Formal representation of knowledge using Z in fast breeder test reactors

Bindu Sankar*, H. Seetha, K.K. Kuriakose, S.A.V. Satyamurty and P. Swaminathan

Indira Gandhi Centre for Atomic Research
Computer Division, Kalpakkam 603 102, India
E-mail: bindu@igcar.gov.in
E-mail: seethah@igcar.gov.in
E-mail: kuriakose@igcar.gov.in
E-mail: satya@igcar.gov.in
E-mail: swamy@igcar.gov.in
*Corresponding author

Abstract: In this paper, knowledge representation using Z, a formal notation, is adapted in mapping Fast Breeder Test Reactor (FBTR) requirements. The most frequent cause of faults in safety-critical real-time computer systems is traced to fuzziness in representing requirements knowledge. Pitfalls using a natural language as a medium for representing the system requirements knowledge were analysed and explored. To ensure the specified safety, it is necessary to represent the system requirements of safety-critical real-time computer systems using formal mathematical methods. This removes the fuzziness in communicating knowledge on reactor system requirements. This paper contains the formal mathematical model for requirement specifications of FBTR systems using Z notation. Finally, the advantages of using formal representation for representing system requirements knowledge of FBTR using Z notations to minimise the ambiguity in knowledge communication and thus improve the safety are summarised.

Keywords: formal methods; notation; knowledge representation; nuclear knowledge management.

Reference to this paper should be made as follows: Sankar, B., Seetha, H., Kuriakose, K.K., Satyamurty, S.A.V. and Swaminathan, P. (2009) 'Formal representation of knowledge using Z in fast breeder test reactors', *Int. J. Nuclear Knowledge Management*, Vol. 3, No. 3, pp.263–283.

Biographical notes: Bindu Sankar is working as a Scientific Officer in the Computer Division of the Indira Gandhi Centre for Atomic Research (IGCAR), India. She finished her Master's in Engineering at Regional Engineering College, Tiruchirapalli, India in 1998. She joined IGCAR in 2001. She has written Fast Breeder Test Reactor (FBTR) and Prototype Fast Breeder Reactor (PFBR) system requirements in Z. Apart from writing requirement specifications, she has taken an active part in the development of Graphical User Interface (GUI) software using Qt library in C++. She has also contributed to animation in graphics using Sherril-Lubinski-Graphical Modelling System (SL-GMS). Her fields of interest include knowledge management, SharePoint 2007, dynamic simulation and graphics.

H. Seetha graduated in Computer Science and Engineering from Bharathidasan University, Tiruchirapalli, India in 1997. She worked as a Lecturer at Government College, Chidabaram, India. She joined the Computer Division of the Indira Gandhi Centre for Atomic Research (IGCAR), India in 2001. She has been involved in 3D modelling of Prototype Fast Breeder Reactor (PFBR) components and the design, development, installation and commissioning of a PFBR full-scope operator training simulator. She has actively participated in the design and development of a knowledge management portal for IGCAR's Computer Division. Her fields of interest include knowledge management, logic modelling, and coding in C and C++. She has published papers in international journals.

K.K. Kuriakose graduated with honours in Electrical Engineering from the Regional Engineering College (now the National Institute of Technology), Calicut, India in 1977. After undergoing a one-year course in Nuclear Science and Engineering from Bhabha Atomic Research Centre (BARC) Training School, he joined the Indira Gandhi Centre for Atomic Research (IGCAR), India in 1979. He had successfully completed a Master of Engineering (first class) in Electrical Communication Engineering from the Indian Institute of Science, Bangalore (1986) and a Master of Business Administration from Indira Gandhi National Open University (2000). Currently he is the Head of the Knowledge Management Section and a doctoral-level research scholar in the area of knowledge management with Homi Bhabha National Institute.

S.A.V. Satyamurthy did his BTech at Jawaharlal Nehru Technological University, India in 1977, for which he was a university gold medalist. Later, he joined a one-year orientation course in Nuclear Science and Engineering (21st Batch) at BARC. He was awarded the Homi Bhabha prize for getting 1st place. He joined the Indira Gandhi Centre for Atomic Research (IGCAR) in 1978. He played a key role in the establishment of a mainframe computer system for IGCAR. He was also instrumental in establishing internet and e-mail facilities at IGCAR. He was responsible for the upgrading of the IGCAR Campus Network. He took keen interest in network security and commissioned many security servers, a high-performance computing facility, a intra-DAE VSAT network and a grid computing facility at IGCAR. He has more than 25 journal publications/conference proceedings and edited one international conference proceedings. At present, he is the Head of the Computer Division at IGCAR.

P. Swaminathan graduated with honours in Electronics and Communication Engineering from Regional Engineering College, Tiruchirapalli, India in 1971. He is a gold medallist of the University of Madras, India. After undergoing a one-year course in Nuclear Science and Engineering from BARC Training School, he joined Indira Gandhi Centre for Atomic Research (IGCAR) in 1972. He further underwent a one-year course in mainframe systems from International Honeywell Bull Training Institute, Paris, France. He is the main architect for the design, development, installation and commissioning of the fault-tolerant safety-critical real-time computer system for the fast breeder test reactor. As Outstanding Scientist and Director of the Electronics & Instrumentation Group at IGCAR, he is engaged in the development of safety instrumentation, a full-scope training simulator and a knowledge management system for a fast breeder reactor programme. He has over 40 publications in international journals/seminars.

