '<C1', '|', 'C0', '>,', 'S,', '<BAR', '|', 'C', '|', "'%'>)"], 'TYPE': ['(O,', 'C1,', 'S,', '<BAR', '|', 'C', '|', "'%'>)"]}}, 'NAME': ['(C,', 'C1,', 'S,', 'UNIQUE', 'AMONG', 'BLOCK', 'ZONE', ')'], 'MODE': ['(C,', 'C1,', 'S,', '<NONEQUIL', '|', 'EQUIL>)'], 'ROOM': ['(O,', 'I0,', 'S,', '>0', ')'], 'END': []}, 'CONN': {'NAME': ['(C,', 'C1,', 'S,', 'UNIQUE', 'AMONG', 'BLOCK', 'ZONE)'], 'FROM': ['(C,', 'C1,', 'S,', 'VALUE', 'SHOULD', 'BE', 'NAME', 'OF', 'ZONE)'], 'TO': ['(C,', 'C1,', 'S,', 'VALUE', 'SHOULD', 'BE', 'NAME', 'OF', 'ZONE)'], 'TYPE': ['(C,', 'C1,', 'S,', '<ATMOS_JU', '|', 'RUPTURE>)'], 'LEN': ['(C,', 'F1,', 'S,', '>0)'], 'SECT': ['(C,', 'F1,', 'S,', '>0)'], 'ZETE': ['(C,', 'F1,', 'S,', '>0)'], 'END': []}}}

The rule dictionary having all the values in the required format (Figure 5.9)

5.2 Reading the input file

Just like how we read the rules file, we need to read the input file in a similar way. That is we need to ensure that we don’t lose the necessary data and we are able to execute all the required functions without any problem. The input file looks like:

```
STRU CONTAINMENT
    STRU MATE
        NAME BET1
        ARR PROGRESS
        CP 869.207
        DL 1.44704
        EMI 1.0
        RHO 2343.32
        MESH 9
        FAC 4.0
    END
    STRU COMP
        NAME WATER
        PHAS FLUID
        MLMA 18.016
        CP FUNCTION
        DL FUNCTION
        ETA FUNCTION
    END
    STRU COMP
        NAME N2
        PHAS GAS
        MLMA 28.01
        CDIF 2.0E-5
        CSTD 0.0
        CDIS 0.0
        CP FUNCTION
        DL FUNCTION
        ETA FUNCTION
    END
    STRU COMP
        NAME STEAM
        PHAS GAS
        MLMA 18.016
        CDIF .58E-4
        CSTD 0.0
        CDIS 0.0
        CP FUNCTION
        DL FUNCTION
        ETA FUNCTION
    END
    STRU COMP
        NAME H2
        PHAS GAS
        MLMA 2.016
        CDIF 6.11E-5
        CSTD 0.0
        CDIS 0.0
        CP FUNCTION
        DL FUNCTION
        ETA FUNCTION
    END
    STRU COMP
        NAME O2
        PHAS GAS
        MLMA 32.0
        CDIF 2.0E-5
        CSTD 0.0
        CDIS 0.0
        CP FUNCTION
        DL FUNCTION
```

```
        ETA FUNCTION
    END
    STRU COMP
        NAME CO
        PHAS GAS
        MLMA 28.01
        CDIF 2.0E-5
        CSTD 0.0
        CDIS 0.0
        CP FUNCTION
        DL FUNCTION
        ETA FUNCTION
    END
    STRU COMP
        NAME CO2
        PHAS GAS
        MLMA 44.01
        CDIF 2.0E-5
        CSTD 0.0
        CDIS 0.0
        CP FUNCTION
        DL FUNCTION
        ETA FUNCTION
    END
    STRU ZONE
        NAME ENVIRON
        MODE NONEQUIL
        ROOM 2
        STRU GEOM
            VOLU 1.0E8
            ELEV 100.0
            SELE 0.0
            ASUM 1000000.0
        END
        STRU INIT
            STRU COND
                COMP PRESS
                VALU 1.013
                UNIT BAR
            END
            STRU COND
                COMP SATUR
                UNIT '%'
                VALU 40.0
            END
            STRU COND
                COMP TEMP
                VALU 25.0
                UNIT C
            END
            STRU COND
                COMP AIR
                TYPE REST
            END
        END
    END

    STRU ZONE
        STRU GEOM
            VOLU 5.0E4
            ELEV 50.0
```

```
            SELE 0.0
            ASUM 1000.0
        END
        STRU INIT
            STRU COND
                COMP PRESS
                VALU 1.0
                UNIT BAR
            END
            STRU COND
                COMP SATUR
                UNIT '%'
                VALU 40.0
            END
            STRU COND
                COMP TEMP
                VALU 23.0
                UNIT C
            END
            STRU COND
                COMP AIR
                TYPE REST
            END
        END
    END
    STRU CONN
        NAME V2
        FROM C1
        TO ENVIRON
        TYPE ATMOS_JUN
        LEN 0.3
        SECT 7.2E-6
        ZETB 5.0
        ZETE 5.0
    END
    STRU CONN
        NAME V1
        FROM C1
        TO ENVIRON
        TYPE RUPTURE
        LEN 0.3
        SECT 1.5
        ZETB 5.0
        ZETE 5.0
        DPBE 4.bar
        DPEB 4.bar
    END
END
```

