FORMAL METHODS IN ACTION:

using runtime verification to automatically address occurrences of labor exploitation crimes

Tesi di Laurea Magistrale in
Computer Science and Engineering

Author: **Federica Suriano**

Student ID: 953085
Advisor: Professor Marcello Maria Bersani
Co-advisors: Professor Damian Andrew Tamburri
Academic Year: 2022-2023

To the daring people who are not afraid to prove their worth.
To my sister who is one of them.

Abstract

Nowadays, software systems are extremely complex: they often have to acquire a myriad of heterogeneous data in a fast and secure way and must ensure that they comply with all the regulations in force for the type of use made of the software itself and of the acquired data. Verifying the software exhaustively is an extremely time-consuming practice. For this reason, faster verification methods than traditional ones have been introduced over time. Among these is runtime verification analysis to verify individual execution paths of a system at runtime.

This research aims to demonstrate that runtime verification method can also be used to **automatically verify the occurrences of events** monitored by the system and that this approach can be extremely useful for **monitoring events attributable to crimes at runtime**. In particular, we evaluated the proposal on a system that monitors labor exploitation in a region of the Netherlands. This choice is a direct consequence of the lack of systems suitable for monitoring such phenomena.

Research has shown that runtime verification is not only an excellent - and commonly used in critical systems - method for verifying the robustness, safety and security of a system under test, but it is also extremely useful for verifying properties of the tracked behavior.

In the aforementioned application context, this translates into the possibility of identifying the occurrence of abuses. Interesting data can emerge from the properties verified, such as the correlation of criminal events in certain situations, or the creation of statistics relating to cases of exploitation in a certain geographical region. The data obtained can be used to make strategic decisions in the context in which the system operates, so that the phenomenon of labor exploitation can be hopefully drastically reduced.

Keywords: RV; runtime verification; labor exploitation; crime detection; labor exploitation.

Abstract in lingua italiana

Al giorno d'oggi i sistemi software sono estremamente complessi: spesso devono acquisire una miriade di dati eterogenei in modo veloce e sicuro e devono garantire che siano conformi a tutte le normative vigenti per il tipo di utilizzo che si fa del software stesso e dei dati acquisiti. Verificare un software è una pratica estremamente dispendiosa in termini di tempo. Per questo motivo nel tempo sono stati introdotti metodi di verifica più rapidi rispetto a quelli tradizionali. Tra questi c'è l'analisi di verifica a runtime per verificare i singoli percorsi di esecuzione di un sistema in fase di esecuzione.

Questa ricerca mira a dimostrare che il metodo di verifica a runtime può essere utilizzato anche per **individuare automaticamente il verificarsi di eventi** monitorati dal sistema e che questo approccio può essere estremamente utile per **monitorare eventi riconducibili a reati a runtime**. In particolare, abbiamo valutato la proposta su un sistema di monitoraggio dello sfruttamento lavorativo in una regione dei Paesi Bassi. Questa scelta è una diretta conseguenza della mancanza di sistemi idonei al monitoraggio di tali fenomeni.

La ricerca ha dimostrato che la verifica a runtime non è solo un metodo eccellente - e comunemente utilizzato in sistemi critici - per verificare la robustezza, la safety e la sicurezza di un sistema sotto test, ma è anche estremamente utile per verificare le proprietà del comportamento tracciato.

Nel contesto applicativo citato, ciò si traduce nella possibilità di individuare il verificarsi di abusi. Dalle proprietà verificate possono emergere dati interessanti, come la correlazione di eventi criminosi in determinate situazioni, o la creazione di statistiche relative a casi di sfruttamento in una determinata regione geografica. I dati ottenuti potranno essere utilizzati per prendere decisioni strategiche nel contesto in cui opera il sistema, affinché si possa ridurre drasticamente il fenomeno dello sfruttamento del lavoro.

Parole chiave: RV; verifica in fase di esecuzione; sfruttamento del lavoro; rilevamento del crimine, sfruttamento lavorativo.

Contents

1 | Introduction

This thesis work was born from the desire to contribute to an extremely important project that can have a strong impact on people's lives.

The aim of this thesis is to automatically verify the occurrences of crimes, in particular crimes related to labor exploitation, starting from a system that monitors them. We will evaluate the possibility of doing this through the runtime verification method applied to SENTINEL Monitor, a system that monitors labor exploitation in a region in the south of the Netherlands.

With this, we aim to demonstrate how in certain contexts the runtime verification analysis of specific software systems is an extremely powerful method for verifying and evaluating the data acquired by a software system. In particular, it is very helpful in contexts in which a great complexity of heterogeneous events must be analyzed and in a context in which there is a need for certainty of results, such as in contexts in which the acquired data are used as evidences useful for investigative activities.

Providing the results of this type of analysis to state bodies, governmental and beyond, that operate in contexts of this type and deal with justice, laws and fundamental rights of human beings is of extreme importance and can positively influence an evolution in how, too often, these inequities are managed.

1.1. Labor exploitation: a widespread phenomenon

Nowadays, labor exploitation is enormously widespread throughout Europe and beyond. Already in 2012 according to the Global Estimate of Forced Labor presented in a Regional Factsheet European Union by the International Labor Organization (ILO), it was estimated that there were 880,000 people in forced labour, i.e. 1.8 per 1000 inhabitants of the member states of the European Union. It was estimated that 30% of them were victims of forced sexual exploitation and 70% were victims of forced labor exploitation (1.26 per 1000 inhabitants).

Today the situation do not seem to have improved much, but there is a willingness on the part of the states of the European Union to collaborate and discuss analytical approaches to address labor exploitation. Precisely in this regard, the EMPACT Analytical Meeting on Labor Exploitation was held in April 2023 in Paris. The event was organized by the European Labor Authority (ELA), in collaboration with the Netherlands' Labor Authority and France's Gendarmerie Nationale. The focus was to identify an analytical approach to identify and sanction labor exploitation. It this context the use of specific software that can detect and collect in ad hoc databases cases of undeclared work in various sectors is essential.

1.1.1. Labor exploitation in the Netherlands

In the Netherlands, the organization which plays a major role in the prevention of labor exploitation is the **Inspectorate of Social Affairs and Employment**. It plays a key role in monitoring, investigating and prosecuting perpetrators. The Inspectorate focuses on crimes such as undeclared work, underpayment and unsafe working conditions. As labor exploitation is considered a criminal offense under Dutch law and falls under the human trafficking article of the Dutch Criminal Code (Article 273f), it is not covered by the monitoring activities of the Inspectorate. The legal article refers to sexual exploitation, removal of organs and labour exploitation.

Section 273f of the Dutch criminal code:

"Exploitation comprises at least the exploitation of another person in prostitution, other forms of sexual exploitation, forced or compulsory labour or services, slavery, slavery like practices or servitude."

The fact that the Inspectorate's monitoring activities do not focus on labor exploitation does not mean that the results of the monitoring process are not used to address this phenomenon. Conversely, inspections by the supervisory division of the Inspectorate can be transferred to the investigation unit for further criminal evaluation.

In 2014, FRANET, the European Union Agency for Fundamental Rights, conducted an interesting research on the "Severe forms of Labor Exploitation" in the Netherlands [1]. Several categories of experts working in the context of labor exploitation took part in the interviews. They include: monitoring bodies (such as labor inspectorates, health and safety bodies), police and law enforcement bodies, victim support organisations, judges and prosecutors, lawyers, recruitment and employment agencies, workers' organisations, trade unions, employers' organisations and national property experts at Member State level.

The interviews revealed that some sectors and professions are more prone to labor exploitation than others. The victims are mostly workers with low levels of education and those who have to do dangerous or demeaning, labour-intensive or high-risk jobs. The sectors most affected by this phenomenon are: agriculture, catering, transport, especially road transport, construction, domestic maintenance and cleaning services, especially in hotels.

When asked about potential risk factors for migrants, respondents identified **poverty** as the major cause of migrants' vulnerability to becoming victims of exploitation. Poverty contributes to migrant risk in different ways. Basically, poverty pushes people from poorer member states to come to the Netherlands. Compounding the situation is the fact that these migrants are willing to accept lower wages and worse working conditions than would be considered acceptable by Dutch workers, as their income level will still be higher than it would be in their country of origin, which is usually a poorer country.

Secondary but still quite influential crucial risk factors among migrants are lack of pre-departure information, **lack of knowledge of labor rights**, lack of knowledge of support organizations and underlying lack of language skills.

There are also legal and institutional risk factors highlighted by the interviewees. In particular, we note the low risk for offenders of being prosecuted and punished or having to compensate exploited workers. Corruption in the police and other parts of the administration has also been reported.

Particularly influential is the **lack of institutions that effectively monitor the situation** of workers in sectors of the economy where labor exploitation occurs. In reality there are sufficient institutions, but these are not functioning effectively enough, thus perceiving the absence of adequate monitoring of the situation in specific sectors. This is also caused by the **absence of adequate systems** specifically created to monitor such phenomena.

1.2. Related works

1.2.1. The need for systems that monitor criminal events

From the analysis of the context, the need for the creation of systems capable of monitoring abuses regarding labor exploitation emerges.

If on one hand technology is widely exploited for the implementation of electronic monitoring of people who have committed a crime and who therefore must be kept under constant control by the police [2], on the other the entire series of events that precede the identification of a possible crime are still largely based on the human factor and on outdated monitoring systems.

The **police strategy is usually based purely on internal investigations and communications**, not taking into account the possible added value that third parties such as citizens could provide. Precisely for this purpose, a system based on **IoT social devices** has been proposed to support the identification and **tracking of criminal events** [3]. The reports are produced by the type of system that collects data starting from sensors and citizen reports. However, these are not complete and **not completely reliable** as possible users of such systems simply report what they believe they have seen, but this may not correspond to reality. Furthermore, the reported situation is very likely not to include a high level of detail, contrary to what happens in SENTINEL, as will be explained later.

In 2019, an app for detecting labour exploitation in supply chains was proposed by Francisca Sassetti, Silvia Mera and Hannah Thinyane [4]. The proposed app "Apprise Audit" is a mobile solution that acquires **workers' feedback** and offers a real-time summary to the reviewer. The app keeps workers at the center of the entire process, and is connected to a content management system that allows authorized users to collect, analyze and manage data obtained from worker interviews. Although it is certainly a useful app for monitoring laboratory feedback, it proves **not** to be **effective** enough **if workers do not tell the truth**. This could happen quite often if we take into account that they may come from a situation where they are for other reasons forced to submit to these abuses. Like this app, many others in a variety of contexts are created on the assumption that exploited people declare they are exploited. Unfortunately, this often does not happen for various reasons. For example, it may happen because those people do not have the legal right to reside or work in a certain country and are therefore afraid of being sent back to their place of origin or because their housing condition strictly depends on their employer and therefore they risk being remains homeless.

1.2.2. Runtime verification: a powerful method for analyzing crimes

Runtime Verification (RV) is a type of analysis usually performed on software systems. It takes care of extracting information from the system at runtime and verifying them.

The power of RV analysis compared to traditional verification techniques, such as theorem proving and testing, is first of all the ability to produce the analysis **while the system**

is running and then the possibility - not to underestimate - to decide exactly which requirements to verify on **real data** produced by the system itself. This makes RV easily scalable and allows us to focus on the characteristics that we are interested in analyzing in detail.

The formulation of the properties to be verified is crucial in this type of analysis and this is why, in their drafting, both external factors to the software system (for example the context in which the software is used, which can be legislative , financial, etc.) and internal factors of the system, i.e. the technical requirements necessary for its correct functioning need to be taken into account.

Given all the reasons above, over the years, the research of monitoring techniques has become increasingly of interest to the scientific community so much so that in 2014 the international **Competition on Runtime Verification** (CRV) was created with the aim of promoting the study and evaluation of new software runtime verification tools [5].

Based on gray literature research, runtime verification techniques are mostly used in in **critical systems** where the presence of errors is not tolerated [6]. This method is in fact used to **verify the robustness of the system** at different times in which the system is running. Having to ensure the **security** and **safety** of such systems pushes the use of formal languages capable of guaranteeing the correctness of the verified properties.

In the existing literature we have not found another application of runtime verification to verify the occurrence of crimes related to labor exploitation, but **similar verification methods** applied to **similar contexts** have certainly been discovered.

For example, the use of a similar formal approach, based on a proposed logic for **computer forensic investigations**, has been found. The proposed language, $S-TLA^{+}$, allows an **unambiguous description of the evidence**. However, the paper proposes its use to **allow hypotheses** to be made whenever there is some **lack of detail** to demonstrate part of an attack scenario [7].

A collaborative approach for incorporating forensic case data into crime investigation using **criminal intelligence analysis** and **visualisation** has been proposed by Rossy and Ribaux [8]. The proposed framework contains useful components for creating a **suitable visual model for expressing investigative problems**. They showed how visualization methods can contribute as external aid to support collaborative thinking and support joint decision-making. However, it is difficult to implement in practice and the process may be the **result of unconscious cognitive biases**.

An article proposed a new method of visualizing and **analyzing crime patterns based**

on geographic data using Formal Concept Analysis, a method that analyses data which describe relationship between a particular set of objects and a particular set of attributes. This method considers the set of common and distinct attributes of crimes such that categorization occurs based on related crime types and builds more defined and conceptual systems for analyzing geographic crime data that can be easily visualized and analyzed intelligently by computer systems [9]. Although it is useful for classifying crimes in a certain geographic area, it **tells us nothing about the number of crime occurrences**, contrary to what can be monitored with RV.

As we have seen, RV is often used to ensure the quality of complex and safety-critical systems. According to Bartocci and Falcone [10], much effort is still needed to make RV attractive and viable methodologies for industrial use. However, some proposal of its **application in the industry** have been found in literature [11].

For example the application of runtime verification, through a Prolog monitor, to a **simplified mobile healthcare IoT system** for the management of diabetic patients composed of sensors, actuators, Node-RED logic on the cloud and smartphones was proposed.[12]. DiaMH is an IoT system that **monitors the patient's glucose level**, sends alerts to the patient and doctor when the glucose level trend is outside a pre-established target range, and adjusts the insulin dosage. This is an excellent example of how runtime verification can be used to monitor a possible event in real time; in this case, the exceeding of the maximum insulin threshold in the patient's blood has been monitored.

With this work we want to apply the RV method to a **legal context**: verifying the behavior recorded by a system to identify the occurrences of crimes being monitored. In the specific context of SENTINEL, we wanted to **automatically identify the violations of labor protection regulations**. These series of correlated violations can be discovered and possibly transformed into statistics in real time.

1.3. Research questions

In this research we will focus on the use of the runtime verification method applied on a software system dealing with many heterogeneous events and in a context where there is a need for certainty of the results. Specifically, our evaluation will take place taking SENTINEL into consideration.

The research questions we intend to answer are shown below:

RQ.1 : Is runtime verification a suitable method to verify system properties to

automatic identify the violations of labor protection regulations?

RQ.2 : What is necessary and **interesting to verify** in a system with a lot of heterogeneous data through runtime verification?

RQ.3 : How does the **context** influence the formulation of properties regarding the properties to be analyzed?

1.4. Structure

This thesis work is divided into three parts. In the first part, traditional software system verification techniques will be introduced and compared to the runtime verification analysis. We will focus more in depth on RV analysis: the basic concepts regarding the runtime analysis of a software system will be explained and the properties that can be analyzed in this type of systems will be presented. Then, the monitoring tool used to apply this type of analysis to a real system will be introduced.

In the second part of the thesis, an evaluation of the proposal will be presented through the application of runtime analysis to **SENTINEL**. Finally, the results obtained from the runtime analysis carried out on the particularly relevant system will be discussed.

In the last part, conclusions and final considerations will be discussed.

Specifically, the structure of the thesis is as follows:

- ***Chapter 1 - Introduction***
- **PART I: Runtime verification analysis**
  - ***Chapter 2 - Runtime verification analysis of a software system***
- **PART II: SENTINEL, an evaluation of the proposal**
  - ***Chapter 3 - SENTINEL: a system to fight labor exploitation***
  - ***Chapter 4 - Analysis of the monitoring problem in SENTINEL***
  - ***Chapter 5 - Verification of events in SENTINEL through MonPoly***
  - ***Chapter 6 - Results***
- **PART III: Conclusions**
  - ***Chapter 7 - Conclusions and final considerations***

Part I

Runtime verification analysis

2 | Runtime verification analysis of a software system

The concept of runtime verification was created to define a type of formal analysis that can be performed on systems to verify software and hardware components. Some synonyms are: runtime monitoring, trace analysis, dynamic analysis, etc.
We can define runtime verification as a formal method that complements other traditional system verification techniques [5].
This type of analysis is particularly important since it is done **at runtime** and therefore it can be used to check the current execution of the system to be sure that it respects the **correctness** criteria imposed [13].

2.1. Traditional verification techniques versus RV

Traditionally, three main verification techniques are considered: *theorem proving*, *model checking*, and *testing* [14].

2.1.1. Theorem proving

Theorem proving is one of the traditional formal techniques used to verify the correctness of systems. **Formal verification** means mathematically demonstrating the correctness of a design against a formal mathematical specification. It allows to prove the correctness of programs in a similar way to how a proof in mathematics shows the correctness of a theorem and is a process that is usually done manually.

The main problem with this type of method is that it is necessary to formally model the system in order to then be able to subject it to verifications. This makes traditional formal methods certainly more complex and model formation is more **prone to errors**. RV, on the other hand, greatly reduces this complexity, as it analyzes few executions of the system and it does it at runtime, at the expense of less coverage.

Furthermore, experience suggests that highly automated techniques are much easier to learn, and often more productive to use, than methods such as theorem proving that rely more on human interaction [15].

2.1.2. Model checking

Model checking, which is an automatic verification technique, is mainly applicable to finite state systems.
In computer science, model checking or property checking is a formal verification technique that exhaustively checks hardware circuits and network protocols against the desired properties [16]. The desired properties are formalized and expressed, for example, in the form of temporal logic. An abstract model of the system to be evaluated is created and finally a formal verification is applied. The outcome of the verification tells us if the specific properties are satisfied (*true* result) or violated (*false* result) and a counterexample is produced, if the *time is exceeded* or the *memory is exhausted* during the test, a state explosion problem is displayed.

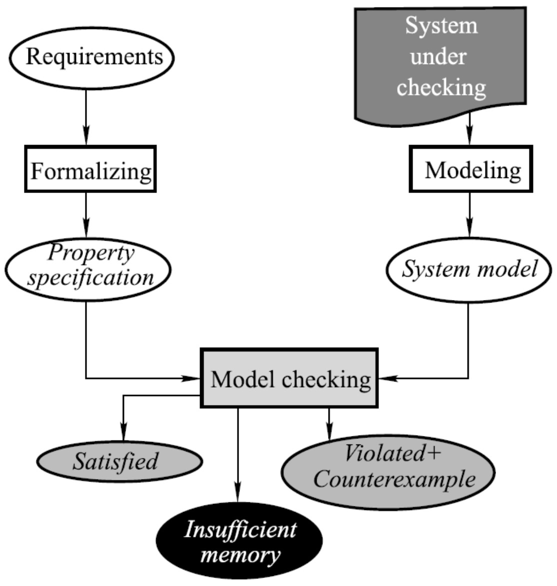


Figure 2.1: Basic principle of model checking [16].

Runtime verification has its origins in model checking, but there are some important differences to note:

- Model checking usually analyzes **infinite traces**, while runtime verification analyzes **finite executions** of the system. Furthermore, model-checking is generally **undecidable** when the verification is applied to particularly expressive codes or models. That is, there is no algorithm by which it is possible to decide in a finite number of steps whether each proposition that can be formulated in the problem is true or false, i.e. satisfied or violated;

- Runtime verification deals only with executions observed by the **real system**. This means that it is also applicable to systems for which an ad hoc verification model has not been built, which is instead necessary if the model checking method is used. In model checking not only an **representative model** of the system must be built, but it must also be built before any executions of the system are made and checked;
- A problem that should not be underestimated is the aforementioned **state explosion** from which model checking suffers. Analyzing all executions of the system means considering generating the entire state space of the underlying system, which is often huge and consequently consuming a large amount of memory (and time for state generation). Considering a single run, on the other hand, usually does not lead to memory issues as when monitoring online only a **finite history** of the executions needs to be stored [14].

2.1.3. Testing

Today testing is the most used technique to verify the correctness of a software. Essentially, a large number of test cases are generated and then verified that the outcome is as expected. This technique is mainly used to discover any bugs present in the analyzed software system and fix them in the development phase.

Looking closely, testing and runtime verification are tightly linked, in fact both methods check the **behavior of the system**. The difference lies in the fact that RV checks the system's behaviour at runtime by listening to system events, while testing takes care of generating an adequate number of test cases during the development phase to be verified [17].

Another similarity between the two methods is given by the fact that RV does not consider every possible execution of the system, but only a finite part of the possible executions [14]. Similarly, testing creates a finite number of test cases that verify the correctness of a finite number of properties with a finite number of executions. The test covers a wide range of different methods, created specifically for a system.

In both cases, however, the verification of the properties is **incomplete**, but it is still much faster than model checking and is often sufficient to guarantee the correctness of the system itself. On the other hand, providing an adequate set of input sequences to "exhaustively" test a system is rarely considered in the scope of runtime verification, while it is desirably pursued in testing.

2.2. Monitoring the behavior of a software system: how RV analysis works

To understand how the runtime verification method works, it is necessary to first introduce some key concepts.

First of all, the method presupposes the observation of a system to understand how it behaves and how its state changes over time and following certain actions. This is what we call an event [5].

Definition 2.2.1: Event

An *event* is an observation about the system.

