APIC'95, El Paso, Extended Abstracts, A Supplement to the international journal of Reliable Computing

Development of Quality Interval Algorithms, Software and Systems

Nik M. Glazunov

***Abstract*— In this paper, we describe a general formalism that is based on our experience in designing interval algorithms, software systems, and controllers for dynamical plants. This formalism can be applied to the following problems:**

(i) requirements and specifications of development units;

(ii) interval approach;

(iii) interval computations and computer algebra;

(iv) verification and validation of software;

(vi) verified numerical results.

I. Thre Types of Desiggn Problems

Design of a complicated system usually includes the design of three types of objects:

- the design of *hardware* that will make this system perform its task (i.e., the hardware that will *control* this system);
- the design of the *algorithms* that will process the data obtained from sensors, and generate the commends for the actuators;
- the design of the *software* that implement these algorithms.

In these three problems, we design different objects (such objects will be called *units*):

- In the first case, we design a *system* itself.
- In the second case, we design a *algorithm*.
- In the third case, we design *software*.

II. These Three Types of Problems Have Something in Common

In spite of the different character of the designed units, the designed units, the design process follows the same pattern:

T We start with the *requirements* T that capture the purpose of this unit.

C Then, we design a *specification* C that describes how exactly the requirements prescribed by T will be met.

S Finally, we design a unit S that satisfies these specifications C.

It is therefore desirable to use this similarily, and to find a general approach to all three types of design problems.

III. A General Approach to All Three Types of Design Problems

In this paper, we describe a general approach to these three types of design problems.

This general approach will be based on *category theory*:

- As *basic* categories, we will consider the categories (Ob, Mor), where:
  - objects (i.e., elements of the set Ob) are *state spaces*, and
  - morphisms (i.e., elements of Mor) are mappings betwen state spaces, that correspond to the *evolution* of the state (i.e., to the changing of state in time).
- Since the state is not exactly known, we will have to introduce the second set of categories, in which:
  - objects are classes whose elements are *sets* of possible states (e.g., if a state is described by a number, possible *intervals*),
  - morphisms describe how these sets change.

In the above three problems, the objects of the corresponding categories will have similar (but somewhat different) interpretations:

- In *system design*, a state will mean a normal state of the system.
- In a *mathematical algorithm*, a state will mean the (ideal) values of all the variables that are used in this algorithm.

The author is with the Glushkov Institute of Cybernetics, Ukrainian Academy of Sciences, 40, Glushkov prospect, 252650, Kiev GPS-650, Ukraine, Tel. 044-266-62-97, fax 044-266-74-18, e-mail glanm@sysres.ms.kiev.ua

- In a *program*, a state will mean the actual values of these variables that are stored in the computer.

Comment. The difference between the second and the third cases is easy to illustrate: if an algorithm consists of adding two numbers, say, 0.3 and 0.4, then according to the algorithm, the result will be exactly 0.7. However, in a typical computer that uses binary representation of real numbers, the result will be slightly different from 0.7.

IV. Results

We have applied the resulting two-layer algebraic (category-theoretical) approach to the design of quality algorithms, software, and systems. In particular, we have applied it to the solution of the following three problems:

- the design of an interval algorithm for verification of an analytical dependency;
- the design of applied interval software;
- interval analysis of dynamical control system with varying parameters.

Let us describe this approach in some detail.

V. Requirements and Specification

In this paper, we consider *requirements* that are expressed in a mathematical notation (such as algebraic category theory, control theory, numerical analysis, etc). These requirements capture the purpose of the unit. We consider three types of requirements: system requirements, program requirements, and functional requirements:

- The expected functions of the unit constitute the *system requirements*.
- If the unit has sensors and actuators, then the part of the system requirements that is related to the states of sensors and actuators forms the *program requirements*.
- *Functional requirements* are the relationships to be established between sensor and actuator values over time.

Specification describes what unit does and how it does it.

VI. Interval Approach

Traditional mathematical techniques (e.g., traditional statistical methods, or methods of designing an optimal controller) are based on the assumption that we know (or at least, we can measure) the exact values of all the necessary parameters. In real life, however, the values are never precisely known. Computers, that are now widely used in real-life systems, contribute to this indeterminacy because their computations are also inherently non-precise.

Interval mathematics has been initially designed to provide the rigorous description and verified analysis of this computer-related type of indeterminacy. However, it is now applied to describe other types of indeterminacy, i.e., indeterminacy caused by the imprecision of measurements.

Many authors have mentioned that there are two aspects of interval mathematics:

- The first aspect, purely *mathematical*, consist of the analysis of interval sets, functions, spaces, maps and equations. Usually, the term "interval mathematics" is reserved only for this type of activity.
- The second aspect is a projection of interval mathematics on computer science; it is usually called *interval computations*. It includes all kinds of theoretical and applied problems related to verified computations.

The first aspect is known to have been described in algebraic terms. Interval computations can also be described in similar algebraic terms, if (as we have explained before), we consider categories in which objects are sets of possible computer representations of real numbers and other (more complicated) mathematical objects.

VII. Categories of Interval Mathematics

Let X be a partially ordered topological space.

PROPOSITION-DEFINITION 1. *Let IX be a class of closed intervals of the space X (e.g., the class of all subsets of the type $\{x \mid a \le x \le b\}$). If $I, J \in IX$ and $I \subseteq J$, then by $i : I \to J$, we denote the natural inclusion. Then, the class of objects IX with morphisms $Mor(I,J) = \{i : I \to J\}$ if $I \subseteq J$ and $Mor(I,J) =$ else, forms a category. We will call this category the interval structure of X.*

Let IR^n be the set of all $n-$dimensional ($n > 0$) intervals over the field of real numbers R with interval operations $+, -, \cdot, /$, and with inclusion $\subseteq$.

PROPOSITION-DEFINITION 2. *Let IR be the class of interval algebraic systems $IR^n = IR^n(+, -, \cdot, /, \subseteq)$, $n > 0$. Let $i_{pq} : IR^p \to IR^q$ ($p \le q$) be the injection, which preserves interval operations. Let's define the set of morphisms as $Mor(IR^p, IR^q) = \{i_{pq}\}$ if $p \le q$ and $Mor(IR^p, IR^q) =$ otherwise (if $p > q$). Then, $CIR = (IR, Mor(.,.))$ is a category.*

Comment. The category CIR can also be considered as a functor from the category of sets IR into the category of algebraic systems $IR(+, -, \cdot, /, \subseteq)$.

VIII. Interval Computations and Computer Algebra

We have designed an *interval extension* of computer algebra system (CAS), i.e., its extension to the case when input parameters can take interval values, and operations are only defined with an interval uncertainty.

The implementation of interval arithmetic (or, more generally, of methods of interval mathematics) in CAS allows to combine computer-algebraic transformations with the strict numerical results obtained during or on completion of the process of computer algebraic transformations.