Formal Verification of the Correctness and Soundness of a Pomset-to-LTS Transformation Algorithm

Asma Bezza, Rohallah Benaboud, Toufik-Messaoud Maarouk, and Rabea Ameur-Boulifa

Original scientific article

***Abstract*—In this paper, we present a comprehensive proof of the correctness and soundness of our previously published algorithm for transforming partially ordered multisets (Pomsets) into labeled transition systems (LTS). Our approach rigorously ensures that the transformation algorithm preserves the behavioral semantics of the original Pomset, ensuring that the resulting LTS accurately represents the concurrent and sequential dependencies inherent in the Pomset. We employ Hoare logic to formally verify the correctness of the algorithm, proving that every valid Pomset is transformed into a corresponding LTS without loss of information. Additionally, we provide detailed proofs of soundness, showing that the algorithm produces an LTS if and only if the input is a valid Pomset. This algorithm was initially proposed as part of our new refinement proof approach, which has already been published. However, the correctness of the algorithm had not been formally proven until now. These results confirm the reliability and robustness of our transformation algorithm, making it a valuable tool for modeling and analyzing concurrent systems.**

***Index Terms*—Formal verification, Correctness, soundness, Pomset, LTS, Hoare-logic.**

I. Introduction

Software is a crucial aspect of nearly all aspects of life. Today, dependency on software programs remains increasing, especially in important domains, wherein harm to human health, effects on the surroundings, or giant financial results can be because of malfunctioning software programs and systems. The ubiquitous presence of software leads to the requirement of being able to prove its correctness. Correctness is an important factor in software engineering. Particularly in embedded systems, ensuring the implementation satisfies the required specification is crucial. Delivering the program together with a formal proof of its correctness with respect to some specification guarantees a correct program, whereas tests can only reduce errors.

Among the various formal verification techniques available, Hoare's logic, introduced in 1969, stands out as a powerful formalism for reasoning about the correctness of computer programs [1] [2].

The essence of Hoare logic lies in its ability to provide a mathematical framework for proving the correctness, soundness, and completeness of algorithms—ensuring that a program behaves as intended. By establishing preconditions and postconditions, developers can delineate the expected state of computation before and after a program's execution. Preconditions serve as a contract, stipulating the requirements that must be met before a function or a procedure is invoked. On the other hand, postconditions define the state of the system after the execution, essentially describing the effects of the computation or the changes brought about by the program segment. The successful application of Hoare logic not only validates the algorithm but also demonstrates the practical utility of formal methods in software development.

The research we present here provides full proof that our algorithm for turning partially ordered multisets (Pomsets) into labeled transition systems (LTS) is correct and sound. Initially proposed as part of our novel refinement proof approach [3].

A Pomset is a mathematical concept utilized for representing systems when certain events occur sequentially while others may occur independently or concurrently. A Pomset facilitates partial ordering of events, distinguishing it from a basic sequence and rendering it suitable for representing concurrency. Originally theoretical, Pomsets have practical uses in distributed systems, concurrent programming, workflow management, and formal verification, where comprehending dependencies and potential parallelism between tasks is crucial.

To illustrate the applicability of our approach, consider a distributed workflow system for handling online orders. The process involves multiple tasks such as payment verification, stock checking, packaging, invoice printing, and shipping. Many of these tasks have natural dependencies (e.g., packaging must wait for payment and stock validation), while others can be performed concurrently (e.g., stock checking and invoice printing). Modeling such a system using Pomsets allows us to precisely capture these partial orders and concurrency relations. This enables formal reasoning about correctness

Manuscript received March 3, 2025; revised April 17, 2025. Date of publication October 20, 2025. Date of current version October 20, 2025.

A. Bezza is with the Department of Mathematics and Computer Science, University of Oum El Bouaghi 04000, Algeria and Department of Mathematics and Computer Science, ICOSI Lab, University Abbes Laghrour Khenchela, BP 1252 EL Houria, Algeria (e-mail: bezza.asma@univ-khenchela.dz).

R. Benaboud is with the Research Laboratory on Computer Science's Complex Systems (RelaCS2), Department of Mathematics and Computer Science, University of Oum El Bouaghi 04000, Algeria (e-mail: benaboud.rohallah@univ-oeb.dz).

T.-M. Maarouk is with the Department of Mathematics and Computer Science, ICOSI Lab, University Abbes Laghrour Khenchela, BP 1252 EL Houria, Algeria (e-mail: maarouk.toufik@univ-khenchela.dz).

R. Ameur-Boulifa is with the LTCI, Télécom Paris, Institut Polytechnique de Paris, France (e-mail: rabea.ameur-boulifa@telecom-paristech.fr).

Digital Object Identifier (DOI): 10.24138/jcomss-2025-0028

properties such as deadlock-freedom, causal consistency, and potential optimizations in task scheduling.

Let us consider a real-world example of a workflow in an e-commerce company:

Process objective

Managing a customer order.

Process steps

1) Verify payment (A)
2) Check item availability (B)
3) Prepare packaging (C)
4) Print invoice (D)
5) Ship the order (E)

Constraints

Tasks A (payment) and B (stock checking) can be performed in parallel.
Task C (packaging) can only start after both payment and stock are validated $\rightarrow$ it depends on A and B.
Task D (invoice printing) can be done as soon as A is completed.
Task E (shipping) depends on both C and D.

Modeling with a Pomset

In this context, each task (A–E) is a labeled event.
The partial order relation between them is: $A \preceq C$, $B \preceq C$, $A \preceq D$, $C \preceq E$, and $D \preceq E$.
However, A and B can be executed in parallel, as can C and D, as long as their respective dependencies are respected. This Pomset captures the true concurrent behavior of the system :not a strict linear sequence (like a trace), but a set of partially ordered events representing causal dependencies.

This modeling approach enables a formal analysis of the process's correctness. For instance, it allows the detection of missing or incorrect dependencies and the verification of properties such as the absence of deadlocks or race conditions. Moreover, Pomsets can be used to compare different versions of a workflow: if a modification is introduced (e.g., postponing task D to occur after task C), the resulting behavioral equivalence or divergence can be formally assessed. Additionally, this representation facilitates execution optimization by identifying independent tasks that can be performed in parallel, thereby improving overall system efficiency.

