RESEARCH-ARTICLE

Using Metamorphic Relations in Redundancy-based Fault/Intrusion Tolerance

FELICITA DI GIANDOMENICO, Institute of Information Science and Technologies "Alessandro Faedo", Pisa, PI, Italy

GIULIO MASETTI

FRANCESCA LONETTI, Institute of Information Science and Technologies "Alessandro Faedo", Pisa, PI, Italy

ANTONIA BERTOLINO, Gran Sasso Science Institute, L'Aquila, AQ, Italy

Accepted: 11 October 2025
Revised: 18 September 2025
Received: 05 February 2025

Using Metamorphic Relations in Redundancy-based Fault/Intrusion Tolerance

FELICITA DI GIANDOMENICO, ISTI-CNR, Italy
GIULIO MASETTI*, Scuola Secondaria di Primo Grado, Monterchi, Italy
FRANCESCA LONETTI, ISTI-CNR, Italy
ANTONIA BERTOLINO†, GSSI, Italy

Redundancy is widely used as a method for fault and intrusion tolerance. However, if the redundant components lack sufficient diversity, potentially dangerous common mode failures may go undetected. To address this issue, the design diversity approach has been proposed in the literature for decades. In this paper, we take an innovative approach to this problem by introducing a broader notion of diversity, which leverages Metamorphic Relations (MRs), i.e., necessary properties that must hold among diverse inputs and diverse outputs. We define two generic categories of MRs that establish data diversity and functional diversity. Furthermore, we elaborate on two corresponding logical architectures, paying particular attention to the necessary conditions for the adjudicator component. Finally, we present an initial evaluation of the proposed architectures, which points out the advantages with respect to their counterparts based on the traditional design diversity method, and discuss future research directions for this novel conceptual approach to redundancy-based fault/intrusion tolerance.

CCS Concepts: • **Computer systems organization → Architectures**; **Dependable and fault-tolerant systems and networks**; • **Software and its engineering → Software organization and properties**; • **Security and privacy → Software and application security**.

Additional Key Words and Phrases: Metamorphic Relations, Fault/Intrusion Tolerance, Data diversity, Functional diversity

ACRONYMS

FV	False Verification
MDDR	Metamorphic Data Diversity Relation
MFDR	Metamorphic Function Diversity Relation
MR	Metamorphic Relation
MT	Metamorphic Testing
NMDP	N Metamorphic Data Programming
NMFP	N Metamorphic Functional Programming
NVP	N-Version Programming
RB	Recovery Block

*Work performed while at ISTI-CNR, Italy.
†Work performed while at ISTI-CNR, Italy.

Authors' Contact Information: Felicita Di Giandomenico, felicita.digiandomenico@isti.cnr.it, ISTI-CNR, Pisa, Italy; Giulio Masetti, 106ohm@gmail.com, Scuola Secondaria di Primo Grado, Monterchi, Pisa, Italy; Francesca Lonetti, francesca.lonetti@isti.cnr.it, ISTI-CNR, Pisa, Italy; Antonia Bertolino, antonia.bertolino@gssi.it, GSSI, L'Aquila, Italy.

ACM 1557-7392/2025/10-ART
https://doi.org/10.1145/3772722

SYMBOLS

Ck	check function Ck : $O \times \{1, \ldots, r\} \times O \rightarrow \{0, 1\}$
C_{att}	(attack) cost to attack a given fault/intrusion tolerance scheme
C_{def}	(defense) cost to implement a given fault/intrusion tolerance scheme
$\hat{c}_i$	(attack) cost to manipulate a program
$\bar{c}_i$	(defense) cost to diversify a program
$\hat{c}_r$	(attack) cost to know a metamorphic relation
$\bar{c}_r$	(defense) cost to establish a metamorphic relation
$\hat{c}_s$	(attack) cost to know the functional specification
$\bar{c}_s$	(defense) cost to define the functional specification
f	number of faults $f \in \mathbb{N}$
g	given function $g : I \rightarrow O$
g_j	$g_j : I \rightarrow O$ is a source function for $j = 0$ and a follow-up function for $j > 0$. Details in Section 3.2
I	input set
i	source input $i \in I$
i_j	follow-up input $i_j : I \rightarrow I$, for $j > 0$. Details in Section 3.1
n	number of programs $n \in \mathbb{N}$
O	output set
o_0	output $o_0 \in O$
o_j	follow-up output $o_j \in O$, for $j > 0$. Details in Section 3.1
o^j	follow-up output $o^j \in O$, for $j > 0$. Details in Section 3.2
p_j	program that implements the evaluation of g on i_j. Details in Section 3.1
p^j	program that implements the evaluation of g_j on i. Details in Section 3.2
R	metamorphic relation. Details are in Sections 3.1 and 3.2
r	number of relations $r \in \mathbb{N}$
T	input set

1 INTRODUCTION

Fault tolerance mechanisms are a key solution for assuring the reliability of a system despite the presence of faults. These faults can be accidental, e.g., due to unintentional mistakes or bad decisions of developers or operators, or malicious, i.e., purposely introduced to alter the system behavior [5]. To complement the classical fault tolerance solutions, since the early nineties, intrusion tolerance approaches have emerged aiming to guarantee that a system works correctly, mainly tailoring accidental faults, but even when some of its parts are compromised due to a malicious fault [43].

Redundancy is typically practiced within a fault tolerant architecture, such that extra components/executions are employed, and the results obtained by such a redundant set of components are combined through an adjudication mechanism to determine the final outcome. Both hardware and software system components can be made redundant for tolerance purposes; in this paper, the focus is on software system components.

Originally, redundancy was obtained using identical copies, as in the Triple Modular Redundancy early envisaged by Von Neumann [44]. However, since the late seventies, it was realized that if a fault-tolerant system uses identical subsystem copies, these may plausibly present identical design faults. Thus, under some circumstances, they could fail in the same manner, providing consistent results that cannot be detected as faulty [35]: these are referred to as common-mode failures.

Indeed, if all redundant components use the same software design, they are likely to fail in the same way. To address the critical issue of common-mode failures arising from design faults, design diversity has been introduced in fault-tolerant systems. Knight [21] summarizes different techniques to achieve design diversity, including: i) for each software instance, adoption of a different development team; ii) employment of different development tools, development methods, or programming languages; iii) using both up-to-date and older versions of the software.

A common requisite of the methods for design diversity redundancy proposed in the literature is that all the obtained software instances must adhere to the same functional specification so that, when correct, they all provide the same output (although, of course, bit-by-bit equality cannot always be guaranteed). This traditional redundancy approach has proven effective in mitigating common mode failures, but identical faults are still possible [6], pointing out the need of further research investigation on diversity. To enhance this aspect, and further empower the scope of fault tolerance mechanisms, in this paper, we take a new, radically different direction to diversity for fault/intrusion tolerance. We target redundancy-based parallel schemes, like the N-version programming (NVP) structure [4], and leverage Metamorphic Relations (MRs) to achieve a broader notion of diversity.

In software engineering research, MRs are the core component of Metamorphic Testing (MT) [13, 14]. MT is a software testing technique used to alleviate the oracle problem by leveraging known relationships — called MRs — between multiple inputs and their expected outputs, rather than relying on a known correct output for a single input. In our proposal the focus is not on MT, but on MRs, namely relations that, given a function, provide the necessary properties over a sequence of inputs and their corresponding outputs [14]. The main idea of our proposal is to use MRs in order to generate either different inputs for the same functional specification, or different functional specifications, in a redundancy-based fault/intrusion tolerance scheme.

We first formally define two new MRs patterns that we call *Metamorphic Data Diversity Relation (MDDR)* and *Metamorphic Function Diversity Relation (MFDR)*, tailored to achieve diversity in data and functional specifications of a redundant scheme, respectively. Then, we propose two new fault/intrusion tolerance architectures that we call *N Metamorphic Data Programming (NMDP)* and *N Metamorphic Functional Programming (NMFP)*, which leverage MDDR and MFDR, respectively, to achieve diversity.

We discuss the conditions of applicability of the proposed architectures and the constraints on the number of MRs to achieve a desired degree of fault/intrusion tolerance.

Exemplary instantiations of the two fault/intrusion tolerance architectures are provided in the context of trigonometric functions and image classification, thus showing how MRs can be employed to achieve either data diversity or functional specification diversity, as well as to provide a final outcome.

Although resorting to data diversity in fault-tolerance schemes to cope with design faults has already been investigated in the literature, the proposed approaches do not use MRs (with the exception of [30] and [48], as we discuss in Section 2). Instead, to the best of our knowledge, diversity at the functional specification level obtained through MRs is an entirely new concept in fault/intrusion tolerance. Adopting different functional specifications as we propose in the NMFP architecture, instead of equivalent but diverse programs (as in traditional NVP architecture or in our NMDP) represents a radical new solution for achieving diversity in NVP fault-tolerant schemes.

The expected benefit is an increase of the degree of diversity among the programs employed in a redundant fault/intrusion tolerance scheme, which translates in a stronger mitigation of the common-mode failure pursued by diversity techniques.

Summarizing, the contribution of this paper includes:

- the formal definition of two new MRs patterns that allow to provide diversity in data and functional specification;

- the definition of two new logical architectures for fault/intrusion tolerance that leverage the two newly defined MRs patterns. Representative examples of MRs instantiating these new logical schemes in order to tolerate different numbers of faults/intrusions are also provided;
- the rigorous definition of an adjudication logic, appropriate for both NMDP and NMFP redundancy-based schemes, valid for an arbitrary number of faults/intrusions;
- a quantitative evaluation of the benefits brought by metamorphic diversity, carried out in terms of the cost an attacker must afford to compromise a system made fault tolerant through the newly proposed NMDP and NMFP redundancy-based schemes, in comparison with the classical NVP scheme. The improvements obtained demonstrate the usefulness of MRs to generate forms of diversity to strengthen fault/intrusion tolerance schemes against common-mode failures;
- two realistic examples of the practical application of the NMDP and NMFP schemes in the domain of Cyber-Physical Systems. Precisely, we discuss how the NMDP scheme could be used for a fault-tolerant implementation of a traffic manager of a system of elevators, and how the NMFP scheme could be applied to ensure the robustness of a nuclear reactor monitoring system, both against one fault/attack.

The paper is structured as follows: in the next section, we provide background information and review related work. In Section 3, we define the two MRs patterns relative to data diversity (MDDR) and functional diversity (MFDR). In Section 4 and Section 5, we present the redundant architectures using MRs, which are NMDP and NMFP, respectively. Then, in Section 6, we elaborate on the logic behind an important component of the defined architectures, i.e., the adjudicator. In Section 7, we perform an evaluation of the benefits brought by metamorphic diversity, and in Section 8 we discuss some examples of possible practical applications of the proposed architectures. Finally, in Section 9, we conclude the paper with a brief summary and hints for future work.

2 BACKGROUND AND RELATED WORK

Our work spans over the two traditionally separated research topics of diversity-based fault tolerance and metamorphic testing. In this section we discuss how our work is related to the literature of both topics, and also report about similar work we are aware of.

2.1 Diversity-based fault tolerance

Fault tolerance is a fundamental means to achieve dependable computing systems [5]. It aims at failure avoidance despite the presence of errors, i.e., to assure that a system works correctly even when some of its parts are experiencing malfunctions. Instead of relying on just a single component, redundancy of components is at the basis of fault tolerance architectures.

The challenge of design faults, typically resulting in common-mode failures, promoted studies on developing redundancy through the design diversity principle: development of functionally equivalent programs, but obtained adopting diversity approaches, referred to as *versions*. Diversity means include different algorithms, different programming languages/compilers, different execution environments, different HW supports, to mention a few. Several papers in the literature propose sets of practices to support design diversity, such as [10, 18, 27, 28]. In addition to the decision on the measures to adopt for enforcing diversity in the redundant structure under development, other design issues around the development of a redundancy-based fault tolerance scheme are: i) the selection of versions to employ in the redundant configuration (e.g., how many versions to employ, each developed according to which diverse methodology, thus showing a required reliability level); ii) the decision on the execution pattern of the selected versions (the two extremes are fully sequential and fully parallel execution); iii) the decision on the adjudicator function to adopt for selecting the only output from the set provided by the employed versions (the two extremes are acceptance test on each single output and voter involving all the

produced outputs). The designer mainly addresses these issues based on the specific needs of the application under development (including dependability requirements, as well as other requirements in the time domain and operational context), the available development environment facilities, and the reference fault tolerance architecture.

The original Recovery Block (RB) approach [32] employs sequential versions execution and acceptance test on each single output. Instead, the original N-Version Programming (NVP) scheme [4] is based on the execution of parallel versions, whose outputs are voted to select the adjudged correct output. A variety of hybrid schemes trades in between these two extremes, both in terms of execution of the available redundancy, and adjudication function to select the final output (e.g., the works in [11, 24, 33]).

An alternative approach involves applying diversity at the input level, known as the "data diversity" approach, as first presented in the work of Knights et al. [3]: instead of feeding all the redundant components with the same input, different inputs obtained through re-expressed formulations of the original one are used in addition to the original input. The paper considers a fault tolerant structure where the same software (replica) executes on each of the diverse inputs. To provide high-assurance arguments against a class of data corruption attacks, this re-expression based data diversity approach has been refined in [31] to deal with N-variant systems, thus leveraging both software diversity and data diversity.

The concept of metamorphic data diversity, utilized in the development of the NMDP fault/intrusion architecture presented in this paper, aims to expand and generalize existing approaches to data diversity. This is achieved by leveraging (in principle, equality and inequality) MRs, which play a dual role: they are used both for generating follow-up inputs and for assisting the adjudicator component in selecting the final output of the redundant architecture. This also includes recalculating a correct outcome as obtained from processing the original input from an output generated by a follow-up input.

The MRs discussed in this paper require more outputs to verify the stated relations. Consequently, the emphasis when proposing the new logical architectures for fault/intrusion tolerance that leverage MRs is placed on redundancy structured in parallel executions, following the NVP style rather than the RB style. Thus, the logical architecture of the classical NVP fault tolerance scheme serves as a baseline for comparing the new solutions proposed in this paper. Its graphical representation is shown in Figure 1. First of all, a terminological note is made: although diverse software components are referred to as *versions* by the dependability community, in this paper the term *programs* is used instead, to keep homogeneous the terminology among the different developments. Starting from the functional specification of the software component to be made fault tolerant and a set of practices to exploit software diversity, the n programs are generated. Each program executes on the same input i, in parallel with all the others. The outputs of these n programs are conveyed to the adjudicator component, responsible for selecting the final scheme's output. Specifically, the adopted adjudication function looks for an output that satisfies the criterion to be judged as correct. A typical adjudicator employed by NVP is a majority voting, implying a number of programs $n = 2f + 1$, where f indicates the maximum number of faults to tolerate. If such an output exists, it is released as the correct output, otherwise an error notification or a default value is issued.