A sample input file (Figure 5.10)

So, we need to ensure that the nesting levels are ensured. Now we need to parse this input file in a similar manner to how we did it with the rule file. So, we create an initial list and add all these values to that list

```
with open('input_file.txt') as input_file:
    for line in input_file:
        if not line.isspace():
            initial_input.append(line.strip())
```

Reading the input file line by line (Figure 5.11)

Now just like we did for the Rules file, we create two lists called binary and sum_one_bin. For forming the list binary, we append 1 to it if we encounter the keyword 'STRU', -1 if we encounter 'END' and 0 in all other cases. Now using a function called as cumulative_sum we sum the values of all elements of bin until the corresponding value and store it in the list sum_one_bin.

```
def cumulative_sum(ind, bin_list):
    cum_sum = 0
    for i, val in enumerate(bin_list):
        if i <= ind:
            cum_sum = cum_sum + val
        else:
            break
    return cum_sum
```

The function cumulative_sum (Figure 5.12)

```
binary = []

for number in final_input:
    if 'STRU' in number:
        binary.append(1)

    elif 'END' in number:
        binary.append(-1)
    else:
        binary.append(0)
```

Creating the list binary (Figure 5.13)

```
for bin in binary:
    sum_one_bin.append(cumulative_sum(na, binary))
    na += 1
```

Creating the list sum_one_bin (Figure 5.14)

Now we need to ensure that the input file holds all these values and can show all the necessary values inside a dictionary. We call that dictionary input and create the key value pairs so that we can check according to the rules file easily so we initialise a dictionary with the name input.

```
input = {}
```

Creating the dictionary input (Figure 5.15)

Now just like we did for the rules file, we need to append the values to the input dictionary in key-value format

```
for i, val in enumerate(final_input):
    if sum_one_bin[i] == 1 and (sum_one_bin[i - 1] == 0 or i == 0):
        temp = val.split()
        input[temp[1]] = {}
```

Initialising the level one of the dictionary input (Figure 5.16)

```
for i in range(len(final_input_two)):
    for j, ip in enumerate(final_input_two[i]):
        if sum_bin_two[i][j] == 2 and sum_bin_two[i][j - 1] == 1:
            temp = ip.split()
            # print(temp[1])
            if level_two[i].count(temp[1]) > 1:
                input[level_one[i]][temp[1]] = []
```

Initialising the level two of the dictionary input (Figure 5.17)

```
for i in range(len(sum_bin_three)):
    for j, val in enumerate(sum_bin_three[i]):
        if 3 in val:
            for k, s in enumerate(val):
                if s == 3 and sum_bin_three[i][j][k - 1] == 2:
                    temp = final_input_three[i][j][k].split()
                    temp[1] = {}

                    input[level_one[i]][level_two[i][j]][x].append(temp[1])
```