1 Introduction

The use of real-time computers for safety-critical systems in strategic applications, such as nuclear, space and defence, is increasing rapidly. They can provide an inexpensive and flexible means of implementing very powerful and complex features. In the case of safety-critical systems, the implementation of system functions must be demonstrated to be safe and reliable with a certain degree of confidence. The software component of computer-based systems is not amenable to a quantitative assessment of reliability. Hence the assessment of software in computer-based systems has to be based on evidence that the software is correct (with respect to software requirements), safe and completely implements the requirements. Basically, the software component of real-time computers is prone to three types of errors: software requirement errors, logic errors and timing errors. When the software requirements knowledge is represented in a natural language, the interpretation by different stakeholders, such as the specifier, the designer, the developer, the tester, and the verification and validation team, may vary. Finally, this may result in the designing of a system with a functionality which is not as expected by the end user. Especially in safety-critical systems, this may affect the safety and functionality of the system. To avoid software requirement errors, the description of software specifications should be detailed, unambiguous, verifiable and complete. Knowledge representation using formal notation meets this goal. This paper describes the knowledge capture and representation of three safety-critical subsystems of a Fast Breeder Test Reactor (FBTR) using formal notation.

2 Brief overview of knowledge representation

A knowledge representation can be defined as a set of knowledge attributes which are necessary for efficiently finding relevant and applicable matches for the context of a knowledge need (Mahesh and Suresh, 2004). Knowledge representation can be a formal notation used to code the knowledge which will be stored in any knowledge-based system. Knowledge is usually stored in implicit form inside our minds or spread in the society as habits, experience, *etc*. To share this knowledge, it has to be made explicit. Producing different knowledge representations is a vital part of intelligence, since the ease of solving a problem is almost completely determined by the way the problem is conceptualised and represented (Shadbolt and Milton, 1999). Knowledge representation is a combination of:

- logic – This analyses inferences and is a source of knowledge for correct reasoning with formal structure.
- ontology – An ontology is a specification of conceptualisation. It is a formal description of concepts and the relationships that exist between entities.
- computation – Calculations or means by which implementation is carried out in computer programs.

Knowledge representation utilises theories and techniques from the above three fields. Knowledge representation can be carried out by two methods:

1 using a natural language

2 using a formal language such as Z.

Natural languages are very expressive; probably everything that can be expressed symbolically can be expressed in natural languages. The problems associated with natural languages are that they are very ambiguous and the syntax and semantics are not fully understood. Also, there is little uniformity in the structure of sentences. Such informal representations are not adequate as they are often inaccurate, inconsistent and unclear. Natural language representations are also very lengthy, making them difficult to check for completeness. These problems can be overcome by the use of formal languages (Holloway, 1997).

The readability of formal requirement specifications is usually poor (Zimmerman *et al.*, 2002). However, by selecting a formal specification language such as Z, which is more intuitive and user friendly with logic-based graphical notations, this could be overcome. Apart from the creation and analysis of requirements knowledge, formal methods can also be applied in a cost-effective way to answer specific questions about the domain (Bhardwaj, 2002).

3 Understanding requirements knowledge

Some of the difficulties associated with representing the requirements knowledge in a natural language which might lead to errors/bugs are contradiction, ambiguity, vagueness, incompleteness and a mixed level of abstraction. In subsequent discussions in this paper, the terms 'requirements knowledge', 'requirement specifications' and 'specifications' are used interchangeably.

3.1 Contradiction

Contradictory statements in a lengthy requirements document are often separated by many pages and remain undetected – until coding.

Here is an example: "The system will always operate on values between levels 0.0 and 4.0…." This statement is later contradicted by another: "Module X will be required to read in data from an external source over the range of integer values 0 to 10."

According to the first quoted statement, only floating point values need to be considered; but later this is contradicted by considering integer values with a different range.

3.2 Ambiguity

Ambiguity is common in written requirements. For example, consider this requirement: "The system will read in 120 initial data values and compute the mean and mode. They will be stored for later analysis."

What exactly must be stored? Is it the raw data or the computed mean and mode values that need to be stored?

3.3 Vagueness

Sometimes lengthy requirements might include vague parts. For example, consider the following requirements: "The storage size should be as small as possible."

Obviously this is useless in a requirement representation, but the danger is that coding might begin before this vagueness is clarified.

3.4 Incompleteness

This is the hardest flaw to deal with when working with requirements. For example: "The system will accept time entries ranging from 03:00 on 1/2/02."

And suppose if someone enters 02/02/02? What happens then? It is not clear from the requirements.

3.5 Mixed levels of abstraction

It can be very hard to see the overall functional architecture when the levels of description are intermixed, like this: "The system will make it easy to access the values in all consumer accounts. This will involve extraction of the 5-digit real values by clicking on the tab to the left of the rightmost frame on the entry page…." This is so obscure that understanding will be difficult.

4 System failure prevention

The three main causes of failure in software systems are physical causes, software errors and human/computer interference. Software errors can be minimised by formal techniques.

Even in a safety-critical application, software may fail frequently, yet still not lead to unsafe behaviour. On the other hand, highly reliable software can be unsafe because, in the rare event of a failure, the effect is dangerous.

Ideally, software designers and manufacturers should demonstrate that software specification forbids actions that lead to catastrophic failures of the system. The software should also protect itself from failures of the other parts of the system. The only way to be certain that a system is specified as safe is to rely on formal techniques. This necessitates the creation of more friendly interfaces for project managers and designers who are not professional mathematicians.

Formal specification of software involves three main concepts:

1. the data invariant – This is a set of rules that governs the range and nature of the data in the system throughout its operation.
2. the state – These are the actual values stored in the system.
3. operations – Each operation has a precondition (*i.e.*, a statement of conditions for the particular action to be allowed to proceed) and a postcondition (*i.e.*, a statement of what happens when the operation is complete).

In the future, we may need to use combinations of formal verification, more advanced testing and other methods to achieve the genuinely safe operation of software-controlled systems.