The event that is observed depends on the level of abstraction of the system we want to consider. Knowing how much information we need with respect to that observation is fundamental because it influences the granularity of the result we want to obtain through the verification.

With the RV method, not a single event is usually analyzed, but a set of events, i.e. a (finite) succession of observations [18], which form a single run of the system. This set of events is called trace.

Definition 2.2.2: Trace

A *trace* is a sequence of observed events.

The key element of RV analysis is called monitor. The monitor is the entity responsible for observing the runtime behavior of the system. It determines whether the system satisfies the given properties. If there is sufficient evidence emerging from the trace analysis, the monitor reaches a verdict of acceptance or rejection [13].

Definition 2.2.3: Monitor

A *monitor* is a device that reads a finite trace and yields a certain verdict.

We can differentiate monitors into two categories:

- **online** monitors: they monitor the ongoing run of the system;
- **offline** monitors: they monitor the functioning of the system starting from a finite

set of traces already recorded.

In runtime verification, monitors are usually generated automatically from some specification written in a given formally defined specification language [14].

At runtime, a monitor attempts to assign a verdict to the current event by determining whether or not that **event is a model of the monitored property** [5]. In other words, if the monitored property, which is typically a logical formula, is true according to an interpretation, which in this case is the observed trace, then that interpretation is called model of that formula.

After having clarified the key points of the RV analysis setup, we can now give a complete definition of it.

Definition 2.2.4: Runtime Verification

Runtime verification is the discipline of computer science that deals with the study, development, and application of those verification techniques that allow checking whether a run of a System Under Scrutiny (SUS) satisfies or violates a given **correctness** property. Its distinguishing research effort lies in synthesizing *monitors* from high level specifications [13].

Figure 2.2 shows the setup necessary to evaluate the behavior of a software system through an RV analysis.

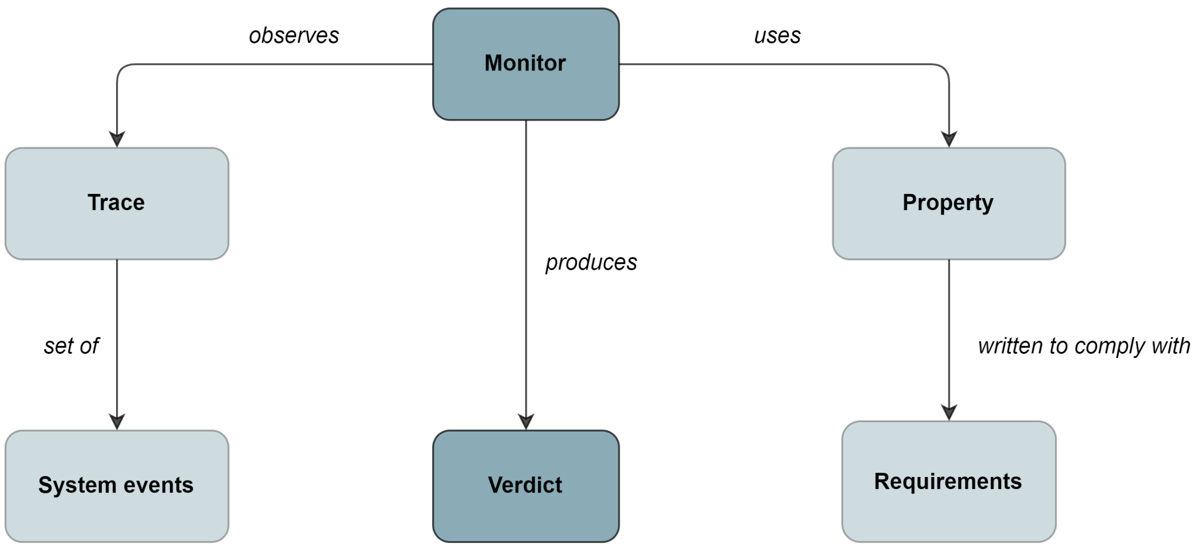


Figure 2.2: Runtime verification setup.

2.3. Properties to verify at runtime

There are several characteristics that a software should have to be considered good. Intuitively, good software is one that meets the needs of its users, performs its intended functions reliably, and is easy to maintain.
Some of the non functional requirements to consider when creating software include functionality, usability, reliability, performance, security, maintainability, reusability, scalability, and testability.

But which of these characteristics can be verified at runtime? In general, verifying a software system means answering the question: *"Did we achieve what we were trying to achieve?"*, i.e. software is correct - its results are correct with respect to the function it must perform - and it meets stakeholder needs. The first aspect is considered in the context of verification while the second aspect is part of the validation.

In fact, the primary goal of the monitor in runtime verification is to detect violation or satisfaction with respect to the given specification [19].

Often in runtime verification of a system, the **safety** property is considered. It includes a series of measures aimed at eliminating the production of irreparable damage within the system. By evaluating it with runtime verification, it is possible to identify safety properties violations of the software system under scrutiny, or verify the occurrence of a system defect.

Verification at runtime can also be used to track what is happening within the system, such as data entry at a particular time of day or from a particular geographic location. It can be used to verify that some **temporal actions** are performed only after others have been completed, for example we could have a system in which access by a user must always precede the entry of personal data, or we could have restrictions on the type of user who can actually log in, and so on.

In general, when the verification problem is decidable, regardless of aspects related to the complexity of the calculation, every property that can be formalized and therefore transformed into a logical formula can also be verified. However, **some violations of the properties may never occur** in the system under analysis, or may not occur in the **current execution** of the system being taken into consideration, i.e. in the trace under analysis.

2.3.1. The influence of the context

It is clear that it is necessary for a system to respect all the basic technical specifications for it to function in a proper way, and it is desirable that this also respects the qualitative specifications.

But **when do desirable specifications become crucial**?
Based on the application context of the system under analysis, it is important to determine which properties and therefore which non-functional requirements of the system must necessarily be taken into consideration and verified.

For example, if we are in a context where software is used to collect data, we will pay particular attention to how this data is processed and the existing privacy regulations in a particular country. If this data also has legal validity, can be used in court and in other judicial contexts by law enforcement, we will also pay particular attention to the legislative aspect, we will make sure that the entity that collects, holds and uses those data can actually do so without incurring sanctions or particular legal charges. We must then consider that it is not certain that the data collected can be used freely, but we must be sure that they are used for all and only the purposes for which the legislative or government body has given permission for use. We must take into account that the various permits vary from country to country, and often also from region to region.

Another example: if the software to be verified is used in a space context, the reliability of the system is certainly one of the fundamental properties to take into consideration, since, if some failure were to occur that was not correctly managed, this could ruin entire space missions, without considering the damage this would cause in terms of costs.

2.3.2. Typical use of runtime verification in existing literature

Runtime verification analysis is used today in applications where aspects relating to the **safety** and **security** of the system must be guaranteed. Indeed, the systems under scrutiny are often complex and critical systems in which errors are not tolerated. The correctness of the functions carried out by these systems is important and necessary for the context in which they operate [6].

The monitoring system is also used to verify the **robustness** of the system over time, i.e. the ability of a software system to continue functioning correctly and reliably even in the face of unexpected or abnormal inputs or situations. It is possible, through online monitoring, to verify the robustness of the system while it is running, but also through offline monitoring, through which the entire trace system is available offline for analysis.

This means that it is possible to go back and forth over time to calculate its robustness to a given specification [20].

In fact, the correctness of real-time systems in the context of runtime verification depends not only on the logical result of the calculation but also on the moment in which the results are produced [21].

2.4. What should we monitor?

As we have seen, the properties of a software system that can be analyzed are many if we consider not only the technical aspects of the system, but also the qualitative aspects desirable from the system. If we then also add the context factor in which the system must be used we see that the granularity of the properties increases more and more and leads us to obtain, if desired, a myriad of different possible scenarios to analyze and verify.

Choosing what to monitor in the system strictly depends on the application context and the entities involved in the process.

It is important to understand what particular aspect we need to focus on. This must start from well-defined reasons which can be:

- **technical** reasons:
  - if one decides to use a certain specification language, he/she can monitor everything that can be expressed with that language,
  - if one decide to use a particular verification method, this may be limiting for verifying some properties, but potentially suitable for verifying others.
  - a key factor to take into consideration is also what data we have available, the method for accessing this data and consequently the response latency to retrieve them, which could be non-negligible.
- **context-related** reasons:
  - if following an initial analysis we realize that in a given context it is more important for a system to comply with some requirements rather than others, we can start from those that are considered most important.
- **functional** reasons (purpose and use of the software system):
  - if the system was created for a specific purpose, it is necessary to verify that that purpose is best pursued. For example, coming back to the legal content

anticipated before, which will be the basis from which we will start in the next chapters to evaluate our example system, the purpose of the system is to collect data on labor exploitation, so we will pay particular attention to how these data are collected, that is, in compliance with current regulations, but also on how they are used and for what purposes, which may include legal actions or simple statistics used to make strategic decisions.

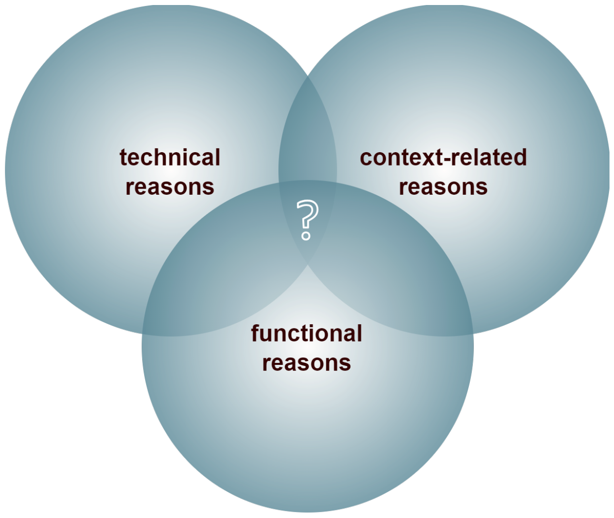


Figure 2.3: How to choose what to monitor.

2.5. Monitorability: what can we monitor?

Since everything that needs to be monitored is translated into the chosen specification language, the latter has a huge influence on what can be monitored. A certain specification language may not be adequate to define a given property. Some specification languages are better suited to specification finite trace sets (e.g., state machines) while others are better suited to specifying infinite trace sets (e.g., temporal logic) [5].

Therefore, the limit of what can be monitored strictly depends on the **expressiveness** of the specification language.

Monitors for real-time property verification can be defined using a variety of languages, ranging from formal languages with strong guarantees, such as temporal logic, to very expressive languages that allow the definition of monitors with rich verdicts, such as scripting languages and programming. If we move towards the use of very expressive languages, however, we will have rich verdicts, but fewer formal guarantees [22].

Part II

SENTINEL, an evaluation of the proposal

3 | SENTINEL: a system to fight labor exploitation

In this chapter we will introduce SENTINEL, a software system whose aim is to monitor labor exploitation in the North Brabant region, a province in the south of the Netherlands, to reduce and hopefully eliminate this type of threat. In fact, the creation of SENTINEL, is a direct consequence of the emerging situation of labor exploitation in the Netherlands previously analyzed. Finally, we will show the first contribution of this thesis work: we will propose a **system modification** to integrate the logs representing system events. The latter will be used to allow the identification of the crime regarding labor exploitation at runtime.

3.1. SENTINEL Monitor

SENTINEL is a software system which aims to combat the spread of labor exploitation [23]. It allows the collection of data relating to labor exploitation in the North Brabant region of the Netherlands.

SENTINEL was created in response to the lack of a system that monitors this extremely widespread phenomenon. When drafting the requirements, the opinions of the municipality of Eindhoven and all the related legal aspects were taken into consideration.

It was designed to be used mainly by local authorities such as the police and all those entities responsible for producing reports on exploitation and investigating them. We therefore assume that users of the system are trusted by default and the truth of the data entered is guaranteed by the status of authorized entity.

3.1.1. Architecture design

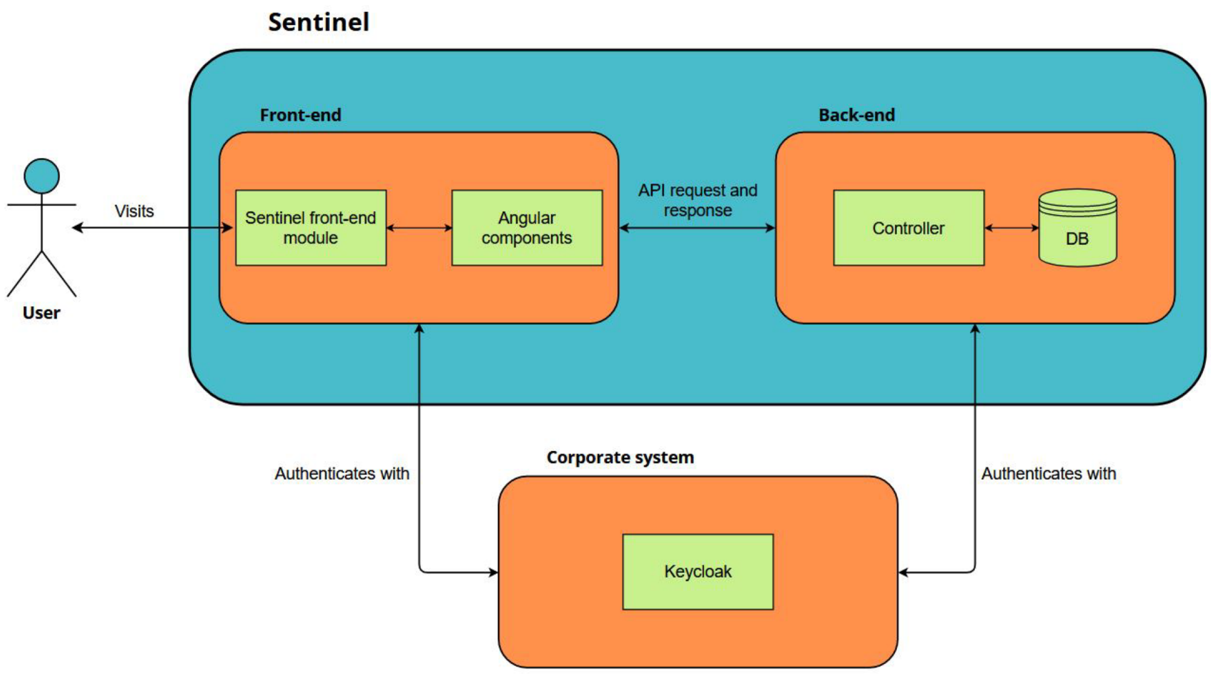


Figure 3.1: The global architecture design of the system [23].

In figure 3.1 the overall architecture design of the system is showed. SENTINEL has two distinct subsystems:

- the front-end which is responsible for interacting with the user who enters the data,
- the back-end where the data is processed and then saved in the database.

The user authentication process takes place through Keycloak, a single sign on solution for web apps and RESTful web services.

The front-end is developed with **Angular** and **Typescript**, while the back-end is developed with **Java** and the Spring framework. More specifically **SpringBoot** has been used, an open source Java-based framework used to create a micro Service and to build stand-alone and production ready spring applications.

3.1.2. Database

Permanent data used by the system must be stored in a database, so that it can be consulted and used by the system itself at any time. All data relating to each report will be saved, both those of the signaler and those relating to the report. Among the fundamental principles on which the Sentinel system is based are **data security** and **integrity**; this implies that the storage of all data must be safe and secure and that the system will guarantee the integrity of the data, in fact the data will be consistent and accurate throughout its life cycle. In short, the system will ensure that data is stored exactly as intended to prevent unauthorized changes to the information [23]. For the design of the database **SQL** and **MySQL** have been used.

3.1.3. Exploitation data collected

The user must initially log in to access the system, as shown in the figure 3.2. Only certain, verified people have permission to access the system.

Then, in figure 3.3 the data regarding the address on which a report of labor exploitation must be made is entered. Data relating to geographical location are necessary for any investigations by the police and for this reason, entering the address where the exploitation situation has been observed is mandatory.

In the end, as shown in figure 3.4, the data related to the situation observed are entered.

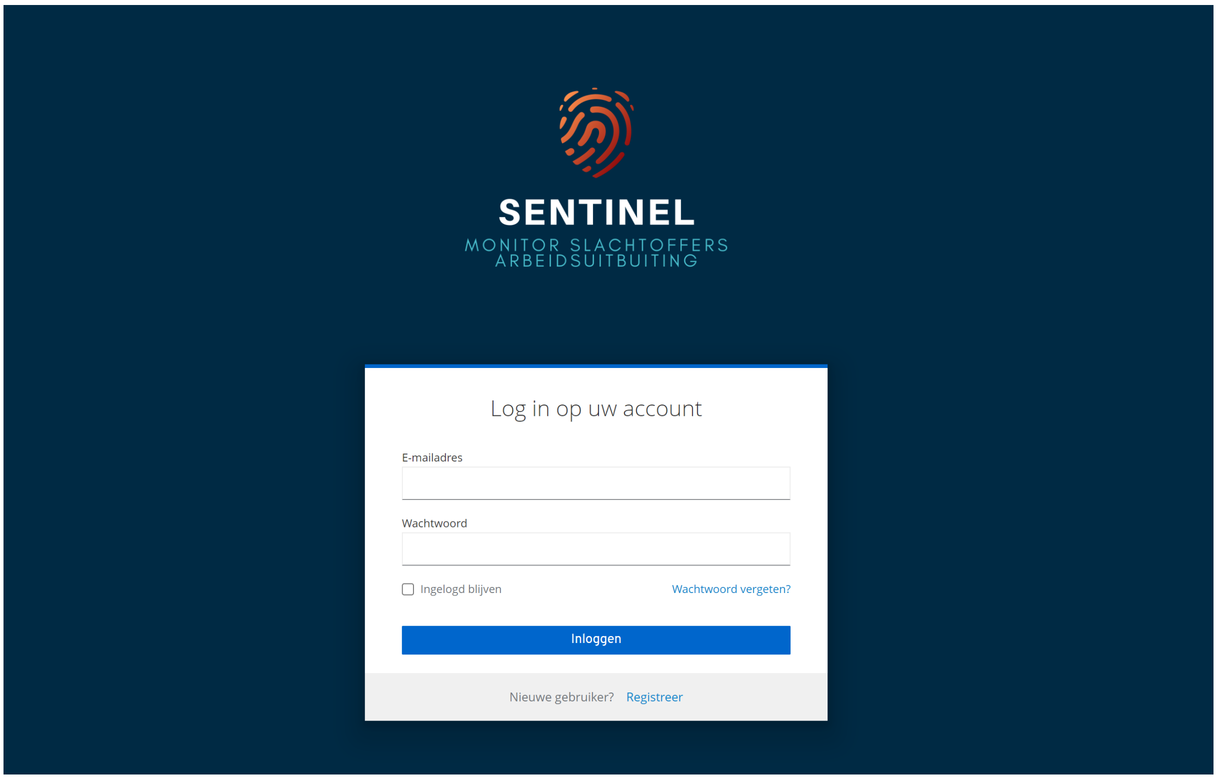
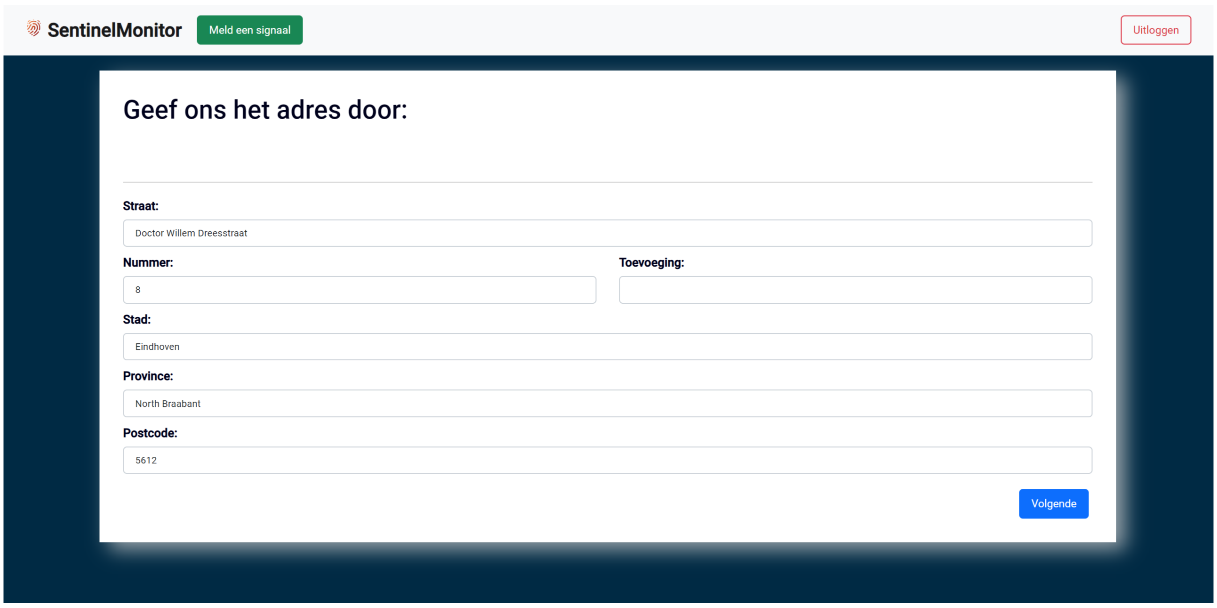


Figure 3.2: Login page.