From the implementation perspective, the NVP scheme needs mechanisms to assure that the same input is given to all the programs and that the adjudication function is able to work in presence of correct but not identical values, as expected in presence of diversity (that is, bit-by-bit voters cannot be employed). Note that in this paper the treatment of the newly proposed fault tolerance schemes is at logical level, whereas the implementation of such mechanisms, beyond what has already been provided in the literature, is left to future work.

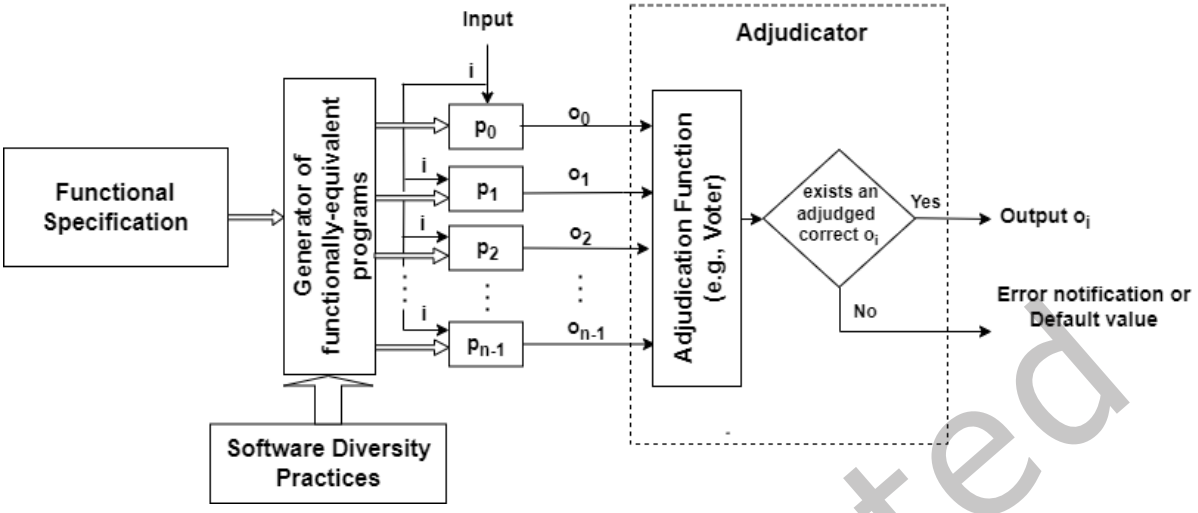


Fig. 1. Logical architecture of the classical NVP fault tolerance scheme.

2.2 Metamorphic Testing

Metamorphic Testing (MT) is based on the concept of Metamorphic Relation (MR), i.e. for a given function g a MR is expressed as a relation among a set of two or more inputs (i, i_1, ..., $i_n \in I$) and their corresponding outputs ($o_0 = g(i)$, $o_1 = g(i_1)$, ..., $o_n = g(i_n) \in O$) related to multiple executions [37]. In MT, a set of *source test cases* is adopted as the seed for deriving new test cases, or *follow-up test cases*, according to the MR, then both source test cases and follow-up test cases are executed on the program and their outputs are checked against MR; if this relation is violated, the test has failed and the program is faulty [37].

MT mainly addresses two fundamental problems of testing that are test cases generation and results verification. In the former case [13], test selection approaches are defined to derive follow-up test cases from successful test cases, i.e. test cases that do not cause software failures. A set of MRs that describe particular properties of the program is identified, then follow-up test cases are generated that may reveal faults that violate these properties. In the literature, many techniques exist to generate source test cases, including random strategies, search-based testing, symbolic execution, or leveraging existing test suites [37]. In the latter case, a set of MRs representing properties of the software under test is used to solve the so-called test oracle problem, i.e., to check the correctness of the system output or behaviour [29]. In particular, MT can help for those "untestable" programs for which identifying a test oracle is hard or inherently impossible [38]. Differently from traditional testing, where the output of a single program execution is checked against an oracle, in Metamorphic Testing, the outputs of multiple program executions are verified to check whether a set of MRs is satisfied or has been violated.

MRs have been used in the literature also for post-deployment testing and run-time monitoring of adaptive systems to detect system faults and to trigger context-specific adaptations [15].

The goal of using MRs in this paper is neither for MT, i.e. for improving fault detection, nor for runtime monitoring, but for defining new NVP schemes for fault/intrusion tolerance. Specifically, in our proposal MRs are leveraged for a twofold objective: i) to achieve diversity in fault/intrusion tolerance schemes. This diversity can deal with either the inputs of an NVP scheme (different inputs of a same program are generated according to a set of MRs), or diverse software programs of an NVP structure (these programs are derived from different specifications that are generated from a source functional specification according to a set of MRs); ii) to check

and provide the correct output of an NVP structure. Details of how MRs are defined to achieve both objectives are presented in Section 3. Then, their usage in defining fault/intrusion tolerance schemes and an adjudicator logic are detailed in Section 4, Section 5 and Section 6.

2.3 Similar work

In addition to the literature on diversity-based Fault Tolerance and Metamorphic Testing already surveyed, to the best of our knowledge, the only attempts to leverage metamorphic relations in the context of fault tolerance are the studies in [30, 48]. In the short paper by Liu et al. [30], the authors introduce a basic metamorphic fault tolerance framework and discuss its constituting components (*Generator*, *Checker*, and *Calculator*), targeting NVP-like organization where redundancy is built with replicas of the same program. Research questions are discussed and directions for potentialities and extensions are informally presented.

In the recent paper [48], the primitive framework proposed in [30] is further elaborated and proposed as a solution to address what the authors define as the oracle problem in fault tolerance. The components of the metamorphic fault tolerance architectures are defined in detail, in particular the output selector in charge of selecting a trustworthy output among those obtained from the original input and from follow-up inputs generated by exploiting metamorphic relations.

We also took inspiration from the framework in [30], but aimed to achieve the goal of pursuing stronger protection against common-mode failures. Additionally, we observe that the two approaches are founded on concepts developed by separate communities (the testing community and the dependability community). This divergence in foundational ideas results in two approaches that, while offering valuable insights, cannot be directly compared due to their differing objectives and underlying principles. In fact, adhering to the theory and practice of fault tolerance as conceptualized in [5], differently from [48], we define architectural configurations in which the number of versions used depends on the number of faults that need to be tolerated, so that the outcome is correct if the stated number of faults is not exceeded (and the adjudicator works correctly). This relevant difference at the design level affects the definition of the involved components and the overall goals of the two fault tolerance solutions. The aim of [48] is to select the most trustworthy output, while this paper focuses on selecting the correct output, under the assumption that a predefined maximum number of faults is not exceeded. Consequently, the two metamorphic fault tolerance frameworks do not appear to be directly comparable.

Moreover, our study has a more comprehensive goal than the paper in [48], namely: i) the adoption of MRs to generate not only diverse inputs, but also diverse programs; ii) the development of formal notations for expressing metamorphic diversity in input data and programs; iii) a detailed presentation of a fault tolerance metamorphic architecture also for metamorphic-based functional diversity.

3 METAMORPHIC RELATIONS TO ACHIEVE DIVERSITY IN FAULT/INTRUSION TOLERANT ARCHITECTURES

In this section, we define the two MRs patterns that are employed in the rest of the paper, namely Metamorphic Data Diversity Relation and Metamorphic Function Diversity Relation. To facilitate comprehension, we also provide some examples for each of them. It is worth noting that while the given definitions are general, their instantiation, and hence the examples we provide, will depend on the assumed fault model and on the logic of the adopted adjudicator (a component of the NMDP and NMFP architectures), which we are going to introduce later in Section 6. For the moment it is sufficient to be aware that when we refer below to a number or a type of MRs, we are assuming the fault model and the adjudicator logic as presented in Section 6.

3.1 Data diversity

Definition 1 (Metamorphic Data Diversity Relation). *Let $g : I \rightarrow O$ a function, i a source input. A Metamorphic Data Diversity Relation (MDDR) to be used in redundancy-based fault/intrusion tolerance schemes is defined as a set $R=\{R_m\}_{m=1,\ldots,r}$ of relations:*

$R_m\big(i, i_1(i), \ldots, i_{n-1}(i), g(i), g(i_1(i)), \ldots, g(i_{n-1}(i))\big) \subset I^n \times O^n$,

where $n \geq 2$, $r \geq 2$, and the elements in $\{i_j(i)\}_{j=1,\ldots,n-1}$, with $i_j : I \rightarrow I$, are the follow-up inputs derived from i according to R.

To always guarantee the identification of the expected output, when fault assumptions are satisfied, $n-1$ relations in R must be of the form:

- $R_1(i, i_1(i), o_0, o_1)$,
- $R_2(i, i_2(i), o_0, o_2)$,
- $\vdots$
- $R_{n-1}(i, i_{n-1}(i), o_0, o_{n-1})$,

where $o_0 = g(i)$, $o_1 = g(i_1(i))$, ..., $o_{n-1} = g(i_{n-1}(i))$.

When $r \geq n$, the remaining $r - n + 1$ elements of R might involve a mix of couples, triples, etc, of follow-up inputs, the simplest examples being:

- $R_n(i_1(i), i_2(i), o_1, o_2)$,
- $R_{n+1}(i_2(i), i_3(i), o_2, o_3)$,
- $\vdots$
- $R_r(i_{n-2}(i), i_{n-1}(i), o_{n-2}, o_{n-1})$.

How to choose the kind and number of metamorphic relations composing MDDR depends on the assumed fault model and on the logic of the adjudicator. When understood, in the following the argument of i_j will be omitted, and $i \in I$ will indicate simply a variable.

Example 1. *For instance, given the function $g(i) = \sin(i)$ and a source input (i), an MDDR can be composed of the following metamorphic relations:*

- *a relation R_1 (e.g., $\sin(i) = \sin(i_1)$) is defined on the inputs i and i_1, as well as the outputs $o_0 = \sin(i)$ and $o_1 = \sin(i_1)$, where ($i_1 = \pi - i$) is a follow-up input derived from i according to R_1.*
- *a relation R_2 (e.g.,$-\sin(i) = \sin(i_2)$) is defined on the inputs i and i_2, as well as the outputs $o_0 = \sin(i)$ and $o_2 = \sin(i_2)$, where ($i_2 = \pi + i$) is a follow-up input derived from i according to R_2.*

To tolerate $f = 1$ faults/intrusions, the two relations defined above are sufficient, but for higher values of f more relations are needed. An example of additional relations that do not involve i and o_0, but the already available follow-up inputs i_1 and i_2, is $R_3 : \sin(i_1) + \sin(i_2) = 0$.

3.2 Function diversity

Definition 2 (Metamorphic Function Diversity Relation). *Let $g_0 : I \rightarrow O$ a source function, i an input. Metamorphic Function Diversity Relation (MFDR) is defined as a set $R=\{R_m\}_{m=1,\ldots r}$ of relations:*

$$R_m\big(i, g_0(i), \ldots, g_{n-1}(i)\big) \subset I \times O \times O_1 \times \cdots \times O_{n-1},$$

where $n \geq 2$, $r \geq 2$ and the elements in $\{g_j(i)\}_{j=1,\ldots,n-1}$ with $g_j : I \rightarrow O_j$, are the follow-up functions derived from $g_0(i)$ and i, and according to R.

To always guarantee the identification of the expected output, when fault assumptions are satisfied, $n-1$ relations in R must be of the form:

- $R_1(i, o^0, o^1)$
- $R_2(i, o^0, o^2)$

⋮

- $R_{n-1}(i, o^0, o^{n-1})$

where $o^0 = g_0(i), o^1 = g_1(i), \ldots, o^{n-1} = g_{n-1}(i)$.

When $r \geq n$, the remaining $r - n + 1$ elements of R involve couples, triples, etc, of follow-up functions, the simplest examples being:

- $R_n(i, o^1, o^2)$
- $R_{n+1}(i, o^2, o^3)$

⋮

- $R_r(i, o^{n-2}, o^{n-1})$

Example 2. *Let us consider the trigonometric functions sine (*sin*), cosine (*cos*), tangent (*tan*) and cosecant (*csc*). For instance, given an existing source function ($g_0 = \sin$), and an input (i), an MFDR can be defined using the following metamorphic relations:*

- *a relation R_1 (e.g., $\sin(i)^2 + \cos(i)^2 = 1$) is defined on* sin *and* cos*, as well as the outputs o^0 and o^1 produced by executing $\sin(i)$ and $\cos(i)$, where $g_1 = \cos$ is a follow-up function derived from $g_0 = \sin$ according to R_1.*
- *a relation R_2 $\left(\text{e.g., } \sin(i) = \pm\tan(i)/\sqrt{1+\tan(i)^2}\right)$ is defined on* sin *and* tan*, as well as the outputs o^0 and o^2 produced by executing $\sin(i)$ and $\tan(i)$, where $g_2 = \tan$ is a follow-up function derived from $g_0 = \sin$ according to R_2.*

To tolerate $f = 1$ fault/intrusion, $n = 3$ and $r = 2$ is sufficient. For $f > 1$, more follow-up functions are needed such as $g_3 = \csc$, for which a relation R_3 (e.g., $\sin(i) \cdot \csc(i) = 1$, with $i \neq k \cdot \pi$ and $k \in \mathbb{Z}$) can be checked on the inputs sin and csc, as well as the outputs o^0 and o^3 produced by executing $\sin(i)$ and $\csc(i)$.

For the two follow-up functions g_1 and g_2, further relations can be defined such as:

- $R_4(g_1, g_2) \ : \ g_2 = \pm\sqrt{\frac{1-{g_1}^2}{{g_1}^2}}$
- $R_5(g_0, g_1, g_2) \ : \ g_2 = \frac{g_0}{g_1}$

4 METAMORPHIC DATA DIVERSITY IN THE NVP ARCHITECTURE

This section focuses on exploiting metamorphic relations to generate diverse input data to feed the n software programs organized according to the NVP structure.

As already discussed, in the traditional NVP scheme, redundancy consists of generating functionally equivalent versions, each one processing the same input. A proposal to extend diversity also to inputs, to enhance the ability of the scheme to cope with specific classes of attacks, is presented in [31], where data variations are created by using different data re-expression functions, such that each variant will operate on different data. Corresponding inverse re-expression functions preserve the semantics of the original data. Although not explicitly discussed by the authors, this re-expression rule can be considered as a specific form of metamorphic relation, that is the equality relation. In this paper, as anticipated in the introduction, we are interested in generalizing the application of transformation relations, borrowing from the rich metamorphic relations theory developed by the software testing community.

