Earliest Deadline Late Server(EDLS)

Masrur Jamil Prochchhod
Department of Electronic Engineering
Hochschule Hamm Lippstadt
masrur-jamil.prochchhod@stud.hshl.de

July 18, 2024

Abstract In real-time systems, meeting task deadlines is crucial to ensure system reliability and performance. In this regard, an early deadline late server scheduling algorithm has long been considered promising to alleviate this problem by using the idea of a "late server" to handle jobs that are past due. This paper gives importance to the Earliest Deadline Late Server scheduling algorithm for efficient handling of periodic and aperiodic tasks in real-time systems. EDLS self-manages its behavior according to system states and workloads in search of idle times to lend them to tasks with the most imminent deadlines. In this paper, we give a thorough study of its theoretical roots, implementation, and real-life performance for applications. This study has highlighted the benefits brought forth by EDLS on efficient deadline management, balanced task scheduling, and improved system utilization, against challenges such as complexity and overhead. This work therefore focuses on advanced scheduling techniques applied within real-time processing environments to come up with improved understanding and application.

I Introduction

Real-time systems focus on the timely completion of tasks within their deadlines in order to guarantee the reliability and functionality of timing-critical applications. This has been of paramount importance in a number of fields, including embedded systems, aerospace, the automobile industry, and telecommunications. Correctness in these systems is not related only to the logic of results from computations but also to the time of computation. Missing deadlines in these environments may mean catastrophic failure, financial loss, or even human lives.

Such challenges in deadline management have motivated the development of the Earliest Deadline Late Server scheduling algorithm. This algorithm is designed to cope efficiently with periodic and aperiodic tasks concerning the system state and workload. This property enables EDLS to optimize the exploitation of idle times and guarantee the execution of tasks with closest coming deadlines. By doing so, EDLS aims to reduce the risk of missed deadlines and improve overall system responsiveness. Many complex control applications include tasks which have to be completed within strict time constraints, called deadlines, Where the meeting of a given deadline is critical for the system's operation, and may cause catastrophic consequences in case of a miss, that deadline is considered to be hard. If missing a deadline is not a serious damage but meeting time constraints is desirable, then that deadline is considered to be soft. Along with their criticalness, tasks that require regular activations are called periodic, whereas tasks which have irregular arrival times are called aperiodic.

This paper provides a comprehensive examination of the EDLS algorithm. It delves into the theoretical underpinnings, explores practical implementation aspects, and evaluates the performance of EDLS in various real-world applications. Figures illustrating the algorithm's functionality, idle time management, and scheduling of aperiodic tasks are included to enhance understanding.

II Importance of Meeting Task Deadlines in Real-Time Systems

Real-time systems are those in which processing and reaction to input are subject to stringent timing constraints. These are usually distinguished as hard, firm, and soft deadlines. Hard deadlines are those where any form of delay is unwanted and could mean system failure, such as in safety-critical applications like a pacemaker or an automotive airbag system. Firm deadlines, even though they are still critical, permit occasional missed deadlines without catastrophic consequences, though with performance degradation. In the soft deadline case, occasional deadline misses only lead to a reduced quality of service rather than an outright failure. [4]

In all these cases, it is essential that the real-time tasks be guaranteed to complete. In hard real-time systems, missing the deadlines could mean loss of life or huge financial loss, while in soft real-time systems, it would mean poor performance and user dissatisfaction. Hence, efficient scheduling algorithms that are strong enough to handle changing loads without missing their deadlines form an integral part of a real-time system.

III Earliest Deadline Late Server

We have made investigations on how to efficiently use the remaining idle time while scheduling periodic tasks to ensure maximum lateness. In that respect, we have come up with a dynamic scheduling algorithm called the EDL server, focusing on the completion of soft aperiodic tasks as early as possible. EDLS Algorithm uses two complementary versions of EDF. It uses EDS—Earliest Deadline First with Slack Stealing—and EDL—Earliest Deadline First with Lateness. [5]

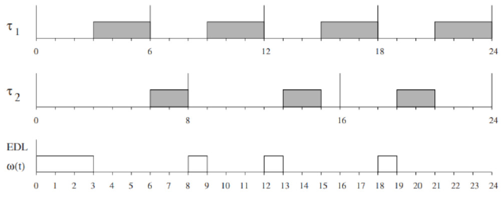


Figure 1: EDL Algorithm [1]

Figure 2: Fundamentals behind how the Earliest Deadline Late Server (EDLS) algorithm schedules any two periodic tasks, 1 and 2, within any 24 time unit period. Solid blocks indicate the execution times of tasks; that is, when these tasks are active.

