electronics

MDPI

Article

Detecting Data Anomalies from Their Formal Specifications: A Case Study in IoT Systems

Benjamin Aziz

School of Computing, University of Portsmouth, Portsmouth PO1 3HE, UK; benjamin.aziz@port.ac.uk

Abstract: We present in this paper a new method in detecting anomalies in datasets representing systems behaviour, which is based on comparing a dataset to the data blueprint of the system representing its normal behaviour. This method removes some of the need for applying complex machine learning algorithms that aim at detecting abnormalities in such datasets and gives a more assured outcome of the presence of abnormalities. Our method first models a system using the formal language of the π-calculus, and then applies an abstract interpretation that ultimately generates an abstract multiset representing the messages exchanged in the system model. We term this multiset as the data blueprint of the system, and it represents the normal behaviour expected. We apply this method to the case of a recent study in literature, which attempts to analyse normal and abnormal behaviour in datasets representing runs of the MQTT protocol, both under attack and no attack conditions. We show that our method is able to detect these conditions in an easier and more straightforward manner than the original case study attempts to.

Keywords: data analysis; data anomalies; formal specifications; IoT systems; process algebra

check for updates

Citation: Aziz, B. Detecting Data Anomalies from Their Formal Specifications: A Case Study in IoT Systems. *Electronics* **2023**, *12*, 630. https://doi.org/10.3390/electronics12030630

Academic Editor: Carlo Olivieri

Received: 30 December 2022
Revised: 20 January 2023
Accepted: 25 January 2023
Published: 27 January 2023

1. Introduction

Despite the increasing popularity of the Internet of Things (IoT) systems, we find that the majority of IoT literature is mostly concerned with either investigating such systems as sources of data and information (e.g., Ref. [1]), often to be used as validation datasets for machine learning algorithms that detect different classes of properties in these datasets (e.g., Refs. [2–4]), or with the classical software engineering aspect of these systems that aims to achieve specific IoT-based solutions to all sorts of problems in different domain of applications (e.g., Refs. [5,6]). There is, in fact, a lack of understanding of the relationship between these two views; the data-based model of IoT systems, and their formal models. Understanding one form of this relationship, and its usefulness, will be the focus of our research here, combining these two views.

In this paper, we propose a novel methodology; that abstract data representing the normal behaviour of a communicating system, derived by statically analysing its formal specification, can be used afterwards to detect and filter out real data representing abnormal behaviour of some implementation of the same system during runtime. We call such abstract data representing the normal behaviour the *data blueprint of the system*, against which data from its runtime behaviour are compared. We demonstrate that even the simplest of static analyses, in this case one that produces a data blueprint of the system on merely what data should be communicated, would suffice in simplifying anomaly detection of the system that would otherwise require the application of complex machine learning methods.

We apply our new method to one popular IoT protocol, namely the MQ Telemetry Transport (MQTT) protocol [7], specifically to the QoS 0 mode of this protocol. We first model the protocol in the formal language of the π-calculus [8], and then use an abstract interpretation of this language that captures message communications to provide the data blueprint for the protocol. We then compare this to some actual message communications

captured during a study in literature, and we demonstrate that our data blueprint is capable easily of detecting anomalies in these communications.

The rest of the paper is structured as follows. In Section 2, we discuss a few examples of literature to our approach. In Section 3, we give an overview of the formal language used for modelling MQTT, the π-calculus. In Section 4, we define a name-substitution abstract semantics for the π-calculus, which allows us to trace where messages are communicated to. In Section 5, we demonstrate how the data blueprint generated from the abstract semantics is used to detect anomalies for the case of an example study from literature. Finally, in Section 6, we conclude the paper and provide directions for future expansion of the ideas within.

2. Related Work

There is much literature [9–14] regarding formally specifying and verifying IoT systems, networks, protocols, and standards, usually for the purpose of understanding the properties of such systems. We give here only a few examples. The author in Ref. [9] proposes the use of the probabilistic model checker, PRISM [10], as a framework for specifying and checking the functional properties of the IoT systems. The work uses probabilities, as quantitative concepts, to be modelled in the formal language. The interaction and interoperability in IoT systems is studied in Ref. [11] using the Tree Query Logic of Ref. [15]. This allows for a multi-layered view of an IoT system to be constructed, and hence the properties verified at the different layers. In Ref. [12], the authors use Event-B [16] to model and verify the properties of IoT communication protocols, such as the presence of duplicate channels, persistent sessions, and message ordering, in MQTT [7], MQTT-SN [17], and CoAP [18]. Other efforts in this area focus on more specific goals, for example, the verification of security properties through the discovery of vulnerability surfaces, as in Refs. [13,14], for example. None of these works, and others, address the question of how formal specifications can be used to benefit the detection of anomalous behaviour in the recorded data.