Geef ons het adres door:

Straat: Doctor Willem Dreesstraat

Nummer: 8

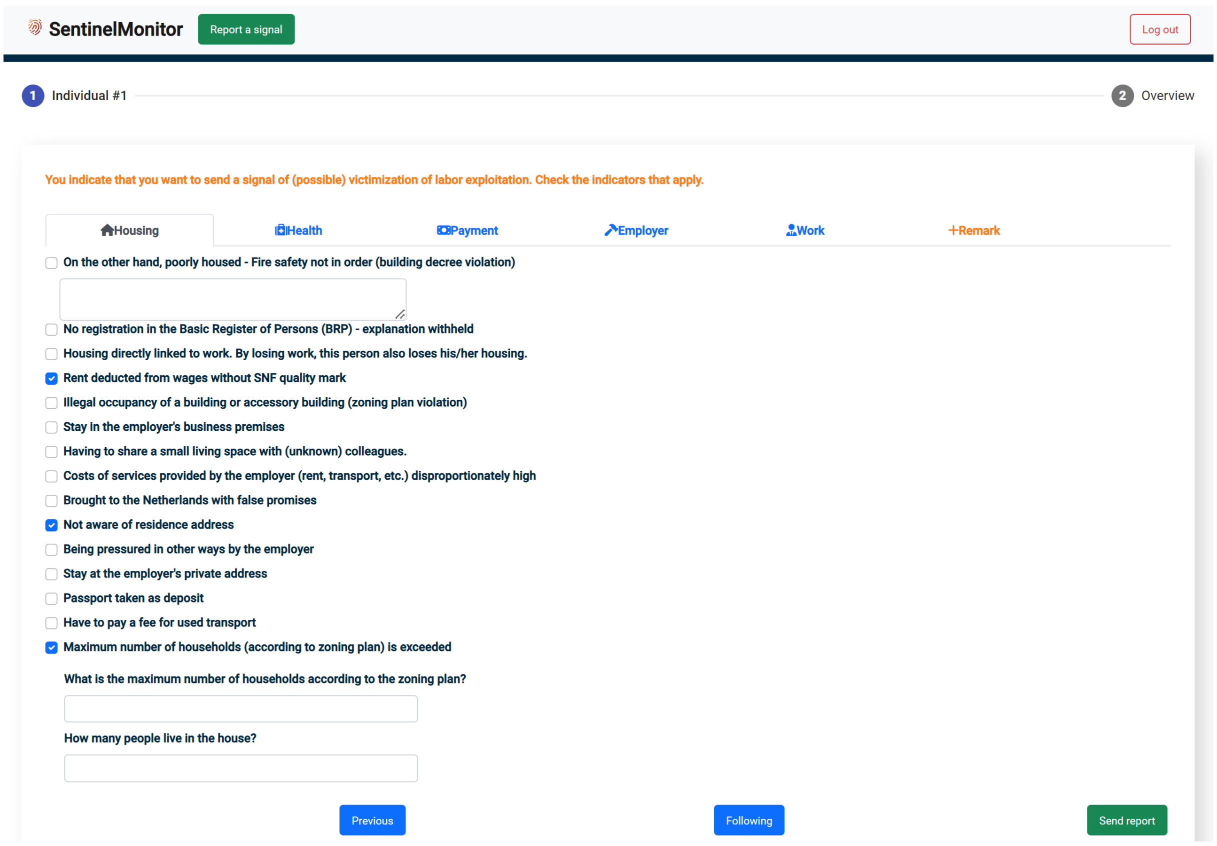
Toevoeging:

Stad: Eindhoven

Province: North Braabant

Postcode: 5612

Figure 3.3: Page collecting location data.



You indicate that you want to send a signal of (possible) victimization of labor exploitation. Check the indicators that apply.

- [ ] On the other hand, poorly housed - Fire safety not in order (building decree violation)
- [ ] No registration in the Basic Register of Persons (BRP) - explanation withheld
- [ ] Housing directly linked to work. By losing work, this person also loses his/her housing.
- [x] Rent deducted from wages without SNF quality mark
- [ ] Illegal occupancy of a building or accessory building (zoning plan violation)
- [ ] Stay in the employer's business premises
- [ ] Having to share a small living space with (unknown) colleagues.
- [ ] Costs of services provided by the employer (rent, transport, etc.) disproportionately high
- [ ] Brought to the Netherlands with false promises
- [x] Not aware of residence address
- [ ] Being pressured in other ways by the employer
- [ ] Stay at the employer's private address
- [ ] Passport taken as deposit
- [ ] Have to pay a fee for used transport
- [x] Maximum number of households (according to zoning plan) is exceeded

What is the maximum number of households according to the zoning plan?

How many people live in the house?

Figure 3.4: Page collecting exploitation data (translated in english).

There are five macro categories that cover different illegal situations related to:

- **housing**: this category includes violations relating to the home of the exploited person. For example, the person may not have a properly registered contract, or the home may not comply with the safety rules, or this person may have to share a small living space with colleagues or stay in the employer's business premises, or he or she may have been forced to leave her passport as a deposit;
- **health**: this category includes violations relating to the health of the exploited person. For example, the person may have been forced to work or be abused by the employer or not have medical insurance;
- **payment**: this category includes violations relating to the payment of the exploited person. For example, the person may not have received a salary and may not even have a contract to prove he or she is entitled to it, or may be paid illegally, or may have to return part of the money earned to the employer illegally;
- **employer**: this category includes violations by the employer against the worker. For example, the worker could suffer abuse from the employer, he or she could be in a situation where he or she is in a position in which he or she cannot do anything else than to undergo exploitation;
- **work**: this category includes workplace abuse against the worker. For example, the worker may be forced to work excessive hours that are not permitted by law, he or she may have been brought to the Netherlands with false promises of work, he or she may have to carry out dangerous or illegal activities..

In Appendix A the detailed possible signs of victimization of labor exploitation present in SENTINEL are listed.

3.2. SENTINEL limitations

In SENTINEL, the occurrence of crimes cannot be verified automatically and at runtime. In fact, the application was created mostly to support the investigative action of the police. In this work we want to take a step forward and **make SENTINEL capable of doing runtime analysis**.

The intent is to identify the properties for SENTINEL runtime verification that must necessarily be respected and verify them through a monitoring tool. The specific properties to be verified depend on the needs and objectives of the system as well as the applicable properties and regulations in the North Brabant region.

3.3. System extension with log files

Event collection is a fundamental part of the runtime verification process of a system, as events represent the actual behavior of the system to be verified. Collect them and transforming them into logs means spending time and memory for the system to process the data and display them in some files so that they can be analyzed offline. This means that the log creation process affects the **complexity** and the **performance** of the system and this must be taken into consideration when deciding to approach the runtime verification of a critical system. It is also true that, in general, the workload required by the system for processing log files is usually negligible when compared to all the rest of the work carried out by the system for its operation.

Specifically, the SENTINEL log file creation process has been integrated into the existing software system. The log files were created following requests performed on the system and contain data regarding the properties we want to **verify** and **track** to possibly automatically verify the occurrence of crimes at runtime.

It is important to understand what to include in the log files, what data is necessary and useful to keep for verification based on the property we want to analyze.

In this case, we decided to use the following approach: save the **entire situation** reported in a report on a single log entry. This means saving much of data entered by the user of the system regarding a report of possible victimization.

The data that we have decided to keep in the log files are data concerning the **position** in which the possible victimization is taking place and the data concerning the **victimization**, i.e. specifically what type of victimization is taking place and what the affected categories are.

A log file generated by our system will contain one or more entries of the type:

- *<**Timestamp**> **situation(**int idReport, int idAddress, String city, String province, boolean housingExpl, boolean healthExpl, boolean paymentExpl, boolean laborExpl, boolean employerExpl, boolean householdsExceeded, boolean workingHoursExceeded, boolean paymentExceeded, boolean noContract, boolean noVacation**)***

In Table 3.1 the parameters listed in a log entry are shown. A brief explanation of their meaning is given and if they refer to one or more categories of exploitation these are visible in the last column. For example, the parameter "householdsExceeded" refers to a type of violation related to housing, therefore it refers to the "housingExpl" category.

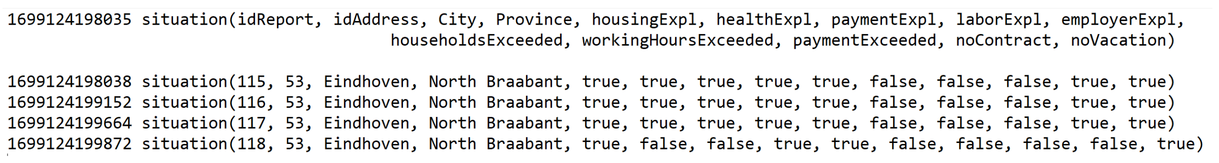
Parameter	Explanation	Referring to
idReport	id of the report submitted	-
idAddress	id of the address in which the exploitation take place	-
city	city in which the exploitation take place	-
province	province in which the exploitation take place	-
housingExpl	category of violations relating to the home of the exploited person	-
healthExpl	category of violations relating to the health of the exploited person	-
paymentExpl	category of violations relating to the payment of the exploited person	-
laborExpl	category of violations relating to the work of the exploited person	-
employerExpl	category of violations by the employer against the exploited person	-
householdsExceeded	violation of the maximum number of households (according to zoning plan) which is exceeded	housingExpl
workingHoursExceeded	violation of the maximum number of working hours (according to Dutch Law) which is exceeded	laborExpl

paymentExceeded	violation of the payment term (according to Dutch Law) which is exceeded	paymentExpl
noContract	violation for not having a registered employment contract (illegal according to Dutch Law)	paymentExpl, employerExpl
noVacation	violation for not having any contractual vacation days (illegal according to Dutch Law)	paymentExpl, employerExpl, laborExpl

Table 3.1: Log entry parameters.

As regards the parameters chosen to be included in the log files and therefore in SENTINEL verification, the choice fell on parameters that could be objective, often of a numerical type. The choice was made only after an analysis of the Dutch exploitation law. Some violations relating to the chosen parameters can be found in table 5.1.

We will therefore have log entries as shown in the figure 3.5. Note that the first line in figure 3.5 is only shown to recall what the parameters of the event refer to and it is not really part of the analysis.



```
1699124198035 situation(idReport, idAddress, City, Province, housingExpl, healthExpl, paymentExpl, laborExpl, employerExpl,
                                   householdsExceeded, workingHoursExceeded, paymentExceeded, noContract, noVacation)

1699124198038 situation(115, 53, Eindhoven, North Braabant, true, true, true, true, true, false, false, false, true, true)
1699124199152 situation(116, 53, Eindhoven, North Braabant, true, true, true, true, true, false, false, false, true, true)
1699124199664 situation(117, 53, Eindhoven, North Braabant, true, true, true, true, true, false, false, false, true, true)
1699124199872 situation(118, 53, Eindhoven, North Braabant, true, false, false, true, true, false, false, false, false, true)
```

Figure 3.5: Log file example.

No personal data or data attributable to specific people is saved or taken into consideration, as our aim is to **trace the occurrence of events**, so as to be able to obtain a general picture that shows how much these victimization phenomena are present, on which territories and for which categories specifically in real time.

Figure 3.6 shows the new SENTINEL controller from an high level point of view.

"Check Address" is in charge of retrieving all the reports related to an address if there are any and if not it directly processes and saves the new inserted address, which is the location in which the exploitation took place.

"Save Reports" is in charge of saving the report with the data entered by the user about the possible violation encountered thanks to a service called "reportService". The logger, whose creation is one of the contributions of this thesis, is shown in green in the figure and must be seen as an extension of SENTINEL. It has been added as a functionality in the "reportService". It was in fact created to monitor the events recorded by the system for runtime verification.

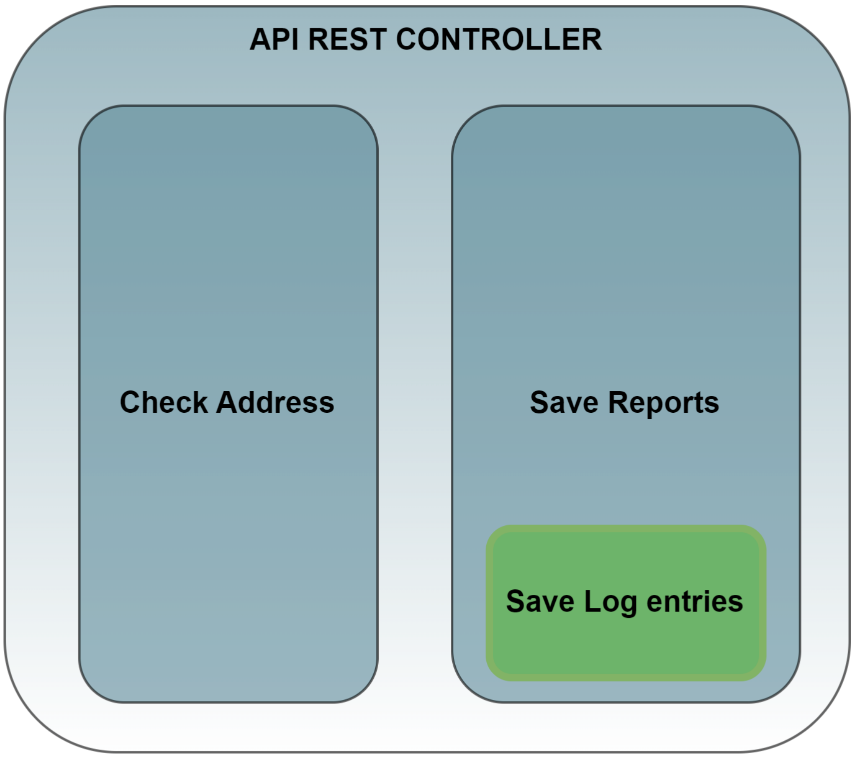


Figure 3.6: High-level view of the new SENTINEL controller.

4 | Analysis of the monitoring problem in SENTINEL

Before delving into the runtime verification of the SENTINEL system, it is necessary to make an analysis of the monitoring problem in SENTINEL carried out with respect to these three factors introduced previously: technical reasons, context-related reasons and functional reasons. Then, it is necessary to understand which tool to use for the verification.

4.1. The monitoring problem in SENTINEL

Starting from the **functional reasons**, i.e why we want to monitor the SENTINEL system, we identify our purpose in automating the process of observation and detection of the occurrences of violations relating to labor exploitation or crimes in general.

Based on the **context**, the following properties will be taken into account:

- **Counting** of specific occurrence of criminal events;
- **Timing** property related to specific events;
- **Security** property related to specific events;
- **Compliance** of properties with respect to the Dutch Law;
- **Privacy** GDPR regulation.

As regards the data to monitor and therefore the **technical reasons** to take into consideration, thanks to SENTINEL we have data divided by area in which the exploitation took place, these may concern the home, health, work, payment or employer category. We can and want to keep track of all this data that cannot be directly traced back to people, but can be used for any subsequent observations and for the creation of statistics.

For the monitoring, since we are in a context in which there is a need for certainty of results - the monitored data can be used as evidences useful for investigative activities -

we will use a type of formal language with strong guarantees, that is temporal logic.

4.1.1. Specification language used for runtime verification of SENTINEL

In runtime verification, which is heavily inspired by model checking, a correctness property is automatically translated into a monitor [14]. Correctness properties are usually formulated in some variant of linear temporal logic, such as LTL [24].
In the case study examined, i.e. in SENTINEL, we observe the events of a system at runtime and therefore we are in the case of analysis of finite traces. However, the size of such traces could, in the real world, be large. For this reason we prefer to use a particular type of LTL for verifying properties.

Linear Temporal Logic is one of the most important logics for software and hardware verification and it is particularly useful when specifying **restricted regular properties** of **discrete time** systems. However, we must keep in mind that not all properties to be monitored during execution can be expressed in LTL.For example LTL can only define non-counting languages. Therefore the following counting language cannot be expressed in LTL: "*An arbitrary but even sequence of α symbols, followed by an infinite sequence of p symbols*" [25]. Also, the "possibility" is not expressible in LTL. Given the formula φ: *Whenever p holds, it is possible to reach a state where q holds*, φ cannot be expressed in LTL.

Linear Temporal Logic: a logic for reasoning about the execution path of a system

Linear Temporal Logic is an extension of modal logic and provides a **linear**, discrete, **future-oriented** and **infinite organization of time**.

LTL provides a number of operators to express temporal properties, i.e. constraints regarding the expected order of the events [26]. Temporal logic has two kinds of operators: logical operators, i.e. logical connectives as standard propositional logic, and modal operators. In table 4.1 the unary temporal operators of LTL are listed and in 4.2 all operators derived from the table 4.1 are shown.

Binary operators

Operator	Explanation
ψ **U** φ	**Until**: ψ has to hold at least until φ becomes true, which must hold at the current or a future position.

ψ **R** φ	**Release**: φ has to be true until and including the point where ψ first becomes true; if ψ never becomes true, φ must remain true forever.
ψ **W** φ	**Weak until**: ψ has to hold at least until φ; if φ never becomes true, ψ must remain true forever.
ψ **M** φ	**Strong release**: φ has to be true until and including the point where ψ first becomes true, which must hold at the current or a future position.

Table 4.1: Binary temporal operators of LTL.

Unary operators

Operator	Explanation
$\bigcirc\varphi$	**Next**: φ has to hold at the next state.
$\bullet\varphi$	**Previous**: φ has hold at the previous state.
$\Diamond\varphi$	**Eventually**: φ eventually has to hold (somewhere on the subsequent path).
$\blacklozenge\varphi$	**Once**: φ once has to hold (somewhere on the previous path).
$\Box\varphi$	**Always**: φ has to hold on the entire subsequent path.
$\blacksquare\varphi$	**Past always**: φ has hold on the entire previous path.
$\forall\varphi$	**All**: φ has to hold on all paths starting from the current state.
$\exists\varphi$	**Exists**: there exists at least one path starting from the current state where φ holds.

Table 4.2: Unary temporal operators of LTL.

MFOTL

The properties of SENTINEL system that we will analyze will be translated in a particular type of temporal logic: the **metric first-order temporal logic** (**MFOTL**).

The first-order characteristics of MFOTL make it particularly suitable for formalizing **relationships** between event arguments [27]. First-order temporal logic allows the expression of complex temporal properties within the context of first-order logic, which is an extension of propositional logic to include universal and existential quantifiers, variables, functions and higher order predicates.

Metric Temporal Logic is an extensions of LTL that allow us to express timing constraints on temporal operators: While in LTL we can express that a proposition eventually becomes true, in a **Metric Temporal Logic** we can express that it becomes true **within in an interval** I. Metric temporal operators can be used to specify qualitative and quantitative temporal constraints between events. For example, it could be used to quantify the time elapsed between two events or the measurement of a quantity at a given instant. In fact, the language includes operators for aggregating data values, which are useful in formulating many types of properties. Also, one can specify that a certain event must occur within a certain time interval or that the time difference between two events must be greater than a certain amount. For example we could verify the following formulas taken form [28]:

- The sum of withdrawals of each user over the last 30 days does not exceed the limit of $10,000.
- The average of the number of withdrawals of all users over the last 30 days should be less than a given threshold of 150.
- For each user, the number of peaks over the last 30 days does not exceed a threshold of 5, where a peak is a value at least twice the average over some time window.

All these characteristics make MFOTL particularly useful in situations where it is necessary to simultaneously verify temporal and metric aspects, as it allows to express a wide range of properties involving both time and measured values.

4.2. MonPoly, a monitoring tool

The tool we will use for SENTINEL runtime verification is MonPoly.

MonPoly is a monitoring tool for checking the compliance of systems to properties, which identifies and reports all property violations [29]. MonPoly performs property verification during program execution. This means it can spot violations in real time, providing valuable insights into the system's behavior while it's running. MonoPoly is flexible and can be used to verify different properties, including safety conditions, correct behavior and compliance with specific rules.

It does this by monitoring a series of events generated by the system and checking which of these events are allowed by the system in terms of property. Events can be monitored online or offline, so this tool allows us to decide whether in our audit we want to verify the system through an **online monitor**, as the events are created starting from the system, or an **offline monitor** through the analysis of the logs produced from the system itself. In both cases, what is being analyzed is exactly the **behavior of the system**. Through specifically written properties, the tool verifies whether or not they are respected.

It should be remembered that the result comes out of the analysis of only the events that occurred in that specific monitored period of time, therefore it is, in general, **not universal** in the sense that it is not certain that the properties violated by analyzing a stream of events, will be also violated by analyzing another stream of events.

However, we can say that if a property does not pass the verification process at least once, it means that the system does not comply with that property.

MonPoly uses **Metric First Order Temporal Logic** (MFOTL) as its specification language. Therefore, it has operators for data aggregation, allowing the specification of properties involving data collection and analysis during system execution.

From a first analysis to understand which properties can be analyzed in SENTINEL, listed in Section 4.1, it is possible to understand how MFOTL is a language suitable for their translation as:

- it is capable of using aggregation operators to obtain the count of specific events;
- it is able to delimit the monitoring in a certain time period;
- it is able to verify that certain rules are respected in a certain log entry guaranteeing the respected security of the monitored event;
- it is suitable for the verification of properties that have to do with legal compliance and certain privacy-related laws, if these are monitorable through the system.

Specifically, MFOTL can be used for the translation of the properties to be verified in SENTINEL which will be shown in Section 5.2.

The MonPoly grammar can be found in Appendix B.

4.2.1. The verification process

MonPoly takes a signature file, a formula file, and a log file as command-line input and it outputs satisfying valuations of the given formula on the given log file [29].

In details:

- The **signature file** describes the 1st-order signature of the formula used. A signature can be seen as a selection of non-logical symbols of a formal language. There can be more than one signature, and in that case, these are sorted. MonPoly currently supports the sorts string, integer, and float.