4.1 Logical architecture of NMDP

The logical architecture of the first fault tolerance scheme we propose, named NMDP, is illustrated in Figure 2. Similarly to the classical NVP scheme in Figure 1, it employs n diverse but functionally equivalent programs

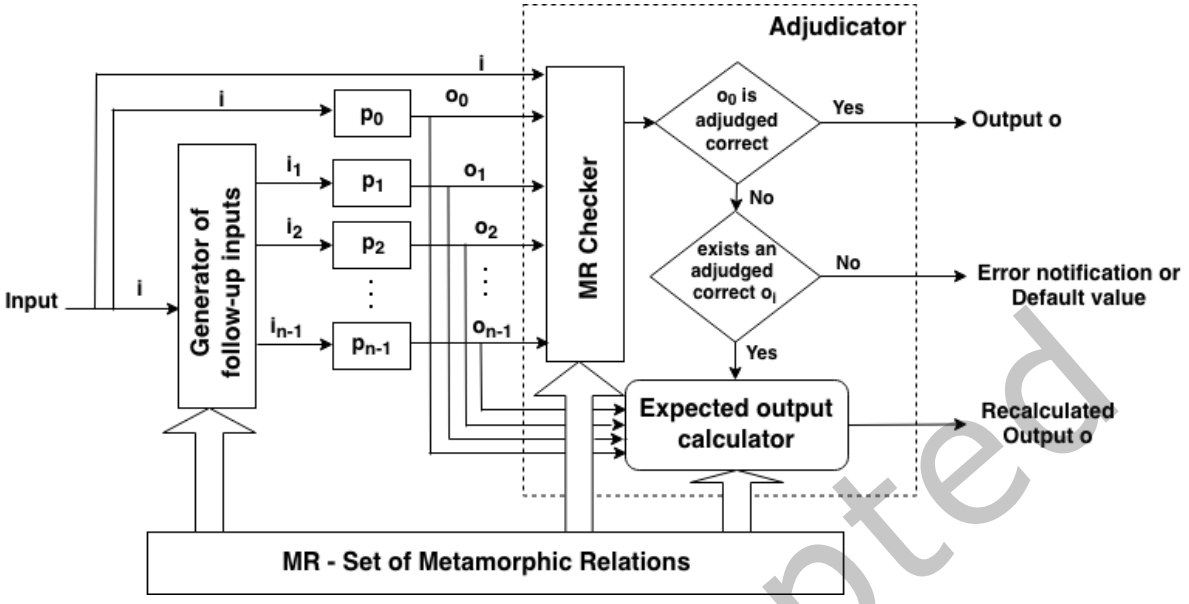


Fig. 2. Logical architecture of the NMDP fault tolerance scheme.

(p_0,..., p_{n-1}) of the software component that is made redundant. For the sake of simplicity, the generator of the different programs is not illustrated in Figure 2, since it can be the same as shown in Figure 1. However, only one program (indicated in the figure as p_0) processes the source input i, while each of the other programs processes a diverse input generated from i by exploiting a set of identified MRs. Each produced output is collected by the Adjudicator component, which performs multiple operations to identify the adjudged correct output (if it exists), again exploiting the MRs. If an adjudged correct output is not found, the outcome of the NMDP structure is typically a default value or an alarm notification.

A more detailed description of the NMDP components is provided in the following. The example illustrated in the next Section 4.2 will then provide specific details on how the scheme works in practice.

4.1.1 MDDR - Set of Metamorphic Relations. The set MDDR includes the r identified metamorphic relations needed for the correct operation of NMDP. A subset $(n-1)$ of such relations is used to generate follow-up inputs from the given source input i (for simplicity, let's call them R_1, ...R_{n-1}). The remaining ones, generically involving n-tuples of the generated inputs (like couples, triples), are used to select an adjudged final output from programs' executions, as detailed next when describing the *Adjudicator* component. According to the definition of metamorphic data diversity already presented in Section 3, the number r of needed relations depends on the assumed fault/intrusion model and the logic of the *Adjudicator*.

4.1.2 Generator of follow-up inputs. This component takes in input the set MDDR and a source input i, and generates a set of follow-up inputs $i_1, i_2, \ldots i_{n-1}$, according to the metamorphic data diversity relations R_1, R_2, ...R_{n-1}, respectively.

According to the convention illustrated in Figure 2, each input in the generated set is processed by one of the $n-1$ programs p_1, ...p_{n-1}, while the source input i is processed by p_0. The programs' outputs are then forwarded to the Adjudicator component. Note that this assignment is not mandatory and a different one would be appropriate as well, since the different versions are functionally equivalent programs (that is, they produce the same output

when given the same input). In more detail, if an input i_j is executed on program p_k or program p_l, the produced output o_j will always satisfy the corresponding metamorphic relation, provided no error occurs in the executed program. However, if p_k is faulty, an erroneous output may or may not be produced, depending on the processed input. From this perspective, it is significant which input is processed by p_k. Nevertheless, if an erroneous output is generated, it does not affect the selection of the correct output, assuming p_k has been counted as one of the f maximum expected faulty programs during each execution.

4.1.3 Adjudicator. This is the component responsible for producing the adjudged correct outcome of the NMDP scheme through the functionalities performed by its constituting sub-components *MR-Checker* and *Expected-output-calculator*.

First, the n outcomes o_0, o_1, ..., o_{n-1} (from the execution of the p_0, p_1, ..., p_{n-1} programs, respectively) are processed by *MR-Checker* to verify which ones (if any) satisfy the appropriate metamorphic relations in MDDR.

Each couple o_0, o_i, for $i = 1, .., n-1$, is first checked against the corresponding R_i relation to adjudge whether o_0 is correct. Then, the outcomes of the checks involving all the other stated relations in MDDR are analyzed, in accordance with the adopted adjudication logic to assess whether an adjudged correct output exists, or the tolerable number of faults/intrusions has been exceeded, and no output can be selected.

To exemplify, in case three programs are employed to tolerate one malfunction (assumption of at most one fault/intrusion), two output couples are checked: (o_0, o_1) against R_1 and (o_0, o_2) against R_2. If the relations are satisfied in both cases, then the three outcomes are all judged as correct. If instead, both are not satisfied, the output o_0 is considered incorrect and the other two correct, under the assumption that, at most, 1 program can fail. In the other two cases where one relation is satisfied, and the other is not, the output o_1 or o_2 involved in the failed relation is judged as incorrect, while the other two are judged as correct.

If o_0 is judged as correct, it is selected as the final output of the NMDP scheme, since it is the output produced by processing the source input i. Instead, when the set of judged correct outputs does not include o_0, an output o_j in this set is selected but requires manipulations to become the final output of the scheme. In fact, it has been produced by processing an input i_j, generated from the source input i through the metamorphic relation R_j. Therefore, it is needed to calculate the expected output, given the source input i, by manipulating o_j through R_j; this is performed by the *Expected-output-calculator* component, according to the definitions in Section 3. Two concrete examples of such processing will be shown in the next Section 4.2.

An error notification, or a default value, is issued when a judged correct outcome is not found (for example, in the case of 5 programs under the assumption of up to two failed programs and the checks of all the metamorphic relations involving the five obtained outputs are not satisfied).

4.2 Application example of NMDP

In this section we show how the logical architecture of the proposed NMDP scheme can be instantiated considering the metamorphic relations of Example 1 of Section 3.1. In this example, three diverse programs p_0, p_1, p_2 implement the target function $g(i) = \sin(i)$.

Specifically, the *Generator* component takes as input:

- the source input (i);
- the metamorphic relation $R_1 : \sin(i) = \sin(i_1)$;
- the metamorphic relation R_2: $-\sin(i) = \sin(i_2)$;

It generates the following new follow-up inputs:

- $i_1 = \pi - i$, according to R_1
- $i_2 = \pi + i$, according to R_2

Table 1. Adjudication table for the case $o_0 = \sin(i)$ and $f = 1$

$R_1(o_0, o_1)$	$R_2(o_0, o_2)$	Correct Output	Incorrect Output	Final Output
✓	✓	o_0, o_1, o_2		o_0
✗	✓	o_0, o_2	o_1	o_0
✓	✗	o_0, o_1	o_2	o_0
✗	✗	o_1, o_2	o_0	Rec(o_1) or Rec(o_2)

The inputs i, π - i, π + i, are executed on programs p_0, p_1, p_2, respectively, and the corresponding outputs o_0 =sin(i), o_1 =sin($\pi - i$) , o_2 = sin($\pi + i$) are collected by the *Adjudicator* component. Note that a different assignment of inputs to programs would be appropriate as well, as already previously discussed.

Moreover, the *MR checker* subcomponent in the *Adjudicator* component checks if the metamorphic relations R_1 and R_2 are satisfied on the outputs o_0, o_1, o_2. Specifically, it checks if the couples (o_0, o_1) and (o_0, o_2) satisfy the metamorphic relations $o_0 = o_1$ and $-o_0 = o_2$ respectively. Table 1 shows the adjudication process[1] presenting: the possible outcomes of the relations checks (first two columns); the identification of correct and incorrect outputs (third and fourth columns); the final output (last column). As depicted in the table, four scenarios are possible:

(1) both the couples (o_0, o_1) and (o_0, o_2) satisfy the relations $o_0 = o_1$ and $-o_0 = o_2$ respectively; in this case o_0, o_1 and o_2 are judged as correct (second row of Table 1).
(2) the couple (o_0, o_1) does not satisfy the relation $o_0 = o_1$, whereas the couple (o_0, o_2) satisfies the relation $-o_0 = o_2$; in this case under the assumption that at most one program can fail, o_0 and o_2 are judged as correct whereas o_1 is considered incorrect (third row of Table 1).
(3) the couple (o_0, o_1) satisfies the relation $o_0 = o_1$ whereas the couple (o_0, o_2) does not satisfy the relation $-o_0 = o_2$; in this case always under the assumption that at most one program can fail, o_0 and o_1 are judged as correct whereas o_2 is considered incorrect (fourth row of Table 1).
(4) neither of the couples (o_0, o_1) and (o_0, o_2) satisfies the respective relation, since $o_0 \neq o_1$ and $-o_0 \neq o_2$; in this case, always under the assumption that at most one program can fail, o_0 that is involved in both unsatisfied relations is considered incorrect whereas o_1 and o_2 are judged as correct (fifth row of Table 1).

In the first three scenarios, *Adjudicator* chooses o_0 as final output since it is the output related to the source input and it has been adjudicated among the correct outputs. In the fourth scenario, the *Expected output calculator* subcomponent computes the *recalculated output o*, to release as final output, from o_1 or o_2 by exploiting the metamorphic relations R_1 (called $Rec(o_1)$) or R_2 (called $Rec(o_2)$) respectively, as:

- $Rec(o_1)$ = sin(i) = sin(i_1) = o_1, i.e. $Rec(o_1)$=o_1
- $Rec(o_2)$ = sin(i) = - sin(i_2) = -o_2, i.e. $Rec(o_2)$=-o_2

4.3 Example of NMDP with inequality relations

Consider the task of classifying images in two categories labeled by $O = \{-1, 1\}$. More formally, the aim is to define a function $I = \mathbb{R}^N \rightarrow O = \{-1, 1\}$, where I is the set of digital images with n pixels (each characterized by a real number), through supervised learning based on a given training set $\{(x_j, y_j)\}_{j=1,\ldots,n_{\text{train}}}$. In particular, the focus of this example is on Support Vector Machines with exponential kernel [17], where the result is equal to 1 if $d(x) \geq 0$ with

$$d(x) := \beta + \sum_{j=1}^{n_{\text{train}}} \alpha_j y_j \kappa(x, x_j),\ \kappa(x, x_j) := \exp\left(-\gamma ||x - x_j||^2\right)$$

[1]This example refers to the case with $f = 1$.

and -1 otherwise. Here, α, β and γ are the parameters obtained through training. To tolerate $f = 1$ faults/intrusions, consider the following set MDDR of metamorphic relations:

- R_1: $d(i_1) - d(i) = \alpha y_j \Big(\exp\big(-\gamma||i||^2\big) - \kappa(i, x_j) \Big)$, where (x_j, y_j) is one of the elements of the training set,
- R_2: $d(i_2) \geq \exp\big(-\gamma||c||^2\big) \cdot \big(d(i) - \beta\big) + \beta$, for a given $c \in \mathbb{R}$, maybe chosen at random

and the corresponding follow-up inputs $i_1 = i + x_j$ and $i_2 = i + c$. The adjudicator is the same as the one detailed in Section 4.2, except for the fact that here $\text{Rec}(o_2)$ works properly only if

$$\frac{d(i_2) + \beta\Big(\exp\big(-\gamma||c||^2\big) - 1\Big)}{\exp\big(-\gamma||c||^2\big)} < 0,$$

and $o = -1$. This example shows that inequality relations can be easily defined (here, R_2 follows from the triangle inequality in vector spaces) when used for judging if o_0 is correct or not, but they can be cumbersome when used to reconstruct o_0. However, in the example, this is not a problem because $\text{Rec}(o_1)$ always works.

5 METAMORPHIC FUNCTIONAL DIVERSITY IN THE NVP-LIKE ARCHITECTURE

This section focuses on exploiting metamorphic relations to generate n diverse software programs to organize according to the NVP structure, starting from a source functional specification. Different from traditional program diversity that relies on the same functional specification, here, the idea is to apply diversity already at the level of functional specification, from which the diverse programs are then implemented.

As already underlined, this is a novel approach, not yet pursued in the context of fault/intrusion tolerance, to the best of the authors' knowledge. The intuition behind the idea of resorting to follow-up specifications is that by raising the level at which diversity is applied (from programs to specifications from which programs are then derived), beneficial effects are expected to mitigate the phenomenon of common-mode failure among the programs employed in the redundant structure. In fact, given a layered organization (in our case, from software specification to executable code), measures taken at a certain layer have the ability to impact the specific level itself and all the underlying levels (as, e.g., discussed in the papers in [27, 40]).

Since the ultimate goal of using diversity is to reduce common-mode failures, exploring diverse functional specifications can be a promising approach to achieving this diversity. This is supported by the initial quantitative analysis presented in Section 7.

5.1 Logical architecture of NMFP

The logical architecture of the proposed fault tolerance scheme, named N Metamorphic Functional Programming (NMFP) is illustrated in Figure 3.

Similarly to the previously presented NMDP scheme, also this architecture resembles NVP in Figure 1 in terms of employed programs of the software component that is made redundant (p^0,..., p^{n-1}). However, this time diversity concerns the functional specifications from which the redundant programs are derived instead of the input data processed by these programs, as was in NMDP.