One of the research works close to ours is that of Ref. [19], where training datasets related to musical songs are used to construct formal specifications in the form of automata machines capable of playing similar genres of music. In some sense, this is almost the reverse of our approach, which uses formal specifications to understand the dataset itself. Similarly, the work of Ref. [20] proposes the formalisation of metadata specifications in order to discover datasets (more robustly) matching those metadata, more specifically, the discovery of geospatial datasets. Both of these works are related to ours, however, with different motivations, and therefore, approaches.

Formal verification techniques have also been used to verify big data-related applications, tools, platforms, and technologies. Although these are not strictly similar in their approach to ours, since they fall under the verification of system specifications category, we mention a few examples here for completeness. In Ref. [21], the authors propose a stochastic model checking using UPPAAL [22] to model the execution of big data applications, in particular, to model the execution strategies in Apache Spark [23]. In addition, in Ref. [24], formal verification is used to evaluate the execution time of Apache Spark applications, using a combination of directed acyclic graphs and constraint linear temporal logic. In Ref. [25], the authors propose an extension of the i^* requirements engineering methodology [26], called BiStar, more adapted for modelling the requirements of big data applications. The integrity property of BiStar is then modelled and verified using bigraphs [27].

To some extent, one may also consider slightly relevant the most recent work of Ref. [28], which extensively covers the relationship between formal verification in computer science and stochastic analysis in mathematics, with application to big datasets related to 3D road networks and household power consumption. Finally, we should mention the works of Refs. [29–31], who provide extensive reviews of the applications of formal methods to machine learning and the relationship between the two areas.

3. Theory

We specify systems using the formal language of the π-calculus [8], as this is a simple and easy language to formulate, using the following syntax:

$$P ::= \mathbf{0} \mid \pi.P \mid (\nu x)P \mid P|Q \mid P + Q \mid !P$$

$$\pi ::= x(y) \mid \overline{x}\langle y \rangle \mid \tau$$

The syntax states informally that processes $P, Q, \ldots \in \mathcal{P}$ are defined as one of the following: an inactive process, $\mathbf{0}$, incapable of performing any further activity, a guarded process, $\pi.P$, which performs an action π and continues with the residue P, the creation of a new name, $(\nu x)P$, with the scope restricted to P, the parallel composition of two processes, $P|Q$, a non-deterministic choice between two processes, $P + Q$, and finally, replication, $!P$, which can spawn any number of copies of P. The actions, π, are defined using names $x, y \ldots \in \mathcal{N}$, in terms of input actions, $x(y)$, output actions, $\overline{x}\langle y \rangle$, or silent unobservable actions, τ. We refer to the set of the free names of a process as $fn(P)$ and that of the bound names as $bn(P)$, and we assume that initially, the bound names of a process are selected such that no two bound names are the same (α-conversion). We write $n(P) = bn(P) \cup fn(n)$.

The standard semantics of processes is given using the classical structural and operational relations, shown in Figure 1, which determine how a process can change its shape and evolve through communications.

Rules of the $\equiv$ relation:

$(R1)$	$P\|Q$	$\equiv$	$Q\|P$	
$(R2)$	$P\|\mathbf{0}$	$\equiv$	P	
$(R3)$	$(P\|Q)\|R$	$\equiv$	$P\|(Q\|R)$	
$(R4)$	$P + Q$	$\equiv$	$Q + P$	
$(R5)$	$(P + Q) + R$	$\equiv$	$P + (Q + R)$	
$(R6)$	$(\nu x)(P\|Q)$	$\equiv$	$Q\|(\nu x)P$	*if* $x \notin fn(Q)$
$(R7)$	$(\nu x)\mathbf{0}$	$\equiv$	$\mathbf{0}$	
$(R8)$	$(\nu x)(\nu y)P$	$\equiv$	$(\nu y)(\nu x)P$	
$(R9)$	$!P$	$\equiv$	$!P \mid P$	

Rules of the $\xrightarrow{\pi'}$ relation:

$(R10)$	$x(y).P \xrightarrow{x(z)} P$			
$(R11)$	$\overline{x}\langle y \rangle.P \xrightarrow{\overline{x}\langle y \rangle} P$			
$(R12)$	$\tau.P \xrightarrow{\tau} P$			
$(R13)$	$P \xrightarrow{\overline{x}\langle y \rangle} P', Q \xrightarrow{x(z)} Q'$	$\Rightarrow$	$(P\|Q) \longrightarrow (P'\|Q'[y/z])$	
$(R14)$	$P \xrightarrow{\overline{x}(y)} P', Q \xrightarrow{x(z)} Q'$	$\Rightarrow$	$(P\|Q) \longrightarrow (\nu y)(P'\|Q'[y/z])$	
$(R15)$	$P \xrightarrow{\pi'} P'$	$\Rightarrow$	$(P\|Q) \xrightarrow{\pi'} (P'\|Q)$	
$(R16)$	$P \xrightarrow{\pi'} P'$	$\Rightarrow$	$(P + Q) \xrightarrow{\pi'} P'$	
$(R17)$	$P \xrightarrow{\overline{x}\langle y \rangle} P'$	$\Rightarrow$	$(\nu z)P \xrightarrow{\overline{x}\langle y \rangle} (\nu z)P'$	*if* $x \neq z \wedge x \neq y$
$(R18)$	$P \xrightarrow{\overline{x}\langle y \rangle} P'$	$\Rightarrow$	$(\nu z)P \xrightarrow{\overline{x}(y)} (\nu z)P'$	*if* $x \neq z \wedge x = y$
$(R19)$	$P \xrightarrow{x(z)} P'$	$\Rightarrow$	$(\nu y)P \xrightarrow{x(z)} (\nu y)P'$	*if* $x \neq y \wedge x \neq z$
			if $x = z$ then apply α-conversion to distinguish the two names	
$(R20)$	$P \xrightarrow{\tau} P'$	$\Rightarrow$	$(\nu x)P \xrightarrow{\tau} (\nu x)P'$	