For our evaluation, we decided to use the following signature based on the information that the logs traced by the new logging system added in SENTINEL have:

– ***situation****(int, int, string, string, string, string, string, string, string, string, string, string, string, string)*

All the topics reported in brackets are necessary for our analysis and represent a specific category or subcategory of labor exploitation possibly violated in an instant of time. The set of topics then forms the set of crimes that possibly occurred. Note that all the boolean value generated in the logs, i.e. the parameters from the 5th to the 14th, are treated as strings whose value can be "true" or "false".

- The **formula file** contains the property as an MFOTL formula, possibly containing aggregation operators. We will see all the formulas in detail in chapter 5.
- A **log file** consists of a sequence of timestamped system events, which are ordered by their timestamps. Events that are assumed to occur simultaneously are grouped together.

Monpoly takes these three files as input and generates an **output**. The output consists of the event, or series of events, for which at least one property is violated. Thus, violating a property means satisfying the evaluation for a given property at a given instant of time.

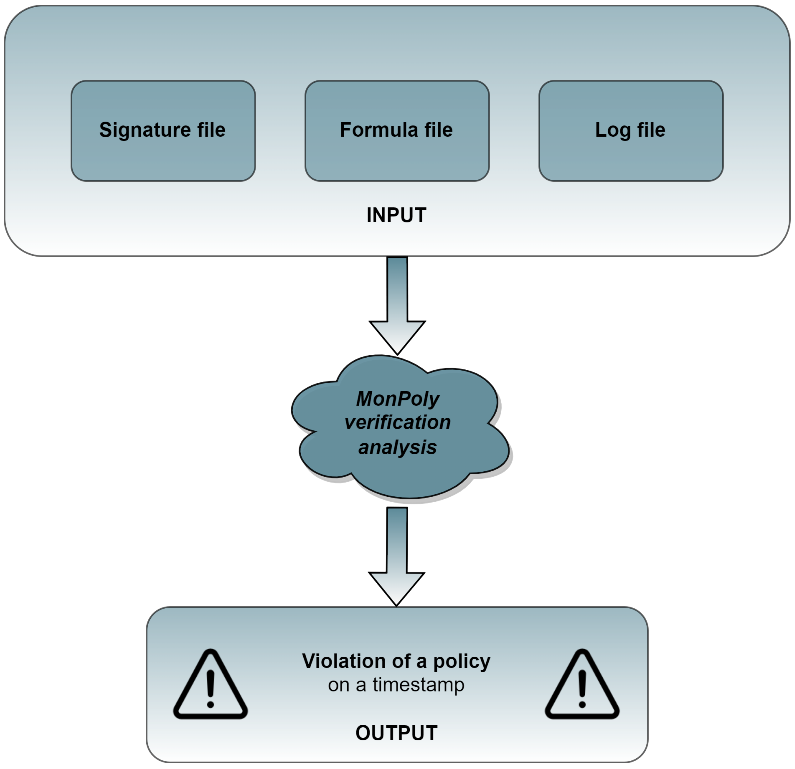

Signature file
Formula file
Log file
INPUT
MonPoly verification analysis
Violation of a policy on a timestamp
OUTPUT

Figure 4.1: The verification process through Monpoly.

5 | Verification of events in SENTINEL through MonPoly

The purpose of this chapter is to show the runtime verification applied to SENTINEL Monitor that can be used to reveal the occurrences of labor exploitation events at runtime. This type of approach can also be used to detect and report SENTINEL runtime errors.

Given the current situation in the Netherlands, where the phenomenon of labor exploitation is still rampant today, it is necessary to promote an effective monitoring system that can be used by the police to limit this phenomenon as much as possible. Since this is a system based on data that records potentially high-risk situations for the individual, identifying and differentiating the areas of exploitation is of fundamental importance in order to have a general picture of the situation and to be able to act on the specific existing problems in a strategic way.

In this context, it becomes essential to have an immediate view of the non-compliance properties, which are extracted at runtime thanks to the verification performed on the system.

5.1. Identification of exploitation events to be monitored

The first step in performing a runtime verification is to understand which events to monitor. This section will describe the choices made and the reason, as well as the property covered by the aforementioned monitored event.

5.1.1. Data gathering

The identification of verifiable properties in SENTINEL occurred only after a careful study of the context in which it proposes to operate and the legislative framework relating to it.

In particular, to understand which properties it is important to verify in a system that monitors the possible victimization of people living in a certain region, it is first necessary to analyze the phenomena that push these people to be subjected to such abuse.

This type of study was previously carried out and the results can be consulted in chapter 4. The **data collection** comes from a research in the gray literature, among which stands out the work of Marije Braakman, Saskia van Bon, Gregor Walz and Igor Boog who contributed to the study on forms of labor exploitation in the Netherlands and generally at European level with their research "*Supporting victims of severe forms of labor exploitation in having access to justice in EU Member States*" [1].

After the analysis of existing data on this phenomenon, it was necessary to **analyze the law** relating to this type of abuse, in particular we referred to the ***Dutch Criminal Code***, according to which labor exploitation is considered a criminal offense under Dutch law and falls under the human trafficking article (Article 273f).

The events to be monitored are therefore defined on the basis of the laws relating to labor exploitation, the current situation of this phenomenon in the Netherlands and the data that the SENTINEL system records through the creation of reports.

5.1.2. Events that can be monitored

We realize that the combinations of events that can be verified and benefited from for a future analysis of the widespread situation in the Netherlands are almost infinite. We have focused on the analysis of the following events in Table 5.1, so that they can be the basis for a more extensive verification of criminal events possibly occured. In Table 5.1 the monitored events are listed with some constraints that apply to the event. The third column indicates the class with respect to the taxonomy that was identified in section 4.1.

Event	**Constraint on event**	**Property**
Number of report entered in the last three months	e.g. > 100	Counting
Number of report entered in the current day	e.g. > 10	Counting
Number of report in a given city	e.g. > 20	Counting
Number of report in a given province	e.g. > 20	Counting

Number of report in a given position	e.g. > 1, the same address has already been entered in (at least) another report	Counting
Number of reports with area of exploitation “Housing”	e.g. > 40, the report contains at least one signal of (possible) victimization of labor exploitation related to Housing.	Counting
Number of reports with area of exploitation “Health”	e.g. > 40, the report contains at least one signal of (possible) victimization of labor exploitation related to Health.	Counting
Number of reports with area of exploitation “Payment”	e.g. > 40, the report contains at least one signal of (possible) victimization of labor exploitation related to Payment.	Counting
Number of reports with area of exploitation “Employer”	e.g. > 40, the report contains at least one signal of (possible) victimization of labor exploitation related to Employer.	Counting
Number of reports with area of exploitation “Work”	e.g. > 40, the report contains at least one signal of (possible) victimization of labor exploitation related to Work.	Counting
Maximum number of households (according to zoning plan) is exceeded	Maximum number of households according to the zoning plan $<$ number of people living in the house	Compliance with Employment Law in the Netherlands
Payment term exceeded	Payment term >30 days	Compliance with Employment Law in The Netherlands
No contract signed	Start of work (at 16 hours p/w) took place more than 3 months ago	Compliance with Employment Law in the Netherlands

No vacation money/vacation days	Labor law in the Netherlands requires that all workers receive a minimum of four weeks paid annual holiday each year. Holiday pay is generally 8% of the gross annual salary and is typically paid in May.	Compliance with Employment Law in the Netherlands
Daily working hours > 8	Standard working hours are usually 36, 38, or 40 hours per week. According to Dutch legislation, full-time employees over 18 years of age may not be asked to work more than 12 hours per day or 60 hours per week. Overtime should also not exceed four hours per day or 24 hours per week. Employees under 18 years old must not be asked to work more than 8 hours per day or 40 hours a week, with no more than two hours' overtime each day and a total of 10 extra hours weekly. These regulations apply only when working beyond normal contractual terms.	Compliance with Employment Law in the Netherlands
Entry of sensitive data concerning a person (employer information: name, agency, branch)	The GDPR (a European privacy regulation) ensures the careful processing of personal data by businesses and organizations. One must have a good reason to process personal data.	Privacy GDPR regulation
A report is always related to a geographical location	Address cannot be empty	Security

Table 5.1: Monitored events.

5.2. From properties to temporal logic formula

The next step to carry out the runtime verification of SENTINEL system is to translate the events to be monitored into formal properties, therefore into temporal logic. Details are available in tables below.

In the following formulas we will use the counting operator CNT. To help the reader understand the semantic of the following formulas, we will take the one in table 5.2 as an example and compare it to an SQL query that would obtain the same result. The corresponding SQL query to determine the violation with respect to property 1 at a specific instant of time is:

SELECT CNT(idR) AS c; idR FROM W GROUP BY idR HAVING CNT(idR) > 100, where W is the dinamically created view consisting of all the situations of all reports within the three months time window relative to the given time.

In all the tables below, the property formalized in MFOTL (in the second row) is the negation of the related property (in the first row). This is a trick that we needed to be able to use the formulas within Monpoly, where they will then be run negated again. In Section 6.3 we will demonstrate why this was a necessary change for making the verification successful.

Property 1	Number of reports entered in total in the last 3 months $>$ 100
Property formalization in MFOTL	$\Box\ \forall c\ [\mathrm{CNT}_{idR} \blacklozenge_{[0,3m]}\ \mathrm{situation}(...)]\ (c) \rightarrow c \leq 100$
Formalized property in the formula file	(c ← CNT idR ONCE[0,3m] (situation(idR, ...))) IMPLIES c ≤ 100 The property is violated when the 101st report in the last 3 months is found.

Table 5.2: From property 1 to temporal logic formula.

In a log file we have a sequence of events differentiated by timestamp and timepoint. The **timestamp** represents the time the event was recorded in a specific format, while the **timepoint** represents the position of that event in the sequence constituting the event log. It is therefore possible to have two events that occur with the same timestamp, but with different timepoints. Having the predicate tpts(i,j) as in the following formula, means grouping the results by differentiating them by timestamp and timepoint.

Property 2	Number of reports entered in the last day $>$ 10
Property formalization in MFOTL	$\square$ $\forall$c [CNT$_{idR}\blacklozenge_{[0,1d]}$ situation(...) $\wedge$ tpts(i,j)] (c) $\rightarrow$ c $\leq$ 10
Formalized property in the formula file	(c $\leftarrow$ CNT idR ONCE[0,1d] (situation(idR, ...) AND tpts(i,j))) IMPLIES c $\leq$ 10 The property is violated when the 11th report on the last day is found.

Table 5.3: From property 2 to temporal logic formula.

Property 3	Number of report in a given city $>$ 20
Property formalization in MFOTL	$\square$ $\forall$c [CNT$_{idR;idC}\blacklozenge$ situation(..., idC, ... $\wedge$ tpts(i,j)] (c) $\rightarrow$ c $\leq$ 20
Formalized property in the formula file	(c $\leftarrow$ CNT idR;idC ONCE (situation(idR, idA, idC, ...) AND tpts(i,j))) IMPLIES c $\leq$ 20 The property is violated when the 21st report with the same city is found.

Table 5.4: From property 3 to temporal logic formula.

Property 4	Number of report in a given province $>$ 20
Property formalization in MFOTL	$\square$ $\forall$c [CNT$_{idR;idP}\blacklozenge$ situation(idR, idA, idC, idP ...) $\wedge$ tpts(i,j)] (c) $\rightarrow$ c $\leq$ 20
Formalized property in the formula file	(c $\leftarrow$ CNT idR;idP ONCE (situation(idR, idA, idC, idP ...) AND tpts(i,j))) IMPLIES c $\leq$ 20 The property is violated when the 21st report with the same province is found.

Table 5.5: From property 4 to temporal logic formula.

Property 5	Number of report in a given position > 1
Property formalization in MFOTL	$\square\ \forall c\ [\mathrm{CNT}_{idR;idA,idC} \blacklozenge$ situation(idR, idA, idC, idP, ...) $\wedge$ tpts(i,j)] (c) $\rightarrow c \leq 1$
Formalized property in the formula file	(c ← CNT idR; idA,idC ONCE (situation(idR, idA, idC, idP, ...) AND tpts(i,j))) IMPLIES c ≤ 1 The property is violated when the 2nd report with the same address (in the same city) is found.

Table 5.6: From property 5 to temporal logic formula.

Property 6	Number of reports with area of exploitation “Housing” in the last month > 40
Property formalization in MFOTL	$\square\ \forall c\ [\mathrm{CNT}_{idR;ho} \blacklozenge_{[0,1m]}$ situation(..., ho, ...) $\wedge$ ho = true $\wedge$ tpts(i,j)] (c) $\rightarrow c \leq 40$
Formalized property in the formula file	(c ← CNT idR;ho ONCE[0,1m] (situation(..., ho, ...) AND ho = true AND tpts(i,j))) IMPLIES c ≤ 40 The property is violated when the 41st report with housing related “problems” in the last month is found.

Table 5.7: From property 6 to temporal logic formula.

Property 7	Number of reports with area of exploitation “Health” in the last month > 40
Property formalization in MFOTL	$\square\ \forall c\ [\mathrm{CNT}_{idR;he} \blacklozenge_{[0,1m]}$ situation(..., he, ...) $\wedge$ he = true $\wedge$ tpts(i,j)] (c) $\rightarrow c \leq 40$
Formalized property in the formula file	(c ← CNT idR;he ONCE[0,1m] (situation(..., he, ...) AND he = true AND tpts(i,j))) IMPLIES c ≤ 40 The property is violated when the 41st report with health related “problems” in the last month is found.

Table 5.8: From property 7 to temporal logic formula.

Property 8	Number of reports with area of exploitation “Payment” in the last month $>$ 40
Property formalization in MFOTL	$\square$ $\forall$c [CNT$_{idR;pa}$$\blacklozenge_{[0,1m]}$ situation(..., pa, ...) $\wedge$ pa $=$ true $\wedge$ tpts(i,j)] (c) $\rightarrow$ c $\leq$ 40
Formalized property in the formula file	(c $\leftarrow$ CNT idR;pa ONCE[0,1m] (situation(..., pa, ...) AND pa $=$ true AND tpts(i,j))) IMPLIES c $\leq$ 40 The property is violated when the 41st report with payment related “problems” in the last month is found.

Table 5.9: From property 8 to temporal logic formula.

Property 9	Number of reports with area of exploitation “Employer” in the last month $>$ 40
Property formalization in MFOTL	$\square$ $\forall$c [CNT$_{idR;em}$$\blacklozenge_{[0,1m]}$ situation(..., em, ...) $\wedge$ em $=$ true $\wedge$ tpts(i,j)] (c) $\rightarrow$ c $\leq$ 40
Formalized property in the formula file	(c $\leftarrow$ CNT idR;em ONCE[0,1m] (situation(..., em, ...) AND em $=$ true $\wedge$ tpts(i,j))) IMPLIES c $\leq$ 40 The property is violated when the 41st report with employer related “problems” in the last month is found.

Table 5.10: From property 9 to temporal logic formula.

Property 10	Number of reports with area of exploitation “Work” in the last month $>$ 40
Property formalization in MFOTL	$\square$ $\forall$c [CNT$_{idR;wo}$$\blacklozenge_{[0,1m]}$ situation(..., wo, ...) $\wedge$ wo $=$ true $\wedge$ tpts(i,j)] (c) $\rightarrow$ c $\leq$ 40
Formalized property in the formula file	(c $\leftarrow$ CNT idR;wo ONCE[0,1m] (situation(..., wo, ...) AND wo $=$ true $\wedge$ tpts(i,j))) IMPLIES c $\leq$ 40

	The property is violated when the 41st report with working related "problems" in the last month is found.

Table 5.11: From property 10 to temporal logic formula.

Property 11	Maximum number of households (according to zoning plan) is exceeded
Property formalization in MFOTL	$\square$ $\forall$c [CNT$_{idR}$$\blacklozenge$ situation(idR, idA, idC, idP, ...) $\wedge$ tpts(i,j) $\wedge$ householdsExceeded=true] (c) $\rightarrow$ c $\leq$ 0
Formalized property in the formula file	(c $\leftarrow$ CNT idR ONCE (situation(idR, idA, idC, idP, ...) AND tpts(i,j)) AND householdsExceeded=true) IMPLIES c $\leq$ 0 The property is violated when a report finds a housing exploitation that the maximum number of inhabitants allowed is exceeded for that dwelling.

Table 5.12: From property 11 to temporal logic formula.

Property 12	Maximum number of households (according to zoning plan) is exceeded in one province more than 50 times
Property formalization in MFOTL	$\square$ $\forall$c [CNT$_{idR;idP}$$\blacklozenge$ situation(idR, idA, idC, idP, ...) $\wedge$ tpts(i,j) $\wedge$ householdsExceeded=true] (c) $\rightarrow$ c $\leq$ 50
Formalized property in the formula file	(c $\leftarrow$ CNT idR;idP ONCE (situation(idR, idA, idC, idP, ...) AND tpts(i,j)) AND householdsExceeded=true) IMPLIES c $\leq$ 50 The property is violated when the 51st report with same province and householdsExceeded is found.

Table 5.13: From property 12 to temporal logic formula.

Property 13	Payment term exceeded
Property formalization in MFOTL	$\square$ $\forall$c [CNT$_{idR;paymentExceeded}$$\blacklozenge$ situation(..., paymentExceeded, ...) $\wedge$ tpts(i,j) $\wedge$ paymentExceeded=true] (c) $\rightarrow$ c $\leq$ 0

Formalized property in the formula file	(c ← CNT idR;paymentExceeded ONCE (situation(..., paymentExceeded, ...) AND tpts(i,j)) AND paymentExceeded=true) IMPLIES c ≤ 0 The property is violated when a report finds a payment exploitation and the payment deadline has been exceeded.

Table 5.14: From property 13 to temporal logic formula.

Property 14	No contract signed
Property formalization in MFOTL	$\square$ ∀c [CNT$_{idR;noContract}$♦ situation(..., noContract, ...) ∧ tpts(i,j) ∧ noContract=true] (c) → c ≤ 0
Formalized property in the formula file	(c ← CNT idR;noContract ONCE (situation(..., noContract, ...) AND tpts(i,j)) AND noContract=true) IMPLIES c ≤ 0 The property is violated when a report finds an exploitation and there isn't a contract signed related to that report.

Table 5.15: From property 14 to temporal logic formula.

Property 15	No vacation money/vacation days
Property formalization in MFOTL	$\square$ ∀c [CNT$_{idR;noVacation}$♦ situation(..., noVacation, ...) ∧ tpts(i,j) ∧ noVacation=true] (c) → c ≤ 0
Formalized property in the formula file	(c ← CNT idR;noVacation ONCE (situation(..., noVacation, ...) AND tpts(i,j)) AND noVacation=true) IMPLIES c ≤ 0 The property is violated when a report finds an exploitation and the "no vacation" box is checked, id equal to true.

Table 5.16: From property 15 to temporal logic formula.

Property 16	Daily working hours $>$ 8
Property formalization in MFOTL	$\Box\ \forall c\ [\mathrm{CNT}_{idR;workingHoursExceeded} \blacklozenge$ situation(..., workingHoursExceeded, ...) $\wedge$ tpts(i,j) $\wedge$ workingHoursExceeded=true] (c) $\rightarrow c \leq 0$
Formalized property in the formula file	(c $\leftarrow$ CNT idR;workingHoursExceeded ONCE (situation(..., workingHoursExceeded, ...) AND tpts(i,j)) AND workingHoursExceeded=true) IMPLIES c $\leq$ 0 The property is violated when a report finds a working exploitation and the allowed working hours are exceeded.

Table 5.17: From property 16 to temporal logic formula.

6 | Results

Given the SENTINEL software system, we created an event simulator, decided which properties to analyze and transformed them into properties in the form of MFOTL logic and finally used the MonPoly tool to perform a runtime verification of the properties.

6.1. Simulator

Before the extension we did on the system through which it is now possible to log its events information, **we did not have log files available** and therefore we simulated the creation of the latter to carry out the verification.

Obviously the **system** and consequently the **creation of logs also work without a simulator**. However, creating a large amount of reports via SENTINEL would have required running the application in the real world for months and would therefore have been a slower process of obtaining the logs to verify. This is the reason why we decided to implement a simulator that could independently creating a series of events recorded by the system.

6.1.1. Netherlands cities database

All the data related to a location necessary to recreate an event have been taken from the online database "***Netherlands Cities Database***" by *SimpleMaps*. It can be found and downloaded *here*. The database can also be consulted in Appendix C.