The strength of Pomsets lies in their power of describing events that are not completely linear but partially ordered. This reflects more the reality of systems where activities can occur in a concurrent way. Whereas the formal verification of Pomsets poses several challenges due to their inherent complexity and the limitations of existing verification tools. Hence, the transformation from Pomsets to LTS is very important, especially in embedded systems and concurrent systems. This allows for a transition from a representation of concurrent behaviors to a representation more suited for formal analysis and verification, thereby facilitating the application of several existing verification algorithms and tools [4]. The transformation algorithm lacked a formal verification of its correctness. Here, we address this gap by leveraging Hoare logic to establish the algorithm's validity. By applying Hoare logic, we provide a structured and formal method to demonstrate that the algorithm meets its specifications. This proof not only shows that our algorithm is reliable, but it also adds to the field of formal verification by showing how Hoare logic can be used in real life to check complex transformations.

Our main contributions are:

- After we determine and demonstrate the invariants of the two nested loops (proving the three properties: initialization, maintenance, and termination) of our algorithm, we use Hoare's axioms to prove partial correctness based on the established invariants.
- We demonstrate termination based on the notion of a variant. Proving the termination of the algorithm requires proving the non-negativity of the two variants and their decrement after each iteration. Thus, we conclude the total correctness of the algorithm.
- We prove the soundness based on Hoare's theorem.
- We calculate the algorithms' spatial and temporal complexity.

We structure the remainder of this paper as follows: We review related works in section II. We present some theoretical background on formal verification and our formal refinement proof approach in section III. In section IV, we present in detail the total correctness proof of the transformation algorithm. Section V will then present the soundness proof. In Section VI, we will present a complexity analysis. Finally, the paper ends with a conclusion and future work.

II. Related Works

Recently, formal verification has gained significant momentum. This is particularly true in embedded systems, where the correction of late-stage errors can be costly, time-consuming, and potentially lead to catastrophic consequences. The task of proving the correctness of software through formal verification has been a long-standing research focus. Nevertheless, formal methods have yet to achieve widespread practical adoption in this field.

Many works have dealt with the subject of program correctness, soundness, and completeness using different formal verification methods, including Hoare logic formalism. We briefly discuss some of the most relevant or recent articles about this below.

The authors of [5] look at how to combine overapproximating and underapproximating logics using a new framework they call Gradual Exact Logic (GEL). This framework aims to bridge the gap between traditional program verification methods (*Hoare logic*) and bug-finding techniques. In this paper, both correctness and soundness are considered. Concerning correctness, GEL inherits Hoare logic's correctness guarantees.

The evolution of Hoare logic has taken place in several expansions and improvements to accommodate diverse programming paradigms, including quantum computing. Quantum Hoare Logic (QHL) has been developed to tackle the

distinct issues presented by quantum algorithms, including superposition and involvement. This change keeps the core ideas of Hoare logic while adding components for quantum systems, making it easier to check quantum programs for correctness. [2] [6].

The authors of [6] describe how to formalize quantum Hoare Logic (QHL), which is an extension of classical Hoare logic that can be used to reason about quantum programs. They formalize the syntax and semantics of quantum programs in Isabelle/HOL, an interactive theorem prover. They verify the soundness and completeness of the deduction system's partial correctness for quantum programs. Then, they apply QHL to verify the correctness of Grover's search algorithm.

In the paper [7], a variant of quantum Hoare logic (QHL) called applied quantum Hoare logic (aQHL) is presented. The goal of this new logic is to make it easier to check quantum programs by limiting QHL to a specific set of preconditions and postconditions called projections. These projections make the verification process more convenient, especially for debugging and testing quantum programs. The usefulness of aQHL is shown by how well it checks two complex quantum algorithms: the Harrow-Hassidim-Lloyd (HHL) algorithm [8] for verifying systems of linear equations and the quantum Principal Component Analysis algorithm (qPCA) [9], which uses quantum Hoare logic.The research by Anika Zaman and Hiu Yung Wong ([8]) is mainly about making sure that the HHL quantum algorithm works correctly and soundly when it is used with IBM-Q hardware. The study verifies the correctness of the HHL algorithm by comparing the results obtained from the IBM-Q hardware implementation with those from a MATLAB simulator. The authors make sure that the errors found in the HHL algorithm are correctly recorded and analyzed. This shows that the algorithm works as expected when errors happen in different situations.

Probabilistic methods, such as probabilistic Relational Hoare Logic (pRHL), make Hoare logic more useful when dealing with randomness and keeping personal information safe. [10] [11]. The article by Gilles Barth et al. [10] focuses on the formal verification of quantum programs using relational program logic, which is based on a quantum analogue of probabilistic couplings, which allows for the verification of non-trivial properties of quantum programs. And the paper by Tetsuya Sato ([11]) presents the formal verification of differential privacy for databases using approximate relational Hoare logic (apRHL).

A recent paper by Lena Vercht and Benjamin Kaminski [12] provides a comprehensive examination of various Hoare-like logics. The paper discusses the soundness and completeness of the logical foundations of these Hoare-like logics. To check the properties of these logics, the authors used two formal verification methods: predicate transformers, which are based on Dijkstra's weakest precondition, are used to express program properties. And Kleen Algebra with Top and Tests (TopKat) is used to model elements of a relational algebra (program and its pre- and postconditions), allowing the expression of program properties as equations between terms.

In the article [13], the authors utilized Hyper Hoare Logic, a modification of classic Hoare logic that facilitates reasoning about hyperproperties, which pertain to multiple program executions. Hyper Hoare Logic extends assertions to features of arbitrary state sets, facilitating the verification of both the absence and presence of (combinations of) executions. The authors demonstrate that this logic is both sound and complete, and they illustrate its application through the Isabelle/HOL theorem.

Other works have used other formal verification methods. Among these works is the paper [14], which proposes that reasoning about program incorrectness can be placed on a logical footing, similar to correctness reasoning but different from it. Correct reasoning requires forgetting information as you go along, while incorrect reasoning requires forgetting some paths. The paper talks about how the under-approximate triple can be used to show that there are bugs and why assertions about successful termination are needed even when errors are the main concern. It also designs a specific logic, the incorrectness logic, along with a semantics and proof theory. The paper explores reasoning idioms, including making connections to concerns in automatic program analysis. The paper also discusses the soundness of the first two iteration rules.

In [15], the authors present a methodology to increase the reliability of the code synthesized through the use of large language models (LLMs). The approach focuses on teaching model checking and runtime verification (RV) algorithms, demonstrating that LLMs can grasp dynamic programming concepts for verification tasks.