Figure 1. Rules of the structural operational semantics for the π-calculus.

We describe first the rules of the structural relation $\equiv$. Rule $(R1)$–$(R3)$ state that $(\mathcal{P}, |, \mathbf{0})$ is a commutative monoid. Rules $(R4)$ and $(R5)$ state that the non-deterministic choice is commutative and associative over $\mathcal{P}$. Rule $(R6)$ (also known as scope extrusion) allows scope restriction on a newly created name to be removed from the left-hand process, if the name is different from any of its free names. Rule $(R7)$ removes name creation from a null process, and rule $(R8)$ allows the order of name restriction to be swapped on the same process. Finally, rule $(R9)$ allows a replication to spawn a copy of its process.

On the other hand, the rules of the operational relation $\xrightarrow{\pi'}$ are explained as follows. Rules $(R10)$–$(R12)$ allow a guarded process to fire its action, making it observable to the context. Rule $(R13)$ is the most important rule, as it allows communications to take place between matching input and output actions. The result is the replacement of the input action parameter with the message carried by the output action. Both processes will continue with their residual part. Rule $(R14)$ is similar to rule $(R13)$, except it deals with bounded outputs, where the scope of the bounded output message is then restricted to the two residual processes. Rules $(R15)$ and $(R16)$ state that internal communications can also occur under parallel composition and non-deterministic choice composition. In the latter case, the process with the internal communication will continue as the main process, removing the option of the other inactive process. $(R17)$ states that name restriction has no impact on the observation of an output action, if that name is different from the names of the channel and message being observed. Rule $(R18)$ turns a free name output into a bounded name output by restricting the scope of the output message. Rule $(R19)$ states that an input action is possible under name restriction, as long as the restriction is not on the name of the communication channel. If the restricted name is the same as the input parameter, then α-conversion is necessary to distinguish the two names. Finally, rule $(R20)$ states that silent internal actions can take place under any name restriction.

In addition to the above semantics, in Ref. [32], we defined a non-standard name-substitution semantics, which when abstracted using an approximation function, yields a meaning represented by the abstract environment $\phi_{\mathcal{A}} : \mathcal{N}^\sharp \to \wp(\mathcal{N}^\sharp) \in D^\sharp_\perp$, which maps each abstract name to a set of abstract names that can replace that name during a process's interpretation. $\mathcal{N}^\sharp$ represents the set of abstract names. Unlike $\mathcal{N}$, $\mathcal{N}^\sharp$ is defined such that it is finite, and therefore, $\wp(\mathcal{N}^\sharp)$ is also finite. The resulting semantic domain, $D^\sharp_\perp$, consists of all such possible environments and guarantees termination for an abstract interpretation computed over it, such as that defined in Ref. [32]. The bottom element of such domain, $\perp_{D^\sharp} = \phi_{\mathcal{A}0}$, is the empty environment where $\forall x \in \mathcal{N}^\sharp : \phi_{\mathcal{A}0}(x) = \{\}$. In Ref. [32], we gave an abstract interpretation of the π-calculus using the semantic domain $D^\sharp_\perp$, and we showed this to be safe with regards to the name-substitution semantics, similar to other analyses we had defined for different variations of the π-calculus in Refs. [33–35]. We next review this abstract interpretation in more detail.

4. An Abstract Semantics

Let us examine in more detail the rules of the abstract interpretation of the π-calculus, $\mathcal{A} : (\mathcal{P} \times \wp(\mathcal{P}) \times D^\sharp_\perp) \to D^\sharp_\perp$, which captures the property of name substitutions in an abstract manner. These rules are shown in Figure 2 below.