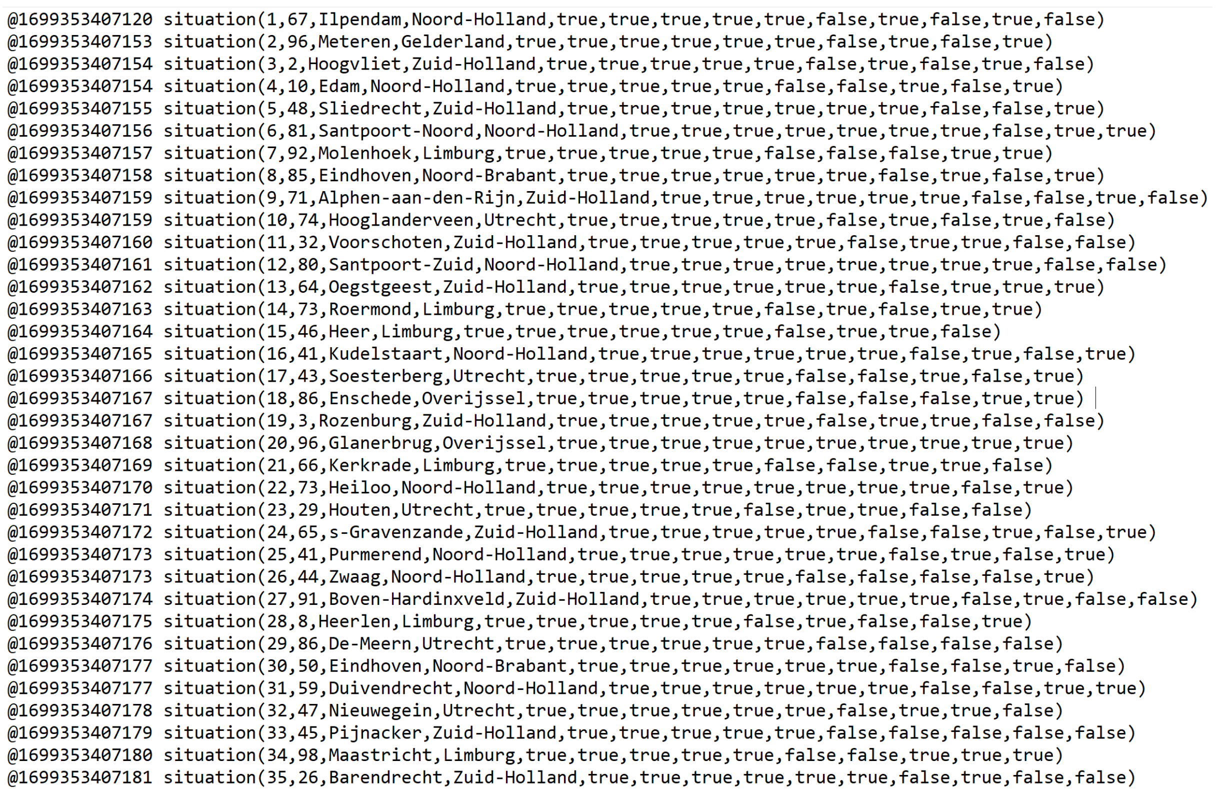
A version containing **229 entries** and therefore **229 cities in the Netherlands** located in different provinces was used. Our simulation of the events took place across the entire territory of the Netherlands, and not just in the specific province of North Braabant, which is where the SENTINEL system initiative started.

6.1.2. Situation simulation

As regards the other parameters present in the logs, whose meaning is explained in Section 3.3, they were simulated randomly taking into consideration that:

- the "**householdsExceeded**" parameter is part of the category "housingExpl" and therefore, if set to true, the "housingExpl" category must necessarily also be set to true;
- the "**workingHoursExceeded**" parameter is part of the category "laborExpl" and therefore, if set to true, the "laborExpl" category must necessarily also be set to true;
- the "**paymentExceeded**" parameter is part of the category "paymentExpl" and therefore, if set to true, the "paymentExpl" category must necessarily also be set to true;
- the "**noContract**" parameter is part of three categories (payment, employer and labor) and therefore, if set to true, the three categories must necessarily also be set to true;
- the "**noVacation**" parameter is part of two categories (payment and employer) and therefore, if set to true, the two categories must necessarily also be set to true;
- in the remaining cases, the parameters can have a true or false value without distinction.

In figure 6.1 a simulated log file is shown.



```
@1699353407120 situation(1,67,Ilpendam,Noord-Holland,true,true,true,true,true,false,true,false,true,false)
@1699353407153 situation(2,96,Meteren,Gelderland,true,true,true,true,true,true,false,true,false,true)
@1699353407154 situation(3,2,Hoogvliet,Zuid-Holland,true,true,true,true,true,false,true,false,true,false)
@1699353407154 situation(4,10,Edam,Noord-Holland,true,true,true,true,true,false,false,true,false,true)
@1699353407155 situation(5,48,Sliedrecht,Zuid-Holland,true,true,true,true,true,true,true,false,false,true)
@1699353407156 situation(6,81,Santpoort-Noord,Noord-Holland,true,true,true,true,true,true,true,false,true,true)
@1699353407157 situation(7,92,Molenhoek,Limburg,true,true,true,true,true,false,false,false,true,true)
@1699353407158 situation(8,85,Eindhoven,Noord-Brabant,true,true,true,true,true,true,false,true,false,true)
@1699353407159 situation(9,71,Alphen-aan-den-Rijn,Zuid-Holland,true,true,true,true,true,true,false,false,true,false)
@1699353407159 situation(10,74,Hooglanderveen,Utrecht,true,true,true,true,true,false,true,false,true,false)
@1699353407160 situation(11,32,Voorschoten,Zuid-Holland,true,true,true,true,true,false,true,true,false,false)
@1699353407161 situation(12,80,Santpoort-Zuid,Noord-Holland,true,true,true,true,true,true,true,true,false,false)
@1699353407162 situation(13,64,Oegstgeest,Zuid-Holland,true,true,true,true,true,true,false,true,true,true)
@1699353407163 situation(14,73,Roermond,Limburg,true,true,true,true,true,false,true,false,true,true)
@1699353407164 situation(15,46,Heer,Limburg,true,true,true,true,true,true,false,true,true,false)
@1699353407165 situation(16,41,Kudelstaart,Noord-Holland,true,true,true,true,true,true,false,true,false,true)
@1699353407166 situation(17,43,Soesterberg,Utrecht,true,true,true,true,true,false,false,true,false,true)
@1699353407167 situation(18,86,Enschede,Overijssel,true,true,true,true,true,false,false,false,true,true)
@1699353407167 situation(19,3,Rozenburg,Zuid-Holland,true,true,true,true,true,false,true,true,false,false)
@1699353407168 situation(20,96,Glanerbrug,Overijssel,true,true,true,true,true,true,true,true,true,true)
@1699353407169 situation(21,66,Kerkrade,Limburg,true,true,true,true,true,false,false,true,true,false)
@1699353407170 situation(22,73,Heiloo,Noord-Holland,true,true,true,true,true,true,true,true,false,true)
@1699353407171 situation(23,29,Houten,Utrecht,true,true,true,true,true,false,true,true,false,false)
@1699353407172 situation(24,65,s-Gravenzande,Zuid-Holland,true,true,true,true,true,false,false,true,false,true)
@1699353407173 situation(25,41,Purmerend,Noord-Holland,true,true,true,true,true,true,false,true,false,true)
@1699353407173 situation(26,44,Zwaag,Noord-Holland,true,true,true,true,true,false,false,false,false,true)
@1699353407174 situation(27,91,Boven-Hardinxveld,Zuid-Holland,true,true,true,true,true,true,false,true,false,false)
@1699353407175 situation(28,8,Heerlen,Limburg,true,true,true,true,true,false,true,false,false,true)
@1699353407176 situation(29,86,De-Meern,Utrecht,true,true,true,true,true,true,false,false,false,false)
@1699353407177 situation(30,50,Eindhoven,Noord-Brabant,true,true,true,true,true,true,false,false,true,false)
@1699353407177 situation(31,59,Duivendrecht,Noord-Holland,true,true,true,true,true,true,false,false,true,true)
@1699353407178 situation(32,47,Nieuwegein,Utrecht,true,true,true,true,true,true,false,true,true,false)
@1699353407179 situation(33,45,Pijnacker,Zuid-Holland,true,true,true,true,true,false,false,false,false,false)
@1699353407180 situation(34,98,Maastricht,Limburg,true,true,true,true,true,false,false,true,true,true)
@1699353407181 situation(35,26,Barendrecht,Zuid-Holland,true,true,true,true,true,true,false,true,false,false)
```

Figure 6.1: Simulated log file example.

It should be noted that a category could still be set to true, even if none of the subcategories explicitly reported in the logs were set to true, as the latter represents only **a portion of all possible reportable violations** of those specific categories.

The full code of the simulator can be found *here*.

6.2. Log analysis and filtering

An event simulation has been generated and saved in a log file with the simulator previously explained.

These events were verified through the Monpoly analysis tool. The formulas in tables 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 5.10, 5.11, 5.12, 5.13, 5.14, 5.15, 5.16, 5.17, were given as input to the tool to verify the compliance of the events with them.

Having simulated the data ourselves, there was no need to do any log filtering, but in a real situation, where we might be dealing with an extremely more complex and log-rich system, we might want to make sure we are using only the logs we need to verify a specific property. In the latter case, it could be useful to first filter the existing log entries and only then give them as input to MonPoly to carry out the runtime verification.

6.3. Presentation of the results

We generated **100 events randomly**, i.e. 100 situations, but certainly the system could also work with a much larger number of events. Each property has been saved in one different formula file. The generated log file can be seen in Appendix D and all the formula files created starting from the properties can be found *here*.

This is the command used to verify the properties in MonPoly:

monpoly -sig <*file*> **-formula** <*file*> **[-negate] [-log** <*file*>**]**

Note that the formula files contain the **negated property** and therefore property violations correspond to **satisfying valuations** for the given formula at a time point [29]. That is why we used the **[-negate]** in the launched commands.

The reason why we use the negated formula and then apply the -negate operator is that if we used the non-negated formula and did not apply the -negate operator, the formula would be **not monitorable**, i.e. not verifiable with MonPoly because at each time point there would be infinitely many tuples satisfying it [29].

To better understand consider the modified formula related to property 1 as an example: c ← CNT idR ONCE[0,3m] (situation(idR, ...))) IMPLIES c > 100

By launching the command without the -negate in MonPoly we would obtain:

```
The analyzed formula is:

(NOT c <- CNT idReport ONCE[0,180] situation(idReport, idAddress, city,
province, housingExpl, healthExpl, paymentExpl, laborExpl, employerExpl,
householdsExceeded, workingHoursExceeded, paymentExceeded, noContract,
noVacation)) OR 100 < c

The sequence of free variables is:  (c)

The analyzed formula is NOT monitorable, because of the subformula:

NOT c <- CNT idReport ONCE[0,180] situation(idReport, idAddress, city,
province, housingExpl, healthExpl, paymentExpl, laborExpl, employerExpl,
householdsExceeded, workingHoursExceeded, paymentExceeded, noContract,
noVacation)
```

Subformulas of the form NOT psi should contain no free variables (except when they are part of subformulas of the form phi AND NOT psi, NOT psi SINCE phi, or NOT psi UNTIL phi). In general, a variable x is free in a formula if it occurs at least once in the formula without being introduced by one of the phrases “for some x” or “for all x.”

On the contrary, in the case of our formula 1, written in a way to let it be monitorable, we would have the following information from MonPoly:

```
The analyzed formula is:

(c <- CNT idReport ONCE[0,180] situation(idReport, idAddress, city,
province, housingExpl, healthExpl, paymentExpl, laborExpl, employerExpl,
householdsExceeded, workingHoursExceeded, paymentExceeded, noContract,
noVacation)) AND (NOT c <= 100)
```

6.3.1. RV of property 1

rv1.mfotl:
(c ← CNT idR ONCE[0,3m] (situation(idR, ...))) IMPLIES c $\leq$ 100

Output:
No output, which means that no contradiction is found in sentinel.log with respect to property 1. Thus, in the last three months the number of events recorded was not greater than 100. In fact, even if all the simulated events in the log file occurred in the last three months, the number of simulated events in total is exactly 100, therefore the condition is never violated.

6.3.2. RV of property 2

rv2.mfotl:
(c ← CNT idR ONCE[0,1d] (situation(idR, ...) AND tpts(i,j))) IMPLIES c $\leq$ 10

Output:
```
@1.69935340716e+12 (time point 10): (11)
@1.69935340716e+12 (time point 11): (12)
@1.69935340716e+12 (time point 12): (13)
...
@1.69935340722e+12 (time point 97): (98)
@1.69935340722e+12 (time point 98): (99)
@1.69935340722e+12 (time point 99): (100)
```

From the 11th log enty to the last log entry, i.e. the 100th, a contradiction is found. In fact, in the last day the number of events recorded was greater than 10. The number in brackets at the end of each line indicates the variable c calculated at that instant which violates the property. For example in this case in the first line it is shown that c is equal to 11 and therefore this represents a violation of the property.

6.3.3. RV of property 3

rv3.mfotl:
(c $\leftarrow$ CNT idR;idC ONCE (situation(idR, idA, idC, ...) AND tpts(i,j))) IMPLIES c $\leq$ 20

Output:
No output, which means that no contradiction is found in sentinel.log with respect to property 3. Thus, the number of events recorded in the log file which had the same city was not greater than 20.

Note that if we slightly modify the property so that the number of equal cities is 2 and not 20, at the last time point we would obtain a result like this:

```
@1.69935340722e+12 (time point 99): (3,Hooglanderveen) (3,Leeuwarden)
```

It represents the contradictions found, i.e. the cities for which, at the end of the parsing of the events, there are more than two events with the same city. In this case the events found are three for both the city "Hooglanderveen" and for "Leeuwarden".

6.3.4. RV of property 4

rv4.mfotl:
(c $\leftarrow$ CNT idR;idP ONCE (situation(idR, idA, idC, idP ...) AND tpts(i,j))) IMPLIES c $\leq$ 20

Output:

```
@1.69935340721e+12 (time point 74): (21,Noord-Holland)
@1.69935340721e+12 (time point 75): (21,Noord-Holland)
@1.69935340721e+12 (time point 76): (21,Noord-Holland)
@1.69935340721e+12 (time point 77): (21,Noord-Holland)
@1.69935340721e+12 (time point 78): (21,Noord-Holland)
@1.69935340721e+12 (time point 79): (21,Noord-Holland)
@1.69935340721e+12 (time point 80): (21,Noord-Holland)
@1.69935340721e+12 (time point 81): (21,Noord-Holland)
```

```
@1.69935340721e+12 (time point 82): (22,Noord-Holland)
@1.69935340721e+12 (time point 83): (22,Noord-Holland)
@1.69935340721e+12 (time point 84): (22,Noord-Holland)
@1.69935340721e+12 (time point 85): (21,Zuid-Holland) (22,Noord-Holland)
@1.69935340722e+12 (time point 86): (21,Zuid-Holland) (22,Noord-Holland)
@1.69935340722e+12 (time point 87): (21,Zuid-Holland) (22,Noord-Holland)
@1.69935340722e+12 (time point 88): (21,Zuid-Holland) (22,Noord-Holland)
@1.69935340722e+12 (time point 89): (21,Zuid-Holland) (22,Noord-Holland)
@1.69935340722e+12 (time point 90): (21,Zuid-Holland) (23,Noord-Holland)
@1.69935340722e+12 (time point 91): (21,Zuid-Holland) (23,Noord-Holland)
@1.69935340722e+12 (time point 92): (21,Zuid-Holland) (23,Noord-Holland)
@1.69935340722e+12 (time point 93): (21,Zuid-Holland) (24,Noord-Holland)
@1.69935340722e+12 (time point 94): (21,Zuid-Holland) (24,Noord-Holland)
@1.69935340722e+12 (time point 95): (21,Zuid-Holland) (24,Noord-Holland)
@1.69935340722e+12 (time point 96): (21,Zuid-Holland) (24,Noord-Holland)
@1.69935340722e+12 (time point 97): (21,Zuid-Holland) (25,Noord-Holland)
@1.69935340722e+12 (time point 98): (21,Zuid-Holland) (25,Noord-Holland)
@1.69935340722e+12 (time point 99): (22,Zuid-Holland) (25,Noord-Holland)
```

A contradiction is found starting from time point 75. By looking at the last time point we can see that the province "Zuid-Holland" appears 22 times in the log file and the province "Noord-Holland" appears 25 time.

Notice that in this property we didn't put a time limit on the ONCE operator, because we wanted to look for the number of times that each province appeared in the entire log file, i.e. in the entire set of recorded events. The result is that a previously found contradiction will remain so until the end of the verification, that is, at every time point.

On the contrary, if we wanted to look for the number of times in which the same province appears in the set of events in a given period, for example in the last 2 hours, we could find a contradiction at a time point, before which the province appeared more than 20 times, but we may not find that contradiction immediately after that time point. In fact, since the latter is ahead in time, it may not include 20 occurrences of the same province in the previous 2 hours (starting from the timestamp associated with the related time point).

6.3.5. RV of property 5

rv5.mfotl:
(c ← CNT idR; idA,idC ONCE (situation(idR, idA, idC, idP, ...) AND tpts(i,j))) IMPLIES c $\leq$ 1

Output:
No output, which means that no contradiction is found in sentinel.log with respect to property 5. Thus, the number of events recorded group by address and city was not greater than 1. This means that it doesn't exists in our set of events, two events with the same address, i.e. same idAddress in the same city.

6.3.6. RV of property 6

rv6.mfotl:
(c ← CNT idR;ho ONCE[0,1m] (situation(..., ho, ...) AND ho = true AND tpts(i,j))) IMPLIES c $\leq$ 40

Output:
@1.69935340718e+12 (time point 40): (41,true)
...
@1.69935340722e+12 (time point 99): (100,true)

A contradiction is found starting from time point 40, which means that from time point 0 to time point 39, there is a violation regarding the category housing was always present.

6.3.7. RV of property 7

rv7.mfotl:
(c ← CNT idR;he ONCE[0,1m] (situation(..., he, ...) AND he = true AND tpts(i,j))) IMPLIES c $\leq$ 40

Output:
@1.69935340718e+12 (time point 40): (41,true)
... *@1.69935340722e+12 (time point 99): (100,true)*

A contradiction is found starting from time point 40, which means that from time point 0 to time point 39, there is a violation regarding the category health was always present.

6.3.8. RV of property 8

rv8.mfotl:
(c $\leftarrow$ CNT idR;pa ONCE[0,1m] (situation(..., pa, ...) AND pa = true AND tpts(i,j))) IMPLIES c $\leq$ 40

Output:
@1.69935340718e+12 (time point 40): (41,true)
... @1.69935340722e+12 (time point 99): (100,true)

A contradiction is found starting from time point 40, which means that from time point 0 to time point 39, there is a violation regarding the category payment was always present.

6.3.9. RV of property 9

rv9.mfotl:
(c $\leftarrow$ CNT idR;em ONCE[0,1m] (situation(..., em, ...) AND em = true $\wedge$ tpts(i,j))) IMPLIES c $\leq$ 40

Output:
@1.69935340718e+12 (time point 40): (41,true)
... @1.69935340722e+12 (time point 99): (100,true)

A contradiction is found starting from time point 40, which means that from time point 0 to time point 39, there is a violation regarding the category employer was always present.

6.3.10. RV of property 10

rv10.mfotl:
(c $\leftarrow$ CNT idR;wo ONCE[0,1m] (situation(..., wo, ...) AND wo = true $\wedge$ tpts(i,j))) IMPLIES c $\leq$ 40

Output:
@1.69935340718e+12 (time point 40): (41,true)
... @1.69935340722e+12 (time point 99): (100,true)

A contradiction is found starting from time point 40, which means that from time point 0 to time point 39, there is a violation regarding the category working was always present.

6.3.11. RV of property 11

rv11.mfotl:
(c ← CNT idR ONCE (situation(idR, idA, idC, idP, ...) AND tpts(i,j)) AND householdsExceeded=true) IMPLIES c ≤ 0

Output:
@1.69935340712e+12 (time point 0): (1)
...

A contradiction is found starting from time point 0, which means that from time point 0 on, there is a violation regarding the sub-category "householdsExceeded".

If we wanted to change the property slightly so that it does the same check but on a time interval of 1 second, therefore having ONCE[0,1s], we would have an output like this:

@1.69935340712e+12 (time point 0): (1)
...
@1.69935340716e+12 (time point 9): (3)
...
@1.69935340716e+12 (time point 14): (2)
...
@1.69935340719e+12 (time point 42): (4)
...

Here can be seen how at each time point we can have a number of events with the constraints above, but only taking into account the events occurred in the previous second with respect to the current timestamp.

6.3.12. RV of property 12

rv12.mfotl:
(c ← CNT idR;idP ONCE (situation(idR, idA, idC, idP, ...) AND tpts(i,j)) AND householdsExceeded=true) IMPLIES c ≤ 50

Output:
No output, which means that no contradiction is found in sentinel.log with respect to property 12. Thus, the number of events recorded in the log file which had the same province and in which the sub-category "householdsExceeded" was set to true was not

greater than 50.

6.3.13. RV of property 13

rv13.mfotl:
(c $\leftarrow$ CNT idR;paymentExceeded ONCE (situation(..., paymentExceeded, ...) AND tpts(i,j)) AND paymentExceeded=true) IMPLIES c $\leq$ 0

Output:

```
@1.69935340712e+12 (time point 0): (1,false)
...
@1.69935340722e+12 (time point 99): (45,false) (55,true)
```

Here at each time point the output displays the occurrence of a false (and of a true) value for the sub-category "paymentExceeded". By looking at them it is possible to know at each time point how many events have a true (and a false) value for "paymentExceeded".

6.3.14. RV of property 14

rv14.mfotl:
(c $\leftarrow$ CNT idR;noContract ONCE (situation(..., noContract, ...) AND tpts(i,j)) AND noContract=true) IMPLIES c $\leq$ 0

Output:

```
@1.69935340712e+12 (time point 0): (1,true)
...
@1.69935340722e+12 (time point 99): (47,true) (53,false)
```

Here at each time point the output displays the occurrence of a false (and of a true) value for the sub-category "noContract". By looking at them it is possible to know at each time point how many events have a true (and a false) value for "noContract".

6.3.15. RV of property 15

rv15.mfotl:
(c $\leftarrow$ CNT idR;noVacation ONCE (situation(..., noVacation, ...) AND tpts(i,j)) AND noVacation=true) IMPLIES c $\leq$ 0

Output:

```
@1.69935340712e+12 (time point 0): (1,false)
...
@1.69935340722e+12 (time point 99): (41,false) (59,true)
```

Here at each time point the output displays the occurrence of a false (and of a true) value for the sub-category "noVacation". By looking at them it is possible to know at each time point how many events have a true (and a false) value for "noVacation".