5 Brief overview on formal methods

Formal methods are based upon elementary mathematics, which is used to produce precise, unambiguous documentation, in which information is structured and presented at an appropriate level of abstraction. Formal methods apply logic and simple mathematics to programming. It particularly concerns discrete mathematics. Formal methods can determine the meaning of a formula, such as a specification or design, without executing it. Usage of formal methods will enable the prediction of what a program will do without running any code. This means that we can discover errors without having to run any tests (Jackey, 1997).

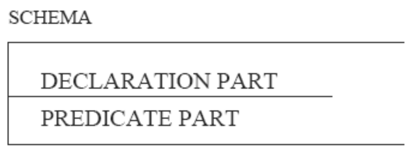
Formal methods can be used for modelling and predicting the system behaviour within a mathematical formalism. In many cases automated tools support formal methods. Such tools provide increased repeatability of analysis, increased soundness and assurance.

6 Z – a formal notation

Formal specification is nothing but expressing system requirements in a well-defined, logically sound mathematical notation.

Z tools are available for this purpose. Z is a model-based notation. In Z, the system model is represented using its state, which is a collection of state variables and their values and some operations that can change the state. A model that is characterised by the operations it describes is called an Abstract Data Type (ADT) (Harrison, 1992). Z has a powerful structuring mechanism. In combination with natural language, it can be used to produce formal specifications. The Z notation is based upon set theory and mathematical logic. The theory includes standard set operators, set comprehensions, cartesian products and power sets. The mathematical logic is a first-order predicate calculus. Mathematical objects and their properties can be collected together in schemas.

A specification is a collection of schemas. Schemas are macros that we can use to abbreviate blocks of mathematical text. A schema introduces some entities and invariant properties. The schema declaration part defines each entity's name and type (syntax). The predicate expresses constraints that determine the values which the variables actually take. The schema language can be used to describe the state of a system, and the ways in which that state may change. It can also be used to describe system properties and to reason about possible refinements of a design (Jackey, 1997). A schema is pictorially represented as shown below.

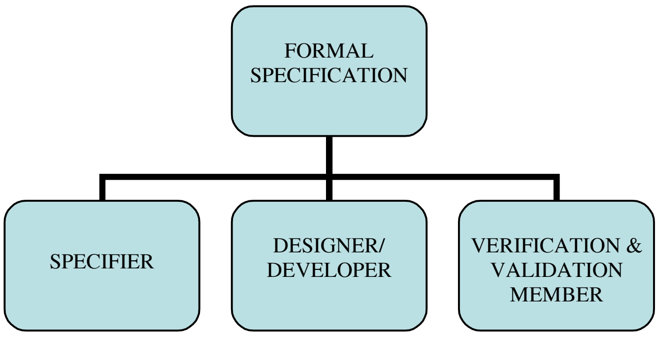


The schema is the characteristic construct of the Z notation.

7 Modelling of requirement specifications for safety-critical systems

If the failure of a computer system may lead to catastrophic consequences, such as loss of human life, damage to the environment or damage to the system itself, such a system is known as 'safety critical'. Nuclear reactors being safety critical, requirements' gatherings must be done with utmost care. Hence, the formal specification language Z was chosen against a natural language (English) for the modelling of requirement specification. The users of formal specifications are given in Figure 1.

Figure 1 User diagram (see online version for colours)



The Z specification document of FBTR was written using Logica's formal methods tool – Z specific formaliser version 7.3.7e – and the document was checked using the provisions given by the formaliser. Formal methods require effort, expertise and significant knowledge in order to be successfully applied to safety-critical systems.

8 Overview of Fast Breeder Test Reactor

The FBTR is a 40 MWt/13 MWe, liquid sodium-cooled plutonium-uranium carbide-fuelled, loop-type fast reactor. The purpose of constructing it is to use it to gain experience in the design, construction and operation of fast breeder reactors. The basic conceptual design of FBTR systems, *viz.* block pile, primary and reactor protection instrumentation, are similar to the French reactor Rapsodie, whereas the secondary, steam-water system and turbo generator are designed in-house (Ramanathan *et al.*, 2004). The schematic diagram of the FBTR is shown in Figure 2.

At the FBTR, for monitoring and regulating the reactor power, neutronic instrumentation is provided. For detecting the fuel pin clad failure, one system based on Delayed Neutron Detection (DND) and another based on cover gas analysis are incorporated. Thermocouples (TCs) are provided to measure the fuel subassembly sodium temperature. Ward-Leonard driven sodium pumps accurately regulate the sodium flow through the core. Real-time fault tolerant computer systems monitor the important safety parameters of the plant and initiate safety actions when required. An online computer system with a hot standby is provided to monitor the plant parameters and initiate safety actions, besides providing vital plant information to the operator. There are around 700 analogue and 300 digital input signals connected in parallel to both computers. The computers scan the signals periodically every 1 s and 20 s, depending on the importance of the parameters, process them through different supervision software tasks, such as supervision of the reactor core against blockage of flow, power excursion and clad hotspot, and perform the safety-related functions of the plant. Online diagnostics check the health of the hardware and proper execution of the application software. If satisfactory, then a watchdog pulse is generated. The watchdog pulses from each computer are processed by switchover logic. The digital outputs from a healthy computer is routed to the plant in the ORING logic (according to ORING logic, based on the health pulse received through switchover logic, either one of the digital outputs received from computer 1 or 2 is routed to the plant) (Figure 3) (Swaminathan, 2004).