We explain these informally as follows. Rules, $(\mathcal{A}1)$ and $(\mathcal{A}2)$, for null processes and input actions, respectively, do not change the value of the $\phi_{\mathcal{A}}$ environment, since no communications take place in these rules. Rule $(\mathcal{A}3)$ deals with output actions; where the meaning of a process guarded by an output action is given as the union of two $\phi_{\mathcal{A}}$ environments. The first environment reflects all possible communications between the output action and matching input actions in ρ. A communication takes place whenever the sets of values of the two communicating channels have similar name values or share a common name value from previous substitutions. This means that the names of the channels are the same (i.e., the channels are free or restricted names) or that there must have been similar name values substituting both channels earlier in the interpretation (i.e., the

channel names are closed under input actions). The effect of the communication is reflected by adding the value of the message to the value of the substituted input parameter in $\phi_{\mathcal{A}}$. The second environment is an unchanged $\phi_{\mathcal{A}}$, reflecting the option that no communication may take place. Rules $(\mathcal{A}4)$–$(\mathcal{A}7)$, are straightforward. Rule $(\mathcal{A}4)$ removes a silent action, and reinterprets the rest of the process. Rule $(\mathcal{A}5)$ does the same for the case of a newly-created name, given that all bound names are distinct. Rules $(\mathcal{A}6)$ and $(\mathcal{A}7)$ distributed the two sides of parallel composition and choice to the rest of the processes in ρ. The rule for replicated processes, $(\mathcal{A}8)$, attaches subscripts to bound names and tags of the spawned processes according to the number of copy of each process. This is necessary to ensure that these names and tags remain distinct throughout the semantics. The rule uses a least fixed-point calculation for a special function, $\mathcal{F} : \mathbb{N} \times \wp(\mathcal{P}) \times D_{\perp}^{\sharp} \rightarrow D_{\perp}^{\sharp}$, which starts by adding a single copy of P then increments till the least-fixed point is reached.

$$
\begin{array}{lll}
(\mathcal{A}1) & \mathcal{A}([\![\mathbf{0}]\!])\rho\phi_{\mathcal{A}} & = \ \phi_{\mathcal{A}} \\
(\mathcal{A}2) & \mathcal{A}([\![x(y).P]\!])\rho\phi_{\mathcal{A}} & = \ \phi_{\mathcal{A}} \\
(\mathcal{A}3) & \mathcal{A}([\![\overline{x}\langle y\rangle.P]\!])\rho\phi_{\mathcal{A}} & = \ (\bigcup_{\substack{\phi_{\mathcal{A}} \\ x'(z).Q \in \rho}} \mathcal{R}([\![\rho[Q/x'(z).Q] \uplus \{\!|P|\!\}]\!])\phi'_{\mathcal{A}}) \cup_{\phi_{\mathcal{A}}} \phi_{\mathcal{A}} \\
 & & \quad \textit{where, } \phi'_{\mathcal{A}} = \phi_{\mathcal{A}}[z \mapsto \phi_{\mathcal{A}}(z) \cup \{y\}] \\
 & & \quad \textit{and } (\exists e, e' : e \in \phi_{\mathcal{A}}(x) \wedge e' \in \phi_{\mathcal{A}}(x') \wedge e = e') \vee (x = x') \\
(\mathcal{A}4) & \mathcal{A}([\![\tau.P]\!])\rho\phi_{\mathcal{A}} & = \ \mathcal{R}([\![\rho \uplus \{\!|P|\!\}]\!])\phi_{\mathcal{A}} \\
(\mathcal{A}5) & \mathcal{A}([\![(\nu x)P]\!])\rho\phi_{\mathcal{A}} & = \ \mathcal{R}([\![\rho \uplus \{\!|P|\!\}]\!])\phi_{\mathcal{A}} \\
(\mathcal{A}6) & \mathcal{A}([\![P \mid Q]\!])\rho\phi_{\mathcal{A}} & = \ \mathcal{R}([\![\rho \uplus \{\!|P|\!\} \uplus \{\!|Q|\!\}]\!])\phi_{\mathcal{A}} \\
(\mathcal{A}7) & \mathcal{A}([\![P + Q]\!])\rho\phi_{\mathcal{A}} & = \ \mathcal{R}([\![\rho \uplus \{\!|P|\!\} \uplus \{\!|Q|\!\}]\!])\phi_{\mathcal{A}} \\
(\mathcal{A}8) & \mathcal{A}([\![!P]\!])\rho\phi_{\mathcal{A}} & = \ \mathcal{F}\ 1\ \rho\ \phi_{\mathcal{A}} \\
 & \textit{where, } \mathcal{F}(n, \rho, \phi_{\mathcal{A}}) = \textit{let } \phi_1 = \mathcal{A}([\![\prod_{i=1}^{n} ren(P, i)]\!])\rho\phi_{\mathcal{A}} \textit{ in} & \\
 & \quad \textit{let } \phi_2 = \mathcal{A}([\![\prod_{i=1}^{n+1} ren(P, i)]\!])\rho\phi_{\mathcal{A}} \textit{ in} & \\
 & \quad \textit{if } \phi_1 = \phi_2 \textit{ then } \phi_1 \textit{ else } \mathcal{F}\ (i+1)\ \rho\ \phi_{\mathcal{A}} & \\
 & \textit{and, } \forall P \in \mathcal{P}, x \in bn(P) : ren(P, i) = P[\alpha_k(x_i)/x] & \\
(\mathcal{R}0) & \mathcal{R}([\![\rho]\!])\phi_{\mathcal{A}} & = \ \bigcup_{\substack{\phi_{\mathcal{A}} \\ P \in \rho}} \mathcal{A}([\![P]\!])(\rho \backslash P)\phi_{\mathcal{A}}
\end{array}
$$

Figure 2. A name-substitution abstract semantics for the π-calculus.

