Object Oriented Commonalities in Universal Generating Function for Reliability and in C++.

Igor Ushakov,
Sumantra Chakravarty[1]

Abstract

The main idea of Universal Generating Function is exposed in reliability applications. Some commonalities in this approach and the C++ language are discussed.

Keywords: Universal Generating function (UGF), C++, reliability.

Introduction

Usually, binary systems are considered in the reliability theory. However, this approach does not describe systems with several levels of performance sufficiently. Analysis of multi-state systems forms now a special branch of the reliability theory.

For analysis of such systems consisting of multi-state subsystems/elements, one can use the method of Universal Generating Functions (UGF), which is described below.

1. Generating function

One frequently uses an effective tool in probabilistic combinatorial analysis: the method of generating functions. For a distribution function of a discrete random variable ξ such that $\Pr\{\xi = k\} = p_k$ for any natural *k*, the generating function has the form

$$\varphi(x) = \sum_k p_k x^k$$

Advantages of using a generating function are well established in this field, and we list a few of those:

(1) For many discrete distributions (e.g., binomial, geometrical, Poisson), there are compact forms of generating functions, which allows one to get analytical solutions quickly and easily.

(2) Moments of statistical distributions can be written in convenient forms. For example, the mathematical expectation of random variable ξ can be found as

$$E\{\xi\} = \frac{\partial}{\partial x}\varphi(x)\Big|_{x=1}.$$

[1] sumontro@hotmail.com

(3) If there are *n* independent random variables ξ_1, ξ_2, ..., ξ_n with the respective generating functions $\varphi_1(x)$, $\varphi_2(x),...,\varphi_n(x)$, then the following generation function can be written for the convolution of these distributions:

$$\varphi(x) = \prod_{j=1}^{n} \varphi_j(x).$$

where $\varphi_j(x) = \sum_k p_{jk} x^k$, and p_{jk} is the probability that *j*-th random variable takes value *k*.

2. Computer alorithm for calculation product of GF's

Let us present a generating function as a set of objects. Each object corresponds to a term in the generating function polynomial. It means that object is a pair of two values: the first is the coefficient, i.e. probability, p, and the second is the power of the argument, a, i.e. the corresponding random variable.

Consider a computational algorithm for calculation of the convolution of two distributions. One makes the following formal operations.

♦ Take two sets of objects: set $\{(p_{11}, a_{11}), (p_{12}, a_{12}),...,$ $(p_{1k}, a_{1k})\}$ for generating function $\varphi_1(x)$, and set $\{(p_{21}, a_{21}), (p_{22}, a_{22}),..., (p_{2m}, a_{2m})\}$ for generating function $\varphi_2(x)$.

♦ Find all cross "interactions" of objects of the first set with all objects of the second set, using the following rule:

[Interacting objects: (p_{1k}, a_{1k}) and (p_{2m}, a_{2m})] →

[Resulting object: $(p_{1k} p_{2m}; a_{1k} + a_{2m})$].

♦ For all resulting objects with different a_{1k_1} for object-1 and a_{2m_2} for object-2, but such that $a_{1k_1} + a_{2m_2} = a$, one forms a new final resulting object: $(\sum p_{1k_1} p_{2m_2};\ a)$. The total set of such final resulting objects gives us the needed solution: from here we can get probabilities for any *a*.

3. Universal generating function

We have described a formalized procedure on sets of objects interaction coresponding to product of polynomials. But in practice, we meet a number of situations when this operation is not enough. Consider the following simple examples.

Example 1. Assume that there is a series connection of two (statistically independent) capacitors (Fig. 1).

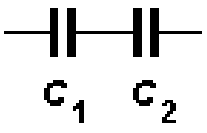

Fig. 1. Series connection of two capacitors.

Assume that c1 and c2 are random with discrete distributions: p1k=Pr{c1=k} and p2j=Pr{c2=j}. One is interested in distribution of total capacity. It is impossible to find the solution with the help of a common generating function. However, there is a possibility to use formal algorithm, described above with the use of corresponding operations over the elements of the objects.