Specifically, starting from the source functional specification g_0, $n-1$ diverse follow-up specifications are derived by exploiting a set of identified metamorphic relations, from which a different program is then obtained. The program obtained from g_0 completes the set of the n programs (for illustrative convenience, in Figure 3 it is p^0). All the programs process the same source input i, and their outputs o^i are collected by the Adjudicator component, which performs multiple operations to identify the adjudged correct output (if it exists), again exploiting the metamorphic relations. In case an adjudged correct output is not found, the outcome of the NMFP structure (as it was for NMDP) is typically a default value or an alarm notification.

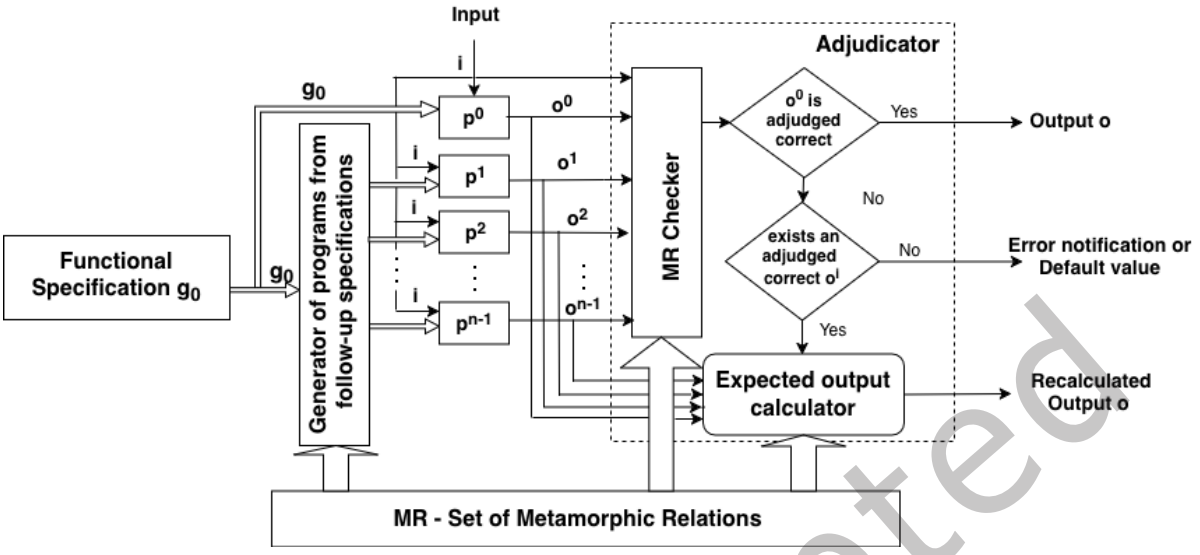


Fig. 3. Logical architecture of the NMFP fault tolerance scheme.

A detailed description of the NMFP components is provided in the following. As expected, they perform similar activities as the corresponding NMDP components already presented; however, to fully understand the impact of the concept of follow-up specifications/programs peculiar to the NMFP architecture, instead of follow-up inputs, their description is also provided.

5.1.1 MFDR - Set of Metamorphic Relations. This set includes the r identified metamorphic relations needed to assure the correct operation of NMFP. Among them, $(n-1)$ relations are used to generate the requested number of follow-up functional specifications from the given source specification g_0, each one then originating a diverse program to obtain the system configuration that satisfies the tolerance goal (for simplicity, let's call them $R_1, ...R_{n-1}$). Other relations, generically involving n-tuples of the generated functional specifications (such as couples or triples), are used to select an adjudged final output from programs' executions, as detailed next when describing the *Adjudicator* component. According to the definitions at the basis of the metamorphic functional diversity already presented in Section 3, the number r of needed relations depends on the assumed fault/intrusion model and the logic of the *Adjudicator*.

5.1.2 Generator of programs from follow-up specifications. This component takes in input the set R and a source functional specification g_0, and generates a set of follow-up functional specifications $g_1, ...g_{n-1}$, according to the metamorphic functional diversity relations $R_1, R_2, ...R_{n-1}$ in R, respectively. Then, each specification g_j is implemented in the program p^j, with p^0 obtained from g_0, according to the convention in Figure 3. The programs elaborate the same input i, and their outputs are then forwarded to the *Adjudicator* component.

5.1.3 Adjudicator. This component performs the selection of the adjudged correct outcome of the NMFP scheme, through the functionalities performed by its constituting *MR-Checker* and *Expected-output-calculator* sub-components.

First, the n outcomes $o^0, o^1, ..., o^{n-1}$ (from the execution of the $p^0, p^1, ..., p^{n-1}$ programs, respectively) are processed by *MR-Checker* to verify which ones (if any) satisfy the appropriate metamorphic relations in R. Specifically, each couple (o^0, o^j), for $j = 1, .., n-1$, is first checked against the corresponding R_i relation, to verify

whether o^0 is correct or not. Then, the outcomes of the checks involving the remaining relations in R are analyzed in accordance with the adopted adjudication logic to assess whether an adjudged correct output exists or the tolerable number of faults/intrusions has been exceeded and no output can be selected.

To exemplify, the same scenario involving three programs, already considered when describing the NMDP scheme, which can tolerate one malfunction, is valid also here: the output couple (o^0, o^1) is evaluated against R_1, and (o^0, o^2) against R_2. If the relations are satisfied in both cases, then the three outcomes are all judged as correct. If both are not satisfied, the output o^0 is considered incorrect and the other two correct, assuming that, at most, 1 program can fail. In the other two cases, i.e., one relation is satisfied, and the other is not, the output o^1 or o^2 involved in the failed relation is judged as incorrect, and the other two as correct.

If o^0 is judged as correct, it is selected as the final output of the NMFP scheme since it is the output produced by the program p^0, which is derived from the original specification g_0. Instead, when the set of judged correct outputs does not include o^0, an output o^j in this set is selected but requires manipulations to become the final output of the scheme. In fact, it has been produced by a program implemented from a follow-up specification of the source specification g_0 through the metamorphic relation R_j. Therefore, it is needed to calculate the expected correct output (i.e., the one that would be provided by the correct execution of the source program p^0) by manipulating o^j through R_j; this is performed by the *Expected-output-calculator* component, following the theoretical formulations in Section 3. A concrete example of such processing will be shown in the next Section 5.2.

As in the NMDP scheme, in case a judged correct outcome is not found, an error notification or a default value is issued.

5.2 Application example of NMFP

In this section, we show how the logical architecture of the proposed NMFP scheme can be instantiated considering the metamorphic relations of Example 2 of Section 3.2. In this example, the *Generator* component takes as input:

- the source functional specification $g_0 = \sin$
- the metamorphic relation R_1: $\sin(i)^2 + \cos(i)^2 = 1$
- the metamorphic relation R_2: $\sin(i) = \pm\frac{\tan(i)}{\sqrt{1+\tan(i)^2}}$

It generates the following new follow-up functional specifications:

- $g_1 = \cos$, according to R_1
- $g_2 = \tan$, according to R_2

The functional specifications g_0, g_1, $g2$ are implemented by the programs p^0, p^1, p^2, respectively. The same input i is executed on programs p^0, p^1, p^2, and the corresponding outputs $o^0 = \sin(i)$, $o^1 = \cos(i)$, $o^2 = \tan(i)$ are collected by the *Adjudicator* component.

Moreover, the *MR checker* subcomponent in the *Adjudicator* component checks if the metamorphic relations R_1 and R_2 are satisfied on the outputs o^0, o^1, o^2. Specifically, it checks if the couples (o^0, o^1) and (o^0, o^2) satisfy the metamorphic relations $\sin(i)^2 + \cos(i)^2 = 1$ and $\sin(i) = \pm\frac{\tan(i)}{\sqrt{1+\tan(i)^2}}$ respectively. The *Adjudicator* is the same as the one described in Section 4.2 and detailed in Table 1.

Also in this case, in the first three scenarios (second, third, and fourth rows of Table 1), the Adjudicator chooses o^0 as the final output. In the fourth scenario the *Expected output calculator* subcomponent computes the *recalculated output* o from o^1 or o^2 by exploiting the metamorphic relation R_1: $\sin(i)^2 + \cos(i)^2 = 1$, or R_2: $\sin(i) = \pm\frac{\tan(i)}{\sqrt{1+\tan(i)^2}}$, respectively, as:

- $Rec(o^1) = sin(i) = \pm\sqrt{1-\cos(i)^2} = \pm\sqrt{1-(o^1)^2}$
- $Rec(o^2) = sin(i) = \pm\frac{\tan(i)}{\sqrt{1+\tan(i)^2}} = \pm\frac{o^2}{\sqrt{1+(o^2)^2}}$

Between the two possible results from each of these recalculation expressions (same value, but with a positive or a negative sign), the selection of the correct one is operated on the basis of the quadrant the executed input (known to the *Adjudicator*) belongs to.

6 AN ADJUDICATOR LOGIC FOR METAMORPHIC-BASED REDUNDANCY

In this section, we first elaborate on aspects around the adjudication in NVP-like fault tolerant schemes and then present the new logic for the adjudicator component of the proposed NMDP and NMFP architectures. Note that the proposed logic is appropriate for both architectures, since the essence of the difference between NMDP and NMFP lies only in the definition of MRs, which for the former is based on the concept of follow-up input, while for the latter on the concept of follow-up specification function. However, both types of metamorphic relation connect inputs with their corresponding outputs. From the perspective of the number of faults / intrusions to be tolerated and consequent redundancy in terms of programs and metamorphic relations used, the two architectures do not differ.

6.1 Adjudicator context

In traditional fault tolerance solutions, the starting point is the definition of the desired tolerance degree, that is, the number f of faults/intrusions to be tolerated; then resource containment and simplicity of adjudication operation are usually privileged. Focusing on the NVP scheme in Figure 1, the simple adjudication logic typically employed is that of a majority voter. Then the number n of needed programs is such that a majority of correct outcomes is guaranteed under the assumption of up to f faults, that is $n = 2f + 1$. To deal with versions instead of replicas, more sophisticated adjudication functions than the bit-by-bit majority have been proposed (as presented in [25]). Data diversity has been managed through the re-expression of computed outputs before being sent to the adjudicator (as presented in [31]).

The introduction of metamorphic-based diversity implies that the outcomes of the programs are no longer directly comparable values since they are either generated from metamorphically related inputs, or from metamorphically related program specifications. This implies that majority voters, as used in NVP architectures, are no longer applicable. Moreover, since the expected output from the redundant scheme is that produced by the original program (or a functionally equivalent one) on the source input i, indicated as o_0, the adjudicator has to be able to recalculate o_0 from any other correct o_j in case o_0 is adjudged as erroneous. This naturally leads to an adjudicator structured in two phases: during the first phase, the focus is on assessing the correctness of o_0, so as to promptly terminate the scheme in case it is correct. Otherwise, the second phase is started, where an adjudged correct output among the remaining ones is selected, if any, and used to recalculate o_0, using the proper metamorphic relation[2].

Fully adhering to the literature on fault tolerance, the adjudicator selects a correct output when the number of failed programs is not higher than the tolerable number f, whichever is their occurrence pattern (including independent and common mode failures). In case this number is exceeded, either the absence of a correct output is detected (in most cases), or an incorrect output is released. Also, the adjudicator's logic is optimistic, so a pattern of output evaluation compatible with up to f failed components is always preferred to an equally possible alternative that implies a number of failed components higher than f.

[2]Note that, as a special case, if the number of tolerable faults $f = 1$, any of the o_j is correct, thus the second phase reduces to taking any of them and recalculating o_0.

6.2 Adjudicator logic

The examples given for the NMDP and NMFD architectures in Sections 4 and 5, respectively, are simple instantiations tailored to a low number of faults (only 1 fault). In such cases the logic of the adjudicator is very simple and intuitive.

As the number of faults grows, so does consequently the number of involved programs and MRs. Hence, it becomes necessary to rigorously define the logic of the adjudicator, in order to properly set the scheme's parameters. Among the possible definitions, in the following an adjudication logic is proposed based on the following principles:

- both at the first and second phase, the adjudicator applies a syntactic and systematic criterion to select the adjudged correct output;
- metamorphic relations and consequent checks for their verification are limited to couples, as defined below.

In particular, for the case of MDDR, the considered metamorphic relations are as follows:

- $n-1$ relations of the form: $R_x(i, i_j(i), o_0, o_j)$, with $j = 1, .., n-1$. In particular, these relations are needed to recompute the expected output from an adjudged correct output o_j different from o_0;
- $n-2$ relations of the form: $R_y(i_1(i), i_j(i), o_1, o_j)$, $j = 2, .., n-1$
- .....
- n-k relations of the form: $R_w(i_{k-1}(i), i_j(i), o_{k-1}, o_j)$, $j = k, .., n-1$, $k = 3, .., n-2$.
- ...
- a last relation of the form: $R_z(i_{n-2}(i), i_{n-1}(i), o_{n-2}, o_{n-1})$.

The above expressions indicate that there is a metamorphic relation between any couple of inputs/outputs, where of course $R(i_h(i), i_j(i), o_h, o_j) = R(i_j(i), i_h(i), o_j, o_h)$.

Summing up, in this adjudicator the total number r of required MRs is thus related to n according to the formula: $r = \frac{n(n-1)}{2}$.

For the case of MFDR, similar relations are established, as also presented in Section 3. For brevity and to keep the notation lighter, in this section only reference to MDDR relations is made, but the same reasoning fully applies to MFDR relations, given the symmetry of the two kinds of relations.

The adjudicator selects the scheme's output on the basis of the outcome of the verification of the available MRs. Let's indicate with

$$\mathrm{Ck} : O \times \{1, \ldots, r\} \times O \rightarrow \{0, 1\}$$

$$\mathrm{Ck}(o_i, x, o_j) = \begin{cases} 1 & \text{if } R_x(o_i, o_j) \text{ is verified} \\ 0 & \text{otherwise} \end{cases}$$

the *check function* that verifies the fulfillment[3] of the metamorphic relation R_x between outputs o_i and o_j. Given the assumed failure model, which includes the common-mode failure, $\mathrm{Ck}(\ldots) = 1$ is not a guarantee that both the outputs verifying the relations are correct, since both could be erroneous values. In the following, the occurrence of a verified relation when both outputs are erroneous (i.e., the manifestation of a common-mode failure) is indicated with False Verification (*FV*). On the other hand, $\mathrm{Ck}(\ldots) = 0$ always indicates that one or both the exercised outputs are erroneous (the possibility of being both correct does not make sense).

The criterion adopted by the proposed adjudicator for adjudging a correct output in the two adjudication phases derives from the same basic principle.

[3]According to the definitions in Section 3, given two outputs, at most one metamorphic relation exists between them. Even though, in principle, more relations could be established, this case is not addressed in this paper.