Figure 2 FBTR schematic (see online version for colours)

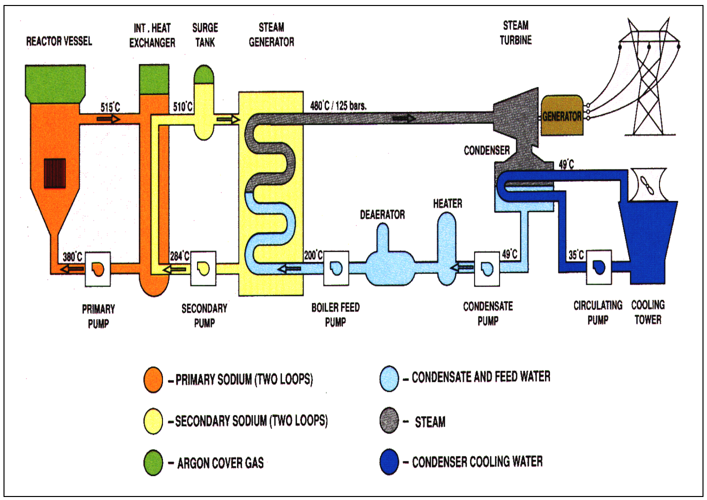
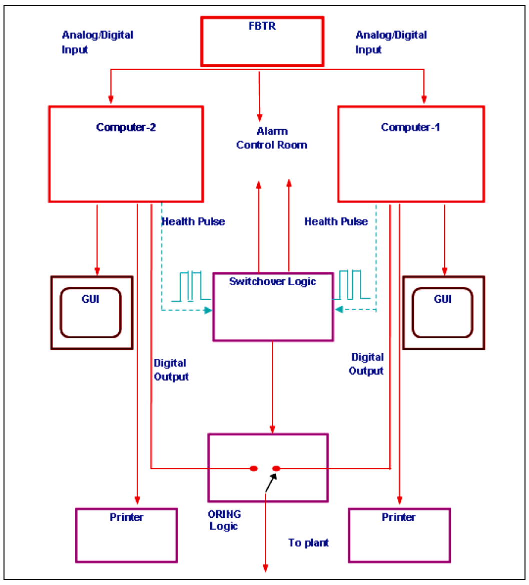


Figure 3 Computer system configuration (see online version for colours)



9 Modelling of system specifications of FBTR

9.1 Case Study 1: core temperature monitoring system

Normally, the reactor core is provided with in-core flux sensors, TCs, flow meters, acoustic sensors, *etc*. The signal from the in-core flux sensors need to be processed for flux mapping, reactor power, period and reactivity. The outlet temperature from each fuel subassembly needs to be monitored to detect flow blockage, clad hotspot and undesirable power excursion. Online statistical analysis of the acoustic sensor signal is required for the detection of a coolant boiling in the core. The availability of powerful computer systems enable online processing of in-core sensor signals.

The Core Temperature Monitoring (CTM) system processes the fuel core outlet temperature signals and reactor inlet signals. It validates all the signals (whether faulty or not) in what is called Pre Treatment Routine (PTR). It has to calculate the core outlet temperature (θm) in the process called Mean Value Sub Routine (MVSR). It also calculates the mean temperature rise across the reactor core (Δθm) in what is called Mean Gradient Sub Routine (MGSR) and detects partial plugging blockage by monitoring the temperature rise across individual subassemblies (δϑi), in what is called Plugging Detection Routine (PDR). It checks against alarm LOR (Lowering of Rods – slow shutdown) and SCRAM (Safety Control Rod Axe Man – fast shutdown) thresholds, generating Alarm, LOR and SCRAM whenever computed values cross the thresholds. In addition, this system has to send the acquired and computed data to the local display and backbone network for the operator's aid in the control room. Since online computation is involved in the process, it was decided to use a computer-based signal processing system for CTM.

9.1.1 Modelling of signal validation (Pre Treatment Routine)

'Value', 'Signal' and 'report' are declared as global, that is, they can be used anywhere in the specification document. 'Value' and 'Signal' are nonnegative integers ($\mathbb{N}$ is a predefined data type of Z):

Value == $\mathbb{N}$

Signal == $\mathbb{N}$

report::=signalvalid|faultysignal|resetalarm|resetlor|resetscram|
orderalarm|orderlor|orderscram|onrequest.

In Z, 'report' is a free type definition (similar to enumerated type). To define a free type, give its name and then, after the definition symbol "::=", list all of its elements. Here, the free type 'report' consists of 'signalvalid', 'faultysignal', 'resetalarm', 'resetlor', 'resetscram', 'orderalarm', 'orderlor', 'orderscram' and 'onrequest' variables. The order here is not significant (no sequence is implied).

'Defscandb' schema implies that at any time the number of valid and invalid signals are less than or equal to the number of raw signals and there will not be any signal which belongs to both valid and invalid signals. This schema is shown below:

```
┌─ Defscandb ─────────────────────
│ Rawsignal :seq Signal
│ Validsignal: seq Signal
│ Invalidsignal : seq Signal
├──────────────────────
│  # Validsignal + # Invalidsignal <=  # Rawsignal
│  disjoint < Validsignal, Invalidsignal>
└──────────────────────────
```

The symbol ‘seq’ is used to define the sequence of any type and the # symbol implies the number of elements in a set.

Every system has a special state in which it starts up. In Z, this state is described by a schema named ‘intscandb’(given below). Initially, ‘rawsignal’, ‘validsignal’ and ‘invalid signal’ must be empty.

```
┌─ intscandb ─────────────────────
│ ΔDefscandb
├──────────────────
│ Rawsignal =  < >
│ Validsignal = < >
│ Invalidsignal = < >
└──────────────────────────
```

The ‘<>’ in the predicate is the empty sequence and ‘Δ’ indicates change.