The following procedure can be suggested:

♦ Take two sets of objects, S_1 and S_2:

$$S_1=\{(p_{11},c_{11}),(p_{12},c_{12}),\ldots,(p_{1k},c_{1k})\}$$

and

$$S_2=\{(p_{21},c_{11}),(p_{22},c_{22}),\ldots,(p_{2m},c_{2m})\},$$

where k is the number of discrete values of the first capacitor, and m is the same for the second one. Here the first element of the object is the probability and the second element is the respective capacity.

♦ Find all cross "interactions", Ω, of objects of set S_1 with all objects of set S_2, using the following rule:

$$\Omega\{(p_{1i},c_{1i}),(p_{2j},c_{2j})\}=(p_{ij}^{*};c_{ij}^{*}).$$

Here p_{ij}^{*} is the resulting probability calculated in accordance with the multiplication rule (under assumption of independence) as

$$p_{ij}^{*}=\Omega_{(p)}\{p_{1i},p_{2j}\}=p_{1i}p_{2j},$$

where $\Omega_{(p)}$ is the rule of interaction of parameters p, which in this particular case is multiplication.

Value of c_{ij}^{*} is the resulting capacity calculated in accordance with the harmonic sum rule for capacities:

$$c_{ij}^{*}=\Omega_{(c)}\{c_{1i},c_{2j}\}=\left(c_{1i}^{-1}+c_{2j}^{-1}\right)^{-1},$$

where $\Omega_{(c)}$ is the rule of interaction of parameters c.

♦ Assume that in result we obtain all $R=km$ possible resulting objects of kind $(p^{*};c^{*})$. Let us order all these resulting pairs in increase of value of c^{*}: $(p_1^{*};c_1^{*})$, ..., $(p_R^{*};c_R^{*})$. For some resulting pairs with numbers, say, i, i+1,… , i+j values of c^{*} can be the same and equal some C. We converge such objects into a single aggregated object with parameters: $(\sum_{i\le s\le i+j}p_s^{*};C)$. The total set of such final resulting objects gives us the needed solution.

The procedure can be easily expanded on a series connection of several independent capacitors.

$$\Omega_{(p)}^{SER}\{p_{1i},\ p_{2j},\ \ldots,\ p_{nr}\}=p_{1i}\cdot p_{2j}\cdot\ldots\cdot p_{nr},$$

and

$$\Omega_{(c)}^{SER}\{c_{1i},\ c_{2j},\ \ldots,c_{nr}\}=\left[c_{1i}^{-1}+c_{2j}^{-1}+\ldots+c_{nr}^{-1}\right]^{-1}.$$

Example 2. Pipeline consists of *n* series sections (pipes). Section *j* is characterized by random capacity, for which each value v is realized with some probability p. In this case,

$$\Omega_{(p)}^{PAR}\{p_{1i},\ p_{2j},\ ...,\ p_{nr}\} = p_{1i}\cdot p_{2j}\cdot ...\cdot p_{nr},$$

and

$$\Omega_{(c)}^{SER}\{v_{1i}, v_{2j}, ..., v_{nr}\} = \min\{v_{1i},\ v_{2j},\ ...,\ v_{nr}\},$$

Example 3. One measures a sum of values, each summand of which is random. With probability p_{js} value *j* is measured with standard deviation (STD) equal to σ_{js}. In this case, using notation similar to above, one has:

$$\Omega_{(p)}^{PAR}\{p_{1i},\ p_{2j}, ...,\ p_{nr}\} = p_{1i}\cdot p_{2j}\cdot ...\cdot p_{nr},$$

and

$$\Omega_{(c)}\{\sigma_{1i},\ \sigma_{2j}, ..., \sigma_{nk_n}\} = \sqrt{\sigma_{1i}^2 + \sigma_{2j}^2 + ... + \sigma_{nr}^2}\ .$$

Examples can be continued and not necessarily with probabilistic parameters.