6.3.16. RV of property 16

rv16.mfotl:

(c ← CNT idR;workingHoursExceeded ONCE (situation(..., workingHoursExceeded, ...) AND tpts(i,j)) AND workingHoursExceeded=true) IMPLIES c $\leq$ 0

Output:

```
@1.69935340712e+12 (time point 0): (1,true)
...
@1.69935340722e+12 (time point 99): (37,true) (63,false)
```

Here at each time point the output displays the occurrence of a false (and of a true) value for the sub-category "workingHoursExceeded". By looking at them it is possible to know at each time point how many events have a true (and a false) value for "workingHoursExceeded".

6.4. Time and space needed for verification

To evaluate the time and space required to verify a policy thanks to MonPoly we have created - with the simulator presented previously - other simulated logs containing 10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8 log entries. We will refer to them respectively with the names ***L1***, ***L2***, ***L3***, ***L4***, ***L5***, ***L6***, ***L7***. Due to the impossibility of simulating a greater quantity of log entries on the local workspace necessary for the evaluation of the memory and time values used by MonPoly, this evaluation is left for future investigation.

For this evaluation we decided to use only one of the previously verified properties, to understand how the tool would have behaved with a much larger amount of logs than the one it was subjected to in our previous analysis. In particular we used ***property 4***: "Number of report in a given province $>$ 20". The choice to use this property comes from the fact that we wanted to test one of the properties having at least two groupings, in this case the results are grouped by report id and then by province id. And furthermore we wanted the verification to be done on all the logs present and therefore we wanted to use a property that did not have time limits within it. However, we could have tested any other properties than those presented above. We are confident in saying that since the properties listed previously are very similar to each other, the result obtained from their possible verification should not differ greatly from that obtained by verifying policy 4.

Obtaining temporal and memory data was possible thanks to the "-mem" option offered by MonPoly and the "time" command offered by Linux. These were added to the launched command:

- ***time***: records the duration of the command's execution. The default output of the time command contains the following information:
  - Real-time (real): The real-life time it takes for the process to run from start to end. This includes any time taken by other processes and the time spent waiting for them to be complete.
  - User time (user). The amount of CPU time spent in user mode during the process. Other processes and blocked time are not included.
  - System time (sys). The total CPU time spent in kernel mode during the process. Similar to user time, other processes and time spent blocked by other processes are not counted.
- ***-mem***: shows memory usage on stderr in KBytes.

In Table 6.1 the time and memory results required for verifying the policy with the different logs are presented.

Log	**Space**	**Real Time**	**User Time**	**Sys Time**
L1	10532 KB	0.005s	0.004s	0.000s
L2	10532 KB	0.050s	0.030s	0.021s
L3	12484 KB	0.554s	0.336s	0.218s
L4	21812 KB	5.385s	2.857s	2.516s
L5	21812 KB	55.785s	29.037s	23.666s
L6	23424 KB	9m 34.474s	4m 47.420s	3m 55.997s
L7	21944 KB	92m 7.794s	48m 51.107s	43m 6.226s

Table 6.1: Time and space required for verification.

Figure 6.2 is a line graph showing changes in the space variable as the number of log entries increases.

Figure 6.3 is a line graph showing changes in the time variables as the number of log entries increases.

Note that in both figures the **horizontal axis** is the one where the logs are reported and it is **logarithmic**, as the increase from one log file to the other is logarithmic. Instead, the **scale** of space and time is **linear** and is reported on the **vertical axis**.

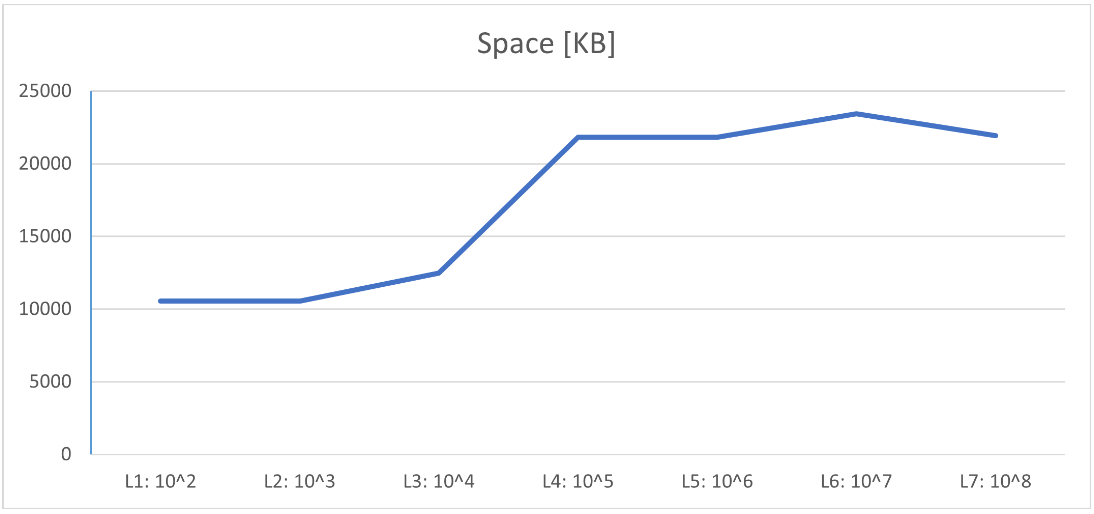


Figure 6.2: Memory usage for verification.

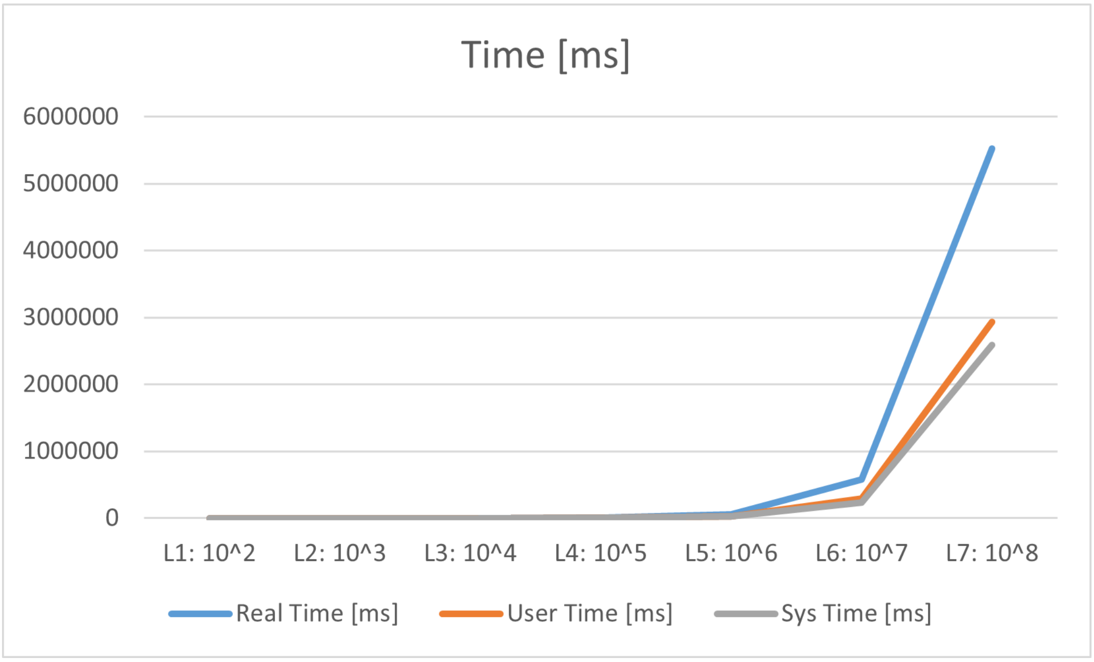


Figure 6.3: Time usage for verification.

6.5. Results analysis

From the runtime verification carried out on the SENTINEL system, the importance of the choice made when drafting the properties emerges. Asking for a certain property to be verified rather than another and translating these properties into formulas that the tool can analyze is fundamental.

By conducting the verification, we observed how the **time needed** to analyze the events is **extremely low**, it is in the order of seconds with a number of log entries less than one million, as the data passed to the tool are simple strings that do not need further excessive processing to be transformed into data that can be analyzed by MonPoly.

The analysis of the results obtained showed that:

- runtime verification is suitable and **useful for analyzing properties** regarding the **granularity of the data** and the **periodicity** with which certain events occur in a certain context, which can be, for example, a certain geographical area;
- through runtime verification it is possible to track the occurrences of labor exploitation crimes at runtime;
- through runtime verification it is also possible to verify that there are no errors in some parts of the event processing, which would therefore lead to the **discovery of system bugs**. For example, we could create a property that verify that a constraint is actually respected. Specifically in SENTINEL, we could verify that there is no event having a parameter of a subcategory set to true and having at the same time the parameter of the corresponding category set to false. In SENTINEL case, no bugs have been found so far;
- runtime verification can be used to verify the **absence of certain values**, for example we could verify that the logs do not contain personal data, by creating a property that verify that the fields relating to the situation do not contain values other than true and false.

6.6. Discussion of the results

The runtime verification carried out on a system like SENTINEL, which is responsible for monitoring events that bring with them a lot of data, is a clear example of how it is possible to analyze that data at runtime.

Some great benefits of this approach are:

- **possibility to choose WHICH property**: it is possible to verify at our convenience only the properties that interest us or that we consider essential to complete a certain task. This means spending time just checking every single property of interest and ignoring the others that are not useful for the purpose we want to achieve;
- **possibility to choose WHEN**: it is possible to analyze the behavior of the system in a specific period of time and instead ignore all events that occurred outside our range of interest. This allows us to focus exactly on verifying the system under certain temporal conditions (giving MonPoly the log file containing only events in a certain period of time on which we want to carry out the analysis);

To understand the true power of RV applied to SENTINEL, we must make a comparison with the SENTINEL system without the proposed extension, i.e. without the creation of the log files that save the events that occur in the system.

Comparing the SENTINEL system in the version without log files, which therefore cannot use MonPoly to implement crime tracking, and still wanting to obtain a similar result, we would have had to extract the data directly from the database, meaning that for each formula created previously we should have written an SQL query.

In this case, problems related to design times and costs must be taken into consideration and therefore also the level of experience required for the creation of SQL queries compared to that necessary for using MonPoly.

In the case of SENTINEL, retrieving the requested data would mean making a selection on the parameters that are the result of selections on other tables on which join operations have been carried out. We would intuitively consume more memory to obtain the same data than we would consume by scrolling through the log strings already created with the extended SENTINEL system. This is because all the data contained in a log entry cannot be obtained from a single table in our existing database, in which the data relating to the report, such as the id, are in a different table than the data relating to the possible violation situation. On the contrary, at runtime, we can write all the data of interest for a specific event in a format that is most convenient for us within the log files, without making a query to the database, but using the data entered at runtime.

The time needed to obtain the result of the verification carried out on the logs also proved to be quite rapid, in the order of seconds with a number of log entries less than one million. For this reason we are confident in saying that **this type of approach can be faster** than a typical approach in which SQL queries are used to retrieve the data to be verified.

We leave the creation of an experimental campaign as a future challenge of this work to verify these claims.

Performing a runtime verification of a software system like SENTINEL, allows one identify any **emerging trends** or **patterns** regarding the events that create the behavior of the system. For example, from the data obtained, we noticed how it is extremely easy for the violation regarding the "housing" category to occur, albeit in a simulation context. In fact, we are aware that some subcategories of the housing category are also part of other categories. This means that once the violation of a subcategory is reported, for example "noContract", the violation of the two categories to which it is connected is also automatically reported.

If it is possible to study the results obtained from the verification in depth, **statistics** can be created regarding the **granularity** and **periodicity** with which the reported phenomena occur.

In fact, we do not study the single event, but **the possibility that a series of events leads to a certain result**. For example, we could verify the periodicity with which a certain event occurs after another has occurred. Some examples of trends we could verify within the SENTINEL context include:

- every time an event of possible victimization is recorded in a certain location, the number of events recorded in the same location in the following three months decreases (i.e. is less) compared to the previous three months;
- every time a potentially fruitful event relating to the home category is recorded, it also records a violation of the health category.

Information like this is extremely important in a context where the data acquired has legal value and can influence decisions that can lead to improving the frequency with which such violations occur and hopefully eliminating them entirely.

6.7. Errors and limitations

Runtime verification also has some limitations when dealing with a real, non-simulated software system:

- it is possible to carry out a check only **based on the logs** generated by the system, so any bugs or non-compliance of events with properties is only found if the system is actually running;
- the system in general is not verified, but for each property translated into logic, only

that property is verified on the basis of the events observed and nothing more;

- only what can be translated into MFOTL logic can be verified, the **monitorability** of the formula is a key point on which the entire approach is based.

Furthermore, there are challenges with this approach that should not be underestimated:

- translating properties into logical formulas is not only **not always possible**, but it is **not always easy** either, especially if the property we are trying to transform into a logical language is particularly complex;
- **falling into error is easy**, especially if considering runtime verification it is not the most used approach to verify a software system and it is also a relatively new one, the probability of making errors is even higher and it is proportional to the probability of noticing them.

6.8. Repeatability and generalizability

The runtime verification method is undoubtedly **repeatable** since, given as input a signature file, a log file and a file containing the formula to be verified, the verification can be carried out again on the same files and will always produce the same result.

Furthermore, it is **generalizable as long as the syntax of the events is consistent** with the signature file. In fact, if we wanted to extend the verification to another log file, this would be possible.

6.9. Answers to research questions

Following the analysis of the results, it is now possible to answer our research questions:

RQ.1: Is runtime verification a suitable method to verify system properties to automatic identify the violations of labor protection regulations?

Runtime verification is a powerful verification method that can be used on software systems and beyond. In general, it has some peculiarities that make it different and attractive compared to the traditional verification methods used today:

- it does not have the problem of creating a mathematical model at the basis of the verification, as instead happens with the theorem proving method;
- verification occurs at runtime on the stream of events produced by the system, unlike testing in which a number of cases are generated to be tested with fictitious data;

- despite being very similar to model checking, runtime verification does not limit itself to analyzing infinite traces generated by the system, but rather verifies finite executions of it, thus reducing the complexity of the method itself.

Runtime verification analysis allows one to verify some properties of a system such as safety and security, but it also allows one to monitor the behavior of a system and keep track of it in order to use the data obtained to identify any patterns of recurring and correlated events.

In the context of SENTINEL, it is fully able to monitor the occurrences of recorded crimes relating to labor exploitation and is therefore a method that can be used to analyze from the data obtained possible recurring events that cause a certain violation of worker rights and human rights.

With RV it is possible and also useful to verify system properties to automatic identify the violations of labor protection regulations. It is therefore not only a suitable method, but also an extremely useful one for automatically identify the violations of labor protection regulations under certain temporal conditions and keeping in mind that the result obtained from the verification is the result of the analysis of a finite trace of events obtained from a system run.

RQ.2: What is necessary and interesting to verify in a system with a lot of heterogeneous data through runtime verification?

To answer this question it is essential to make a premise: as shown through the research carried out, before understanding which properties are interesting to monitor, we must be aware of what is possible to monitor. We have identified three main factors that limit this decision. Specifically, the choice of properties to verify is influenced by:

- technical reasons: the choice of specification language and its expressiveness is decisive in the choice of properties;
- reasons relating to the context: any limits dictated by the context in which the system operates must be taken into consideration;
- functional reasons: it is necessary to understand what the primary purpose of the system is and therefore identify what is useful to verify in it.

Having said this premise, it is clear that the identification of properties is a direct consequence of the analysis of the three motivations above.

In general, in a data-rich system, we need to ensure that this data is consistent with the given specifications.

In the context of runtime verification it is possible to identify any system **bugs** through the analysis of the logs which constitute the evolution of the system states at a certain instant of time. It is possible to evaluate the **performance** relating to the system, as we can analyze the number of events recorded in a given period of time. It is very useful, and often used in critical systems, to verify the **stability** and **robustness** of the system following certain events or certain temporal conditions. Finally, it is useful to keep track of the data saved in the logs so that they can be used at runtime to see not only the performance of the system, but also the progress of the situation being monitored in real life.

We must remember that behind the data monitored and saved by the SENTINEL Monitor, there are real people who may have suffered violations of their rights. Evaluating the trend in real time can be extremely useful in choosing the decisions to make at a strategic level to limit this phenomenon. Furthermore, the creation and evolution in real time of patterns associated with the periodicity with which these phenomena occur can be decisive for the resolution of the aforementioned abuses.

RQ.3: How does the context influence the formulation of properties regarding the properties to be analyzed?

As anticipated in the previous answer, the context in which a software system operates is one of the foundations on which the choice of properties to verify must be made.

The reason is that, even if in theory it is possible to monitor everything that can be translated into the chosen specification language, it is important to deduce when the verification of a certain property becomes crucial in the context in which the system operates.

For example, if the system under analysis contains data that can be legally processed, as in the case of SENTINEL Monitor, it is important to ensure that these are not corrupted or have not been subsequently modified and it can be achieved through runtime verification.

In general, the context in which a system operates can increase the importance of some properties to be respected and decrease that of others that do not contribute to the criticality of the system.

Part III

Conclusions

7 | Conclusions and final considerations

This thesis work was born with the aim of contributing to the automatic tracking of crimes and in particular those relating to labor exploitation.

Our proposal was to monitor possible abuses related to labor exploitation occurred in a region of the Netherlands recorded through SENTINEL Monitor and subsequently verified with the runtime verification method.

7.1. Discussion

This research made it possible to analyze the role that runtime verification can have within complex systems in which the correctness of the assimilated data is fundamental and the possibility of tracking it is extremely useful.

It was possible to understand how the identification of properties can be obtained only after a careful study of the context in which the system operates, the purpose for which it was created and the technical feasibility deriving from the specification language used to translate the high-level properties into logical formulas.

Furthermore, two key points on runtime verification were highlighted. First of all, the possibility of choosing which property to test with a certain stream of events allows us to isolate the type of study we want to conduct and the data we want to track and analyze. And secondly, it is possible to choose the time range in which we want to carry out the verification. It is therefore not necessary to verify the entire stream of events that a system produces and it is not mandatory to analyze all the properties that the system must respect. This allows us to significantly reduce the complexity of this verification method compared to other traditional methods.

Although the runtime verification method is usually used to verify the robustness and safety properties of the system, we wanted to study how useful the same method could be for studying the behavior obtained from events recorded by SENTINEL over time, i.e. the sequence of log identifying possible crimes. In fact, the research showed that through the verification carried out on the simulated SENTINEL logs, it is possible to extrapolate patterns representing the set of crimes committed within the same event. For example, identifying a pattern whereby every time there is a violation relating to an aspect of workers' payment, in the same report there is also a violation relating to workers' health. The analysis of these patterns is useful for making strategic decisions regarding the analyzed phenomenon.

Furthermore, the analysis showed that when comparing the SENTINEL system in the version without log files, which therefore cannot use MonPoly to implement crime tracking, and still wanting to obtain a similar result, we would have had to extract the data directly from the database, in the sense that for each formula created previously we would have to write an SQL query. Retrieving data from the database can have a strong impact on memory and time, especially if the data of interest is scattered across several tables or can only be obtained through cross-checking between multiple tables. To obtain the same data intuitively we would consume more memory than we would consume by scrolling through the log strings already created with the extended SENTINEL system using the data inserted at runtime.

In general, some characteristics of the RV method can be seen as limitations, such as the fact that the correctness of a property is specific to the analysis performed on a finite trace of system events. The same property applied on other logs could lead to a different verification result.

However, if we evaluate the runtime verification method in the context for which SENTINEL was created, i.e. the monitoring of possible criminal events, we are aware that it is not possible to create a universal and certain model of crimes which can then be subsequently verified, at the more this could be a probabilistic model. This means that, in our analysis case this limitation turns into an advantage, as it makes sense to apply runtime verification and not for example other verification methods such as model checking where the creation of a model is necessary.

As always, the power of the method used lies in the knowledge of what can be done through that method. Being aware of the limitations of runtime verification allows to focus on what it is most useful for and therefore take advantage of the speed of the method to analyze finite traces of events.

7.2. Conclusions

In recent years, online monitoring of real-time data has gained more and more attention from experts, so much that a workshop on RV was born in 2001 and then it has been transformed into a conference held every year from 2010 onwards. The reason for this increasing attention is that runtime monitors are essential for evaluating the performance and for evaluating the integrity of a running system and are an integral part of the detection of failures and consequently for the implementation of the necessary countermeasures when such failures occur at runtime.

In a fast-paced world, having access to the data extrapolated from real-time runtime verification is extremely useful for better managing the actions to be taken in the immediate future to preserve the system and make the best use of it.

In such a context, this research has demonstrated how it is possible to use the application of a verification method to SENTINEL system not only to verify its basic properties, but also to evaluate the succession of events that are recorded by it, i.e. the succession of possible crimes.

This allows, in the specific case of SENTINEL Monitor, to make the necessary considerations on the system and also on the criminal events monitored in real time, to make strategic decisions and drastically reduce this phenomenon.

7.3. Final considerations

The verification of SENTINEL events is carried out taking into account the laws of the Netherlands, but could also be extended to all member states of the European Union in the future.
A single system that offers an overall vision would make it possible to identify the differences between the various states in terms of labor exploitation and would allow us to emphasize the differences between neighboring states. Having a general picture of the situation would allow us to understand the phenomena underlying this problem in the Netherlands, but also in other European Union countries.

A big step forward would be to create a free and accessible platform that declares the average working conditions in a given sector and in a given region in order to make people more aware of what they deserve based on their skills and put them in a position to claim it.

Bibliography

[1] Marije Braakman, Saskia Van Bon, Gregor Walz, and Igor Boog. Social fieldwork research (franet) severe forms of labour exploitation supporting victims of severe forms of labour exploitation in having access to justice in eu member states. 2014.