Some more global variables defined in the Z specification document of FBTR systems were:

tempoutletsignal == $\mathbb{N}$

tempinletsignal == $\mathbb{N}$

subassemblies == $\mathbb{N}$

tempvalue== $\mathbb{N}$

limitvalue== $\mathbb{N}$

ratiovalue == $\mathbb{N}$

errorvalue == $\mathbb{N}$

powervalue == $\mathbb{N}$.

The axiomatic definition is a Z paragraph that is set off from the surrounding informal prose by an open box. Its general form is (Woodcock and Davis, 1996);

```
│  declarations
├──────────────────────
│  predicates
│
```

In the CTM system:

```
│ Median : seq ℕ  → ℕ
│
```

The axiomatic definition of median is implied above with no predicate part, with ‘$\rightarrow$’ indicating total function.

```
┌─Defcore ──────────────────────
│ inlettemp : seq tempinletsignal
│ Rawsuboutlet : subassemblies
│           ⇸       seq tempoutletsignal
│ alarmltMGSR_core, alarmltMVSR_core,
│ alarmltPDS_core : limitvalue
│ inletavg : tempvalue
│ ratio : seq ratiovalue
├───────────────────────────────
│ inletavg = median inlettemp
└───────────────────────────────
```

We have introduced a sequence for inlet temperatures and defined a partial function from subassemblies to outlet temperatures. The median of inlet temperatures was calculated in the above schema.

In the schema ‘Signalvalidation’, Ξ Defcore is an operation on ‘Defcore’ that does not change the value of any state variable. In this schema the outlet temperature signal of fuel subassemblies is compared with the median of the reactor inlet temperature. The reactor outlet temperature signal is declared valid if it satisfies the following equation:

outlet temperature > median of inlet temperatures – x°C

where x°C is taken into account due to the inaccuracy in inlet and outlet temperature signals. At low power, the inlet and outlet temperatures are nearly equal.

If the outlet temperature signal does not satisfy the equation, then the signal is declared as faulty. In the schema below, contforvalidity is assigned as x°C.

```
┌─ Signalvalidation ─────────────────────
│ ΞDefcore
│ outletsignal : seq tempoutletsignal
│ validsuboutlet : subassemblies
│           ⇸       seq tempoutletsignal
│ contforvalidity : ℕ
├────────────────────────────────────────
│ ∀i : 1..# subassemblies •
│   (validsuboutlet i A ⊆ Rawsuboutlet i
│     ∧ (outletsignal = validsuboutlet i
│         ∧ ( ∀j : 1 .. # outletsignal •
│                 outletsignal j – inletavg
│                          + contforvalidity > 0)))
└────────────────────────────────────────
```

In the predicate, if any of the TCs at the outlet of the subassemblies reads less than the inlet temperature, that TC is considered as faulty and that value should not be taken into account for further calculation purposes. The symbol $\forall$, for all, is the universal quantifier. Here it is used to introduce the bound variable ‘i’ into the predicate. This bound variable does not model some particular component of the system we are trying to describe; it is merely a place holder that stands for 1 to the total number of subassemblies. The general form of a universally quantified predicate is:

$\forall$ declaration • predicate.

The character • is just a delimiter. The variables declared in declaration part of a schema are called bound variables. This quantified predicate means that it is true for all values of the bound variables that are admitted by the declaration. The scope of the bound variables is limited to the predicate; outside this scope, the bound variables are undefined.

$\subseteq$ implies set inclusion, $\wedge$ implies Logical conjunction (and).

If all the thermocouples in any one of the fuel subassemblies are declared faulty by signal validation, then a permanent scram order is given. The equivalent Z statement is shown below.

```
┌─ SCRAMORDERING ──────────────────
│ Signalvalidation
│ rep! : report
├──────────────────────────
│ ∃ i : 1 .. # (dom validsuboutlet) •
│  (# (validsuboutlet i) = 0
│     ∧ rep! = permanent_scramorder)
└──────────────────────────────
```

Symbol $\exists$ implies existential quantification and symbol ! implies output. 'Dom' means domain.

9.1.2 Modelling of MVSR (θm), MGSR (Δθm) and PDR (δθi)

A real-time computer scans each subassembly outlet temperature at the desired frequency and calculates the core outlet temperature (θm), the mean temperature rise in the core (Δθ m) and the deviation in individual subassembly outlet temperature over the expected value (δθi).

Mean value sub routine

The average value (θm) of validated outlet temperature signals for fuel subassemblies shall be computed every second. This shall be compared against the Alarm, LOR and SCRAM thresholds. The equivalent Z statement is shown below.

```
┌─NormalMVSR_alarm ──────────────────
│
│   Signalvalidation
│   outletavg : tempvalue
│
├──────────────────────────
│   outletavg = setaverage validsuboutlet
│   outletavg   ≤ alarmltMVSR_core
│
└──────────────────────────────
```

where setaverage is given by the axiomatic definition:

```
| setaverage : (ℕ ⇸ seq ℕ) ⇸ ℕ
```

```
NormalMVSR_LOR ≙
NormalMVSR_alarm[alarmltMVSR_core /
lorltMVSR_core ]
```

```
NormalMVSR_SCRAM ≙
NormalMVSR_alarm[alarmltMVSR_core
/scramltMVSR_core]
```

Here the schemas NormalMVSR_LOR and NormalMVSR_SCRAM are horizontal schemas which are similar to NormalMVSR_alarm, except for the change in the alarm limit to 'lor' and 'scram' limits respectively. Horizontal schemas are usually in a single-line horizontal format in which the schema name appears to the left of the definition symbol '≙'.

Separate schemas were written indicating whether θm exceeds the alarm, LOR or SCRAM limits. If the values were found to exceed them, then alarm, LOR or SCRAM shall be energised. Also, a corresponding error message shall be displayed and printed.