Table I provides a comparative analysis of the studies discussed in this work, using specific evaluation criteria. We first examine the formal verification method, which may include Hoare logic, incorrectness logic, model checkers, or other methods. The second criterion examines the formal properties validated, including soundness, safety, correctness, and completeness.

III. Theoretical Backgrounds

The algorithm to be verified was proposed in a previous refinement proof approach for embedded systems [3]. The details of the transformation algorithm from Pomset to LTS, along with the refinement proof approach, are described in [3].

A brief description of the approach and the algorithm itself is provided in this section. As well as terminologies about the Hoare logic, which will be used in order to prove the correctness and soundness of the algorithm. Before presenting the refinement proof approach and the transformation algorithm from Pomsets to LTSs, let's first define a Pomset and an LTS:

Definition 1 (LTS)**.** *A labeled system of transitions is a quintuplet:* $\langle Q, q_0, L, T\rangle$ *where:*

- Q *is the set of states.*
- $q_0 \in Q$ *is the initial state.*
- L *is the set of labels.*
- $T \subseteq Q \times L \times Q$ *is the transition relation.*

An element $(q, \alpha, q') \in T$ will be noted $q \xrightarrow{\alpha} q'$.

TABLE I
COMPARISON OF WORKS ON FORMAL ALGORITHM VERIFICATION

Paper	Verified Formal Properties	Formal verification method
[5]	correctness, soundness	Hoare logic and incorrectness logic
[10]	Uniformity, reliability, security	Relational Program Logic
[6]	Correctness, Soundness, Completeness	Quantum Hoare Logic (QHL)
[7]	Correctness, Soundness, Completeness	Quantum Hoare Logic (QHL)
[8]	Correctness,Soundness	Applied Quantum Hoare Logic (aQHL)
[16]	Correctness	Conformal Prediction (CP)
[14]	Incorrectness, Soundness	Incorrectness logic
[15]	Correctness	Large Language Models (LLMs), Model Checking, Runtime Verification (RV)
[12]	Partial Correctness, Total Correctness, Incorrectness	Predicate Transformers and (TopKAT)
[13]	Correctness, Soundness, Completeness	Hyper Hoare Logic

Definition 2 (Labeled Partial Order)**.**
Called also LPO or a labeled partial order, it is a 4-tuple $\langle V, A, \preceq, \mu\rangle$ *where:*

- V *a finite set of events.*
- A *a finite set of actions.*
- $\preceq \subseteq V \times V$ *a relation of order on the set* V*.*
- μ*:* $V \to A$ *a surjective labeling function assigning an action to an event.*

A. Formal refinement proof approach

In order to prove communication refinement transformations proposed by Mokrani in [17]. We have proposed a refinement proof approach as presented in the figure 1.

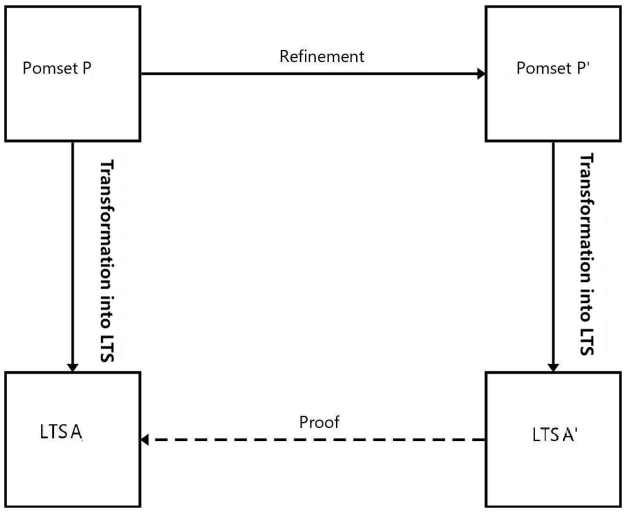

Fig. 1. Refinement proof process

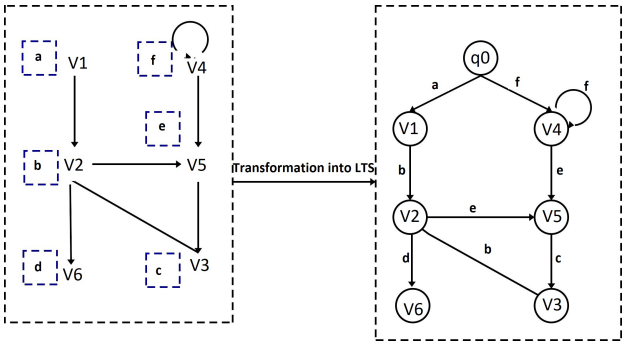

Fig. 2. Example of a Pomset and his corresponding LTS

Given a Pomset P, we shall obtain, after refinement, a Pomset P'. Using the transformation algorithm (see algorithm1), we shall obtain two LTS, A and A', corresponding, respectively, to the Pomsets P and P'. Thereafter, we can prove that A is refined by A' by applying Lanoix's refinement proof algorithm [4]. The following figure (2)presents an example of the transformation to an LTS $A = \langle Q, q_0, L, T\rangle$ corresponding to the Pomset $Pom = (V, A, \preceq, \mu)$

The Transformation Algorithm from Pomset to LTS: As presented in the algorithm 1, it takes a Pomset $Pom = (V, A, \preceq, \mu)$ and produces an LTS $A = (q_0, Q, L, T)$. Let us define $Pred : V \to \mathbb{N}$ as the number of predecessors of a node v, such that $Pred(v) = |\{v' \in V | v' \prec v\}|$.
We designate by I a function that returns the set of initial nodes of the Pomset Pom. An initial node is defined as all nodes that constitute the abstract Pomset without any refinement transformation.