The abstraction function, $\alpha_k : \mathcal{N} \rightarrow \mathcal{N}^{\sharp}$, constrains the number attached to a new copy of a bound name created during the replication process over some maximum permitted number of copies, $k \in \mathbb{N}$. Hence, $\forall P \in \mathcal{P}, x_i \in bn(P)$, we can define α_k as:

$$\alpha_k(x_i) = \begin{cases} x_i & \textit{if } i \leq k \\ x_k & \textit{otherwise} \end{cases}$$

For example, if we set $k = 3$, and we apply the renaming mechanism to the structural semantics relation, we would get the following congruence:

$$!(x(y).P) \ \equiv \ !(x(y).P) \mid x(y_1).P \mid x(y_2).P \mid x(y_3).P \mid x(y_3).P \mid x(y_3).P \mid \ldots$$

where every copy after the third copy is still approximated to being the third copy.

To simplify our analysis of the next section, we define a variation of $\phi_{\mathcal{A}}$, which instead of having number-distinguished copies of the same name, it contains the same name multiple times. Such a *multiset* version, written as $\phi^{k}_{\mathcal{A}}$, can be defined as follows:

$$\forall x \in (dom(\phi_{\mathcal{A}}) \cup ran(\phi_{\mathcal{A}})) : \ \phi^{k}_{\mathcal{A}} = \phi_{\mathcal{A}}[x/x_i]_{i=1}^{k}$$

where a copy of a name, x_i, is replaced by the original root name, x, up to the maximum k^{th} copy of x occurring in $\phi_{\mathcal{A}}$, where x_i is either an input parameter or a newly created name. We now arrive at the definition of a *data blueprint* for some process.

Definition 1. *Define the data blueprint, $\psi^k_{\mathcal{A}blueprintP}$, for a process, P, as follows:* $\psi^k_{\mathcal{A}blueprintP} = \{y : \forall x \in bn(P).y \in \phi^k_{\mathcal{A}}(x) \wedge (\phi_{\mathcal{A}} = \mathcal{A}([P])\{\}\phi_{\mathcal{A}0})\}$

Hence, the blueprint environment only reflects the communicated messages, ignoring the input parameters these instantiate.

5. Example: MQTT Data Analysis

The MQ Telemetry Transport (MQTT) protocol [7] is described as a lightweight broker-based publish/subscribe messaging protocol that was designed to allow devices with small processing power and storage, such as those which the IoT is composed of, to communicate over low-bandwidth and unreliable networks. The publish/subscribe message pattern [36], on which MQTT is based, provides for one-to-many message distribution with three varieties of delivery semantics, based on the level of quality of service expected from the protocol. These include the "exactly once" delivery semantics, the "at least once" delivery semantics and the "at most once" delivery semantics. We focus here on the "at most once" semantics, as this is the one most relevant to our example later.

In the "at most once" case, the protocol is configured to deliver messages with the best effort of the underlying communication network, and given that many networks are unreliable, there would be no guarantee that the MQTT messages will be delivered. This protocol, also termed the *QoS 0* protocol, is represented by the following flow of messages and actions:

$$\begin{array}{rcl} \textit{Client} \rightarrow \textit{Server} & : & \textbf{CONNECT} \\ \textit{Server} \rightarrow \textit{Client} & : & \textbf{CONNACK} \\ \textit{Client} \rightarrow \textit{Server} & : & \textbf{PUBLISH} \\ \textit{Server} & : & \textit{Publish message to subscribers} \end{array}$$

The protocol defines the message communications between *Clients*, i.e., end-devices responsible for generating data from their domain (the data source), and *Servers*, i.e., brokers responsible for collecting source data from clients, based on specific topics, and publishing this data to interested subscribers. We can define the π-calculus model of the QoS 0 protocol as shown in Figure 3.

QoS 0 Protocol:

$$\textit{Client} \mid \textit{Server} \mid \textit{Subscriber}$$

where:

$$\textit{Client} \stackrel{\texttt{def}}{=} \bar{c}\langle \textit{Connect} \rangle.c(z).!((\nu \textit{Publish})\bar{c}\langle \textit{Publish} \rangle.\mathbf{0})$$

$$\textit{Server} \stackrel{\texttt{def}}{=} c(y).\bar{c}\langle \textit{Connack} \rangle.!(c(x).\overline{pub}\langle x \rangle.\mathbf{0})$$

$$\textit{Subscriber} \stackrel{\texttt{def}}{=} !(pub(u).\mathbf{0})$$

Figure 3. The MQTT QoS 0 protocol as modelled in the language of the π-calculus.

In this model, the protocol consists of three top-level processes: *Client*, *Server* and *Subscriber*. The *Client* process represents any IoT device, which after connecting to the *Server*, will always publish some data by sending them in a *Publish* message, and so on. The *Server* process, on the other hand, after acknowledging the connection message sent by the *Client*, will always receive the published data and send these to the *Subscriber* process. Finally, the *Subscriber* process is always waiting to receive the published data. The model is abstract, but sufficient enough to capture anomalies in data as we show in Section 5.2.