In the first phase, the adjudicator uses the outcomes of $\mathrm{Ck}(o_0, x, o_j)$, with $j = 1, \ldots, n-1$. Thus, $2f$ checks are performed, involving all the $n = 2f + 1$ programs. Given that f faults can occur, in the worst case generating common-mode failures, the criterion applied by the adjudicator to adjudge the correctness of o_0 is that out of the $n - 1$ obtained check's outcomes, at least f are 1, meaning that f couples of outputs satisfy their metamorphic relation.

The second phase is launched if the output o_0 is adjudged as erroneous, and consequently, the remaining $n - 1$ outputs are considered, among which at most $f - 1$ can be erroneous. In particular, this time the focus is on the $\frac{n(n-3)+2}{2}$ outcomes of $\mathrm{Ck}(o_j, x, o_h)$, with $j, h = 1, \ldots, n-1$, and $j \neq h$. In this phase, a correct output o_i has to be selected among the population of $n - 1$ available ones. Since each output is metamorphically checked against any other output, the same kind of information as for the case of o_0 at the first phase is available to the adjudicator for each o_j, $j = 1, \ldots, n-1$. Therefore, the same criterion can be applied, but considering $f - 1$ possible erroneous outputs. Specifically, the adjudicator selects as correct any output o_j, $j = 1, \ldots, n-1$ for which, out of the $n - 2$ obtained check's outcomes involving it, at least $f - 1$ are 1. Then, applying the same reasoning as at the first phase, the minimum number of checks required to cope with common-mode failures is $2(f - 1)$, involving $n = 2f - 1$ programs. Actually, the number of programs requested for the first phase (i.e., $n = 2f + 1$) always assures even more than the minimum number of requested programs in the second phase, being $2f > 2f - 1$[4].

Concerning the recalculation of the expected final output if o_0 is wrong and o_m correct, and R_m is an equality relation [14], then the expected output is computed solving the equation defined by R_m, where i, i_m and o_m are treated as parameters. Even when there is no closed formula for o_0 it is often possible to evaluate it through an iterative procedure[5]. If the metamorphic relations are not equality relations, then recalculation is more complex (an example is in Section 4.3), or not possible at all.

Summarizing, the proposed adjudication logic requires $n = 2f + 1$ and $r = \frac{n(n-1)}{2}$ to successfully adjudge a correct output in presence of up to f faults/intrusions.

Some illustrative examples are shown in Figures 4 and 5, assuming $f = 3$, $n = 7$ and $r = 21$. Being all the considered MRs binary relations, the data structure employed to illustrate the adjudication algorithm is a colored regular polygon (in the specific example, a hexagon), where o_0 is the center and o_j are the vertices, for $j = 1, \ldots, 6$. In the left side of both figures, edges represent the outcomes of the check function $\mathrm{Ck}(\ldots)$ when applied to the two outputs at the extremes: a solid black line indicates a verified check, a solid red line indicates a non verified check, while a dotted line indicates no check (because not pertinent to the considered phase). In the right side, outputs judged correct are checkmarked in green, while those judged wrong are crossed in red; finally, question marks are used when the adjudicator cannot yet issue a decision.

More in detail, Figure 4 focuses on the first phase, where the left part of the figure represents the outcomes (here the radiuses of the hexagon) of the check functions $\mathrm{Ck}(o_0, x, o_j)$, with $j = 1, \ldots, 6$, while the left part represents the decision of the adjudication after the application of the criterion used in the first phase. For the cases, 1), 2), and 3), o_0 is adjudged correct since there are at least $f = 3$ verified metamorphic relations involving the considered outputs. Instead, 4), 5) and 6) represent cases where the number of not verified MRs is greater or equal to $f + 1$, thus o_0 is erroneous and the second phase is launched. In the figure, the occurrences of FVs in cases 4), 5) are also indicated.

Figure 5 illustrates the second phase of adjudication, focusing on case 6) of Figure 4. The meaning of the left and right part of the figure is the same as for Figure 4, but referred to the second phase. The figure depicts a few basic cases for a generic output o_i, showing when it is judged as correct, since it satisfies at least $f - 1$ metamorphic

[4]The additional redundancy available at the second phase could be used to possibly detect patterns of faults/intrusions in number greater than $f - 1$. Otherwise, if efficiency is a major concern, a refined adjudication logic that restricts the outputs and associated relations to the minimum to be considered for the experienced pattern of failure manifestations can be developed. In this paper, the objective was to present a basic adjudication logic, and any smart implementation of it is postponed to future work.

[5]For instance, if $g \in C^1(\mathbb{R})$ then o_0 can be retrieved through the Newton-Raphson method [34].

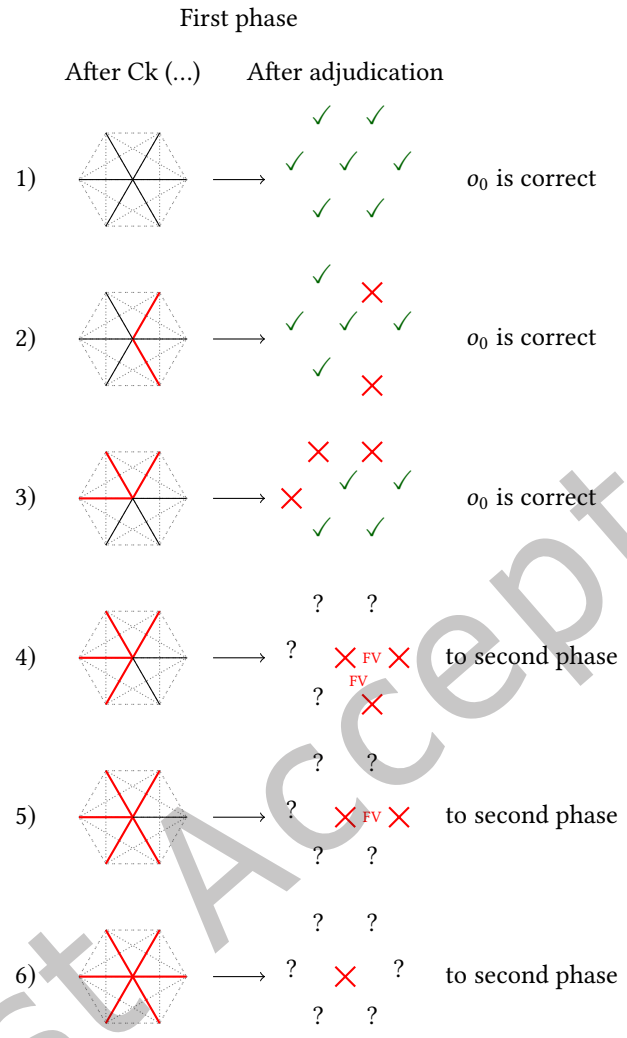


Fig. 4. Examples of cases resulting from the first phase of the adjudication when $f = 3$, $n = 7$ and $r = 21$. Here, o_0 is always placed at the hexagon center. The left side shows the outcomes of Ck, while the right part shows the outcomes of the adjudication criterion.

relations (cases 6.1, 6.2, 6.3, 6.4), and when it is judged as incorrect (case 6.5, where only 1 metamorphic relation is satisfied). Note that case 6.4 indicates that there are at least 3 erroneous outputs at the second phase, and the o_i under consideration could be actually incorrect although adjudged as correct. However, this is not a failure of the tolerance scheme, since the maximum number of faults/intrusions is exceeded (the working assumption is $f = 3$), so any output is admissible, even an incorrect one.

6.3 Observations on the adjudicator logic

A few final observations regarding the proposed adjudication logic are drawn in the following.

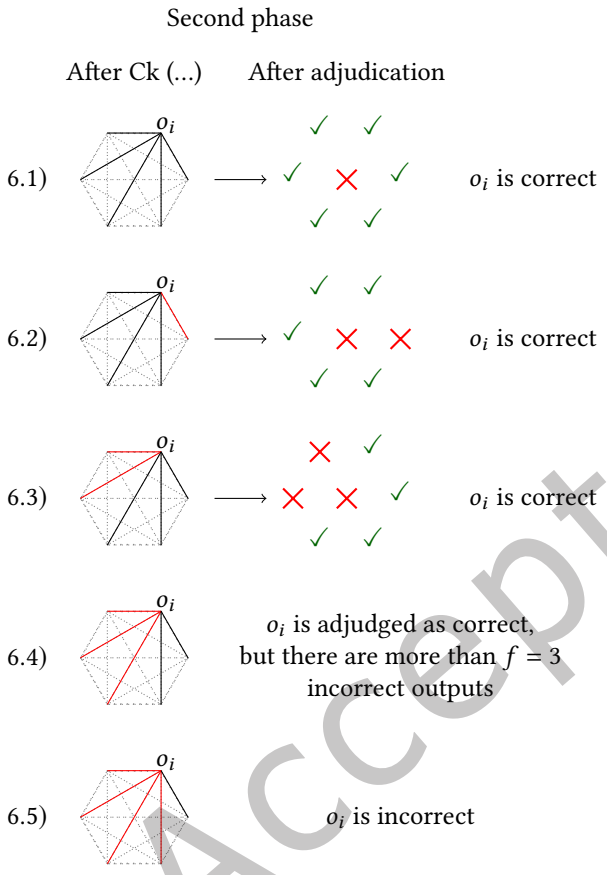


Fig. 5. Evolution of case 6) of Figure 4 for a generic output o_i at the second phase of adjudication. Only basic cases are shown; symmetric ones are easily understandable.

The expected output calculator component automatically rebuilds the output generated by the source input (or source function) from the one generated by a follow-up input or follow-up function, using the pertinent metamorphic relation (see the examples in Sections 4.2 and 4.3 for NMDP, and Section 5.2 for NMFP). Once the MRs are set up, the recalculation functions can also be automatically defined. In addition, there are MRs for which recalculation can be performed more easily, and others for which it is more cumbersome (an example is given in Section 4.3). All these aspects should be taken into account when selecting the MRs to employ among the potential candidates, and also when selecting an adjudged correct output for recalculation of the output to deliver.

The adjudicator uses metamorphic relations to assess the correctness of the received outputs and select the (adjudged) correct result to deliver as the final output of the redundant scheme. Therefore, in principle, the higher the number of established (MDDR or MFDR) metamorphic relations (e.g., involving not only couples but also triples, quadruples of elements, etc..), the richer the information available to the adjudicator to decide the final output reliably. However, such expected benefits are counterbalanced by a more sophisticated logic necessary to take advantage of the more informative but more intricate data, which is generally more prone to faults. In

fact, the adjudicator has to be a highly reliable component since its role is crucial for the reliability of the fault tolerance scheme. Such requirement of high reliability has been traditionally addressed either by resorting to a simple adjudication logic to promote highly controllable operation (as in the case of majority voter) or by making the adjudicator itself a fault-tolerant component, with consequent higher complexity of the overall fault tolerant scheme. With the intent to primarily focus on the new concept of metamorphic diversity and gradually increase the complexity of the resulting scheme, simplicity of operation has been privileged in the definition of the logic proposed in this paper, namely the usage of simple criteria for deciding at the first and second phases. Applying more sophisticated adjudication algorithms is expected to potentially save on employed resources (either programs or relations) but at the cost of potentially impairing the adjudicator's reliability.

Differently from the majority voting adjudicator employed in classical NVP, the one defined here encompasses both semantic and syntactic aspects: the verification of relations has a semantic nature, while a selection of an output having a stated number of relations verified is purely syntactic. This combination of semantic and syntactic features is beneficial to higher accuracy of the adjudication decision and possibly to reduce FVs, responsible for common-mode failures. While further calculations are necessary to check the results for selecting the final output, they are conducted using a systematic and rigorously verifiable procedure. Consequently, the added complexity seems manageable overall. Anyway, a focus on the reliability of the proposed adjudicator is postponed to an implementation phase in a specific application context, where it needs to be assessed whether it allows to meet the requirements of that particular application, or appropriate measures need to be taken to enhance its reliability.

Finally, an aspect that could become critical is the number of needed metamorphic relations as the number of faults/intrusions to be tolerated grows. For example, $f = 2$ requires $r = 10$, and $f = 3$ requires $r = 21$. While this is, in principle, a potential limitation, the fault tolerance degree pursued in concrete applications rarely exceeds $f = 2$ (with most applications showing the need to cope with $f = 1$ only), as addressed in the fault tolerance architectures in References [2, 42, 45, 46]. Moreover, the literature on metamorphic testing has been investigating since rather long time the world of metamorphic relations in a variety of application contexts (e.g., [37]). In the last decade, the automated discovery of metamorphic relations is a growing important research topic for allowing full test automation. According to a recent systematic review on metamorphic relations' generation [26], until June 2024, there have been 81 papers introducing techniques for systematically generating metamorphic relations, among which 63 were published between 2019 and 2024 and this publication trend is growing. Some approaches define new metamorphic relations by composing existing ones [16]; others leverage machine-learning approaches [19] or even LLMs [39] for deriving specific types of metamorphic relations, other works leverage search-based algorithms to automatically generate metamorphic relations for numerical programs (for instance GENMORPH [9], or AUTOMR [47]), whereas other ones generate metamorphic relations for specific domains as cyber-physical systems (GASSERTMRS [8]) or query-based systems [36]. In addition, there exist some publicly available repositories of metamorphic relations, such as that in [12], which provides 76 system-agnostic MRs for automated security testing in Web systems. The expectation is that these solutions can be of interest to, and relatively easily leveraged by, the newly defined fault tolerance architectures, especially by NMDP. Resorting to alternative adjudication logics that aim to minimize metamorphic relations is another direction to overcome this limitation, but very likely at the expense of a higher number of programs. The definition of other adjudication algorithms will be addressed in future work.

7 EVALUATION OF FUNCTIONALLY EQUIVALENT DIVERSITY VS MDDR AND MFDR

In this section, a quantitative comparison is performed between the newly introduced MDDR and MFDR to generate diversity-based redundancy and the functionally equivalent diversity in traditional NVP. The objective is to understand how the two approaches relate to one another regarding their efficacy in mitigating common-mode failures.

Since the most prone to generate common-mode failures are intentional attacks, we focus on them and conduct the evaluation in terms of a cost function representing the effort (and skill) an attacker needs to compromise the intrusion tolerant scheme such that a common-failure is induced, resulting in the adjudicator selecting an erroneous final output. For the moment, the probability of an attacker successfully intruding into a program is not considered; instead, in this scenario, the attacker is always successful if equipped with the minimum necessary resources for the intrusion. Hence, the analysis is optimistic from the attacker's point of view.