Algorithm 1 Pomset to LTS transformation

Input: pom $= \{V, A, \preceq, \mu\}$ // pom is a Pomset
Output: $\mathcal{A} = \{q_0, Q, L, T\}$ // A is an LTS

$n \leftarrow$ length(V)
$L \leftarrow A$
$Q \leftarrow \{q_0\} \cup \{V\}$
$T \leftarrow \emptyset$ // T is initialized as an empty set
for $i = 1$ **to** n **do**
  $m \leftarrow \mathbf{pred}(v_i)$
  if $m \neq 0$ **then**
    for $j = 1$ **to** m **do**
      $T \leftarrow T \cup (v_j, a, v_i)$ //vj is a predecessor of vi
    end for
    if $v_i \subseteq I$(pom) **then**
      $T \leftarrow \cup(q_0, \epsilon, v_i)$
    end if
  end if
  if $m = 0$ **or** $(m = 1 \wedge v_i \preceq v_i \wedge \mu(v_i) = a)$ **then**
    $T \leftarrow T \cup (q_0, a, v_i)$
  end if
end for

TABLE II
HOARE'S AXIOMATIC SYSTEM FOR PARTIAL CORRECTNESS [19]

$[Ass_P]$	$\{P[x \mapsto A[a]]\}x := a\{P\}$
$[Skip_P]$	$\{P\}skip\{P\}$
$[Comp_P]$	$\frac{\{P\}S_1\}Q\}, \{Q\}S_2\{R\}}{\{P\}S_1; S_2\}R\}}$
$[if_P]$	$\frac{\{\beta[b] \wedge P\}S_1\{Q\}, \{\neg\beta[b] \wedge P\}S_2\{Q\}}{\{P\}\text{if b then } S_1\text{else } S_2\{Q\}}$
$[while_P]$	$\frac{\{\beta[b] \wedge P\}S\{P\}}{\{P\}\text{while b do S } \{\neg\beta[b] \wedge P\}}$
$[Cons_P]$	$\frac{\{P'\}S\{Q'\}}{\{P\}S\{Q\}}$ if $P \Rightarrow P'$ and $Q' \Rightarrow Q$

B. Hoare Logic

Hoare logic ([18]) is a formal framework for reasoning inductively about the correctness of computer programs. Hoare logic has established the basis for formal methods in software development. It is based on preconditions, postconditions, and loop invariants. Preconditions are conditions that must be true before the execution of a program or a program segment. They represent the assumptions about the initial state of the program. On the other hand, postconditions refer to the conditions that must hold true following the execution of a program or a specific segment of it. They represent what the program guarantees to achieve. Invariants are conditions that remain true throughout the execution of a loop or a block of code. They help in reasoning about loops and recursive functions [19].

The correctness of an algorithm consists of demonstrating that it works by answering these two questions: Did the algorithm answer the question correctly? In other words, did it calculate the correct result? And therefore corresponds to the *partial correction*. The second question is: does the algorithm terminate, or does it loop indefinitely? The answer implies that the algorithm responds within a finite time frame. Here we talk about *termination*. Thus, *total correctness is partial correctness plus termination*. These aspects can be written formally as a Hoare triple [20]:

$$\{P\}S\{Q\}$$

The formulas P and Q represent the precondition and the postcondition, respectively, while S represents the program. Pre- and postconditions are formulas in first-order logic.
An inference system made up of axioms and rules [19] is shown in the table II. It specifies the partial correctness assertions. We will use this table in the next section to prove the partial correctness of our algorithm.

IV. CORRECTNESS PROOF OF THE TRANSFORMATION ALGORITHM

Based on Hoare logic, an algorithm's correctness is demonstrated using a loop invariant. The algorithm must align this invariant, which is a property or a set of properties of type boolean, with its goal. It must remain true both before and after each transformation step. This property should describe the main idea of the transformation and make sure that the LTS transitions correctly reflect the order of events in the Pomset. In the following section, we will describe the steps to find and prove the loop invariants of the transformation algorithm 1. To prove a piece of code containing a while (or for) loop, it is essential to identify an invariant, denoted as inv. It is crucial to establish that this invariant holds before the loop's first iteration, that each following iteration maintains the invariant, and that the invariant finally provides a significant property to confirm the algorithm's correctness at the loop's termination. Therefore, to establish an invariant, we must prove the following result.

Theorem 1. *Partial correctness via a loop invariant Let be b the loop condition, C: the body of the loop, $post$ is the postcondition,pre is the precondition, and inv is the loop invariant. An algorithm with a loop is said to be partially correct if and only if its invariant satisfies the following three lemmas:*

Lemma 1. *: Initialization: The invariant must holds prior to the first iteration of the loop. Which means $\{\mathbf{inv} \wedge \mathbf{b}\}\mathbf{C}\{\mathbf{inv}\}$ must be provable.*

Lemma 2. *Maintenance : Assume the invariant holds before an iteration k then, it must hold before the next iteration $k+1$. In other word: the invariant inv must be strong enough to imply the postcondition $\{\mathbf{inv} \wedge \neg\mathbf{b}\} \Rightarrow \mathbf{post}$.*

Lemma 3. *: Termination: the invariant holds when the loop terminates $\{\mathbf{pre} \Rightarrow \mathbf{inv}\}$.*

To prove this theorem we will prove the three lemmas for each loop of the algorithm.

A. Find and prove the loops invariants

As it is shown in the algorithm 1, contains two nested loops:

1) Outer loop: iterates over the nodes $v_i \in V$ (indexed by i).
2) Inner loop: Iterates over the predecessors m of each node v_i.

In the following, we will find and prove the invariants of the two loops, staring by the outer loop (see algorithm 1 lines 9-18).

1) Define the Outer Loop Invariant:

- Initial case : At the start, we have $T = \emptyset$, which means $length(T) = 0$.
- After the i th iteration : we have two cases for each node v_i:
  Case 1 : if
  $$pred(v_i) \neq 0$$
  (lines 5-11) . If the node has one or greater than one predecessor
  $$(m = pred(v_i) \geq 1)$$
  , the inner loop executes m time. That means in each iteration of the inner loop adds at least one transaction to T. $length(T) = length(T) + m$.
  Case 2: if $pred(v_i) = 0$ or 1 (lines 13-15). If a node has no predecessors $(m = 0)$ or only one self-predecessor $(m = 0 \wedge v_i \preceq v_i)$, exactly one transaction added to $T \Rightarrow length(T) = length(T) + 1$.