Task τ_1 is run during intervals such as 0–4, 6–10, 12–16, and 18–22. These intervals show the time blocks within which task 1 is running actively on the CPU. Task τ_2 executes in intervals 0-2, 8-10, 16-18, and 24-26—examples of when it is scheduled to execute. Next to these rows of tasks, one can notice the EDL line, (t), which represents all the CPU idle times that are not occupied by the periodic tasks. These idle slots arrive, for instance, in the intervals 4-6, 10-12, 16-18, and 22-24 and represent periods when no periodic tasks are executed; thus, the CPU is free.

This figure shows how the EDLs algorithm forces the execution of periodic tasks within their specified times to discover the idle periods that can be used in the execution of aperiodic tasks.

Whenever a new aperiodic request arrives into the system, it calculates the idle times of an EDL scheduler applied to the current periodic task set and uses this to schedule the pending aperiodic requests. The idle times of EDLS schedular used in the periodic task set are then used to schedule the aperiodic requests task pendings. Figure 2 and 3 shows and example of EDL service Mechanism.

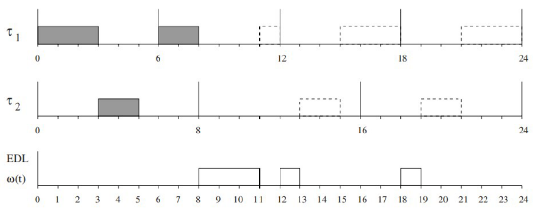


Figure 2: Idle Time Available [1]

This figure illustrates the scheduling of an aperiodic task. The aperiodic task is executed in the idle times computed by the EDL algorithm without affecting the periodic tasks execution, as shown in.

All these numbers taken together reflect that EDFS has to deal with periodic and aperiodic tasks and use system resources effectively for executing all tasks on time according to their deadlines.

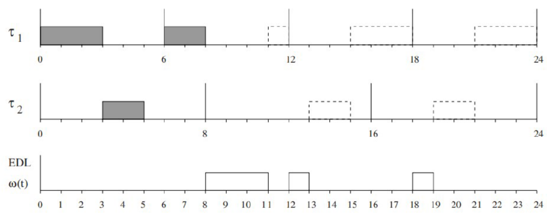


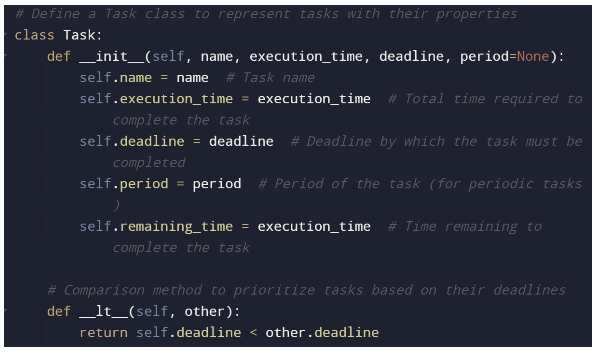
Figure 3: Scheduling of aperodic task [1]

This figure shows the timing chart for an aperiodic task. The aperiodic task will execute in the idle times found by the EDL algorithm. Thus, the aperiodic task does not interfere with the execution of periodic tasks and the whole system's efficiency and reliability are maintained.

These figures show how EDLS manages periodic and aperiodic tasks jointly to ensure good utilization of system resources and timely execution of all tasks without missing any deadlines. In the later part we will talk about the whole algortihm of the process using python to get a far more better understanding the whole scenario of how the tasks are dealt and how they are being done.