Initialising the level three of the dictionary input (Figure 5.18)

```
for i, one in enumerate(final_input_three):
    for j, two in enumerate(final_input_three[i]):
        if 4 in sum_bin_three[i][j]:
            for k, three in enumerate(sum_bin_three[i][j]):
```

```
                if three == 4 and sum_bin_three[i][j][k - 1] == 3:
input[level_one[i]][key[1]][z][1][level_three[i][1]][temp[1]].append([])
```

Initialising the level four of the dictionary input (Figure 5.19)

After following the mentioned steps, we arrive at the dictionary input which has been populated with the key value pairs accordingly

{'CONTAINMENT': {'MATE': {'NAME': 'BET1', 'ARR': 'PROGRESS', 'CP': '869.207', 'DL': '1.44704', 'EMI': '1.0', 'RHO': '2343.32', 'MESH': '9', 'FAC': '4.0'}, 'COMP': [[{'NAME': 'WATER'}, {'PHAS': 'FLUID'}, {'MLMA': '18.016'}, {'CP': 'FUNCTION'}, {'DL': 'FUNCTION'}, {'ETA': 'FUNCTION'}], [{'NAME': 'N2'}, {'PHAS': 'GAS'}, {'MLMA': '28.01'}, {'CDIF': '2.0E-5'}, {'CSTD': '0.0'}, {'CDIS': '0.0'}, {'CP': 'FUNCTION'}, {'DL': 'FUNCTION'}, {'ETA': 'FUNCTION'}], [{'NAME': 'STEAM'}, {'PHAS': 'GAS'}, {'MLMA': '18.016'}, {'CDIF': '.58E-4'}, {'CSTD': '0.0'}, {'CDIS': '0.0'}, {'CP': 'FUNCTION'}, {'DL': 'FUNCTION'}, {'ETA': 'FUNCTION'}], [{'NAME': 'H2'}, {'PHAS': 'GAS'}, {'MLMA': '2.016'}, {'CDIF': '6.11E-5'}, {'CSTD': '0.0'}, {'CDIS': '0.0'}, {'CP': 'FUNCTION'}, {'DL': 'FUNCTION'}, {'ETA': 'FUNCTION'}], [{'NAME': 'O2'}, {'PHAS': 'GAS'}, {'MLMA': '32.0'}, {'CDIF': '2.0E-5'}, {'CSTD': '0.0'}, {'CDIS': '0.0'}, {'CP': 'FUNCTION'}, {'DL': 'FUNCTION'}, {'ETA': 'FUNCTION'}], [{'NAME': 'CO'}, {'PHAS': 'GAS'}, {'MLMA': '28.01'}, {'CDIF': '2.0E-5'}, {'CSTD': '0.0'}, {'CDIS': '0.0'}, {'CP': 'FUNCTION'}, {'DL': 'FUNCTION'}, {'ETA': 'FUNCTION'}], [{'NAME': 'CO2'}, {'PHAS': 'GAS'}, {'MLMA': '44.01'}, {'CDIF': '2.0E-5'}, {'CSTD': '0.0'}, {'CDIS': '0.0'}, {'CP': 'FUNCTION'}, {'DL': 'FUNCTION'}, {'ETA': 'FUNCTION'}]], 'ZONE': [[{'GEOM': {'VOLU': '1.0E8', 'ELEV': '100.0', 'SELE': '0.0', 'ASUM': '1000000.0'}}, {'INIT': {'COND': [[{'COMP': 'PRESS'}, {'VALU': '1.013'}, {'UNIT': 'BAR'}], [{'COMP': 'SATUR'}, {'UNIT': "'%'"}, {'VALU': '40.0'}], [{'COMP': 'TEMP'}, {'VALU': '25.0'}, {'UNIT': 'C'}], [{'COMP': 'AIR'}, {'TYPE': 'REST'}]]}},