So, the $length(T)$ can be expressed by i, which represents the number of nodes processed by the algorithm. And the variable $m = pred(v_i)$, which represents the number of predecessors of a node v_i . To resume, at the end of the i-th iteration we have the following expression:

$$length(T) = \sum_{i=1}^{n} \begin{cases} 1, & \text{if } pred(v_i) = 0 \text{ or } 1, \\ pred(v_i), & \text{otherwise} \end{cases}.$$

$$length(T) = pred(v_1) + pred(v_2) + ... + pred(v_i) + q$$

Such as q represents the number of transitions added with the condition $pred(v_i) = 0$ or 1. which means q is the number of nodes where $pred(v_i) = 0$ or $pred(v_i) = 1$. And since these transactions will be with the initial node q_0. So, at least we will have one transaction which relate the node q_0 with any other node v_i. Hence, the predicate of the invariant can be refined as follows:

$$length(T) = \sum_{i=1}^{n} pred(v_i) + q$$

And since $q \geq 1$ and $pred(v_i) = m$. So, we have:

$$inv_{outer} = length(T) \geq \sum_{i=1}^{n} pred(v_i) + 1 \qquad (1)$$

2) Proof of the Outer Loop Invariant:: To prove the outer invariant, we must prove the satisfaction of the three lemmas cited above:

Proof. initialization - Before the first iteration of the loop we have $i = 0$, which means no nodes have been processed yet so the set T is empty; $length(T) = 0 = i$. The invariant at this point $length(T) = \sum$(contributions of nodes from $i = 1$ to $0) = 0$. This is trivially true since no nodes have been processed yet. - After the first iteration $(i = 1)$, node v_1 is processed: $m = pred(v_1)$ (number of predecessors of nodev_1). Lets examine the cases:

- If $pred(v_1) = 0$ (no predecessors), 1 transaction is added.
- If $pred(v_1) \geq 1$ (more than one predecessor), $m = pred(v_1)$. m transactions are added.

So, after processing node v_1, the invariant is $length(T) = \sum 1($ if $m = 0$ or $1), m($if $m \geq 1)$. This is true because for v_1, the correct contribution (either 1 or $pred(v_1)$ has been added to the set T).That means $length(T) \geq pred(v_1) + 1$ is provable. Thus, the loop invariant inv_{outer} holds initially. □

Proof. Maintenance: To show that each iteration maintains the invariant, we suppose that it holds for $i = k$, then we prove that it holds if $i = k + 1$. For $i = k$, we have: $length(T) = \sum_{i=k}^{n} pred(v_i) + m$

$$length(T) = \begin{cases} 1, & \text{if } pred(v_i) = 0 \text{ or } pred(v_i) = 1, \\ pred(v_i), & \text{otherwise.} \end{cases}$$

for $i = 1$ to k
This implies $length(T) \geq \sum_{i=1}^{k} pred(v_i)$. If $i = k + 1$ then we have:

$$length(T) = length(T)_{\text{for k nodes}} + length(T)_{\text{for the node k+1}}$$

so:

$$length(T) \geq \sum pred(v_i) \text{for node } v_1 \text{to } v_{k+1}$$

□

Proof. Termination At the end of the loop $(i = n)$, all nodes $(i = 1$ to n) have been processed. The invariant guarantees: According to the invariant, the set T must contains at least n transitions. So, $length(T) \geq pred(v_i)$. This matches the postcondition, proving that the loop correctly computes the length of the set T.

□

3) Define the Inner Loop Invariant: - The initial case: at the start of $j - th$ iteration of the inner loop, the set T contains all transitions for the predecessors of v_i up to the $j - 1 - th$ predecessor. So, we have $T = T_0$ where $length(T_0) = pred(v_1) + pred(v_2) + ... + pred(v_{j-1})$. That means, the algorithm has processed $j - 1$ predecessors of v_i. And, each predecessor contributes one transition to T.
- During the $j - th$ iteration: a transition for $m[j]$ (the curent predecessor) is added to T, such that:

$$length(T) >= length(T_0) + (j)$$

such as $length(T_0) = \sum_{k=1}^{i-1} pred(v_k) + 1$

$$inv_{inner} = \{length(T) \geq (\sum_{k=1}^{i-1} pred(v_k) + 1) + (j)\} \qquad (2)$$

4) Proof of the Inner Loop Invariant: Similarly, we must prove the three previously cited lemmas for the inner loop invariant.

Proof. Initialisation At the begenning of the inner loop $(j = 1)$, no predecessors of v_i have been processed yet, so the summation remains unchanged, and $length(T)$ reflects transitions from $v_1, v_2, ..., v_{i-1}$. Which means $length(T) = length(T_0) = inv_{outer}$. Thus, the inner loop invariant inv_{inner} holds initially. □

Proof. Maintenance - Assume the invariant holds at the start of $j - th$ iteration. -during the iteration, a transition for j is added to T, increasing $length(T)$ by 1. -By the end of the iteration, $length(T)$ reflects correctly the transitions for j predecessors of v_i. □

Proof. Termination -When $j = m + 1$, all predecessors of v_i have been processed. The summation correctly includes all m predecessors of v_i. That means, $length(T) \geq \sum_{i=1}^{n} pred(v_i) + m$ reflects correctly the postcondition. □

B. Using the Invariants to prove the Algorithm Correctness

In this section, we will prove the correctness of the algorithm. We use the axiomatic semantics for verifivation to prove the algorithm correctness. Hence, we shall prove the following assertions:

$$\{\mathbf{n} = \mathbf{length(V)} \wedge \mathbf{L} = \mathbf{A} \wedge \mathbf{length(Q)} = \mathbf{n} + \mathbf{1}\}$$

$$T := \emptyset$$

$$\textit{for } j = 1 \textit{ to } m \textit{ do}$$
$$m \leftarrow pred(v_i)$$
$$\textit{for } j = 1 \textit{ to } m \textit{ do}$$
$$T := T \cup (v_j, a, v_i) \textit{ EndFor}$$
$$\textit{if } v_i \in I(Pom) \textit{ then}$$
$$T := T \cup (q_0, \epsilon, v_i) \textit{ EndIf EndFor}$$

$$\{\mathbf{length(T)} \geq (\mathbf{length}(\preceq) + \mathbf{1}) \wedge \mathbf{L} = \mathbf{A} \wedge \mathbf{length(Q)} = \mathbf{n} + \mathbf{1}\} \tag{3}$$

The inference of this algorithm proceeds in a number of stages. After we have defined the predicates inv_{outer} and inv_{inner} that are the invariants of the outer and inner loops respectivly. We shall now consider the bodies of the loops. We start by the condition expressed by the lines (12-14)in the transformation algorithm (1.
From $[skip_p]$ (see the tableII),we have:

$$\vdash_p \{\mathbf{i} \leq \mathbf{n}\} skip \{\mathbf{i} \leq \mathbf{n}\}$$

We put $b1 = pred(v_1) \neq 0 \wedge (pred(v_i) = 1 \wedge v_i \preceq v_i)$ Since $(\neg\beta[b1] \wedge i \leq n\} \Rightarrow \{i \leq n\}$, we can apply the rule of consequence $[Cons_p]$ and get:

$$\vdash_p \{\neg\beta[\mathbf{b1}] \wedge \mathbf{i} \leq \mathbf{n}\} Skip \{\mathbf{i} \leq \mathbf{n}\} \tag{4}$$