To simplify the scenario, the analysis is performed under the following assumptions: i) the adjudicator component cannot be attacked; it is either run on a Trusted Execution Environment, or it is, in turn, made intrusion tolerant; ii) given a specification, the cost to attack a program implementing that specification successfully is always the same (no matter how different are the redundant programs adhering to the same specification; upon discovering a vulnerability of the program under attack, the attacker successfully intrudes the software at the first attempt, so the effort spent is the same for any program); iii) similarly, the cost to discover metamorphic relations among outputs is always considered the same, independently from the specific programs that have produced the outputs. Adhering to such assumptions is reasonable since the objective is to conduct a comparative analysis rather than an accurate estimate for each scheme.

The definition of the cost functions for the attacker depends on the scheme under consideration, in particular:

- for NVP based on the (inexact) majority voting, the attacker first has to acquire knowledge of what is the functional specification of the targeted subsystem and then has to compromise $f+1$ programs (out of the n composing the scheme to tolerate f faults/attacks), in order to forge their outcome so that the adjudicator selects such erroneous value as final output. Therefore, the function for calculating the incurred cost by the attacker is:

$$C_{att}(\text{NVP}) = \hat{c}_s + (f+1)\hat{c}_i$$

  where $\hat{c}_s$ is the cost to know the functional specification and $\hat{c}_i$ is the cost to attack a program and manipulate its outcome successfully;
- for NMDP adopting the adjudicator's logic described in Section 6, the case of minimal effort needed by the attacker is when the adjudicator ends at the first phase, having observed that at least f relation checks involving o_0 are 1. Therefore, first, as for NVP, the attacker has to acquire knowledge of which is the functional specification of the targeted subsystem, but, in addition, has to get knowledge of the metamorphic relations between the output o_0 (the outcome of the program processing the source input i) and all the other outputs (from the remaining $n-1$ programs in the fault tolerant architecture). Then, the attacker has to compromise all the n programs and forge their outputs to satisfy the adjudicator's condition to end at the first phase (at least f metamorphic relations satisfied). Accounting for all of this makes the function for calculating the incurred cost as:

$$\begin{aligned} C_{att}(\text{NMDP}) &= \hat{c}_s + n\hat{c}_i + (n-1)\hat{c}_r \\ &= \hat{c}_s + (2f+1)\hat{c}_i + 2f\hat{c}_r \end{aligned}$$

  where $\hat{c}_r$ is the cost to know a metamorphic relation between two outputs.
  Under the very optimistic assumption that the attacker knows which is the program producing o_0, the attack cost reduces according to the expression:

$$C_{att}(\text{NMDP}) = \hat{c}_s + (f+1)\hat{c}_i + f\hat{c}_r$$

- since they share the same adjudicator's logic, also for NMFP, the best strategy for the attacker is to induce the adjudicator to end at the first phase. However, for this scheme, additional costs are needed to know the n functional specifications since, in this scheme, metamorphic diversity is applied at the level of functional

specifications. Therefore, the cost function is defined as:

$$\begin{aligned} C_{att}(\text{NMFP}) &= n\hat{c}_s + n\hat{c}_i + (n-1)\hat{c}_r \\ &= (2f+1)\hat{c}_s + (2f+1)\hat{c}_i + 2f\hat{c}_r \end{aligned}$$

where the meaning of the symbols is the same as for NMDP.
Also, in this case, under the very optimistic assumption that the attacker knows which is the program implementing the source functional specification g_0, the attack cost reduces according to the expression:

$$C_{att}(\text{NMFP}) = (2f+1)\hat{c}_s + (f+1)\hat{c}_i + f\hat{c}_r$$

As it is immediate to observe, independently of the values (all reasonably assumed greater than 0) assigned to the cost factors $\hat{c}_s$, $\hat{c}_i$ and $\hat{c}_r$, the relation existing between the cost needed to an attacker to compromise the three analyzed fault tolerance organizations is:

$$C_{att}(\text{NVP}) < C_{att}(\text{NMDP}) < C_{att}(\text{NMFP})$$

These relations are verified even under the very optimistic assumption for the attacker for which the cost expressions related to NMDP and NMFP are reduced (see above). This result sustains our claim that metamorphic diversity is more effective than NVP in facing intentional attacks. This is not surprising. In fact, regarding NMDP, the effectiveness of data diversity for intrusion tolerance was already investigated and demonstrated in [31]. Regarding NMFP, the metamorphic functional diversity raises the level at which diversity is addressed from implementation to functional specification, which is a recognized direction to amplify the impact of an action in a hierarchically structured organization (as, e.g., discussed in [27, 40]). The expected higher degree of diversity is beneficial in contrasting intentional attacks.

To provide numerical examples, Figure 6 shows the costs incurred by the attacker in basic configurations of the three schemes. They are calculated considering $\hat{c}_i$ as the most impacting factor in the cost function, and $\hat{c}_s$ and $\hat{c}_r$ related to $\hat{c}_i$ by a factor. In other words, in the figures $\hat{c}_i$ has been set equal to 1, thus $\hat{c}_s$ and $\hat{c}_r$ are measured in units of $\hat{c}_i$. Figure 6a shows how, at increasing of $\hat{c}_r$, $C_{att}(NMDP)$ increases with respect to $C_{att}(NVP)$ (that does not depend on $\hat{c}_r$): $C_{att}(NMDP)$ is about twice as expensive as $C_{att}(NVP)$ for $\hat{c}_r = 0.5$, and almost three times more expensive for $\hat{c}_r = 1$. Figure 6b indicates that $\hat{c}_s$ affects the change rate of $C_{att}(NMFP)/C_{att}(NVP)$, as the slope of the solid line differs from that of the dotted line.

Of course, the demonstrated higher cost for the attacker needs to be evaluated against the cost for the defender in employing the NMDP or NMFP schemes instead of NVP. Therefore, similar to the analysis of the cost for the attacker, a preliminary assessment of the cost for the defender is presented for the three schemes in the following section. In particular:

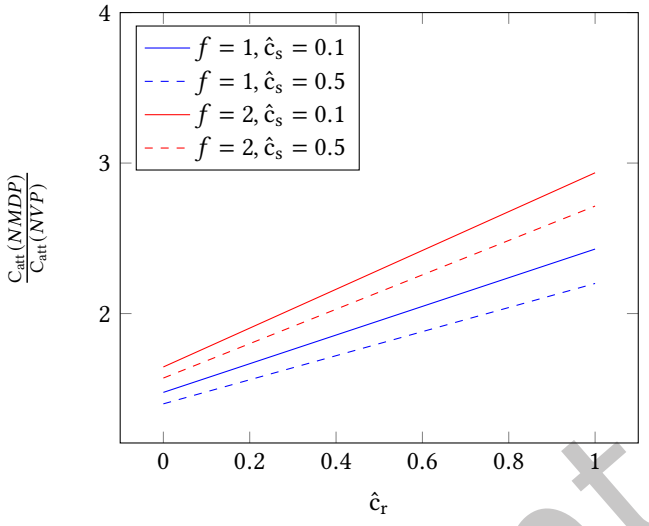
- the cost for the defender to set up an NVP configuration based on the (inexact) majority voting is given by the following expression:

$$C_{def}(\text{NVP}) = \bar{c}_s + (2f+1)\bar{c}_i$$

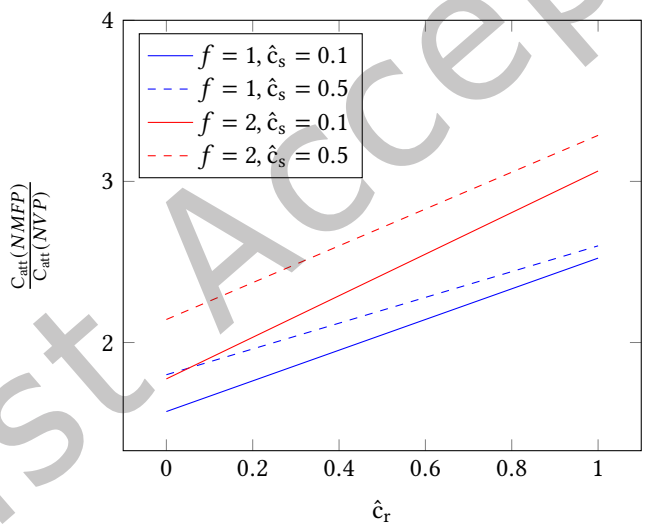
  where $\bar{c}_s$ is the cost to process a functional specification, and $\bar{c}_i$ is the cost to implement a program.
- the cost for the defender to set up an NMDP configuration based on the adjudicator's logic described in Section 6 is given by the following expression:

$$\begin{aligned} C_{def}(\text{NMDP}) &= \bar{c}_s + n\bar{c}_i + \frac{n(n-1)}{2}\bar{c}_r \\ &= \bar{c}_s + (2f+1)\bar{c}_i + (2f+1)f\bar{c}_r \end{aligned}$$

  where $\bar{c}_s$ is the cost to process a functional specification, $\bar{c}_i$ is the cost to implement a program, and $\bar{c}_r$ is the cost to generate metamorphic relations.



(a) Attack cost of NMDP over NVP



(b) Attack cost of NMFP over NVP

Fig. 6. Attack cost of the proposed NMDP and NMFP with respect to the attack cost of NVP. Blue lines indicate the relative cost for $f = 1$, red lines for $f = 2$.

- the cost for the defender to set up an NMFP configuration based on the adjudicator's logic described in Section 6 is given by the following expression:

$$\begin{aligned} C_{def}(\text{NMFP}) &= n\bar{c}_s + n\bar{c}_i + \frac{n(n-1)}{2}\bar{c}_r \\ &= (2f+1)\bar{c}_s + (2f+1)\bar{c}_i + (2f+1)f\bar{c}_r \end{aligned}$$

From the defender's perspective, the relation among costs is the same as for the case of the attacker, with NVP being the less expensive, followed by NMDP, and then NMFP. Again, this is unsurprising since higher protection is unavoidably paid by higher costs. To get an idea of the cost values to be afforded, a few examples are analyzed, based on the same configurations exercised to assess attacker's costs, with results shown in Figure 7. Figure 7 confirms that the key parameter that the defender has to fix is the number of tolerated attacks because f determines the rate of change of $C_{def}(NMDP)/C_{def}(NVP)$ and $C_{def}(NMFP)/C_{def}(NVP)$. If $\bar{c}_r$ is small, then $C_{def}(NMDP)$ and $C_{def}(NMFP)$ are close to $C_{def}(NVP)$; otherwise $C_{def}(NMDP)$ and $C_{def}(NMFP)$ can be considerably greater that $C_{def}(NVP)$, depending on f.

A more refined analysis that accounts for the probability of attack success through properly defined stochastic models is postponed as future work. Although more accurate results are expected, the trend just shown will not change. Actually, since the above-reported analysis is based on optimistic assumptions for the attackers, the effort estimates for inducing common-mode failure through attacks in more realistic conditions are expected to be higher than those shown in this paper.

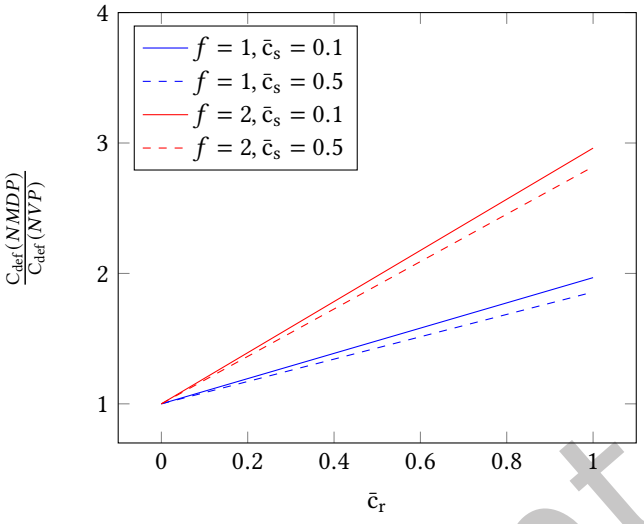
8 DISCUSSION ON REALISTIC APPLICATIONS OF THE PROPOSED METAMORPHIC-BASED DIVERSITY APPROACHES

We recall that the primary motivation for our study was to improve the effectiveness of redundancy-based fault tolerance in addressing common-mode failures, which are increasingly impacting systems due to the proliferation of deliberate attacks across various application contexts, including those critical to dependability. Similar to the study in [48], our approach aims to achieve the planned goal in a general context, making it applicable wherever fault tolerance through redundancy is required or desired. Therefore, rather than staying restricted within a specific targeted application, we presented the proposed ideas and architectures at a logical level. To convey the concepts behind our proposals, we included simple examples commonly found in the metamorphic testing literature, aiming at making them understandable to a broad range of readers. While we postpone deeper investigations of real application contexts as future work, in this section we discuss the instantiation of the proposed redundancy approaches in selected realistic scenarios.

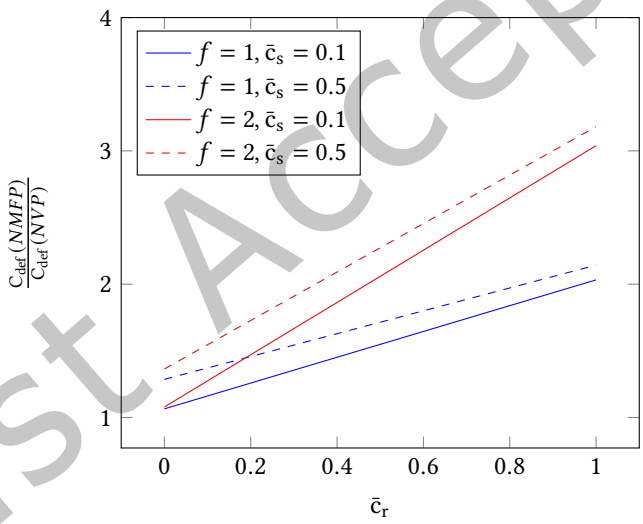
A relevant domain where metamorphic testing has been successfully applied is that of Cyber-Physical Systems (CPS), which are complex systems that integrate both physical and software components and are widely represented in many application contexts, including critical ones such as railways, smart grids, and nuclear power plants, among others. As discussed in [8], in the CPS domain, metamorphic testing has been applied for testing wireless sensor networks, or autonomous drones, or self-driving cars, and has stimulated research on automatic generation of effective MRs. In general, when the structure and behavior of the system under development are characterized by a model that includes well-established laws or general principles (such as the power flow equilibrium in the electrical sector, or the law of the conservation of mass in the chemical sector), as is rather common in CPS, metamorphic relations can be established based on those principles. Obviously, this does not preclude that other application domains could be of interest, especially in view of the active research in MT and automated derivation of MRs, which can be fruitful also to our proposed approach. Here, we take CPS as the reference domain for outlining two examples of definitions of the MRs needed to instantiate the two redundancy schemes NMDP and NMFP, respectively, for the case of tolerance of one fault/attack.