IV EDLS Algorithm

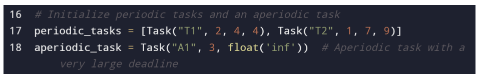
The EDLS algorithm is designed to manage and schedule both periodic and aperiodic tasks in real-time systems. It dynamically utilizes idle times (when no periodic tasks are ready to execute) to process aperiodic tasks, ensuring efficient use of system resources while meeting task deadlines.



```python
# Define a Task class to represent tasks with their properties
class Task:
    def __init__(self, name, execution_time, deadline, period=None):
        self.name = name  # Task name
        self.execution_time = execution_time  # Total time required to
            complete the task
        self.deadline = deadline  # Deadline by which the task must be
            completed
        self.period = period  # Period of the task (for periodic tasks
            )
        self.remaining_time = execution_time  # Time remaining to
            complete the task

    # Comparison method to prioritize tasks based on their deadlines
    def __lt__(self, other):
        return self.deadline < other.deadline
```

Figure 4: Defining the Functions

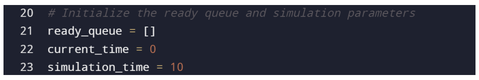
The `Task` represents any task with at least the key properties like`name`, `execution_time`, `deadline`, and optionally `period` for periodic tasks. The `remaining_time` attribute keeps track of how much time is remaining to finish the task. The `__lt__` method is overridden to make the tasks comparable and sortable according to their deadlines, so that the ones with earlier deadlines can be executed first. This class forms the base for handling tasks scheduling in the EDLS algorithm.



```python
# Initialize periodic tasks and an aperiodic task
periodic_tasks = [Task("T1", 2, 4, 4), Task("T2", 1, 7, 9)]
aperiodic_task = Task("A1", 3, float('inf'))  # Aperiodic task with a
    very large deadline
```

Figure 5: Initialization

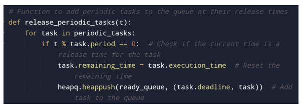
Two periodic, T1 and T2, and one aperiodic, A1, tasks are initialized. The aperiodic task has an infinite deadline, and thus it will only execute during idle times..



```python
# Initialize the ready queue and simulation parameters
ready_queue = []
current_time = 0
simulation_time = 10
```

Figure 6: Queue

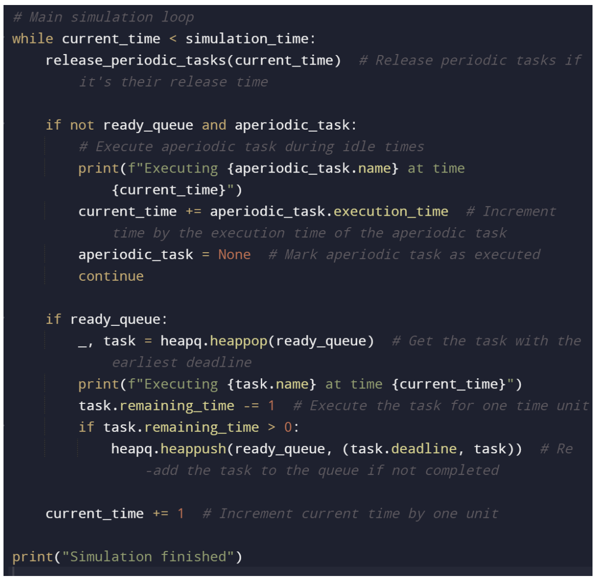
A priority queue (`ready_queue`) is used to manage tasks based on their deadlines. The simulation runs for a specified duration (`simulation_time`), with `current_time` tracking the elapsed time.



```python
# Function to add periodic tasks to the queue at their release times
def release_periodic_tasks(t):
    for task in periodic_tasks:
        if t % task.period == 0:  # Check if the current time is a
            release time for the task
            task.remaining_time = task.execution_time  # Reset the
                remaining time
            heapq.heappush(ready_queue, (task.deadline, task))  # Add
                task to the queue
```