If θm falls below the alarm, SCRAM or LOR limits, fault clear messages shall be displayed and printed out.

Mean gradient sub routine

The average temperature rise:

$$\Delta\theta\ m = \theta m - \text{inlet temperature}$$

shall be calculated every second and compared against the alarm, LOR and SCRAM limits. The thresholds (alarm, LOR and SCRAM limits) are variables which are related to the reactor power campaign. If Δθ m exceeds the alarm, LOR or SCRAM thresholds, corresponding action shall be initiated. Relevant fault messages shall be printed and displayed. If Δθ m decreases below the threshold level of alarm or LOR or SCRAM, corresponding reset action shall be initiated. Relevant fault messages shall be printed and displayed. This can be shown by the schema below named NormalMGSR_alarm.

```
┌── NormalMGSR_alarm ─────────────────
│ signalvalidation
│ diffinoutavg : tempvalue
│ outletavg : tempvalue
├──────────────────────────────
│ outletavg = setaverage validsuboutlet
│ diffinoutavg = outletavg − inletavg
│ diffinoutavg ≤ alarmlt MGSR_core
│
└─────────────────────────────────
```

NormalMGSR_LOR $\widehat{=}$
NormalMGSR_alarm[alarmltMGSR_core /
lorltMGSR_core]

NormalMGSR_scram $\widehat{=}$
NormalMGSR_alarm[alarmlt_MGSR_core /
scramltMGSR_core]

Here the schemas NormalMGSR_LOR and NormalMGSR_SCRAM are horizontal schemas which are similar to NormalMGSR_alarm except for the change in the alarm limit to 'lor' and 'scram' limits respectively.

Separate schemas were written indicating whether $\Delta\theta$ m exceeds alarm, LOR or SCRAM limits. If the values were found to exceed them, then alarm, LOR or SCRAM shall be energised. Also, a corresponding error message shall be displayed and printed.

If $\Delta\theta$ m falls below the alarm, SCRAM or LOR limits, fault clear messages shall be displayed and printed out.

Plugging detection supervision

The plugging detection software shall be operational only after the reactor power exceeds 2 MWt in the FBTR. The expected temperature rise in each subassembly shall be calculated with the following formula:

$$\text{Expected temperature rise} = \text{ai} \times \text{Average temperature rise.}$$

The term 'ai' is the ratio of the temperature rise of an individual subassembly to the mean temperature rise in the core, which is determined for a fresh core and updated periodically. To start with, the computed ai shall be available as 'software data'. However, there shall be online provision to calculate ai using the following equation:

$$\text{ai} = \frac{\text{Temperature rise in the i}^{\text{th}}\text{ fuel subassembly}}{\text{Average temperature rise in the core}}.$$

There shall be two sets of ai. The first is named the computational set and the other, the operational set. Plugging detection software shall use ai values only of the operational set.

To start with, both sets shall contain the same values. But at any time, the operator can order online computation of ai. The computer shall calculate ai and store the values only in the computational ai set. After transferring the values to the operational set, the computer system shall print out the contents of the updated operational set. If any TC is declared faulty by the signal validation (PTR) software, then the signal shall be assumed to have crossed the SCRAM limit. In the schema, if the error between the actual temperature rise and the expected temperature rise exceeds 'k', alarm shall be energised. If the error exceeds 'm' in any one TC, then alarm is given and if both TC values exceed value 'm', SCRAM is given.

Here, k = 5, m = 10 and power p = 2.0 Mwt.

```
──────PluggingDetectionSoftware ──────────
Signalvalidation
expectedtemprise, Actualtemprise : seq tempvalue
error :seq errorvalue
power: powervalue
rep! :report
request ?: report
k,m,p : ℕ
──────────────────────────────
k< m
power ≥ p
    ∀ i :1 .. # (dom validsuboutlet ) •
    (Actualtemprise i
      = average (validsuboutlet i ) – inletavg
     ∧ expectedtemprise i
        = ratio i
       *   (setaverage validsuboutlet – inletavg)
       ∧ error i = expectedtemprise i – Actualtemprise i
       ∧ (request? = onrequest
       ∧ ratio' i = Actualtemprise i div expectedtemprise i
              ∨ ratio' i =ratio i))
```

In the PluggingDetectionSoftware schema, the predicate request ? implies input and the symbol * means multiplication. Alarm and SCRAM schemas were also written.

9.2 *Case Study 2: discordance supervision*

There are three fission gas chambers surrounding the reactor core at 120°. The neutron emission is continuously monitored in these three chambers through discordance supervision. The function of this supervision is to find out the discordance among the triplet channels of neutronic parameters. Any discordance among the three channels will be notified through alarm messages. Figure 4 shows a schematic representation of the reactor core and the placement of the fission chambers at locations A, B and C.

1 The discordance threshold is calculated as:

Discordance threshold = median value of the three channels × threshold percentage of parameter