5.1. The QoS 0 Protocol Data Blueprint

Applying a non-uniform abstract interpretation for some value of $k > 1$, e.g., $k = 3$, reveals the following $\phi_{\mathcal{A}}$ environment:

$$\phi_{\mathcal{A}}[y_1 \mapsto Connect, z_1 \mapsto Connack,$$
$$x_1 \mapsto Publish_1, x_2 \mapsto Publish_2, x_3 \mapsto Publish_3, u_1 \mapsto Publish_1, u_2 \mapsto Publish_2, u_3 \mapsto Publish_3]$$

where its $\phi^3_{\mathcal{A}}$ equivalent is defined as follows:

$$\phi^3_{\mathcal{A}}[y \mapsto Connect, z \mapsto Connack,$$
$$x \mapsto \{|Publish, Publish, Publish|\}, u \mapsto \{|Publish, Publish, Publish|\}]$$

From this, we can obtain the following data blueprint for the QoS 0 protocol:

$$\psi^3_{\mathcal{A}blueprintQoS0} = \{|Connect, Connack, Publish, Publish, Publish|\}$$

In this case, we see that there are three copies of the *Publish* message, and a single copy of each of the *Connect* and *Connack*. We consider this analysis as providing the *data blueprint* corresponding to the normal behaviour of any system running an MQTT-based network in the QoS 0 mode up to the choice of $k = 3$. The choice of k will depend on the trade-off between precision and efficiency; larger values of k produce analyses with higher precision, however, these would take a longer time to run. For $k = 1$ (i.e., a uniform analysis), it is impossible for any attacks relying on the repetition or multiplicity of messages to look anomalous with regards to the resulting data blueprint. Therefore, we avoid the case of a uniform analysis.

Property 1 (Normal Behaviour)**.** *We characterise a protocol run as being* normal*, if and only if, for some abstract dataset, $\psi^k_{\mathcal{A}}$, representing a run of the protocol, we have that:*

$$\psi^k_{\mathcal{A}} = \psi^k_{\mathcal{A}blueprint}$$

for some approximation number, k.

5.2. A Case of Intrusion Detection

Now let us review here one example in recent literature, where a study was presented in Ref. [37], as a typical example of the application of machine learning algorithms in analysing dataset features and extraction of interesting information. We use this case study to demonstrate how our concept of data blueprints renders the analysis of intrusions straightforward, avoiding all the complexities associated with a machine learning-based approach, which reveal no better knowledge here compared to what is revealed by our approach.

In Ref. [37], the authors used Wireshark to sniff packets (i.e., messages) exchanged over an MQTT network, both under normal and attack circumstances. This resulted in the following cases.

5.2.1. The Normal Case

In the normal case, the analysis of Ref. [37] (§4.A) captures the following sequence of messages, representing the normal case scenario:

1. `Connect Command`
2. `Connect Ack`
3. `Publish Message`
...
24. `Publish Message`

Applying our abstraction $k = 3$, which limits the instances of the communicated messages to 3, we would obtain the following abstract representation of the above dataset:

$$\psi^3_{\mathcal{A}Normal} = \{|Connect, Connack, Publish, Publish, Publish|\}$$

We can see here that $\psi^3_{\mathcal{A}Normal} = \psi^3_{\mathcal{A}blueprintQoS0}$, and therefore, it indicates normal behaviour of the protocol according to Property 1.

5.2.2. First Attack Scenario

Now, let us consider the second dataset generated in Ref. [37] (§4.B), which contains the following sequence of messages:

1. `Connect Command`
2. `Connect Ack`
...
17. `Connect Command`
18. `Connect Ack`
19. `Publish Message`
...
24. `Publish Message`

Abstracting this dataset using $k = 3$, gives us the following abstract environment:

$$\psi^3_{\mathcal{A}Attack1} = \{|Connect, Connect, Connect, Connack, Connack, Connack, Publish, Publish, Publish|\}$$

In this case, we can clearly see that $\psi^3_{\mathcal{A}Attack1} \neq \psi^3_{\mathcal{A}blueprintQoS0}$ and therefore there is some anomalous behaviour as captured by the dataset of [37] (§4.B).

5.2.3. Second Attack Scenario

Finally, let us consider the dataset generated in Ref. [37] (§4.C), which gives us the following sequence of messages:

1. `Connect Ack`
2. `Connect Command`
3. `Connect Ack`
4. `Connect Command`
5. `Connect Ack`
6. `Publish Message`
7. `Connect Command`
8. `Connect Ack`
9. `Connect Command`
10. `Connect Ack`
11. `Publish Message`
12. `Connect Command`
13. `Connect Ack`
14. `Connect Command`
15. `Connect Ack`
16. `Publish Message`
17. `Connect Command`
18. `Connect Ack`
19. `Connect Command`

which when abstracted, for $k = 3$, we obtain the following environment:

$$\psi^3_{\mathcal{A}Attack2} = \{|Connect, Connect, Connect, Connack, Connack, Connack, Publish, Publish, Publish|\}$$

Again here, we can see that $\psi^3_{\mathcal{A}Attack2} \neq \psi_{\mathcal{A}blueprintQoS0}$, and therefore the dataset of [37] (§4.C) reveals abnormal behaviour in the protocol run.