[2] Russell G. Smith Matt Black. Electronic monitoring in the criminal justice system. *U.S. Department of justice, Office of justice programs*, 2003.

[3] Mühlhäuser M. Tundis A, Kaleem H. Detecting and tracking criminals in the real world through an iot-based system. 2020.

[4] Silvia Mera Francisca Sassetti and Dr Hannah Thinyane. Apprise audit impact assessment: Detecting labour exploitation in supply chains. 2019.

[5] Ezio Bartocci, Ylies Falcone, Adrian Francalanza, and Giles Reger. Introduction to runtime verification. 2001.

[6] Joshua Schneider, David Basin, Srđan Krstić, and Dmitriy Traytel. A formally verified monitor for metric first-order temporal logic. pages 310–328, 2019.

[7] Slim REKHIS and Noureddine BOUDRIGA. A formal logic-based language and an automated verification tool for computer forensic investigation. page 287–291, 2005.

[8] Quentin Rossy and Olivier Ribaux. A collaborative approach for incorporating forensic case data into crime investigation using criminal intelligence analysis and visualisation. *Science Justice*, 54(2):146–153, 2014.

[9] Quist-Aphetsi Kester. Visualization and analysis of geographical crime patterns using formal concept analysis. *CoRR*, abs/1307.8112, 2013.

[10] Ezio Bartocci and Yliès Falcone. Runtime verification and enforcement, the (industrial) application perspective (track introduction). 9953:333–338, 10 2016.

[11] Ahrendt Wolfgang Bartocci Ezio Bianculli Domenico Colombo Christian Falcone Yliès Francalanza Adrian Krstić Srđan Lourenço Joao M. Nickovic Dejan Pace Gordon J. Rufino Jose Signoles Julien Traytel Dmitriy Weiss Alexander Sánchez César,

Schneider Gerardo. A survey of challenges for runtime verification from advanced application domains (beyond software). *Formal Methods in System Design*, 2019.

[12] Luca Franceschini Dario Olianas Marina Ribaudo Filippo Ricca Maurizio Leotta, Davide Ancona. Towards a runtime verification approach for internet of things systems. 2018.

[13] Martin Leucker. Teaching runtime verification. 7186:34–48, 2012.

[14] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, may 2009.

[15] John Harrison. Theorem proving for verification (invited tutorial). pages 11–18, 2008.

[16] HaiboYU Hao ZHONG Jianjun ZHAO Anil Kumar KARNA, Yuting CHEN. The role of model checking in software engineering. 2018.

[17] Christian Colombo, Department of Computer, and Science. Combining testing and runtime verification.

[18] Klaus Havelund, Giles Reger, and Grigore Roşu. Runtime verification past experiences and future projections. pages 532–562, 10 2019.

[19] Mounier Laurent Falcone Yliès, Fernandez Jean-Claude. What can you verify and enforce at runtime? *International Journal on Software Tools for Technology Transfer*, 2012.

[20] Georgios Fainekos, Bardh Hoxha, and Sriram Sankaranarayanan. Robustness of specifications and its applications to falsification, parameter mining, and runtime monitoring with s-taliro. pages 27–47, 2019.

[21] Yliès Falcone and Srinivas Pinisetty. On the runtime enforcement of timed properties. pages 48–69, 2019.

[22] Hazem Torfah. Stream-based monitors for real-time properties. pages 91–110, 2019.

[23] Davide Russo. *Sentinel: A dashboard to fight labour exploitation.* 2022.

[24] Amir Pnueli. The temporal logic of programs. *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57, 1977.

[25] Tobias Salzmann Bernd Finkbeiner, Felix Klein. Automata, games, and verification. 2015.

[26] Christian Colombo and Gordon J. Pace. *Linear Temporal Logic*, pages 109–122. Springer International Publishing, Cham, 2022.

[27] David Basin, Matúš Harvan, Felix Klaedtke, and Eugen Zalinescu. Monpoly: Monitoring usage-control policies. 7186:360–364, 01 2012.

[28] David Basin, Felix Klaedtke, Srdjan Marinovic, and Eugen Zălinescu. Monitoring of temporal first-order properties with aggregations. pages 40–58, 2013.

[29] David Basin, Felix Klaedtke, and Eugen Zlinescu. The monpoly monitoring tool. *Kalpa Publications in Computing*, 2017.

A | Appendix A: Possible victimization in SENTINEL

HOUSING

- Poorly housed - Fire safety not in order (violation of the Building Decree)
- No registration in the Basic Register of Persons (BRP) - explanation withheld
- Housing directly linked to activities. By losing the work, this person also loses his/her housing
- Rent deducted from wages without SNF quality mark
- Illegal occupation of (outside) building (violation of zoning plan)
- Staying in the employer's business premises
- Having to share a small living space with (unknown) colleagues
- Costs of services provided by the employer (rent, transport, etc.) are disproportionately high
- Brought to the Netherlands with false promises
- Not aware of residence address
- Being pressured in other ways by the employer
- Staying at the employer's private address
- Passport taken as deposit
- Have to pay compensation for used transport
- Maximum number of households (according to zoning plan) is exceeded
- What is the maximum number of households according to the zoning plan?
- How many people live in the house?

HEALTH

- Anxious for employer
- Threatened by employer
- Forced to work
- Abused by employer
- Can't do anything else (withdraw) than to undergo exploitation
- Being pressured in other ways by the employer
- Passport taken as deposit
- Person required care (accident at work or otherwise required physical care) and turned out not to be insured without being aware of this and to have paid for insurance
- Person does not have health insurance and is aware of this
- Living situation very unhygienic (unhealthy living conditions such as mold, etc.)
- Visible physical injury (e.g. broken arm)

PAYMENT

- Having to pay a fine that is not stated in the employment contract
- Contract only in NL and person does not speak NL
- No contract signed
- No payslip received
- No vacation money / vacation days
- Rent deducted from wages without SNF quality mark
- Unable to show contract
- Costs of services provided by the employer (rent, transport, etc.) are disproportionately high
- Not being able to have your own bank account
- Being pressured in other ways by the employer
- Payment term exceeded (>30 days)

- Passport taken as deposit
- Debt to employer
- Have to pay compensation for used transport
- Black paid

EMPLOYER

- On the other hand, poorly housed - Unsafe / unhealthy / dangerous living situation
- Anxious for employer
- Threatened by employer
- Having to pay a fine that is not stated in the employment contract
- Contract only in NL and person does not speak NL
- Blackmailed by employer
- Forced to work
- No contract signed (>3 mdn work at 16 hours p/w)
- No registration in the Basic Register of Persons (BRP) - explanation withheld
- No payslip received
- No explanation about BRP registration + rental property rights
- No vacation money / vacation days
- Having to do dangerous work (due to employer negligence)
- Staying in the employer's business premises
- Brought to the Netherlands with false promises
- Abused by employer
- Can't do anything else (withdraw) than to undergo exploitation
- Not aware of residence address
- Not aware of where work will take place p/d
- Being pressured in other ways by the employer
- Staying at the employer's private address

- Passport taken as deposit
- Debt to employer
- Have to pay compensation for used transport
- Person does not have health insurance and is aware of this

WORK

- Anxious for employer
- Contract only in NL and person does not speak NL
- Blackmailed by employer
- Forced to work
- No contract signed (>3 mdn work at 16 hours p/w)
- Having to do dangerous work (due to employer negligence)
- Housing directly linked to activities. By losing the work, this person also loses his/her housing
- Brought to the Netherlands with false promises
- Abused by employer
- Can't do anything else (withdraw) than to undergo exploitation
- Not aware of where work will take place p/d
- Being pressured in other ways by the employer
- Having to do unhealthy work
- Passport taken as deposit
- Debt to employer
- Have to pay compensation for used transport
- Working days $>$ 8 hours

B | Appendix B: MonPoly Grammar

The Monpoly Grammar has been taken from reference [29].

B.1. Signatures

<signature> ::= <predicate> <signature> | <empty>
<predicate> ::= <string> '(' <sorts> ')'
<sort-list> ::= <sort> ','<sort-list> | <sort> | <empty>
<sort> ::= 'string' | 'int' | 'float'

B.2. Log Entries

<log-entry> ::= '@' <ts> <db>
<ts> ::= <integer> | <float>
<db> ::= <table> <db>
<table> ::= <string> <relation>
<relation> ::= <tuple> <relation> | <empty>
<tuple> ::= '(' <fields> ')'
<fields> ::= <string> ','<fields> | <string> | <empty>

B.3. Property Specication Language

<formula> ::=
| '(' <formula> ')'
| 'FALSE'
| 'TRUE'
| <predicate>
| <term> '=' <term>

| <term> ‘<’ <term>
| <term> ‘>’ <term>
| <term> ‘<=’ <term>
| <term> ‘>=’ <term>
| <formula> ‘EQUIV’ <formula>
| <formula> ‘IMPLIES’ <formula>
| <formula> ‘OR’ <formula>
| <formula> ‘AND’ <formula>
| ‘NOT’ <formula>
| ‘EXISTS’ <var-list> ‘.’ <formula>
| ‘FORALL’ <var-list> ‘.’ <formula>
| <var> ‘<-’ <aggreg> <var> ‘;’ <var-list> <formula> // aggregation formula
| <var> ‘<-’ <aggreg> <var> <formula> // variant with no group-by variables
| ‘NEXT’ <interval-opt> <formula>
| ‘PREV’ <interval-opt> <formula>
| ‘EVENTUALLY’ <interval-opt> <formula>
| ‘ONCE’ <interval-opt> <formula>
| ‘ALWAYS’ <interval-opt> <formula>
| ‘PAST ALWAYS’ <interval-opt> <formula>
| <formula> ‘SINCE’ <interval-opt> <formula>
| <formula> ‘UNTIL’ <interval-opt> <formula>
<predicate> ::=
| <string> ‘(’ <term-list> ‘)’
| ‘tp’ ‘(’ <term> ‘)’ // time point predicate
| ‘ts’ ‘(’ <term> ‘)’ // timestamp predicate
| ‘tpts’ ‘(’ <term> ‘,’<term> ‘)’ // time point and timestamp predicate
<aggreg> ::=
| ‘CNT’ // counting aggregation operator
| ‘MIN’ // minimum aggregation operator
| ‘MAX’ // maximum aggregation operator
| ‘SUM’ // sum aggregation operator
| ‘AVG’ // average aggregation operator
| ‘MED’ // median aggregation operator
<interval-opt> ::= <lbound> ‘,’<rbound> | <empty>
<lbound> ::= ‘(’ <bound> | ‘[’ <bound>
<rbound> ::= <bound> ‘)’ | <bound> ‘]’ | ‘*)’
<bound> ::= <integer><unit> | <integer>

```
<unit> ::= ‘s’ | ‘m’ | ‘<’ | ‘d’
<term-list> ::= <term> ‘,’<term-list> | <term> | <empty>
<var-list> ::= <var> ‘,’<var-list> | <var> | <empty>
<term> ::=
| ‘(’ <term> ‘)’
| <term> ‘+’ <term>
| <term> ‘-’ <term>
| <term> ‘*’ <term>
| <term> ‘/’ <term>
| <term> ‘MOD’ <term> // modulo operation
| ‘-’ <term>
| ‘f2i’ ‘(’ <term> ‘)’ // float to integer conversion
| ‘i2f’ ‘(’ <term> ‘)’ // integer to float conversion
| <cst>
| <var>
<cst> ::= <integer> | <rational> | ‘"’ <string> ‘"’
<var> ::= ‘ ’ | <string>
```

C | Appendix C: Cities database

City	Lat	Lng	Country	Province	Population
Amsterdam	523.728	48.936	Netherlands	Noord-Holland	1459402
Rotterdam	519.167	45.000	Netherlands	Zuid-Holland	631155
The Hague	520.800	43.100	Netherlands	Zuid-Holland	548320
Utrecht	520.833	51.167	Netherlands	Utrecht	359370
Maastricht	508.500	56.833	Netherlands	Limburg	277721
Eindhoven	514.333	54.833	Netherlands	Noord-Brabant	235691
Groningen	532.167	65.667	Netherlands	Groningen	233273
Tilburg	515.500	50.833	Netherlands	Noord-Brabant	221947
Almere	523.667	52.167	Netherlands	Flevoland	214715
Breda	515.889	47.758	Netherlands	Noord-Brabant	184126
Nijmegen	518.475	58.625	Netherlands	Gelderland	177659
Arnhem	519.833	59.167	Netherlands	Gelderland	164096
Haarlem	523.833	46.333	Netherlands	Noord-Holland	162902
Apeldoorn	522.167	59.667	Netherlands	Gelderland	161156
Enschede	522.167	69.000	Netherlands	Overijssel	159703
Hoofddorp	523.061	46.907	Netherlands	Noord-Holland	157789
s-Hertogenbosch	516.833	53.000	Netherlands	Noord-Brabant	157486
Amersfoort	521.500	53.833	Netherlands	Utrecht	157462
Zaanstad	524.697	47.767	Netherlands	Noord-Holland	156901
Zwolle	525.167	61.000	Netherlands	Overijssel	129840
Zoetermeer	520.667	45.000	Netherlands	Zuid-Holland	125267
Leiden	521.600	44.900	Netherlands	Zuid-Holland	124899
Leeuwarden	532.000	57.833	Netherlands	Fryslan	124481
Dordrecht	517.958	46.783	Netherlands	Zuid-Holland	119115
Alphen aan den Rijn	521.333	46.667	Netherlands	Zuid-Holland	112587
Alkmaar	526.333	47.500	Netherlands	Noord-Holland	109896
Emmen	527.833	69.000	Netherlands	Drenthe	107471
Delft	520.117	43.592	Netherlands	Zuid-Holland	103581

Deventer	522.500	61.500	Netherlands	Overijssel	101236
Helmond	514.833	56.500	Netherlands	Noord-Brabant	92627
Amstelveen	523.000	48.500	Netherlands	Noord-Holland	91675
Hilversum	522.333	51.667	Netherlands	Noord-Holland	91235
Heerlen	508.833	59.833	Netherlands	Limburg	86936
Purmerend	525.000	49.500	Netherlands	Noord-Holland	81515
Hengelo	522.656	67.931	Netherlands	Overijssel	81049
Lelystad	525.000	54.833	Netherlands	Flevoland	79811
Schiedam	519.167	44.000	Netherlands	Zuid-Holland	79279
Zaandam	524.333	48.333	Netherlands	Noord-Holland	78682
Vlaardingen	519.000	43.500	Netherlands	Zuid-Holland	73924
Gouda	520.111	47.111	Netherlands	Zuid-Holland	73681
Hoorn	526.500	50.667	Netherlands	Noord-Holland	73619
Almelo	523.500	66.667	Netherlands	Overijssel	73132
Spijkenisse	518.500	43.333	Netherlands	Zuid-Holland	72740
Assen	530.000	65.667	Netherlands	Drenthe	68836
Velsen-Zuid	524.667	46.167	Netherlands	Noord-Holland	68617
Capelle aan den IJssel	519.333	45.833	Netherlands	Zuid-Holland	67319
Veenendaal	520.167	55.500	Netherlands	Utrecht	66912
Katwijk	522.000	44.167	Netherlands	Zuid-Holland	65995
Zeist	520.906	52.331	Netherlands	Utrecht	65043
Nieuwegein	520.333	50.833	Netherlands	Utrecht	63866
Scheveningen	521.081	42.731	Netherlands	Zuid-Holland	58850
Heerhugowaard	526.667	48.333	Netherlands	Noord-Holland	58387
Roermond	512.000	59.833	Netherlands	Limburg	57308
Oosterhout	516.431	48.569	Netherlands	Noord-Brabant	56206
Rijswijk	520.456	43.300	Netherlands	Zuid-Holland	55220
Houten	520.333	51.667	Netherlands	Utrecht	50223
Sittard	510.000	58.667	Netherlands	Limburg	49483
Harderwijk	523.500	56.167	Netherlands	Gelderland	48726
Barendrecht	518.500	45.333	Netherlands	Zuid-Holland	48673
IJmuiden	524.586	46.194	Netherlands	Noord-Holland	48320
Middelburg	515.000	36.167	Netherlands	Zeeland	47939
Zutphen	521.400	61.950	Netherlands	Gelderland	47423
Soest	521.833	53.000	Netherlands	Utrecht	46906
Ridderkerk	518.667	46.000	Netherlands	Zuid-Holland	46671
Schagen	527.833	48.000	Netherlands	Noord-Holland	46553

Kerkrade	508.667	60.667	Netherlands	Limburg	45642
Veldhoven	514.200	54.050	Netherlands	Noord-Brabant	45500
Zwijndrecht	518.167	46.500	Netherlands	Zuid-Holland	44775
Zevenaar	519.167	60.667	Netherlands	Gelderland	44096
Noordwijk	522.333	44.333	Netherlands	Zuid-Holland	44062
Etten-Leur	515.667	46.333	Netherlands	Noord-Brabant	43869
Tiel	518.833	54.333	Netherlands	Gelderland	41978
Beverwijk	524.833	46.500	Netherlands	Noord-Holland	41863
Huizen	523.000	52.500	Netherlands	Noord-Holland	41090
Hellevoetsluis	518.167	41.333	Netherlands	Zuid-Holland	40312
Maarssen	521.351	50.413	Netherlands	Utrecht	39752
Wageningen	519.667	56.667	Netherlands	Gelderland	39635
Heemskerk	525.167	46.667	Netherlands	Noord-Holland	39191
Veghel	516.167	55.500	Netherlands	Noord-Brabant	38077
Teijlingen	522.150	45.103	Netherlands	Zuid-Holland	37791
Landgraaf	509.083	60.297	Netherlands	Limburg	37591
Gorinchem	518.306	49.742	Netherlands	Zuid-Holland	37410
Hoogvliet	518.667	43.500	Netherlands	Zuid-Holland	35575
Berg en Dal	518.167	59.167	Netherlands	Gelderland	35010
Maassluis	519.333	42.333	Netherlands	Zuid-Holland	33567
Bussum	522.833	51.667	Netherlands	Noord-Holland	32410
Papendrecht	518.333	46.833	Netherlands	Zuid-Holland	32290
Aalsmeer	522.667	47.500	Netherlands	Noord-Holland	31991
Oldenzaal	523.167	69.333	Netherlands	Overijssel	31840
Vught	516.500	53.000	Netherlands	Noord-Brabant	31669
Nieuw-Vennep	522.644	46.347	Netherlands	Noord-Holland	31415
Waddinxveen	520.500	46.500	Netherlands	Zuid-Holland	31342
Diemen	523.333	49.667	Netherlands	Noord-Holland	31334
Hendrik-Ido-Ambacht	518.500	46.333	Netherlands	Zuid-Holland	31258
Rosmalen	517.167	53.667	Netherlands	Noord-Brabant	31219
Best	515.167	54.000	Netherlands	Noord-Brabant	30216
Uithoorn	522.333	48.333	Netherlands	Noord-Holland	30206
Krimpen aan den IJssel	519.167	45.833	Netherlands	Zuid-Holland	29410
Culemborg	519.500	52.333	Netherlands	Gelderland	29121
Brunssum	509.500	59.667	Netherlands	Limburg	28103
Geldrop	514.222	55.578	Netherlands	Noord-Brabant	28001
Langedijk	526.936	47.944	Netherlands	Noord-Holland	27992

Vleuten	521.081	50.150	Netherlands	Utrecht	27815
Heemstede	523.500	46.167	Netherlands	Noord-Holland	27545
Leiderdorp	521.667	45.333	Netherlands	Zuid-Holland	27377
Blerick	513.667	61.500	Netherlands	Limburg	27330
Pijnacker	520.167	44.333	Netherlands	Zuid-Holland	27130
Dongen	516.333	49.333	Netherlands	Noord-Brabant	26368
Voorschoten	521.333	44.500	Netherlands	Zuid-Holland	25650
Sliedrecht	518.167	47.667	Netherlands	Zuid-Holland	25597
Oegstgeest	521.833	44.667	Netherlands	Zuid-Holland	25064
Stein	509.667	57.667	Netherlands	Limburg	24961
Baarn	522.167	52.833	Netherlands	Utrecht	24792
Oud-Beijerland	518.167	44.000	Netherlands	Zuid-Holland	24301
Heiloo	526.000	47.167	Netherlands	Noord-Holland	24144
Borne	523.000	67.500	Netherlands	Overijssel	23668
Lisse	522.500	45.500	Netherlands	Zuid-Holland	22982
Volendam	524.994	50.675	Netherlands	Noord-Holland	22715
Hillegom	522.833	45.833	Netherlands	Zuid-Holland	22197
s-Gravenzande	520.000	41.667	Netherlands	Zuid-Holland	22190
De Meern	520.781	50.281	Netherlands	Utrecht	21815
Nuenen	514.733	55.467	Netherlands	Noord-Brabant	20580
Alblasserdam	518.702	46.667	Netherlands	Zuid-Holland	20136
Weesp	523.000	50.500	Netherlands	Noord-Holland	19334
Nootdorp	520.333	44.000	Netherlands	Zuid-Holland	19000
Krommenie	525.000	47.667	Netherlands	Noord-Holland	18955
Naaldwijk	519.931	42.050	Netherlands	Zuid-Holland	18858
Edam	525.167	50.500	Netherlands	Noord-Holland	18828
Enkhuizen	527.000	53.000	Netherlands	Noord-Holland	18637
Hardinxveld-Giessendam	518.167	48.333	Netherlands	Zuid-Holland	18413
Naarden	522.953	51.622	Netherlands	Noord-Holland	17555
Waalre	514.000	54.667	Netherlands	Noord-Brabant	17544
Rijen	515.833	49.500	Netherlands	Noord-Brabant	16800
Glanerbrug	522.150	69.742	Netherlands	Overijssel	16715
Beek	509.333	58.000	Netherlands	Limburg	15929
Schaesberg	509.000	60.167	Netherlands	Limburg	15900
Boskoop	520.667	46.500	Netherlands	Zuid-Holland	15045
Westervoort	519.667	59.667	Netherlands	Gelderland	15014
Sassenheim	522.258	45.225	Netherlands	Zuid-Holland	14886