And from $[Ass_p]$ we have:

$$\{\beta[\mathbf{b}] \wedge \mathbf{i} \leq \mathbf{n}\} T := T \cup (q_0, v, a) \{\mathbf{i} \leq \mathbf{n}\} \tag{5}$$

We can now apply the rule $[if_p]$ for the two above assertions 4 and 5, and we get:

$$\{\mathbf{i} \leq \mathbf{n}\}$$
$$\textit{b1 then } T := T \cup (q_0, v_i, a)$$
$$\textit{skip endIf}$$
$$\{\mathbf{i} \leq \mathbf{n}\}$$

Now we consider the conditionnel asserssion expressed by lines (11-13)(see1)let be $b2 = v \in I(Pom)$ and $P2 = j \leq m \wedge i \leq n$. By applying the rule$[Ass_p]$ we get:

$$\vdash_p \{\beta[\mathbf{b2}] \wedge \mathbf{P2}\} T := T \cup (q_0, \epsilon, v) \{\mathbf{P2} \wedge \mathbf{T} = \mathbf{T} \cup (\mathbf{q_0}, \epsilon, \mathbf{v})\} \tag{6}$$

And from $[Skip_p]$, we have:

$$\vdash_p \{\mathbf{P2}\} Skip \{\mathbf{P2}\}$$

And since $P2 \wedge T = T \cup (q_0, \epsilon, v) \Rightarrow P2$, we can apply the consequence rule $[Cons_p]$ and get:

$$\vdash_p \{\mathbf{P2}\} skip \{\mathbf{P2} \wedge \mathbf{T} = \mathbf{T} \cup (\mathbf{q_0}, \epsilon, \mathbf{v})\} \tag{7}$$

We are now in a position to use the rule for the conditional $[if_p]$ to the two assertions above 6 and 7 and get:

$$\vdash_p \{\beta[\mathbf{b2}] \wedge \mathbf{P2}\}$$
$$\textit{if } (b) \textit{then } T := T \cup (q_0, \epsilon, v) \textit{else skip endIf} \tag{8}$$
$$\{\mathbf{P2} \wedge \mathbf{T} = \mathbf{T} \cup (\mathbf{q_0}, \epsilon, \mathbf{v})\}$$

Now we consider the inner loop assertion expressed by lines (8-10). We have defined the predicate inv_{inner}(see2), that is the invariant of this loop. Considering the body of the loop. Using $[ass_p]$, we get:

$$\vdash_p \{\mathbf{inv_{inner}}[\mathbf{T} \leftarrow \mathbf{T} \cup (\mathbf{v_j}, \mathbf{a}, \mathbf{v_i})]\}$$
$$T := T \cup (v_j, a, v_i)$$
$$\{\mathbf{inv_{inner}}\}$$

It is easy to verify

$$(\neg j \geq m) \wedge inv_{inner} \Rightarrow (inv_{inner}[T \leftarrow T \cup (v_j, a, v_i)])$$

So, using the rule $[cons_p]$ we get:

$$\vdash_p \{\neg(\mathbf{j} \geq \mathbf{m}) \wedge \mathbf{inv_{inner}}\} T := T \cup (v_j, a, v_i) \{\mathbf{inv_{inner}}\}$$

We are now in a position to use the rule $[while_p]$ and get:

$$\vdash_p \{\mathbf{inv_{inner}}\}$$
$$\textit{for } j = 1 \textit{to } m \textit{ do}$$
$$T := T \cup (v_j, a, v_i) \textit{ EndFor}$$
$$\{\neg(\neg(\mathbf{j} \geq \mathbf{m}) \wedge \mathbf{inv_{inner}})\}$$

Clearly we have:

$$\neg(\neg(j \geq m) \wedge inv_{inner}) \Rightarrow (j > m \wedge inv_{inner})$$

So, applying the rule $cons_p$ we get:

$$\vdash_p \{\mathbf{inv_{inner}}\}$$
$$\textit{for } j = 1 \textit{to } m \textit{ do} T := T \cup (v_j, a, v_i) \textit{ EndFor} \tag{9}$$
$$\{\mathbf{j} > \mathbf{m} \wedge \mathbf{inv_{inner}}\}$$

We shall now apply the axiom $[comp_p]$ to the two assertions above 9 and 8, and we get:

$$\vdash_p \{\mathbf{i} \leq \mathbf{n}\}$$
$$\textit{for } j = 1 \textit{ to } m \textit{ do}$$
$$T := T \cup (v_j, a, v_i) \textit{ EndFor}$$
$$\textit{if } v_i \in I(Pom) \textit{ then}$$
$$T := T \cup (q_0, \epsilon, v_i) \textit{ EndIf}$$
$$\{\mathbf{i} \leq \mathbf{n} \wedge \mathbf{j} > \mathbf{m}\} \tag{10}$$

Using $[ass_p]$, we get:

$$\vdash_p \{\mathbf{inv_{outer}}[\mathbf{m} \leftarrow \mathbf{pred(v_i)}] \wedge \mathbf{length(T)} \geq \mathbf{0} \wedge \mathbf{i} \leq \mathbf{n}\}$$
$$m \leftarrow pred(v_i) \tag{11}$$
$$\{\mathbf{length(T)} \geq \mathbf{0} \wedge \mathbf{i} \leq \mathbf{n}\}$$

So, by applying the rule $[Comp_p]$ to the two above assertions 11 and 10, we get:

$$\vdash_p \{\mathbf{i} \leq \mathbf{n} \wedge \mathbf{length(T)} \geq \mathbf{0}\}$$
$$m \leftarrow pred(v_i)$$
$$\textit{for } j = 1 \textit{ to } m \textit{ do}$$

$$T := T \cup (v_j, a, v_i)\ \textit{EndFor}$$

$$\textit{if } v_i \in I(Pom)\ \textit{then}$$

$$T := T \cup (q_0, \epsilon, v_i)\ \textit{EndIf}$$

$$\{\mathbf{i} \leq \mathbf{n} \wedge \mathbf{j} > \mathbf{m} \wedge \mathbf{length(T)} \geq \mathbf{0}\}$$