4. Formal description of the Method of Universal Generating Functions

After these simple examples, let us begin with formal description of the Method of Universal Generating Function (UGF[2]). For a more vivid presentation, let us use special terminology to distinguish the UGF from the common generation function. This will relieve us from using traditional terms in a new sense, which may lead to some confusion. Moreover, we hope that this new terminology can help us, in a mnemonic sense, to remember and perhaps even to explain some operations.

In the ancient Roman army, a cohort (C) was the main combat unit. Each cohort consisted of maniples (M), which were independent and sometimes specialized combat units with several soldiers of different profiles. Several cohorts composed a legion (L). The use of this essentially military terminology appears to be convenient in this essentially peaceful mathematical application. A legion is close by its sense to a generating function, a cohort is close to a term of the generating function written in the form of expanded polynomial, and a maniple is close to a parameter of each term.

Starting with polynomial multiplication, in our approach, we will consider less restrictive operations (not only multiplication of terms) and more general parameters. For instance, multiplication of polynomials assumes getting products of coefficients and summation of powers. In our case, we will expand on such restrictive limits on operations.

Let's denote legion j by Lj. This legion includes vj different cohorts, Cjk:

$$L_j = (C_{j1}, C_{j2}, ..., C_{jv_j}).$$

The number of cohorts within different legions might be different. However, in our approach, maniples, which consist of a cohort, must be similar by its structure.

Each cohort C_{jk} is composed of some maniples, M, each of which represents different parameters, special characteristics, and auxiliary attributes. Each cohort consists of the same set of maniples:

$$C_{jk} = (M_{jk}^{(1)}, M_{jk}^{(2)}, ..., M_{jk}^{(s)}).$$

[2] UGF might be also read as Ushakov's Generating Function ☺.

To make description of the method more transparent, let us start with the examples of two legions, L_1 and L_2: each of which consists of the following cohorts, $L_1=(C_{12},C_{12},C_{13})$ and $L_2=(C_{21},C_{22})$, and each cohort C_{jk} includes two maniples $M_{jk}^{(1)}$ and $M_{jk}^{(2)}$,i.e. $C_{jk}=(M_{jk}^{(1)},M_{jk}^{(2)})$. Denote the operation of legion interaction by Ω_L. This operator is used to obtain the resulting legion L_{RES}. In this simple case, one can write:

$$L_{RES} = \Omega_L \{L_1, L_2\}. \quad (1)$$

This interaction of legions produces six pairs of interactions between different cohorts, which generate the following resulting cohorts:

$$C_{RES-1} = \Omega_C \{C_{11}, C_{21}\}, C_{RES-2} = \Omega_C \{C_{11}, C_{22}\},$$

$$C_{RES-3} = \Omega_C \{C_{12}, C_{21}\}, C_{RES-4} = \Omega_C \{C_{12}, C_{22}\},$$

$$C_{RES-5} = \Omega_C \{C_{13}, C_{21}\}, C_{RES-6} = \Omega_C \{C_{13}, C_{22}\}.$$

Here $\Omega_C \{\bullet\}$ denotes the interaction of cohorts.

Interaction of cohorts consists of interaction between its costituent maniples. All cohorts contain maniples of the same types though with individual values of parameters. Let us take, for instance, resulting cohort C_{RES-5}, which is obtained as interaction of cohorts C_{13} and C_{21}. In turn, interaction of these particular cohorts consists in interaction of their corresponding maniples:

$$M_{RES-5}^{(1)} = \Omega_M^{(1)} \{M_{13}^{(1)}, M_{21}^{(1)}\}$$

$$M_{RES-5}^{(2)} = \Omega_M^{(2)} \{M_{13}^{(2)}, M_{21}^{(2)}\}$$

The rules of interaction between maniples of different types, i.e. $\Omega_M^{(1)} \{M_{1i}^{(1)}, M_{2j}^{(1)}\}$ and $\Omega_M^{(2)} \{M_{1i}^{(2)}, M_{2j}^{(2)}\}$ are (or might be) different.