{'NAME': 'ENVIRON'}, {'MODE': 'NONEQUIL'}, {'ROOM': '2'}], [{'GEOM': {'VOLU': '5.0E4', 'ELEV': '50.0', 'SELE': '0.0', 'ASUM': '1000.0'}}, {'INIT': {'COND': [[{'COMP': 'PRESS'}, {'VALU': '1.0'}, {'UNIT': 'BAR'}], [{'COMP': 'SATUR'}, {'UNIT': "'%'"}, {'VALU': '40.0'}], [{'COMP': 'TEMP'}, {'VALU': '23.0'}, {'UNIT': 'C'}], [{'COMP': 'AIR'}, {'TYPE': 'REST'}]]}}, {'NAME': 'C1'}, {'MODE': 'NONEQUIL'}, {'ROOM': '1'}]], 'CONN': [[{'NAME': 'V2'}, {'FROM': 'C1'}, {'TO': 'ENVIRON'}, {'TYPE': 'ATMOS_JUN'}, {'LEN': '0.3'}, {'SECT': '7.2E-6'}, {'ZETB': '5.0'}, {'ZETE': '5.0'}], [{'NAME': 'V1'}, {'FROM': 'C1'}, {'TO': 'ENVIRON'}, {'TYPE': 'RUPTURE'}, {'LEN': '0.3'}, {'SECT': '1.5'}, {'ZETB': '5.0'}, {'ZETE': '5.0'}, {'DPBE': '4.bar'}, {'DPEB': '4.bar'}]]}}

A simplified version of the final value obtained after processing the input file (Figure 5.20)

Future Work

The code developed so far needs to be improved to meet the PEP8 (Python Enhancement Program) standards which recommend the styling aspects about the code. These are minor modifications. For proper publication the code needs to be styled better.

As far as the algorithm is concerned, the formation of a binary list and sum of those binary values in itself is an elegant solution to the nested inputs and rules given. Having said that, The Complexity of the files to be read and parsed, makes it imminent to retain the valuable data as given in the input and rules files respectively.

Conclusion

To conclude, the module developed will be able to read inputs, store the data given in them in the necessary format so that no information is lost. Then it will be able to read the rules, and store the data in a similar way. Next the comparison of the input file against the rule file can be implemented in various ways. And thus, this module become critical to the development of the Nuclear Plant Analyser Code which simulates the Stability of the nuclear power plants.

This will then be presented as a module which can be imported by the developers working to check for the stability conditions of the running PHWR in India and we will be able to predict, simulate and stop any accidents which could have happened if we weren't able to develop a simulation code. Although, RELAP5 is an option, that code is built for PWR type of reactors and so it is not directly applicable for the PHWR type of reactors which are the major chunk of the Nuclear Reactors in India at the point. And if we want to use RELAP5 for PHWR type of reactors, we need to customize and modify the code to suit the PHWR type reactor's requirements, which was a very long and gruesome process. So, this NuPAC is an attempt at developing an indigenous code worthy of replacing the established code RELAP5 to address the safety aspects of Nuclear Reactors.

References

1. AERB Safety Analysis and Research (2018)
   https://www.aerb.gov.in/images/PDF/Annual_report/ar2018/chapter7-annualreport2018.pdf

2. AERB Wikipedia
   https://en.wikipedia.org/wiki/Atomic_Energy_Regulatory_Board

3. AERB Safety Guide (SG/D-25)
   AERB Safety Guide for Computer based systems of Pressurised Heavy Water Reactors

4. AERB safety Guide(SG/D-10)
   AERB Safety Systems for Pressurised Heavy Water Reactors

5. Python Crash Course, 2nd Edition: A Hands-On, Project Based Introduction to Programming Book by Eric Matthes

6. Reactor Building Wikipedia
   https://en.wikipedia.org/wiki/Reactor_building#:~:text=A%20reactor%20building%20is%20a,to%20call%20them%20containment%20buildings.

7. Spent Fuel Pool Wikipedia
   https://en.wikipedia.org/wiki/Spent_fuel_pool#:~:text=Spent%20fuel%20pools%20(SFP)%20are,spent%20fuel%20from%20nuclear%20reactors.&text=Such%20pools%20are%20used%20for,radiation%20emanating%20from%20the%20rods.

8. United States Nuclear Regulatory Commission https://www.nrc.gov/reading-rm/basic-ref/glossary/auxiliary-building.html#:~:text=Auxiliary%20building,and%20emergency%20cooling%20water%20systems

9. Nuclear Power Plant Wikipedia https://en.wikipedia.org/wiki/Nuclear_power_plant

10. Relap5 https://en.wikipedia.org/wiki/RELAP5-3D