Then, we can apply the rule $[while_p]$ and get:

$$\vdash_p \{\mathbf{i} \leq \mathbf{n} \wedge \mathbf{length(T)} \geq \mathbf{0}\} \wedge \mathbf{inv_{outer}}$$

$$m \leftarrow pred(v_i)$$

$$\textit{for } j = 1 \textit{ to } m \textit{ do}$$

$$T := T \cup (v_j, a, v_i)\ \textit{EndFor}$$

$$\textit{if } v_i \in I(Pom)\ \textit{then}$$

$$T := T \cup (q_0, \epsilon, v_i)\ \textit{EndIf}$$

$$\{\mathbf{i} \leq \mathbf{n} \wedge \mathbf{j} > \mathbf{m} \wedge \mathbf{length(T)} \geq \mathbf{0} \wedge \mathbf{inv_{outer}}\}$$

We shall now consider the lines (1-4) in the algorithm 1. By applying the axiom $[ass_p]$ to the statement $T := \emptyset$, we get:

$$\vdash_p \{\mathbf{inv_{outer}}[\mathbf{T} \mapsto \emptyset]\}T := \emptyset\{\mathbf{inv_{inner}}\}$$

Using that:

$$\{n = length(V) \wedge L = A \wedge length(Q) = n{+}1\} \Rightarrow \{inv_{outer}\}$$

Together with $[Consp_P]$, we get:

$$\vdash_P \{\mathbf{n = length(V)} \wedge \mathbf{L = A} \wedge \mathbf{length(Q) = n + 1}\}$$

$$T := \emptyset$$

$$\{\mathbf{inv_{outer}}\}$$

Finally, we can use the rule $[Consp_P]$ and the equation 3 as required. Hence, we can say now that the transformation algorithm 1 is correct.
The next step is to prove the termination of the algorithm since *partial correctness + termination = Total Correctness*

C. Termination Proof

To prove the Termination of an algorithm, the following result will be proved.

Theorem 2. *Termination If there exists a function $V(x)$ that maps the state x to a natural number such that:*

- *$V(x)$ is always non-negative.*
- *$V(x)$ strictly decreases on each loop iteration. Then, the loop must terminate after a finite number of iterations.*

Proof. To prove the termination, first we have to find a *variant (called also ranking function)* for each loop. Then prove the decrease of these variants.
We define V_{outer} as the variant of the outer loop and V_{inner} as the variant of the inner loop.

- The outer loop runs exactly n times, which is trivially bounded. So, the variant function is defined as:
  $V_{outer} = (n - i)$ where $n = length(V)$ is the number of nodes, and i is a natural number which increments at each iteration. So, the outer loop ensures that $(n-i)$ decreases by 1 per iteration. Hence, the outer loop terminates.
- The inner loop depends on $m = pred(v_i)$ which is bounded (the number of predecessors of a node v_i)because it cannot exceed the number of nodes n, which means that $m \leq n$, so the variant function for this loop is defined as follows:
  $V_{inner} = m - j$ deceases by 1 per iteration. So the inner loop also terminates.

□

We can say mow, the algorithm is *partially correct and terminates* so the algorithm is *totally correct*.

V. Soundness proof

After proving the partial correctness of our algorithm using Hoare logic, proving its soundness requires showing that our reasoning is valid and that all derivations made using Hoare logic are correct.
We have:

Theorem 3. *Soundness For all partial correctness assertions $\{P\}S\{Q\}$ we have:*

$$\models_p \{P\}S\{Q\}\ \textit{If and only if}\ \vdash_p \{P\}S\{Q\}$$

We have to prove the following lemma:

Lemma 4. *The inference system of (tableII)is sound that is for every partial correctness formula $\{p\}A\{Q\}$ we have:*

$$\vdash_P \{p\}A\{Q\} \Rightarrow \models_P \{p\}A\{Q\}$$

Proof. Proving soundness means that if we can derive $\{p\}A\{Q\}$ (a Hoare using the rules of Hoare logic (see table of axioms II), then the algorithm actually behaves correctly in all execution paths, assuming the precondition holds. In other words, since $\{p\}$*Our transformation algorithm*$\{Q\}$ is provable (the equation equation3), then running the algorithm from a state satisfying P will always lead to a state satisfying Q, provided that the algorithm terminates as required. □

VI. Complexity Analysis of the Pomset to LTS Transformation Algorithm

This section presents the analysis of the spatial and temporal complexity of our algorithm.

A. Temporal Analysis

Let $n = |V|$ be the number of nodes in the Pomset and d the average degree of the nodes.

1) Initialization step:

- $n \longleftarrow |V|$: $O(1)$
- $L \longleftarrow A$: $O(1)$
- $Q \longleftarrow \{q_0\} \cup V$: $O(n)$
- $T \longleftarrow \emptyset$: $O(1)$

The total complexity of initialization step: $O(n)$

2) Main Loop: (n iterations)

- Retrieval of predecessors $m \longleftarrow pred(v_i)$: assumed to be $O(1)$ on average.
- If $m \neq 0$:
  - Loop over the m predecessors: $O(m)$.
  - Add an initial transition if $v_i \in I(pom)$: O(1).
- If $m = 0$ or special condition ($m = 1 \wedge v_i \preceq v_i \wedge \mu(v_i) = a$): adding a transition in $O(1)$

Complexity of one iteration: $O(m)$, where m is the number of predecessors of v_i. Since $m \leq d$, the complexity becomes $O(d)$ per iteration.

3) Total Complexity: The loop runs n times, so the overall complexity is:

$$O(nd) \tag{12}$$

Special cases::

- Best case (linear chain): $O(n)$.
- Worst case (dense, complete Pomset): $O(n^2)$.

B. Spatial Analysis

1) Data Structures:

- $Q = q_0 \cup V$:$O(n)$.
- $L = A$: negligible.
- T (set of transitions): $O(nd)$.
- $\preceq$ (partial order relation):
  - *Adjacency matrix*: $O(n^2)$.
  - *Adjacency list*: $O(nd)$.
- μ (labeling function):$O(n)$.