Interaction of n legions can be written as:

$$L = \Omega_L (L_1, L_2, ..., L_n).$$

Operator Ω_L denotes a kind of "n-dimensional Cartesian product" of legions and special final "reformatting" of the resulting cohorts (like converging polynomial terms with the equal power for a common generating function). Since each legion j consists of v_j cohort, the total number of resulting cohorts in the final legion (after all legion interaction) is equal to

$$v = \prod_{1 \le j \le n} v_j .$$

Number v corresponds to the total number of cohorts' interactions.

5. Implementing UGF philosophy in computer language C++

We would like use the UGF (Universal Generating Function) philosophy in an analysis tool and perform reliability calculations for real-world systems. Because we are talking about an (reliability) engineering discipline, all philosophies present the need to be converted into numerical results and predictions. Thus, the UGF philosophy begs an implementation! The implementation task is to identify objects (maniple, cohort, legion) and program all interactions between them. Unfortunately, we run into a combinatoric explosion of possible interactions for a sysem consisting of a large number of (atomic) units. Even moderm computers are not able to enumerate astronomically large (21000) number of interaction states in system consisting of 1000 binary atomic units. Fortunately, for a class of frequently occuring practical systems, the situation is not as hopeless as it may first appear. For a system to be useful in engineering, it may only fail very infrequently. In a highly reliable system, the failure probability of all atomic units much smaller that the system failure probability. This fact makes most of the interactions exceedingly rare and they can be systematically ignored in an approximation scheme that retains only the dominant contributions.

Let us proceed to find an approximate implementation of the UGF philosophy for highly relaible systems in a system simulator. It should be reasonably easy to identify an atomic unit in reliability theory as a maniple. Independence of the maniples corresponds to statistical independence of the atomic units. A cohort is defined to be a collection of maniples. The same definition holds in the context of reliability theory, where the collection is defined by a failure criterion. In a series system, each atomic unit is assumed to provide distinct and critical functionality. This maps on to the notion of specialized combat units. In a parallel system, all atomic units are statistically identical. This improves survival probability during operation, either in the military or in system reliability! Thus, we may identify a subsystem in reliability engineering as a cohort in UGF formalism.

Interactions between the objects are identified in the simulator by their natural reliability names. k-out-of-n combinations are of primary interest. But this class includes the two most frequently appearing reliability structures: series (n-out-of-n) and parallel (1-out-of-n). In fact, probability of failure of a parallel system is negligible (higher order in numerical smallness) with an additional assumption of high availability of the atomic units. Obviously a series system can be made up of distinct units providing separate functionality to the system.

As an illustration let us consider a system S of two subsystems A and B in series. Let A be atomic and B be composed of two atomic units X and Y in parallel. One possible C++ coding for this (simple) system is

$$B=Parallel(X,Y);\ S = Series(A,B);$$

Properties (MTBF, MTTR etc.) of all atomic units are specified at the start of analysis. Operations like Series and Parallel are C++ member functions for the instances of class “unit”. We will not specify unit composition rules in this work. Most of these rules can be found in standard textbooks on reliability engineering. Interested readers may find the remaining ones (involving switching time and PEI) in Chakravarty and Ushakov (2000, 2002).

It remains to identify the “legion”. The preceding paragraphs almost suggest that a legion be identified with the entire system in reliability theory, where the system is further assumed to be represented by its generating function. We would like to note that that this analogy cannot be taken literally sometimes. It is common for a real world reliability system to have deeper hierarchies (e.g., system, equipment shelves, equipment racks, electornic cards) like modern day militaries. In such an elaborate system, we still identify the atomic units as maniples. At the other end, we identify the entire system as a “legion”! All intermediate stages in the hierarchy are

considered generalized "cohorts".