8.1 Elevators Traffic Manager

A practical example of MDDR in the context of CPS deals with the traffic manager of a system of elevators in charge of managing the passengers' flow, developed in [7], where it has been adopted for MT purposes. Among the components of this traffic manager, a critical one is the dispatching algorithm, namely the software module that decides which elevator should satisfy each call. The implementation of the elevators' dispatching algorithm



(a) Defense cost of NMDP over NVP



(b) Defense cost of NMFP over NVP

Fig. 7. Defense cost of the proposed NMDP and NMFP with respect to the defense cost of NVP. Blue lines indicate the relative cost for $f = 1$, red lines for $f = 2$.

is highly complex and strongly depends on the types of elevator installations as well as the building requirements that it needs to satisfy. Although in general the criticality level of such an application could not be so high as to deserve fault tolerance techniques, there are contexts (such as hospitals) where it makes sense. Therefore, we adapted the approach in [7] to our context of MDDR. In [7], some QoS metrics are defined to detect faulty

behavior or incorrect dispatching of this algorithm. Among these QoS metrics, there is the Total Movements (TM), which is the sum of all movements (i.e., engine start-ups) of all elevators in the building. This metric may reveal inefficient dispatching or buggy behaviours of the dispatching algorithm when comparing the results obtained from processing different passengers' calls.

To leverage the benefits of this metric in the fault tolerance context and take advantage of the metamorphic relations already defined, we have to extend the dispatching algorithm by making the TM relative to each execution available as an output, which is provided to the adjudicator component together with the assignments of elevators to each call. Then, the same MRs defined in [7] on TM can be used for an NMDP instantiation that replicates the dispatching algorithm to tolerate one fault.

In other words, given an initial configuration of elevator positions and an initial set of elevator calls, the dispatching algorithm determines the assignment of elevators to calls and, as part of its output, computes the TM required to satisfy the specified calls. During the execution window of the dispatching algorithm, additional elevator calls may be generated beyond those present in the initial set.

The TM serves as a quantitative metric for evaluating the correctness of the dispatching algorithm. The fundamental objective of TM assessment is to verify the consistency between the initial TM—computed with respect to the initial set of calls—and the resulting TM, which is computed after incorporating additional calls issued during the execution window.

Within our NMDP instantiation, a set of metamorphic relations is defined to formally assess this consistency between the initial and resulting TM values. Any violation of these relations may indicate a potential defect or anomalous behavior in the dispatching algorithm.

Specifically, we suppose that:

- User(s) can make elevator call(s) from an elevator lobby on a floor, called *call point*.
- In the geographical area under the control of the considered elevator dispatching algorithm, there are several *call points*. The map of *call points* and the number of elevators serving each *call point* is fixed and does not change during the stated operation window.
- An *elevator call* is defined by means of a tuple that expresses the *call point*, and the floor that the passenger wants to reach.
- An *elevator position* is defined by means of a tuple that expresses the elevator identifier and the floor on which this elevator is located in the geographical area under the control of the dispatching function.
- An *elevator assignment* is defined by means of a topological structure that includes, among other information, the *elevator calls*, and for each *elevator call* taken as input, the elevator assigned to that *elevator call*.
- TM is defined as the total number of movements (i.e., engine start-ups) of all elevators involved in an execution of the dispatching algorithm.

Let us define:

- $dispatch : T^{in} \times T^{e} \rightarrow T^{out} \times \mathbb{N}$ is the dispatching function, where T^{in} is the set of *elevator call(s)*, T^{e} is the set of *elevators position(s)* and T^{out} is the set of *elevator assignment(s)*.
  Then, the output of *dispatch* is a tuple where the first element represents a set of *elevator assignment(s)* and the second represents the TM needed to obtain that set of *elevator assignment(s)*.
- C_k is the number of additional calls from the *call point(s)*;
- $TMworst(C_k)$ is the number of movements in the worst case for executing the additional C_k calls, namely, $TMworst$ is always 2 movements per call in the worst case, one to reach the call floor and the other to get to the destination.

- The function $g : T^{in} \rightarrow \mathbb{N}$ is defined as

$$\begin{aligned} \left(t_{out}, TM_j\right) &= dispatch(i_j, t_e) \\ g(i_j) &= TM_j \end{aligned}$$

  where i_j is the input of the *dispatch* function, being g a sub-function of *dispatch* in charge of computing TM.

Given the function *dispatch*, $t_e \in T^e$ and a source input ($i \in T^{in}$), an MDDR can be composed of the following metamorphic relations:

- a relation R_1 (e.g., $TM_1 - TM_0 \geq 0$) is defined on the inputs i, i_1 as well as the outputs $o_0 = TM_0$ and $o_1 = TM_1$, where $i_1 = i \cup t_l^{in}$ is a follow-up input derived from $i \in T_i^{in}$ according to R_1, adding t_l^{in} calls.
- a relation R_2 (e.g., $TM_2 - TM_0 \leq TMworst(C_k)$) is defined on the inputs i, i_2 as well as the outputs o_0=TM_0 and o_2=TM_2, where $i_2 = i \cup t_r^{in}$ is a follow-up input derived from i according to R_2, adding t_r^{in} calls.

The two MRs defined above allow for setting an instantiation of the NMDP architecture consisting of three versions of the dispatching algorithm to enhance its robustness against the occurrence of a fault/attack. The source input and the two follow-up inputs, generated according to the two MRs, respectively, are given as input to the three redundant versions, according to the logical scheme in Figure 2. The three outputs obtained are managed by the adjudicator component in a manner similar to the example detailed in Table 1. Note that, in case the version processing the source input is deemed incorrect, the expected correct output can be reconstructed from any of the two other outputs, since the adjudicator knows the elevator calls included in the source input, which were also included in the two follow-up inputs.

8.2 Nuclear Reactor Safety System

As a realistic example of MFDR, in this subsection an integrated monitoring system that calculates different critical quantities for reactor safety according to nuclear physics principles is presented.

Very briefly, in simplified but realistic terms, the key reactor components are [23]:

- Reactor core: contains fuel assemblies with fissile material (Uranium 235) where nuclear fission occurs. The neutron flux (ϕ) represents the intensity of neutron bombardment causing fission reactions.
- Control rods: made of neutron-absorbing materials (e.g., boron, cadmium). When inserted into the core, they reduce the neutron flux and control the reaction rate. Rod position ($\rho_{\text{rod}} \in [0, 1]$) determines their effectiveness.
- Coolant system: circulates water through the reactor core to remove heat generated by fission. Primary coolant temperature (T_{coolant}) and pressure (P_{primary}) are critical operating parameters.
- Safety systems: multiple independent systems designed to shut down the reactor and maintain cooling during emergencies.

The input of the safety system is the tuple of measurements $i = (\phi, T_{\text{coolant}}, P_{\text{primary}}, \rho_{\text{rod}}) \in \mathbb{R}_+^4$. The source function g_0 and the follow-up functions g_1 and g_2, needed to tolerate one fault, are:

- $g_0 : \mathbb{R}_+^4 \rightarrow \mathbb{R}$ evaluates [22] the safety margin to critical power:

$$\text{safety\_margin} = g_0(i) = 1 - \frac{\phi \cdot \sigma_f \cdot N_{\text{fuel}}}{P_{\text{max}}}$$

  where:
  - $\sigma_f = 585 \times 10^{-24}$ [cm^2] is the fission cross-section for Uranium 235
  - $N_{\text{fuel}} = 2.5 \times 10^{22}$ [nuclei/cm^3] is the fuel density
  - $P_{\text{max}} = 3000$ [MW] is the maximum thermal power of the reactor

safety_margin indicates how far the reactor operates from its maximum safe power limits: 1 means the reactor is completely shut down (zero power), 0.8 means the reactor is operating at 20% of maximum power (normal operation), 0.2 means the reactor is operating at 80% of maximum power (approaching limits), less than 0 means the reactor is exceeding safe operating limits. Thus, safety_margin is the primary indicator for triggering emergency procedures when approaching dangerous power levels;

- the emergency shutdown, called Safety Control Rod Axe Man (SCRAM), is characterized by two time values in [s]: t_{scram} is the time required for control rods to fully insert into the reactor core; t_{cooling} is the time needed to remove residual heat from the reactor core after shutdown. These values are computed through $g_1 : \mathbb{R}^4_+ \to \mathbb{R}^2$:

$$t_{\text{scram}} = \alpha \cdot e^{-\beta \cdot \rho_{\text{rod}}} \cdot \left(\frac{\phi}{\phi_{\text{nominal}}} \right)^{-0.3}$$
$$t_{\text{cooling}} = \frac{m_{\text{thermal}} \cdot c_p}{h_{\text{exchange}} \cdot \Delta T_{\max}}$$
$$g_1(i) = (t_{\text{scram}}, t_{\text{cooling}})$$

  where:
  - $\alpha = 10.0$ [s], $\beta = 0.5$ are calibration constants for SCRAM
  - $\phi_{\text{nominal}} = 3.0 \times 10^{13}$ [neutron/(cm$^2 \cdot s$)] is the nominal flux
  - $m_{\text{thermal}} = 1.2 \times 10^6$ [kg] is the thermal mass of the core
  - $c_p = 4186$ [$J/(\text{kg} \cdot K)$] is the specific heat of water
  - $h_{\text{exchange}} = 5000$ [$W/(m^2 \cdot K)$] is the heat exchange coefficient
  - $\Delta T_{\max} = 50$ [K] is the maximum allowable temperature gradient

  based on delayed neutron kinetics and thermal balances [23];
- $g_2 : \mathbb{R}^4_+ \to \{\text{LOW}, \text{MEDIUM}, \text{HIGH}, \text{CRITICAL}\}$ evaluates the categorical alert level:

$$\text{score} = w_1 \frac{\phi}{\phi_{\max}} + w_2 \frac{T_{\text{coolant}}}{T_{\max}} + w_3 \frac{P_{\text{primary}}}{P_{\max}}$$

  where:
  - $w_1 = 0.5$, $w_2 = 0.3$, $w_3 = 0.2$ are the alert system weights
  - $\phi_{\max} = 5.0 \times 10^{13}$ [neutron/(cm$^2 \cdot s$)] is the maximum allowable flux
  - $T_{\max} = 320$ [C] is the maximum coolant temperature
  - $P_{\max} = 155$ [bar] is the maximum primary circuit pressure

$$g_2(i) = \begin{cases} \text{LOW} & \text{if score} < 0.3 \\ \text{MEDIUM} & \text{if } 0.3 \leq \text{score} < 0.6 \\ \text{HIGH} & \text{if } 0.6 \leq \text{score} < 0.8 \\ \text{CRITICAL} & \text{if score} \geq 0.8 \end{cases}$$

The metamorphic relations are:

- $R_1(i, o^0, o^1)$ deals with the consistency between margin and scram[6]

$$\text{if safety\_margin} < 0.2 \Rightarrow t_{\text{scram}} < 5.0 \text{ [s]}$$

[6]Based on reactor control theory, see chapter 7 of [23]

- $R_2(i, o^0, o^2)$ deals with the consistency between margin and alert[7]

$$(\text{safety\_margin} < 0.3 \wedge \text{alert\_level} \in \{\text{HIGH}, \text{CRITICAL}\}) \vee (\text{safety\_margin} > 0.7 \wedge \text{alert\_level} \in \{\text{LOW}, \text{MEDIUM}\})$$

- $R_3(i, o^1, o^2)$ deals with the consistency between time and alert[8]

$$\text{if alert\_level} = \text{CRITICAL} \Rightarrow (t_{\text{scram}} < 2.0 \wedge t_{\text{cooling}} < 300)$$
$$\text{if alert\_level} = \text{LOW} \Rightarrow t_{\text{scram}} > 10.0$$

The MRs defined above between the source function g_0 and the follow-up functions g_1 and g_2 allow the setting of an instantiation of the NMFP architecture consisting of three programs, each implementing a different specification of the nuclear reactor monitoring system, to enhance its robustness against the occurrence of a fault/attack, following the logical scheme in Figure 3. The three outputs obtained are processed by the adjudicator component, similar to the example detailed in Table 1. Due to the inherently unequal nature of the stated metamorphic relations, in case the program implementing the source function is adjudged as incorrect, the recalculation of the expected correct value for it may present degraded accuracy. While, on the one hand, reasonable tolerances to specified degrees of inaccuracy can be expected to be accepted, on the other hand, the availability of the additional R_3 metamorphic relation with respect to the two ones strictly needed to tolerate 1 fault, offers the possibility of also strengthening the adjudication operations for this critical scenario.

9 CONCLUSIONS AND FUTURE WORK

We have introduced a novel notion of redundancy-based fault/intrusion tolerance, by which we propose to use MRs to enhance diversity across redundant executions. In particular, we have defined two MRs patterns leveraging data diversity and function diversity, and based on them, we have then presented the logical scheme of the two architectures NMDP and NMFP that can be adopted with the aim to reduce common-mode failures. We have also discussed in detail the logic behind the adjudicator to be included within these two logical architectures, and have developed two realistic examples for the two redundancy schemes NMDP and NMFP in the domain of Cyber-Physical Systems.

The goal of this paper is inherently foundational: we aimed to introduce the logical framework of MRs based fault tolerance by providing a rigorous definition and characterizing its possible instantiation in two logical architectures. Examples of applications are also included, as well as a preliminary evaluation of the benefits deriving from metamorphic diversity in contrasting intentional attacks, quantified in terms of the higher effort the attacker needs to compromise the newly proposed NMDP and NMFP architectures with respect to the classical NVP architecture based on functionally equivalent programs, given a maximum number of attacks/faults to tolerate. Overall, the achieved results so far are promising and position the proposed architectures as superior alternatives for fault/intrusion tolerance in place of traditional NVP-like solutions based on design diversity to enhance system protection, particularly when addressing deliberate attacks. Of course, the applicability depends on whether the requested multiplicity of metamorphic data or function diversity relations can be found for the specific application context under development. However, a relevant positive side effect of the metamorphic functional diversity is that the identified set of metamorphically related functions can satisfy the needs of all the applications where any of the functions in the set is of interest and must be fault-tolerant. For example, the set of trigonometric functions sin, cos, tan, csc, among which proper metamorphic relations can be identified, is built once and then adopted to satisfy the tolerance needs of any of the four included functions. This "reuse" property brings a considerable advantage from the development point of view.

[7]Standard nuclear safety criterion, see appendix A of [23]
[8]Temporal requirements for nuclear safety systems, see IEEE-279 standard.