Figure 7: Release Periodic Task Function

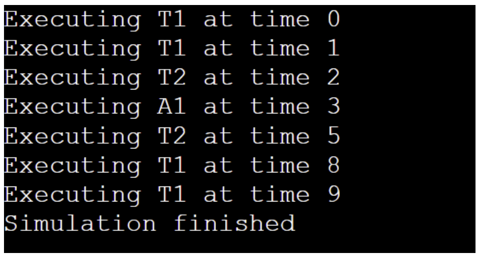
This function puts periodic tasks into the ready queue at their release times. It checks whether the current time is a multiple of the task's period—the time when a new instance of the task should be released.



```python
# Main simulation loop
while current_time < simulation_time:
    release_periodic_tasks(current_time)  # Release periodic tasks if
        it's their release time

    if not ready_queue and aperiodic_task:
        # Execute aperiodic task during idle times
        print(f"Executing {aperiodic_task.name} at time
            {current_time}")
        current_time += aperiodic_task.execution_time  # Increment
            time by the execution time of the aperiodic task
        aperiodic_task = None  # Mark aperiodic task as executed
        continue

    if ready_queue:
        _, task = heapq.heappop(ready_queue)  # Get the task with the
            earliest deadline
        print(f"Executing {task.name} at time {current_time}")
        task.remaining_time -= 1  # Execute the task for one time unit
        if task.remaining_time > 0:
            heapq.heappush(ready_queue, (task.deadline, task))  # Re
                -add the task to the queue if not completed

    current_time += 1  # Increment current time by one unit

print("Simulation finished")
```

Figure 8: Main Simulation Loop

The main loop simulates the running of tasks against time. It maintains periodic task release, schedules, and executes tasks by deadline, and executes aperiodic tasks during idle time.



```
Executing T1 at time 0
Executing T1 at time 1
Executing T2 at time 2
Executing A1 at time 3
Executing T2 at time 5
Executing T1 at time 8
Executing T1 at time 9
Simulation finished
```

Figure 9: Output

This example illustrates how the EDL scheduling algorithm performs the scheduling of two periodic tasks, T1 and T2, and the execution of the aperiodic task, A1, in idle time so that tasks are executed based on their deadlines. The simulation is stopped after the execution of all the scheduled tasks in the given simulation time. This shows the following algorithm of the whole process works in the correct manner in python and how they should be implemented.

V Multimedia Streaming System

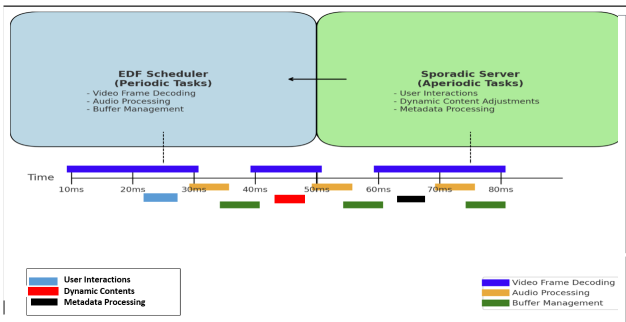


Figure 10: Application of EDLS in Multimedia

This diagram illustrates the implementation of Earliest Deadline Late Server in a multimedia application streaming server. In this configuration, it will support periodic and aperiodic tasks for optimal performance and resource utilization.

A multimedia system comprises two basic components. The execution of periodic tasks is assured uninterruptedly by the EDF Scheduler. The periodic tasks will thus be video frame decoding, audio processing, and buffer management. All these tasks are scheduled at fixed intervals so that most of them can meet their deadlines and ensure real-time performance.

The Sporadic Server processes aperiodic tasks. These are tasks of an irregular occurrence in nature, such as user interactions, dynamic content adjustments, and metadata processing. They are executed during idle time, so they will not interfere with periodic tasks.

The bottom timeline of the diagram illustrates the run of tasks against time. Video frame Decoding is executed from 10ms to 20ms, followed by user interactions. At 30ms, audio processing occurs. At 40ms, it is buffer management that is executed, followed by dynamic content adjustments. Video frame decoding occurs again from 50ms to 60ms. At 70ms, audio processing is followed by metadata processing. Finally, buffer management is executed at 80ms.

This execution sequence proves that the EDLS algorithm is dynamic in handling periodic tasks based on deadlines and tries to fit aperiodic tasks into idle times to ensure optimal resource utilizations.

To summarize, the EDLS algorithm ensures the running of periodic tasks by priorities within their deadline and utilizes the idle times for aperiodic tasks. The approach ensures high responsiveness and performance of the system, as required in real-time multimedia applications.