In Chakravarty and Ushakov (2000) implementation, any subsystem can be composed from other subsystems at the next lower level of hierarchy (or atomic units which are always at the lowest level). A newly formed subsystem provides an effective reliability description of all units that compose this subsystem. This composition can be continued indefinitely to obtain an effectiveness measure for the entire system. They have shown that this can be recast as an approximation from a system generating function when all atomic units satisfy binary failure criteria (on/off) they are statistically independent, the system itself is highly reliable and reliability design of the system consists of hierarchical blocks.

6. Reliability analysis of GlobalstarTM Gateways

Globalstar is a low-earth-orbit (LEO) based telephony system with global coverage. The gateways make its ground segment that connect to the orbiting satellites. The gateways are cpmlex systems with more than a thousand components (e.g., electronic cards). Ushakov (1998), Chakravarty and Ushakov (2002) used the UGF approach for the reliability (performance) analysis of GlobalstarTM gateways (fixed ground segment of a low earth orbit satellite communications system). Given the prominence of object oriented abstractions and operations in Globalstar design, it should not be surprising that the reliability analysis naturally fits into the UGF philosophy. Further, these ideas can be naturally implemented in the computer using an object oriented language.

Because of the object oriented nature of system reliability design in Globalstar (interaction between objects like system, racks, shelves, cards are triggered by failure, switching of failed units and changing user demand), Ushakov (1998) proposed that a system reliability simulator should be coded in an object oriented computer language like C++. Later, Chakravarty and Ushakov (2002) implemented a simulator for the GlobalstarTM Gateway in C++.

In Chakravarty and Ushakov implementation for Globalstar, C++ objects are in one-to-one correspondence with reliability objects. An object is specified by mean time between failures (MTBF), mean time to repair/replace (MTTR) and an effectiveness weight (partial effectiveness index: PEI). By definition, PEI=1 for binary atomic units. All failure distributions are implicitly assumed to be Exponential. If failed units were to be automatically swapped, a switching time was also assigned by Chakravarty and Ushakov (2000). Even small switching time is important because it changes a parallel system "on paper" to a series system with small MTTR. This may have dramatic effect overall on system reliability.

References

1. Chakravarty, S., and Ushakov, I. *Reliability measure based on average loss of capacity. International Transactions in Operational Research*, Vol. 9, No. 2, 2002.
2. Chakravarty, S., and Ushakov, I. *Effectiveness Analysis of GlobalstarTM Gateways. Proceedings of Second International Conference on Mathematical Methods in Reliability (MMR'2000)*, Vol. 1, Bordeaux, France, 2000.
3. Ushakov, I.A. *The Method of Generating Sequences. European Journal of Operational Research*, Vol. 125/2, 2000.
4. Ushakov, I.A. *An object oriented approach to generalized generating function*. *Proc. of the ECCO-XI Conference (European Chapter on Combinatorial Optimization)*, Copenhagen, May 1998.
5. Ushakov, I.A. *Reliability Analysis of Multi-State Systems by Means of a Modified Generating Function. Journal of Information Processes and Cybernetics* (Germany), Vol.24, No.3, 1988.
6. Gnedenko, B.V., and I.A. Ushakov. *Probabilistic Reliability Engineering.* John Wiley & Sons, New York.1995.

7. Ushakov, I.A. *Solving of Optimal Redundancy Problem by Means of a Generalized Generating Function. Journal of Information Processes and Cybernetics* (Germany), Vol.24, No.4-5,1988.
8. Ushakov, I.A. *Solution of Multi-Criteria Discrete Optimization Problems Using a Universal Generating Function*. *Soviet Journal of Computer and System Sciences* (USA), Vol. 25, No. 5, 1987.
9. Ushakov, I.A. *Optimal Standby Problem and a Universal Generating Function*. *Soviet Journal of Computer and System Sciences* (USA), Vol. 25, No. 4, 1987.
10. Ushakov, I.A. *A Universal Generating Function*. *Soviet Journal of Computer and System Sciences* (USA), Vol. 24, No. 5, 1986.