Julianadorp	528.833	47.333	Netherlands	Noord-Holland	13925
Badhoevedorp	523.333	47.833	Netherlands	Noord-Holland	13290
Raamsdonksveer	516.833	48.667	Netherlands	Noord-Brabant	12470
Rozenburg	519.058	42.486	Netherlands	Zuid-Holland	12455
Blaricum	522.667	52.500	Netherlands	Noord-Holland	11952
Schoonhoven	519.500	48.500	Netherlands	Zuid-Holland	11922
Laren	522.500	52.333	Netherlands	Noord-Holland	11400
Koog aan de Zaan	524.667	48.000	Netherlands	Noord-Holland	11225
Doesburg	520.167	61.333	Netherlands	Gelderland	11148
Leidschendam	520.833	44.000	Netherlands	Zuid-Holland	10482
Heerlerbaan	508.692	60.103	Netherlands	Limburg	9872
Nieuw-Lekkerland	518.833	46.833	Netherlands	Zuid-Holland	9541
Kudelstaart	522.339	47.483	Netherlands	Noord-Holland	9250
Zaandijk	524.667	48.000	Netherlands	Noord-Holland	8600
Zwanenburg	523.833	47.500	Netherlands	Noord-Holland	7935
Limmen	525.667	47.000	Netherlands	Noord-Holland	7635
Bolnes	518.947	45.788	Netherlands	Zuid-Holland	7300
Santpoort-Noord	524.225	46.261	Netherlands	Noord-Holland	7240
Soesterberg	521.194	52.831	Netherlands	Utrecht	7210
Reuver	512.850	60.792	Netherlands	Limburg	6120
Surhuisterveen	531.833	61.667	Netherlands	Fryslan	6050
Susteren	510.667	58.667	Netherlands	Limburg	5965
Hintham	517.000	53.500	Netherlands	Noord-Brabant	5910
Meteren	518.639	52.825	Netherlands	Gelderland	5398
Teteringen	516.167	48.333	Netherlands	Noord-Brabant	5371
Duivendrecht	523.333	49.333	Netherlands	Noord-Holland	5300
Empel	517.311	53.272	Netherlands	Noord-Brabant	5160
Arnemuiden	515.000	36.667	Netherlands	Zeeland	5055
Pernis	518.833	43.833	Netherlands	Zuid-Holland	4860
Soestdijk	521.908	52.822	Netherlands	Utrecht	4830
Aerdenhout	523.644	45.972	Netherlands	Noord-Holland	4730
Munstergeleen	509.747	58.667	Netherlands	Limburg	4610
Den Dolder	521.064	52.403	Netherlands	Utrecht	4420
Rijsenhout	522.667	47.000	Netherlands	Noord-Holland	4103
Overveen	523.922	46.175	Netherlands	Noord-Holland	4040
Molenhoek	517.661	58.739	Netherlands	Limburg	3930
De Kwakel	522.408	47.908	Netherlands	Noord-Holland	3880

Kwintsheul	520.167	42.500	Netherlands	Zuid-Holland	3560
Heelsum	519.822	57.525	Netherlands	Gelderland	3540
Nijkerkerveen	521.950	54.667	Netherlands	Gelderland	3530
Santpoort-Zuid	524.167	46.333	Netherlands	Noord-Holland	3370
Goutum	531.786	58.064	Netherlands	Fryslan	3310
Nieuwstadt	510.333	58.667	Netherlands	Limburg	3260
Zwaag	526.679	50.757	Netherlands	Noord-Holland	3155
Oerle	514.222	53.706	Netherlands	Noord-Brabant	2940
Elden	519.667	58.833	Netherlands	Gelderland	2865
Giesbeek	519.944	60.694	Netherlands	Gelderland	2695
Boekelo	522.044	68.019	Netherlands	Overijssel	2595
Vlijmen	516.953	52.119	Netherlands	Noord-Brabant	2577
Bemmel	518.917	58.958	Netherlands	Gelderland	2532
Kralingse Veer	519.249	45.071	Netherlands	Zuid-Holland	2411
Born	510.333	58.111	Netherlands	Limburg	2405
Neerbeek	509.506	58.153	Netherlands	Limburg	2370
Boven-Hardinxveld	518.256	48.381	Netherlands	Zuid-Holland	2227
Huissen	519.333	59.333	Netherlands	Gelderland	2116
Geleen	509.667	58.333	Netherlands	Limburg	2106
Lutjebroek	526.992	52.067	Netherlands	Noord-Holland	2105
Diepenveen	522.894	61.500	Netherlands	Overijssel	2101
Haalderen	518.867	59.300	Netherlands	Gelderland	2061
Helden	513.167	60.000	Netherlands	Limburg	2015
West-Souburg	514.650	35.908	Netherlands	Zeeland	1960
Mijdrecht	522.000	48.667	Netherlands	Utrecht	1960
Assendelft	524.667	47.500	Netherlands	Noord-Holland	1946
Ilpendam	524.667	49.500	Netherlands	Noord-Holland	1845
Lonneker	522.506	69.119	Netherlands	Overijssel	1770
Rijpwetering	522.000	45.833	Netherlands	Zuid-Holland	1640
Halsteren	515.167	42.667	Netherlands	Noord-Brabant	1555
Berghem	517.700	55.747	Netherlands	Noord-Brabant	1550
Aadorp	523.786	66.272	Netherlands	Overijssel	1525
Hooglanderveen	521.889	54.292	Netherlands	Utrecht	1515
Grootebroek	527.000	52.167	Netherlands	Noord-Holland	1454
IJsselmonde	519.000	45.500	Netherlands	Zuid-Holland	1437
Oudshoorn	521.333	46.667	Netherlands	Zuid-Holland	1262
Oud-Vroenhoven	508.478	56.514	Netherlands	Limburg	1220

Rijnsburg	521.833	44.333	Netherlands	Zuid-Holland	1171
Heer	508.406	57.269	Netherlands	Limburg	1152
Nieuwendam	523.917	49.406	Netherlands	Noord-Holland	1120
Ubach over Worms	509.167	60.500	Netherlands	Limburg	1100
Limbricht	510.117	58.369	Netherlands	Limburg	1100
Driehuis	524.472	46.367	Netherlands	Noord-Holland	1080
Zwammerdam	521.058	47.272	Netherlands	Zuid-Holland	1052
Hoensbroek	509.208	59.267	Netherlands	Limburg	1043
Ellecom	520.333	60.833	Netherlands	Gelderland	1035
Bentveld	523.658	45.717	Netherlands	Noord-Holland	1025
Boukoul	512.167	60.500	Netherlands	Limburg	1010
Hoogkarspel	526.947	51.778	Netherlands	Noord-Holland	884
Bovenkarspel	527.000	52.500	Netherlands	Noord-Holland	773
Zuid-Scharwoude	526.833	48.167	Netherlands	Noord-Holland	642
Poortugaal	518.667	44.000	Netherlands	Zuid-Holland	640
Odijk	520.503	52.333	Netherlands	Utrecht	339

D | Appendix D: Simulated log file

```
@1699353407120 situation(1, 67, Ilpendam, Noord-Holland, true, true, true, true, true, false, true, false, true, false)
@1699353407153 situation(2, 96, Meteren, Gelderland, true, true, true, true, true, true, false, true, false, true)
@1699353407154 situation(3, 2, Hoogvliet, Zuid-Holland, true, true, true, true, true, false, true, false, true, false)
@1699353407154 situation(4, 10, Edam, Noord-Holland, true, true, true, true, true, false, false, true, false, true)
@1699353407155 situation(5, 48, Sliedrecht, Zuid-Holland, true, true, true, true, true, true, true, false, false, true)
@1699353407156 situation(6, 81, Santpoort-Noord, Noord-Holland, true, true, true, true, true, true, true, false, true, true)
@1699353407157 situation(7, 92, Molenhoek, Limburg, true, true, true, true, true, false, false, false, true, true)
@1699353407158 situation(8, 85, Eindhoven, Noord-Brabant, true, true, true, true, true, true, false, true, false, true)
@1699353407159 situation(9, 71, Alphen-aan-den-Rijn, Zuid-Holland, true, true, true, true, true, true, false, false, true, false)
@1699353407159 situation(10, 74, Hooglanderveen, Utrecht, true, true, true, true, true, false, true, false, true, false)
@1699353407160 situation(11, 32, Voorschoten, Zuid-Holland, true, true, true, true, true, false, true, true, false, false)
@1699353407161 situation(12, 80, Santpoort-Zuid, Noord-Holland, true, true, true, true, true, true, true, true, false, false)
@1699353407162 situation(13, 64, Oegstgeest, Zuid-Holland, true, true, true, true, true, true, false, true, true, true)
@1699353407163 situation(14, 73, Roermond, Limburg, true, true, true, true, true, false,
true, false, true, true)
@1699353407164 situation(15, 46, Heer, Limburg, true, true, true, true, true, true, false,
true, true, false)
@1699353407165 situation(16, 41, Kudelstaart, Noord-Holland, true, true, true, true, true,
true, false, true, false, true)
@1699353407166 situation(17, 43, Soesterberg, Utrecht, true, true, true, true, true, false,
false, true, false, true)
@1699353407167 situation(18, 86, Enschede, Overijssel, true, true, true, true, true, false,
false, false, true, true)
@1699353407167 situation(19, 3, Rozenburg, Zuid-Holland, true, true, true, true, true,
false, true, true, false, false)
@1699353407168 situation(20, 96, Glanerbrug, Overijssel, true, true, true, true, true, true,
true, true, true, true)
@1699353407169 situation(21, 66, Kerkrade, Limburg, true, true, true, true, true, false,
false, true, true, false)
@1699353407170 situation(22, 73, Heiloo, Noord-Holland, true, true, true, true, true, true,
true, true, false, true)
@1699353407171 situation(23, 29, Houten, Utrecht, true, true, true, true, true, false, true,
true, false, false)
@1699353407172 situation(24, 65, s-Gravenzande, Zuid-Holland, true, true, true, true,
true, false, false, true, false, true)
@1699353407173 situation(25, 41, Purmerend, Noord-Holland, true, true, true, true, true,
true, false, true, false, true)
@1699353407173 situation(26, 44, Zwaag, Noord-Holland, true, true, true, true, true,
false, false, false, false, true)
@1699353407174 situation(27, 91, Boven-Hardinxveld, Zuid-Holland, true, true, true,
true, true, true, false, true, false, false)
@1699353407175 situation(28, 8, Heerlen, Limburg, true, true, true, true, true, false, true,
false, false, true)
@1699353407176 situation(29, 86, De-Meern, Utrecht, true, true, true, true, true, true,
false, false, false, false)
@1699353407177 situation(30, 50, Eindhoven, Noord-Brabant, true, true, true, true, true,
true, false, false, true, false)
@1699353407177 situation(31, 59, Duivendrecht, Noord-Holland, true, true, true, true,
true, true, false, false, true, true)
@1699353407178 situation(32, 47, Nieuwegein, Utrecht, true, true, true, true, true, true,
false, true, true, false)
```

```
@1699353407179 situation(33, 45, Pijnacker, Zuid-Holland, true, true, true, true, true,
false, false, false, false, false)
@1699353407180 situation(34, 98, Maastricht, Limburg, true, true, true, true, true, false,
false, true, true, true)
@1699353407181 situation(35, 26, Barendrecht, Zuid-Holland, true, true, true, true, true,
true, false, true, false, false)
@1699353407182 situation(36, 58, Alkmaar, Noord-Holland, true, true, true, true, true,
true, false, false, true, true)
@1699353407183 situation(37, 58, Rozenburg, Zuid-Holland, true, true, true, true, true,
true, false, true, false, true)
@1699353407183 situation(38, 35, Hengelo, Overijssel, true, true, true, true, true, false,
false, false, false, true)
@1699353407184 situation(39, 89, Hooglanderveen, Utrecht, true, true, true, true, true,
false, false, true, true, true)
@1699353407185 situation(40, 1, Apeldoorn, Gelderland, true, true, true, true, true, true,
true, true, false, false)
@1699353407185 situation(41, 83, Haarlem, Noord-Holland, true, true, true, true, true,
true, true, true, false, true)
@1699353407186 situation(42, 52, Huissen, Gelderland, true, true, true, true, true, false,
true, true, false, true)
@1699353407186 situation(43, 64, Nieuwstadt, Limburg, true, true, true, true, true, false,
true, true, true, false)
@1699353407187 situation(44, 8, Scheveningen, Zuid-Holland, true, true, true, true, true,
false, false, false, true, false)
@1699353407188 situation(45, 88, Nuenen, Noord-Brabant, true, true, true, true, true,
false, false, false, true, true)
@1699353407188 situation(46, 57, s-Hertogenbosch, Noord-Brabant, true, true, true, true,
true, false, false, false, false, true)
@1699353407189 situation(47, 9, IJmuiden, Noord-Holland, true, true, true, true, true,
false, false, true, false, true)
@1699353407189 situation(48, 86, Teijlingen, Zuid-Holland, true, true, true, true, true,
false, false, false, false, true)
@1699353407190 situation(49, 72, Nieuw-Vennep, Noord-Holland, true, true, true, true,
true, true, true, true, true, true)
@1699353407190 situation(50, 12, Hooglanderveen, Utrecht, true, true, true, true, true,
true, false, false, false, true)
@1699353407191 situation(51, 98, Santpoort-Noord, Noord-Holland, true, true, true, true,
true, false, false, false, false, true)
@1699353407192 situation(52, 71, Oerle, Noord-Brabant, true, true, true, true, true, false,
false, true, true, false)
@1699353407192 situation(53, 76, Enkhuizen, Noord-Holland, true, true, true, true, true,
true, false, true, true, true)
@1699353407193 situation(54, 30, Best, Noord-Brabant, true, true, true, true, true, false,
false, false, true, true)
@1699353407194 situation(55, 33, Geleen, Limburg, true, true, true, true, true, false, true,
true, true, false)
@1699353407195 situation(56, 10, Leeuwarden, Fryslan, true, true, true, true, true, true,
false, false, false, false)
@1699353407196 situation(57, 25, Rijswijk, Zuid-Holland, true, true, true, true, true,
false, true, true, false, false)
@1699353407196 situation(58, 98, Bussum, Noord-Holland, true, true, true, true, true,
false, true, true, true, true)
@1699353407197 situation(59, 46, Weesp, Noord-Holland, true, true, true, true, true,
false, false, true, false, true)
@1699353407198 situation(60, 62, Zwolle, Overijssel, true, true, true, true, true, false,
false, true, true, true)
@1699353407198 situation(61, 45, Hoogkarspel, Noord-Holland, true, true, true, true,
true, false, false, false, true, false)
@1699353407199 situation(62, 32, Heelsum, Gelderland, true, true, true, true, true, false,
false, true, false, true)
@1699353407199 situation(63, 94, Waddinxveen, Zuid-Holland, true, true, true, true, true,
true, false, true, true, true)
@1699353407200 situation(64, 79, Beek, Limburg, true, true, true, true, true, false, false,
true, false, false)
@1699353407200 situation(65, 52, Haalderen, Gelderland, true, true, true, true, true, true,
true, true, false, false)
@1699353407201 situation(66, 58, Katwijk, Zuid-Holland, true, true, true, true, true, true,
false, false, false, true)
@1699353407201 situation(67, 46, Hoogkarspel, Noord-Holland, true, true, true, true,
true, false, true, true, true, true)
@1699353407202 situation(68, 42, Heerhugowaard, Noord-Holland, true, true, true, true,
true, true, false, true, false, true)
@1699353407202 situation(69, 43, Enschede, Overijssel, true, true, true, true, true, false,
true, true, false, false)
```

```
@1699353407203 situation(70, 24, Dongen, Noord-Brabant, true, true, true, true, true, false, true, false, false, false)
@1699353407203 situation(71, 45, Delft, Zuid-Holland, true, true, true, true, true, true, true, false, true, false)
@1699353407204 situation(72, 26, Molenhoek, Limburg, true, true, true, true, true, true, false, true, false, true)
@1699353407205 situation(73, 22, Ellecom, Gelderland, true, true, true, true, true, false, true, false, false, true)
@1699353407205 situation(74, 77, Elden, Gelderland, true, true, true, true, true, false, true, false, true, true)
@1699353407206 situation(75, 39, Heiloo, Noord-Holland, true, true, true, true, true, true, true, false, false, true)
@1699353407206 situation(76, 46, Limbricht, Limburg, true, true, true, true, true, false, false, false, false, true)
@1699353407207 situation(77, 4, Bemmel, Gelderland, true, true, true, true, true, true, false, true, false, true)
@1699353407207 situation(78, 95, Leeuwarden, Fryslan, true, true, true, true, true, false, false, false, true, false)
@1699353407208 situation(79, 26, Krimpen-aan-den-IJssel, Zuid-Holland, true, true, true, true, true, false, false, true, false, true)
@1699353407208 situation(80, 91, Heer, Limburg, true, true, true, true, true, true, true, true, true, false)
@1699353407209 situation(81, 41, Roermond, Limburg, true, true, true, true, true, false, false, false, true, false)
@1699353407209 situation(82, 74, Raamsdonksveer, Noord-Brabant, true, true, true, true, true, true, false, false, true, false)
@1699353407210 situation(83, 10, Zuid-Scharwoude, Noord-Holland, true, true, true, true, true, true, true, true, true, false)
@1699353407210 situation(84, 73, Katwijk, Zuid-Holland, true, true, true, true, true, true, false, false, false, true)
@1699353407212 situation(85, 14, Capelle-aan-den-IJssel, Zuid-Holland, true, true, true, true, true, true, true, false, false, true)
@1699353407214 situation(86, 15, Bolnes, Zuid-Holland, true, true, true, true, true, true, false, true, false, false)
@1699353407215 situation(87, 51, Goutum, Fryslan, true, true, true, true, true, true, false, false, false, false)
@1699353407216 situation(88, 28, Houten, Utrecht, true, true, true, true, true, false, true,
false, true, true)
@1699353407217 situation(89, 53, Apeldoorn, Gelderland, true, true, true, true, true,
true, false, true, true, false)
@1699353407217 situation(90, 78, Oerle, Noord-Brabant, true, true, true, true, true, true,
false, false, true, true)
@1699353407218 situation(91, 33, Ilpendam, Noord-Holland, true, true, true, true, true,
true, false, false, false, false)
@1699353407218 situation(92, 44, Assen, Drenthe, true, true, true, true, true, false, true,
true, false, true)
@1699353407219 situation(93, 44, Leeuwarden, Fryslan, true, true, true, true, true, false,
true, true, true, true)
@1699353407219 situation(94, 31, Zwanenburg, Noord-Holland, true, true, true, true,
true, true, false, true, true, true)
@1699353407220 situation(95, 1, Kerkrade, Limburg, true, true, true, true, true, true,
true, false, true, false)
@1699353407221 situation(96, 2, Deventer, Overijssel, true, true, true, true, true, true,
false, true, true, false)
@1699353407222 situation(97, 11, Nieuwegein, Utrecht, true, true, true, true, true, true,
true, true, false, false)
@1699353407222 situation(98, 23, Heemstede, Noord-Holland, true, true, true, true, true,
false, false, true, true, true)
@1699353407222 situation(99, 65, Hoensbroek, Limburg, true, true, true, true, true, true,
false, false, true, true)
@1699353407223 situation(100, 36, Scheveningen, Zuid-Holland, true, true, true, true,
true, true, false, false, false, true)
```

List of Figures

List of Tables

List of Symbols

Variable	Description
$\bigcirc\varphi$	**Next**: φ has to hold at the next state.
$\bullet\varphi$	**Previous**: φ has hold at the previous state.
$\Diamond\varphi$	**Eventually**: φ eventually has to hold (somewhere on the subsequent path).
$\blacklozenge\varphi$	**Once**: φ once has to hold (somewhere on the previous path).
$\Box\varphi$	**Always**: φ has to hold on the entire subsequent path.
$\blacksquare\varphi$	**Past always**: φ has hold on the entire previous path.
$\forall\varphi$	**All**: φ has to hold on all paths starting from the current state.
$\exists\varphi$	**Exists**: there exists at least one path starting from the current state where φ holds.

List of Acronyms

Short name	Long name	Description
RV	Runtime Verification	A computing system analysis and execution approach based on extracting information from a running system and using it to detect and possibly react to observed behaviors satisfying or violating certain properties.
CRV	Competition on Runtime Verification	Competition created with the aim of promoting the study and evaluation of new software runtime verification tools.
LTL	Linear Temporal Logic	Extension of modal logic and provides a linear, discrete, future-oriented and infinite organization of time.
MFOTL	Metric First Order Temporal Logic	Extensions of LTL that allow us to express timing constraints on temporal operators and suitable for formalizing data relationships.

Acknowledgements

I would like to express special thanks to Prof. Marcello Maria Bersani and Prof. Damian Andrew Tamburri for the constant support, professionalism and precious advice they gave me for the realization of this thesis. It was a pleasure collaborating with you.

I want to thank Prof. Fabiano Pecorelli, who enormously influenced the success of this work.

A thanks goes to Politecnico di Milano and to all the people who work with dedication to make it a place of professional and personal growth.

Finally, I want to thank my family immensely for always being by my side, through all ups and downs of this journey which comes to an end today and which I will certainly never forget.