VI Uppaal Model of EDLS and Sequence Diagram

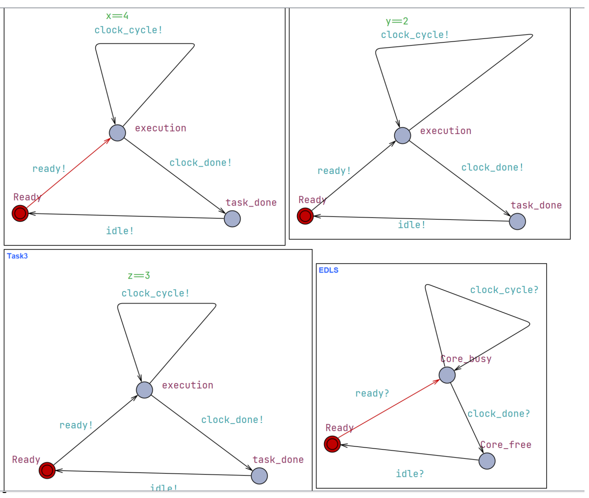


Figure 11: Uppaal Diagram

The EDL scheduling algorithm can easily be represented using the UPPAAL Modeling Diagram software, whereby it becomes easy to model the states of the task clearly. The figure shown is the designed UPPAAL simulation model for the scheduling of periodic and aperiodic tasks in a runtime period.

In this particular model:

- **TaskOne** and **TaskTwo** are periodic tasks, with durations of 4 and 2 time units respectively.
- **TaskThree** is an aperiodic task, with a duration of 3 time units.
- The entire process runs for a total of 10 time units.

The model developed in simulation also elaborately showcases the utilization status of the core. At times of task execution, the core will be marked utilized. Once the task completes, the core will be marked free and would remain ready for the execution of the next task.

This is a simulation model in UPPAAL of how the EDL algorithm works in scheduling periodic and aperiodic tasks, stating how long it takes to perform any particular task, time taken by the system as a whole for runtime, and the status of the core's utilization in the middle of performing tasks. This graphical display helps in comprehending the way the scheduling of tasks occurs within the provided runtime to help in efficient management and running of tasks.

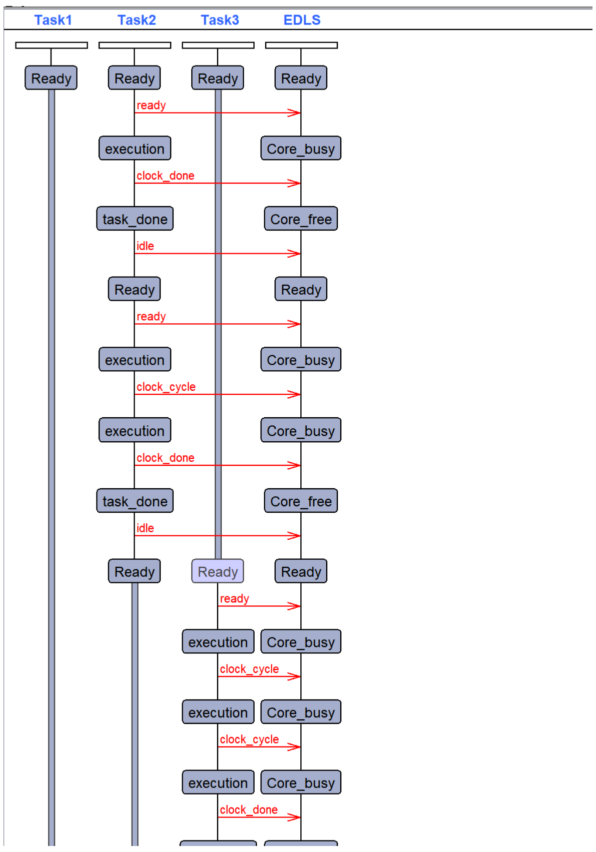


Figure 12: EDLS Sequence

The sequence diagram of the runtime simulation generated by the Uppaal model is shown in Figure 12 above. First, it will be the duration of 2 time units Task2 to be run on the CPU core. Upon the completion of Task2, the aperiodic Task3 of duration 3 time units is scheduled and executed. It is this time that the EDL algorithm exploits by executing the aperiodic task whenever the core is available. The periodic Task1 of duration 4 time units gets scheduled and executed after the execution of Task3. This example explains how EDL efficiently manages periodic and aperiodic tasks through dynamic allocation of CPU time depending on the task readiness and deadlines.