6. Conclusions

We presented in this paper a new method for detecting anomalies in datasets representing systems behaviour using a direct comparison with analysis of their formal specifications. We highlight that this method is different but more robust than fuzzy methods that use machine learning algorithms as it relies directly on formal specification and verification methods in detecting anomalous data. Such methods provide verifiable evidence for the presence of anomalies, unlike the fuzzy methods, which by their nature, can only hint at such anomalies. Furthermore, we applied our method to a case of IoT systems that use the MQTT protocol, and particularly, to a recent study in literature that presents multiple normal and abnormal datasets for this protocol. One of the shortcomings of our approach, at its current level of information that the analysis produces, is that it does not indicate "which" kind of attack does the abnormal state generated through the analysis provide, only that there is something abnormal (i.e., possibly some kind of attack).

We plan in the future to extend this method, with more definitions of the semantic properties underlying the formal language used to obtain more variations of *normality* in datasets, and hence extend the ability to detect more anomalous data. Because of the generality of this method, we also plan to apply this method to other systems and protocols.

Funding: This research received no external funding

Institutional Review Board Statement: Not applicable

Informed Consent Statement: Not applicable

Data Availability Statement: Not applicable

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hajjaji, Y.; Boulila, W.; Farah, I.R.; Romdhani, I.; Hussain, A. Big data and IoT-based applications in smart environments: A systematic review. *Comput. Sci. Rev.* **2021**, *39*, 100318.
2. Vaccari, I.; Chiola, G.; Aiello, M.; Mongelli, M.; Cambiaso, E. MQTTset, a new dataset for machine learning techniques on MQTT. *Sensors* **2020**, *20*, 6578.
3. Balducci, F.; Impedovo, D.; Pirlo, G. Machine learning applications on agricultural datasets for smart farm enhancement. *Machines* **2018**, *6*, 38.
4. Durga, S.; Nag, R.; Daniel, E. Survey on machine learning and deep learning algorithms used in internet of things (IoT) healthcare. In Proceedings of the 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 27–29 March 2019; pp. 1018–1022. .
5. Kumar, A.; Salau, A.O.; Gupta, S.; Paliwal, K. Recent trends in IoT and its requisition with IoT built engineering: a review. *Adv. Signal Process. Commun.* **2019**, *2019*, 15–25.
6. Madni, A.M.; Madni, C.C.; Lucero, S.D. Leveraging digital twin technology in model-based systems engineering. *Systems* **2019**, *7*, 7.
7. Locke, D. *MQ Telemetry Transport (MQTT) V3.1 Protocol Specification*; Technical Report; IBM Corporation: Armonk, NY, USA, 2010.
8. Milner, R.; Parrow, J.; Walker, D. A Calculus of Mobile Processes. *Inf. Comput.* **1992**, *100*, 1–77.
9. Ouchani, S. Ensuring the functional correctness of IoT through formal modeling and verification. In Proceedings of the International Conference on Model and Data Engineering, Chengdu, China, 25–27 August 2018; pp. 401–417.
10. Kwiatkowska, M.; Norman, G.; Parker, D. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In Proceedings of the Computer Aided Verification, Snowbird, UT, USA, 14–20 June 2011; Gopalakrishnan, G.; Qadeer, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 585–591.
11. Marir, S.; Belala, F.; Hameurlain, N. A formal model for interaction specification and analysis in IoT applications. In Proceedings of the International Conference on Model and Data Engineering. Springer, 2018, pp. 371–384.
12. Diwan, M.; D'Souza, M. A framework for modeling and verifying IoT communication protocols. In Proceedings of the International Symposium on Dependable Software Engineering: Theories, Tools, and Applications, Beijing, China, 25–27 November 2017; pp. 266–280.