Due to its foundational scope, this work paves the way for several directions for future research. Pursuing a more detailed assessment of the benefits brought by metamorphic diversity by refining the preliminary analysis conducted in Section 7 is the first immediate extension. Moreover, in this initial study, the classical NVP fault tolerant architecture (depicted in Figure 1) has been considered as the reference scheme to be evolved by introducing the metamorphic relations concept. However, the metamorphic diversity practice can be applied as well to alternative organizations of redundant programs proposed in the literature, which adopt a comparison-like adjudication function. Among them, N Self-checking Programming (NSCP) [24] and Self-Configuring Optimistic Programming (SCOP) [11] are indicated as good representative examples. The former tries to combine advantageous features of NVP and RB, while the latter resembles NVP but is structured in more execution phases to improve efficiency. The precise structure of metamorphic-based alternatives of NSCP and SCOP is postponed as future work, though it appears to be a rather straightforward development.

A more challenging direction would be to build hybrid solutions that combine MDDR e MFDR within a single scheme. The expected benefits are twofold. A first advantage grounds on the consideration that the number of metamorphic relations required for assuring either data diversity or functional diversity can become too onerous as the desired tolerance level increases (for the adjudicator presented in Section 6, the formula that relates r with n is defined); depending on the specific application at hand, it could be even impossible to obtain the needed set. Covering the targeted diversity partly by data diversity and partly by functional diversity seems to be a viable solution to overcome such a problem and to obtain a suitable/affordable balance between the two types of relations. The second foreseen advantage concerns the stronger degree of diversity so achievable through mixing data and functional diversity, again especially desirable for intrusion tolerance purposes since the attacker is required to be able to gain knowledge on both MDDR and MFDR relations (thus increasing the cost of compromising the fault tolerant system, determined through an extension of the cost functions defined in Section 7).

Another challenging task for applying the proposed fault/intrusion tolerance schemes is the automated identification of the MRs. In the illustrative examples of this paper, they have been manually derived, leveraging the domain knowledge, but manual derivation is hardly viable when the needed MRs is high, in accordance with the faults/intrusions to be tolerated and the adjudicator logic. However, given the affinity with the usage of MRs in MT, which in recent years has proven to be an effective strategy to reveal errors in programs in many domains, we expect that the software engineering community in the large will be increasingly interested in exploring and exploiting MRs wherever they bring advantages, including the derivation of data and functional diversity for fault/intrusion purposes, as proposed in this paper. In fact, MT has been applied for testing of real-world software systems such as Google and Yahoo search engines [49]. Recently, MT has been adopted at Facebook for industrial-scale deployment of web-enabled simulation [1]. The growing interest for MT in these real-world and industrial environments has triggered interesting studies to identify MRs automatically, mainly in specific domains such as numerical programs [47] or query-based systems [36]. Some approaches leverage machine learning methods [20] and more recently even LLMs [39], genetic programming [8], or information derived from inputs and outputs domains [41] for automatically generating MRs. In the future, we plan to investigate the applicability of these existing approaches in deriving the metamorphic relations required by the proposed redundancy schemes, in addition to devising new methods specifically tailored for achieving metamorphic diversity in the proposed architectures. A more ambitious goal would be the development of general guidelines for systematically identifying or deriving follow-up specifications starting from the metamorphic relations.

Finally, although our current examples are effective in illustrating the concepts behind our proposals and provide straightforward scenarios, we intend to identify and select more complex software systems in real application contexts to implement and evaluate the proposed architectures in future work.

REFERENCES

[1] John Ahlgren, Maria Berezin, Kinga Bojarczuk, Elena Dulskyte, Inna Dvortsova, Johann George, Natalija Gucevska, Mark Harman, Maria Lomeli, Erik Meijer, et al. 2021. Testing web enabled simulation at scale using metamorphic testing. In *Proc. of 43rd International Conference on Software Engineering: Software Engineering in Practice*. 140–149.

[2] AM Amendola, L Impagliazzo, P Marmo, G Mongardi, G Sartore, and Ansaldo Trasporti. 1996. Architecture and safety requirements of the ACC railway interlocking system. In *Proc. of IEEE International Computer Performance and Dependability Symposium*. IEEE, 21–29.

[3] Paul Eric Ammann and John C Knight. 1988. Data diversity: an approach to software fault tolerance. *IEEE Trans. Comput.* 37, 4 (1988), 418–425. https://doi.org/10.1109/12.2185

[4] Algirdas Avizienis. 1985. The N-Version Approach to Fault-Tolerant Software. *IEEE Transactions on Software Engineering* 11, 12 (1985), 1491–1501.

[5] Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing* 1, 1 (2004), 11–33.

[6] Algirdas Avizienis, Michael R Lyu, and Werner Schutz. 1988. In search of effective diversity: a six-language study of fault-tolerant flight control software. In *Proc. of 18th Int. Symposium on Fault-Tolerant Computing*. 15–22.

[7] Jon Ayerdi, Sergio Segura, Aitor Arrieta, Goiuria Sagardui, and Maite Arratibel. 2020. Qos-aware metamorphic testing: An elevation case study. In *Proc. of IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. 104–114.

[8] Jon Ayerdi, Valerio Terragni, Aitor Arrieta, Paolo Tonella, Goiuria Sagardui, and Maite Arratibel. 2021. Generating metamorphic relations for cyber-physical systems with genetic programming: an industrial case study. In *Proc. of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1264–1274.

[9] Jon Ayerdi, Valerio Terragni, Gunel Jahangirova, Aitor Arrieta, and Paolo Tonella. 2024. Genmorph: Automatically generating metamorphic relations via genetic programming. *IEEE Transactions on Software Engineering* (2024).

[10] Benoit Baudry and Martin Monperrus. 2015. The multiple facets of software diversity: Recent developments in year 2000 and beyond. *ACM Computing Surveys (CSUR)* 48, 1 (2015), 1–26.

[11] Andrea Bondavalli, Felicita Di Giandomenico, and Jie Xu. 1993. A Cost-Effective and Flexible Scheme for Software Fault Tolerance. *Journal of Computer Systems Science and Engineering* 8 (1993), 234–244.

[12] Nazanin Bayati Chaleshtari, Fabrizio Pastore, and Yoann Marquer. 2025. *Catalog of Metamorphic Relations for MST*. https://github.com/MetamorphicSecurityTesting/CWE

[13] Tsong Yueh Chen, Shing C Cheung, and Shiu Ming Yiu. 2020. Metamorphic Testing: A New Approach for Generating Next Test Cases. *CoRR* abs/2002.12543 (2020). https://arxiv.org/abs/2002.12543

[14] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, TH Tse, and Zhi Quan Zhou. 2018. Metamorphic Testing: A Review of Challenges and Opportunities. *ACM Compututing Surveys* 51, 1 (2018).

[15] Byron DeVries and Erik M Fredericks. 2024. Triggering Adaptation via Contextual Metamorphic Relations. In *Proc. of IEEE 24th International Conference on Software Quality, Reliability and Security (QRS)*. 105–114.

[16] Guowei Dong, Baowen Xu, Lin Chen, Changhai Nie, and Lulu Wang. 2008. Case studies on testing with compositional metamorphic relations. *Journal of Southeast University (English Edition)* 24, 4 (2008), 437–443.

[17] Anurag Dwarakanath, Manish Ahuja, Samarth Sikand, Raghotham M Rao, RP Jagadeesh Chandra Bose, Neville Dubash, and Sanjay Podder. 2018. Identifying Implementation Bugs in Machine Learning Based Image Classifiers Using Metamorphic Testing. In *Proc. of 27th Int. Symposium on Software Testing and Analysis*. 118–128.

[18] Ilir Gashi, Andrey Povyakalo, and Lorenzo Strigini. 2016. Diversity, Safety and Security in Embedded Systems: Modelling Adversary Effort and Supply Chain Risks. In *Proc. of 12th European Dependable Computing Conference*. 13–24.

[19] Upulee Kanewala and James M Bieman. 2013. Using machine learning techniques to detect metamorphic relations for programs without test oracles. In *Proc. of IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. 1–10.

[20] Upulee Kanewala, James M Bieman, and Asa Ben-Hur. 2016. Predicting metamorphic relations for testing scientific software: a machine learning approach using graph kernels. *Software testing, verification and reliability* 26, 3 (2016), 245–269.

[21] John C Knight. 2011. Diversity. In *Dependable and historic computing*. Springer, 298–312.

[22] J.R. Lamarsh. 1966. *Introduction to Nuclear Reactor Theory*. Addison-Wesley, Reading, MA.

[23] J.R. Lamarsh and A.J. Baratta. 2001. *Introduction to Nuclear Engineering* (3rd ed.). Prentice-Hall, Upper Saddle River, NJ.

[24] Jean-Claude Laprie, Jean Arlat, Christian Beounes, and Karama Kanoun. 1990. Definition and analysis of hardware- and software-fault-tolerant architectures. *Computer* 23, 7 (1990), 39–51.

[25] Gholamreza Latif-Shabgahi, Julian M Bass, and Stuart Bennett. 2004. A taxonomy for software voting algorithms used in safety-critical systems. *IEEE Transactions on Reliability* 53, 3 (2004), 319–328.

[26] Rui Li, Huai Liu, Pak-Lok Poon, Dave Towey, Chang-Ai Sun, Zheng Zheng, Zhi Quan Zhou, and Tsong Yueh Chen. 2025. Metamorphic Relation Generation: State of the Art and Research Directions. *ACM Transactions on Software Engineering and Methodology* 34, 5 (2025), 1–25.

[27] Bev Littlewood and Lorenzo Strigini. 2000. *A discussion of practices for enhancing diversity in software designs.* Technical Report DISPO. Centre for Software Reliability, City University.
[28] Bev Littlewood and Lorenzo Strigini. 2004. Redundancy and Diversity in Security. In *Proc. of 9th European Symposium on Research Computer Security*. 423–438.
[29] Huai Liu, Fei-Ching Kuo, Dave Towey, and Tsong Yueh Chen. 2013. How effectively does metamorphic testing alleviate the oracle problem? *IEEE Transactions on Software Engineering* 40, 1 (2013), 4–22.
[30] Huai Liu, Iman I Yusuf, Heinz W Schmidt, and Tsong Yueh Chen. 2014. Metamorphic Fault Tolerance: An Automated and Systematic Methodology for Fault Tolerance in the Absence of Test Oracle. In *Companion Proc. of the 36th International Conference on Software Engineering*. 420–423.
[31] Anh Nguyen-Tuong, David Evans, John C. Knight, Benjamin Cox, and Jack W. Davidson. 2008. Security through redundant data diversity. In *Proc. of International Conference on Dependable Systems and Networks*. 187–196. https://doi.org/10.1109/DSN.2008.4630087
[32] Brian Randell. 1975. System structure for software fault tolerance. *IEEE Transactions on Software Engineering* 1, 2 (1975), 220–232.
[33] Brian Randell and Jie Xu. 1994. *The evolution of the recovery block concept.* Vol. 3. John Wiley & Sons Ltd, Chichester, 1–22.
[34] Walter Rudin et al. 1976. *Principles of mathematical analysis.* Vol. 3.
[35] Kizito Salako and Lorenzo Strigini. 2013. When Does" Diversity"'in Development Reduce Common Failures? Insights from Probabilistic Modeling. *IEEE Transactions on dependable and secure computing* 11, 2 (2013), 193–206.
[36] Sergio Segura, Juan C Alonso, Alberto Martín López, Amador Durán Toro, Javier Troya, and Antonio Ruiz Cortés. 2022. Automated Generation of Metamorphic Relations for Query-Based Systems. In *Proc. of 7th Int. Workshop on Metamorphic Testing*. 48–55.
[37] Sergio Segura, Gordon Fraser, Ana B Sanchez, and Antonio Ruiz-Cortés. 2016. A Survey on Metamorphic Testing. *IEEE Transactions on Software Engineering* 42, 9 (2016), 805–824.
[38] Sergio Segura, Dave Towey, Zhi Quan Zhou, and Tsong Yueh Chen. 2018. Metamorphic testing: Testing the untestable. *IEEE Software* 37, 3 (2018), 46–53.
[39] Seung Yeob Shin, Fabrizio Pastore, Domenico Bianculli, and Alexandra Baicoianu. 2024. Towards generating executable metamorphic relations using large language models. In *Proc. of International Conference on the Quality of Information and Communications Technology*. Springer, 126–141.
[40] Lorenzo Strigini and Felicita Di Giandomenico. 1991. Flexible Schemes for Application-Level Fault Tolerance. In *Proc. of Tenth Symposium on Reliable Distributed Systems, SRDS*. IEEE Computer Society, 86–95.
[41] Chang-Ai Sun, An Fu, Pak-Lok Poon, Xiaoyuan Xie, Huai Liu, and Tsong Yueh Chen. 2019. METRIC $^{+}$: A Metamorphic Relation Identification Technique Based on Input Plus Output Domains. *IEEE Transactions on Software Engineering* 47, 9 (2019), 1764–1785.
[42] Pascal Traverse, Isabelle Lacaze, and Jean Souyris. 2004. Airbus Fly-By-Wire: A Total Approach To Dependability. In *Building the Information Society*, Jacquart, Renè (Ed.). Springer US, Boston, MA, 191–212.
[43] Paulo Esteves Veríssimo, Nuno Ferreira Neves, and Miguel Pupo Correia. 2003. Intrusion-tolerant architectures: Concepts and design. In *Architecting dependable systems*. Springer, 3–36.
[44] J. Von Neumann. 1956. *Probabilistic Logics.* Vol. 34. Priceton University Press.
[45] Richard Thomas Wood, Randy Belles, Mustafa Sacit Cetiner, David Eugene Holcomb, Kofi Korsah, Andy Loebl, Gary T Mays, Michael David Muhlheim, James Allen Mullens, Willis P Poore III, et al. 2010. *Diversity strategies for nuclear power plant instrumentation and control systems.* Technical Report. Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States).
[46] Ying C Yeh. 1996. Triple-triple redundant 777 primary flight computer. In *Proc. of IEEE Aerospace Applications Conference*, Vol. 1. 293–307 vol.1.
[47] Bo Zhang, Hongyu Zhang, Junjie Chen, Dan Hao, and Pablo Moscato. 2019. Automatic discovery and cleansing of numerical metamorphic relations. In *Proc. of IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 235–245.
[48] Zheng Zheng, Daixu Ren, Huai Liu, and Tsong Yueh Chen. 2024. Metamorphic Fault Tolerance: Addressing the Oracle Problem of Reliability Assurance for Contemporary Software Systems. *Computer* 57, 7 (2024), 77–86.
[49] Zhi Quan Zhou, ShuJia Zhang, Markus Hagenbuchner, TH Tse, Fei-Ching Kuo, and Tsong Yueh Chen. 2012. Automated functional testing of online search services. *Software Testing, Verification and Reliability* 22, 4 (2012), 221–243.