VII Advantages and Disadvantages

Advantages

- **Efficient Deadline Management:**EDLS addresses the jobs according to the earliest due date; therefore, all the critical tasks are executed on time.
- **Balance Task Scheduling:** Dynamic priority adjustments by EDLS balance the scheduling of tasks with respect to their deadlines, ensuring no deadline is missed.
- **Improved System Utilization:** EDLS can ensure higher system utilization by very effectively handling idle times and ensuring that tasks get executed at the earliest possible.
- **Quality of Service:** EDLS reduces the number of deadline misses, hence improving the quality of service by allowing more predictable and reliable task execution.
- **Flexibility:** EDLS makes itself versatile to different sets of tasks and system loads by managing periodic and aperiodic tasks.

Disadvantages

- **Complex Implementation:** EDLS requires sophisticated algorithms to track the various tasks for continuous deadline monitoring, making implementation complex.
- **Overhead:** The dynamic priority adjustments and frequent context switches in EDLS introduce many computational overhead.
- **Scalability Issues:** This is one of the major drawbacks. EDLS potentially suffers from inefficiency in scaling with a large number of tasks since its overhead and complexity increase linearly with the number of tasks.

VIII Applications in Real Life

Embedded Systems

EDLS can be used for the management of tasks in automotive control systems, for example, embedded systems in automotive applications. EDLS schedules periodic tasks, such as sensor readings and control signals, and aperiodic tasks—the so-called user inputs and emergency alerts—in order to enable these tasks to run with the best possible efficiency in an ECU(engine control units) or an ADAS(advanced driver-assistance systems). [2]

Industrial Automation

EDLS coordinates tasks within industrial automation in such a way that periodic tasks—like sensor data acquisition or routine movements—and aperiodic tasks, such as avoiding unexpected obstacles or executing an emergency stop, are managed. This enables efficient and safe operation of such robots in an industrial environment.

Cloud Computing

EDLS optimizes resource management in cloud computing environments, including periodic—regular backups, maintenance scripts—and aperiodic tasks—on-demand scaling and resource reallocation. This ensures maximum resource usage and better system performance in the cloud. [3]

Aerospace and Defense

In the aerospace and defense industries EDLS manages tasks in unmanned aerial vehicles. It schedules periodic tasks, like navigation and altitude control, and aperiodic tasks, such as collision avoidance and mission updates, for execution to guarantee safe and efficient operation in different mission-critical scenarios for unmanned aerial vehicles.

IX Conclusion

In this paper, we have illustrated the Earliest Deadline Late Server scheduling algorithm and its application in real-time systems. EDLS optimizes idle time usage by efficiently managing periodic and aperiodic tasks to ensure that tasks with the most imminent deadline are scheduled first. Dynamic adjustment of the system increases responsiveness and reliability, and hence EDLS finds its use in a variety of domains like embedded systems, industrial automation, cloud computing, and aerospace and defense.

Our results showed that EDLS offers enormous benefits with respect to deadline management and the balancing of task scheduling, all while improving system utilization. On the other hand, it had a few drawbacks: the implementation process was complex, and computational overheads were huge. In spite of all odds, the benefit in using EDLS for timely task completion and ensuring the high performance of the system is very high.

In the future, how to solve the issues in scalability and the computational overhead of EDLS will be the foci. Further research may have to do with the integration of EDLS with other advanced scheduling techniques so that there can be improved performance in an ever-increasingly complicated environment. In general, the EDLS algorithm represents a robust and flexible solution for the management of real-time tasks, thus offering high system reliability and efficiency.

References

[1] Giorgio C Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications.* Springer Science & Business Media, 2011.

[2] Robert I Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys (CSUR)*, 43(4):1–44, 2011.

[3] James T DeRacy, Nik Bessis, and Jamil Bentahar. Cloud computing scheduling algorithms: A review. *IEEE Access*, 8:127751–127768, 2020.

[4] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[5] Lui Sha, Ragunathan Rajkumar, and John P Lehoczky. Real-time scheduling theory and optimality: Static priority versus dynamic priority. *Real-Time Systems*, 10(1):23–27, 2004.

X AFFIDAVIT

I, Masrur Jamil Prochchhod, hereby confirm that I have written this paper independently and have not used any sources or aids other than those indicated. All statements taken from other sources in wording or sense are clearly marked. Furthermore, I assure that this paper has not been part of a course or examination in the same or a similar version.

[illegible]

Masrur Jamil Prochchhod

Lippstadt, 18.07.2024