2) Total Space Complexity:

- Optimized case (adjacency list, sparse Pomset): $O(nd)$.
- Worst case (adjacency matrix, dense Pomset): $O(n^2)$

VII. Conclusion and Future Works

In conclusion, the research paper establishes the correctness and soundness of the algorithm for converting Pomsets to LTS. The algorithm's rigorous demonstration ensures the preservation of behavioral properties and dependencies from the original Pomset to the resulting LTS. Hoare logic is used for formal verification, ensuring every valid Pomset is transformed into an LTS without information loss. Detailed proofs confirm that the algorithm generates an LTS only for valid Pomsets, capturing all behaviors expressible in the Pomset. These results confirm the reliability and robustness of our transformation algorithm, making it a valuable tool for modeling and analyzing concurrent systems. In this paper, we used Hoare logic to give our previously suggested transformation algorithm from Pomsets to LTS a formal verification. The demonstration of the algorithm's partial correctness was based on the invariants proof. Then we have demonstrated the termination of the algorithm using the variant. Consequently, we have proven the algorithm's total correctness. Finally, we deduced the algorithm's soundness. In our future work, we plan to continue working on the completeness proof of our algorithm. Additionally, we want to make our algorithm and our refinement proof approach more flexible, so we plan to add the idea of time to it. This will cause timed automata to be generated instead of LTS. This extension allows us to effectively capture the dynamic behavior of real-world systems, such as real-time controllers and embedded systems.

References

[1] C. A. R. Hoare, "An axiomatic basis for computer programming," *Software Pioneers*, pp. 367–383, 2002.

[2] X. Le, S. Lin, J. Sun, and D. Sanán, "A quantum interpretation of separating conjunction for local reasoning of quantum programs based on separation logic," *Proceedings of the Acm on Programming Languages*, vol. 6, pp. 1–27, 2022.

[3] A. Bezza, E. Merah, R. Ameur-Boulifa, R. Benaboud, and T. M. Maarouk, "Formalization and refinement proof for embedded systems," in *2020 4th International Symposium on Informatics and its Applications (ISIA)*, pp. 1–6, IEEE, 2020.

[4] A. Lanoix, "Verifier le raffinement de maniere compositionnelle,"

[5] C. Zimmerman and J. DiVincenzo, "Gradual exact logic: Unifying hoare logic and incorrectness logic via gradual verification," *arXiv preprint arXiv:2412.00339*, 2024.

[6] J. Liu, B. Zhan, S. Wang, S. Ying, T. Liu, Y. Li, M. Ying, and N. Zhan, "Formal verification of quantum algorithms using quantum hoare logic," in *Computer Aided Verification: 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II 31*, pp. 187–207, Springer, 2019.

[7] L. Zhou, N. Yu, and M. Ying, "An applied quantum hoare logic," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 1149–1162, 2019.

[8] A. Zaman and H. Y. Wong, "Study of error propagation and generation in harrow-hassidim-lloyd (hhl) quantum algorithm," in *2022 IEEE Latin American Electron Devices Conference (LAEDC)*, pp. 1–4, IEEE, 2022.

[9] C. He, J. Li, W. Liu, J. Peng, and Z. J. Wang, "A low-complexity quantum principal component analysis algorithm," *IEEE transactions on quantum engineering*, vol. 3, pp. 1–13, 2022.

[10] G. Barthe, J. Hsu, M. Ying, N. Yu, and L. Zhou, "Relational proofs for quantum programs," *Proceedings of the ACM on Programming Languages*, vol. 4, pp. 1–29, 2019.

[11] T. Sato, "Approximate relational hoare logic for continuous random samplings," *Electronic Notes in Theoretical Computer Science*, vol. 325, pp. 277–298, 2016.

[12] L. Verscht and B. L. Kaminski, "A taxonomy of hoare-like logics: Towards a holistic view using predicate transformers and kleene algebras with top and tests," *Proceedings of the ACM on Programming Languages*, vol. 9, no. POPL, pp. 1782–1811, 2025.

[13] T. Dardinier and P. Müller, "Hyper hoare logic:(dis-) proving program hyperproperties," *Proceedings of the ACM on Programming Languages*, vol. 8, no. PLDI, pp. 1485–1509, 2024.

[14] P. W. O'Hearn, "Incorrectness logic," *Proceedings of the ACM on Programming Languages*, vol. 4, no. POPL, pp. 1–32, 2019.

[15] I. Cohen and D. Peled, "Llm-based scheme for synthesis of formal verification algorithms," in *International Conference on Bridging the Gap between AI and Reality*, pp. 167–182, Springer, 2024.

[16] L. Lindemann, X. Qin, J. V. Deshmukh, and G. J. Pappas, "Conformal prediction for stl runtime verification," in *Proceedings of the ACM/IEEE 14th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2023)*, pp. 142–153, 2023.

[17] H. Mokrani, *Assistance au raffinement dans la conception des systèmes embarqués*. PhD thesis, Télécom ParisTech, 2014.

[18] C. A. R. Hoare, "An axiomatic basis for computer programming," *Communications of the ACM*, vol. 12, no. 10, pp. 576–580, 1969.

[19] H. Nielson and F. Nielson, "Semantics with applications: A formal introduction, revised edn.(july 1999)," 1992.

[20] M. Rainer-Harbach, *Methods and tools for the formal verification of software: an analysis and comparison*. PhD thesis, 2011.

Asma Bezza is currently pursuing her Ph.D. in Computer Science at the University of Oum El Bouaghi, Algeria, and conducting research at the ICOSI Laboratory, Abbes Laghrour University, Khenchela. She earned her Magister degree in 2013. Since 2014, she has served as an assistant professor at Abbes Laghrour University, where she contributes to research on e-learning, personalisation, formal methods, and formal verification. Her work focuses on formal refinement proof in embedded systems.

Rohallah Benaboud is a senior lecturer at the Department of Mathematics and Computer Sciences - University of Oum El Bouaghi (Algeria). He obtained his PhD degree in Computer Science from University Abdelhamid Mehri Constantine 2, Algeria in 2016. He is currently a member of Distributed-Intelligent Systems Engineering (DISE) team at ReLa(CS)2 Laboratory - University of Oum El Bouaghi. Rohallah Benaboud has published many articles in many International Conferences and Journals. His research interests include Internet of Things, Service Oriented Computing, Machine/Deep Learning and Multi-Agents Systems.

Toufik Messaoud Maarouk is a Professor at the Department of Computer Science, Faculty of Sciences and Technology, at the University of Khenchela, Algeria. He received his Ph.D. in Computer Science from Constantine University, Algeria, in 2012. He currently serves as the Director of the ICOSI Laboratory . His research focuses on formal methods for the specification and verification of concurrent and distributed systems, with particular emphasis on concurrency theory, formal semantics, and process algebras. He has supervised several doctoral theses in the fields of formal verification and multi-agent systems, leading to applications in intelligent systems and real-time environments.

Ameur-Boulifa currently works at the Department of Communications Electronics, Telecom Paris, Institut Polytechnique de Paris. Rabéa does research in the design, testing, and verification of reliable computer systems. Her current projects include the development of safe and secure embedded systems.