13. Mohsin, M.; Anwar, Z.; Husari, G.; Al-Shaer, E.; Rahman, M.A. IoTSAT: A formal framework for security analysis of the internet of things (IoT). In Proceedings of the 2016 IEEE Conference on Communications and Network Security (CNS), Philadelphia, PA, USA, 17–19 October 2016; pp. 180–188.
14. Alhanahnah, M.; Stevens, C.; Bagheri, H. Scalable analysis of interaction threats in iot systems. In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual, 18–22 July 2020; pp. 272–285.
15. Conforti, G.; Ghelli, G.; Flesca, S.; Greco, S.; Saccà, D.; Zumpano, E. Spatial tree logics to reason about semistructured data. *Language* **2003**, *17*, 16.
16. Abrial, J.R. *Modeling in Event-B: System and Software Engineering*; Cambridge University Press: Cambridge, MA, USA, 2010.
17. Stanford-Clark, A.; Truong, H.L. Mqtt for sensor networks (mqtt-sn) protocol specification. *Int. Bus. Mach. (IBM) Corp. Version* **2013**, *1*, 1–28.
18. Bormann, C.; Castellani, A.P.; Shelby, Z. Coap: An application protocol for billions of tiny internet nodes. *IEEE Internet Comput.* **2012**, *16*, 62–67.
19. Valle, R.; Donzé, A.; Fremont, D.J.; Akkaya, I.; Seshia, S.A.; Freed, A.; Wessel, D. Specification mining for machine improvisation with formal specifications. *Comput. Entertain.* **2016**, *14*, 1–20.
20. Mechouche, A.; Abadie, N.; Prouteau, E.; Mustière, S. Ontology-Based Formal Specifications for User-Friendly Geospatial Data Discovery. In *Advances in Knowledge Discovery and Management*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 151–176.
21. Mandrioli, C.; Leva, A.; Maggio, M. Dynamic models for the formal verification of big data applications via stochastic model checking. In Proceedings of the 2018 IEEE Conference on Control Technology and Applications (CCTA), Copenhagen, Denmark, 21–24 August 2018; pp. 1466–1471.
22. Bengtsson, J.; Larsen, K.; Larsson, F.; Pettersson, P.; Yi, W. UPPAAL—A tool suite for automatic verification of real-time systems. In Proceedings of the International Hybrid Systems Workshop, Ithaca, NY, USA, October 1996; pp. 232–243.
23. Zaharia, M.; Xin, R.S.; Wendell, P.; Das, T.; Armbrust, M.; Dave, A.; Meng, X.; Rosen, J.; Venkataraman, S.; Franklin, M.J.; et al. Apache spark: A unified engine for big data processing. *Commun. ACM* **2016**, *59*, 56–65.
24. Baresi, L.; Bersani, M.M.; Marconi, F.; Quattrocchi, G.; Rossi, M. Using formal verification to evaluate the execution time of Spark applications. *Form. Asp. Comput.* **2020**, *32*, 33–70.
25. Djeddi, C.; Zarour, N.E.; Charrel, P.J. Formal verification of the extension of iStar to support Big data projects. *Comput. Sci.* **2021**, *22* .
26. Yu, E. Modeling Strategic Relationships for Process Reengineering. *Soc. Model. Requir. Eng.* **2011**, *11*, 66–87.
27. Jensen, O.H.; Milner, R. *Bigraphs and Mobile Processes (Revised)*; Technical Report; University of Cambridge, Computer Laboratory: Cambridge, MA, USA, 2004.
28. Cosentino, F. Formal Verification Meets Stochastic Analysis. Ph.D. Thesis, University of Oxford, Oxford, UK, 2021.
29. Urban, C.; Miné, A. A review of formal methods applied to machine learning. *arXiv* **2021**, arXiv:2104.02466.
30. Krichen, M.; Mihoub, A.; Alzahrani, M.Y.; Adoni, W.Y.H.; Nahhal, T. Are Formal Methods Applicable To Machine Learning And Artificial Intelligence? In Proceedings of the 2022 2nd International Conference of Smart Systems and Emerging Technologies (SMARTTECH), Riyadh, Saudi Arabia, 22–24 May 2022; pp. 48–53.
31. Huang, X.; Ruan, W.; Tang, Q.; Zhao, X. Bridging formal methods and machine learning with global optimisation. In Proceedings of the International Conference on Formal Engineering Methods, Madrid, Spain, 28–30 September 2022; pp. 1–19.
32. Aziz, B.; Hamilton, G. Detecting Man-in-the-Middle Attacks by Precise Timing. In Proceedings of the 2009 Third International Conference on Emerging Security Information, Systems and Technologies, Athens/Vouliagmeni, Greece, 18–23 June 2009; IEEE Computer Society: Washington, DC, USA, 2009; pp. 81–86.
33. Aziz, B. A Static Analysis Framework for Security Properties in Mobile and Cryptographic Systems. Ph.D. Thesis, School of Computing, Dublin City University, Dublin, Ireland, 2003.
34. Aziz, B.; Hamilton, G.; Gray, D. A Static Analysis of Cryptographic Processes: The Denotational Approach. *J. Log. Algebr. Program.* **2005**, *64*, 285–320.
35. Aziz, B.; Hamilton, G. The Modelling and Analysis of PKI-based Systems Using Process Calculi. *Int. J. Found. Comput. Sci.* **2007**, *18*, 593–618.
36. Birman, K.; Joseph, T. Exploiting Virtual Synchrony in Distributed Systems. *SIGOPS Oper. Syst. Rev.* **1987**, *21*, 123–138.
37. Siddharthan, H.; Deepa, T.; Chandhar, P. SENMQTT-SET: An Intelligent Intrusion Detection in IoT-MQTT Networks Using Ensemble Multi Cascade Features. *IEEE Access* **2022**, *10*, 33095–33110.