2 The difference among the three channels is:

D1 = Channel A – Channel B

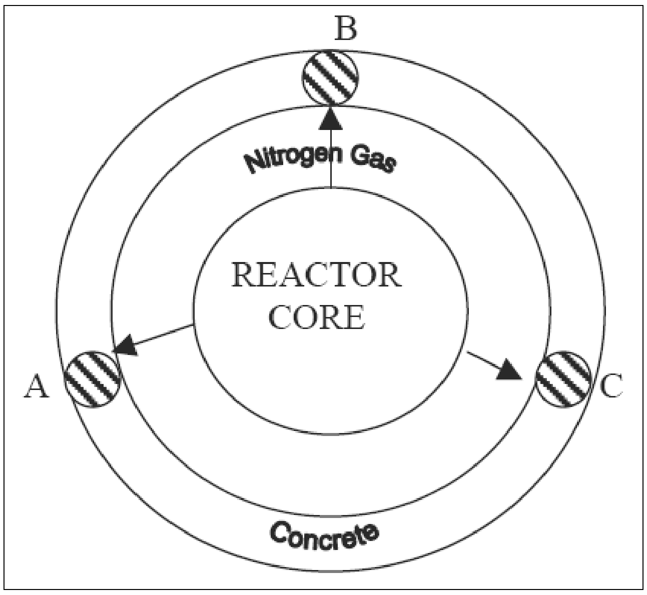
D2 = Channel B – Channel C

D3 = Channel C – Channel A

3 Compare D1, D2 and D3 with the discordance threshold and generate alarm messages as per the algorithm below:

- If no difference crosses the discordance threshold – NO discordance
- If two differences cross the discordance threshold – ONE discordance
- If all the three differences cross the discordance threshold – TWO discordance.

Figure 4 Reactor core, cross-sectional view



The modelling of the above English specification in Z is shown below:

CHANNEL == ℕ

ALARMLT == ℕ

value == ℕ

Let us define the channels in a schema called ‘DefChannels’. The input for discordance supervision is ‘chA’, ‘chB’ and ‘chC’. ‘S’ belongs to ‘Alarmlt’.

```
_____ DefChannels _________
|
| chA, chB, chC: CHANNEL
| s: ALARMLT
|___________________________
```

The axiomatic definition of the absolute difference between two channels is indicated below:

```
| absdiff : CHANNEL x CHANNEL ↔ value
```

We can define product types using the cross product symbol ×. In a formal description of a software system, we may wish to associate two or more objects of the same kind or different kinds, respecting order and multiplicity. To support this structuring mechanism, the Z notation includes Cartesian products. These are sets of tuples: ordered lists of elements, one list drawn from each of the component sets.

If ‘a’ and ‘b’ are two sets, then the Cartesian product ‘a × b’ consists of all tuples of the form (x, y) , where ‘x’ is an element of ‘a’ and ‘y’ is an element of ‘b’.

The symbol '$\leftrightarrow$' denotes binary relations that can be many to many. If 'X' and 'Y' are sets, then 'X$\leftrightarrow$Y' denotes the set of all relations between 'X' and 'Y'. The relation symbol may be defined by generic abbreviation:

$$X \leftrightarrow Y == \mathbb{P}\ (X \times Y)$$

Any element of 'X$\leftrightarrow$Y' is a set of ordered pairs in which the first element is drawn from 'X' and the second from 'Y'; that is, a subset of the Cartesian product set 'X$\leftrightarrow$Y'.

The modelling of different discordance scenarios like NO discordance, ONE discordance and TWO discordance are given below.

The schema for NO discordance is:

```
┌──── NODiscordance ──────
│ Ξ DefChannels
├────────────────────────
│ absdiff (chA, chB) ≤ s ∧ absdiff (chB, chC) ≤
│ s ∧ absdiff (chC, chA) ≤ s
└────────────────────────
```

The schema for ONE discordance for channels A, B and C are:

```
┌──── ONEDiscordanceA ──────
│ Ξ DefChannels
├────────────────────────
│ absdiff (chA, chB) > s ∧ absdiff (chB, chC) < s
│ ∧ absdiff (chC, chA) > s
└────────────────────────
```

```
┌──── ONEDiscordanceB ──────
│ Ξ DefChannels
├────────────────────────
│ absdiff (chA, chB) > s ∧ absdiff (chB, chC) > s
│ ∧ absdiff (chC, chA) < s
└────────────────────────
```

```
┌──── ONEDiscordanceC ──────
│ Ξ DefChannels
├────────────────────────
│ absdiff (chA, chB) < s ∧ absdiff (chB, chC) > s
│ ∧ absdiff (chC, chA) > s
└────────────────────────
```

ONEDiscordance ≙ ONEDiscordanceA V
ONEDiscordanceB V ONEDiscordanceC

The horizontal schema 'ONEDiscordance' will take place if ONEDiscordanceA or ONEDiscordanceB or ONEDiscordanceC will occur.

The schema for TWO discordance is:

TWODiscordance

Ξ DefChannels

---

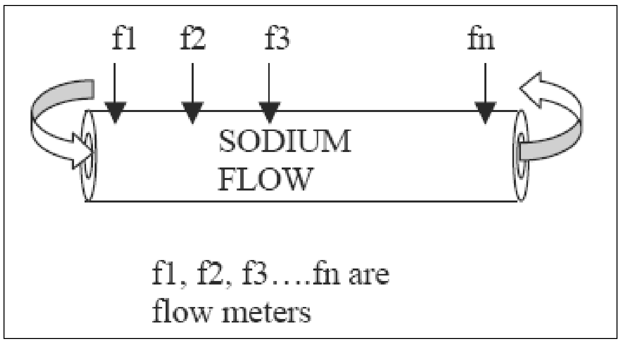
absdiff (chA, chB) $>$ s $\wedge$ absdiff (chB, chC) $>$ s
$\wedge$ absdiff (chC, chA) $>$ s

The modelling of schemas for indicating alarm, clearing alarm, *etc*., based on the above schemas was also done.

9.3 Case Study 3: general supervision

Sodium is pumped through the reactor core by the primary sodium pumps. There is a secondary sodium pump which pumps sodium through a secondary loop. The measurement of sodium flow is done using sodium flow meters like magnet flow meters and/or eddy current flow meters. Consider a case in which sodium flows through a pipe as shown in Figure 5. Flow meters f1, f2…*etc*. measure the sodium flow. Flow supervision falls under general supervision. The flow is checked against the threshold generating alarm, if required. In general supervision, process signals of similar nature are grouped together and compared with thresholds (upper thresholds and lower thresholds).

Figure 5 Sodium flow inside a pipe



If the sodium flow is less than lower thresholds, it indicates that the sodium pump might be tripped out (not working) or faulty. If the sodium flow crosses higher thresholds, it implies that the pump is working at a higher speed. In both cases (sodium flow crossing the lower or higher thresholds), alarms are generated.

The above specifications are modelled in Z as:

group == seq $\mathbb{N}$

signal == $\mathbb{N}$

UT == $\mathbb{N}$

LT == $\mathbb{N}$

Report :: = ResetAlarm

on == 1

off == 0

boolean : : = on | off

In ‘DefGensupervisionSignal’ each group of the process signal has a common upper threshold and lower threshold. So here we can consider a one-to-one relationship (indicated by the symbol ⤖) between ‘group’ and ‘upper Threshold’ (UT) and ‘group’ and ‘Lower Threshold’ (LT). Here ‘c’ is a constant.

```
_____________ DefGensupervisonSignal _____________
|
|  setofgroup : seq group
|  ProcessSignal : group  seq signal
|  UpperLimit : group ⤖  UT
|  LowerLimit : group ⤖  LT
|_________________________________________________
```

Now we can define two schemas called Normal_GS and NotNormal_GS.

```
____________ Normal_GS ____________
|
| Ξ DefGensupervisionSignal
| processSig : seq signal
| Upper : UT
| Lower : LT
|__________________________________
| (∀ i : 1.. # setofgroup ● processSig =
| ProcessSignal (setofgroup i) ∧
| Upper = UpperLimit (setofgroup i) ∧
| Lower = LowLimit (setofgroup i)) ∧
| (∀ j : 1..#processSig ● processSig j ≤ Upper ∧
| processSig j ≥ Lower)
|__________________________________
```

```
____________ NotNormal_GS ____________
| Ξ DefGensupervisionSignal
| processSig : seq signal
| Upper : UT
| Lower : LT
|__________________________________
| (∃ i : 1.. # setofgroup ● processSig =
| ProcessSignal (setofgroup i) ∧
| Upper = UpperLimit (setofgroup i) ∧
| Lower = LowLimit (setofgroup i)) ∧
| (∃ j : 1..#processSig ● processSig j > Upper ∨
| processSig j < Lower)
|
|__________________________________
```

Based on the above schemas, we can extend the Z modelling for ‘alarms’ and ‘alarm clear’.

10 Conclusion

The advantage of using formal methods was demonstrated in the knowledge representation of FBTR system specifications. Knowledge representation of specifications was done formally using Z, with clear and elegant semantics. Formal methods can help in creating software that could be understood before the actual execution. The usage of formal methods requires the programmers to use mathematical symbols to represent the program's logic before coding. Like a mathematical theorem, knowledge representation using Z can be checked to verify that specifications form logically correct statements. Once the programmer is sure that there is no logical flaw in the knowledge representation of specifications of the reactor, it is relatively simple to convert Z symbols into a programming code. Hence, this is a way to eliminate bugs even before writing the actual code. Also, this eliminates the necessity of trial and error to validate and improve the systems. The main advantages of using formal specifications are as follows:

- higher quality software
- verifiability
- insight and understanding
- minimised maintenance and cost
- formal analysis
- guidance for testing
- reduced liability and risks
- standard satisfaction.

Case studies of three FBTR systems were taken to demonstrate the knowledge representation using Z code. Similarly, formal representation of the system requirement knowledge of the remaining systems of FBTR also successfully used Z notation.

References

Bhardwaj, R. (2002) 'Formal analysis of domain models', *Proc. International Workshop on Requirements for High Assurance Systems (RHAS'02)*, Essen, Germany, pp.1–6.

Harrison, M.D. (1992) 'Engineering human error tolerant software', in J.E. Nicholls (Ed.) *Z user Workshop, York 1991*, Workshops in Computing, Springer-Verlag, pp.191–204.

Holloway, M. (1997) 'Why engineers should consider formal methods', *16th Digital Avionics Systems Conference*, 27–30 October, pp.5–7.

Jackey, J. (1997) *The Way of Z*, Cambridge University Press, pp.3–13, 49–51.

Mahesh, K. and Suresh, J.K. (2004) 'What is the K in KM Technology?', *Electronic Journal of Knowledge Management*, Vol. 2, No. 2, p.16.

Ramanathan, V., Pillai, C.P., Rajendran, B., Ramalingam, P.V. and Bhoje, S.B. (2004) 'Commissioning and operating experience on I&C systems of fast breeder test reactor', *IAEA/RCA Technical Meeting on Research Reactor Instrumentation and Control*, Korea Atomic Energy Research Institute (KAERI), Daejeon, Korea, 3–7 May, pp.1–3.

Shadbolt, N. and Milton, N. (1999) 'From knowledge engineering to knowledge management', *British Journal of Management*, Vol. 10, pp.309–322.

Swaminathan, P. (2004) 'Computer based on-line monitoring system for Fast Breeder Test Reactor, India', *Technical Meeting on Increasing Instrument Calibration Interval through Online Calibration Technology*, Halden, Norway, 27–29 September, p.3.

Woodcock, J. and Davis, J. (1996) *Using Z: Specification, Refinement, and Proof*, Prentice Hall.

Zimmerman, M.K., Lundqvist, K. and Leveson, N. (2002) 'Investigating the readability of state based formal requirements specification languages', *Proc. International Conference on Software Engineering (ICSE 2002)*, Orlando, Florida, USA, May, pp.10–11.