Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 159 (2019) 1277–1286

Procedia Computer Science

www.elsevier.com/locate/procedia

23rd International Conference on Knowledge-Based and Intelligent Information & Engineering Systems

Model Checking for Data Anomaly Detection

Madalina G. Ciobanu[a], Fausto Fasano[b], Fabio Martinelli[c], Francesco Mercaldo[c,b,*], Antonella Santone[b,*]

[a]*Department of Medicine and Health Sciences "Vincenzo Tiberio", University of Molise, Campobasso, Italy*
[b]*Department of Bioscience and Territory, University of Molise, Pesche (IS), Italy*
[c]*Institute for Informatics and Telematics, National Research Council of Italy (CNR), Pisa, Italy*

Abstract

Data tipically evolve according to specific processes, with the consequent possibility to identify a profile of evolution: the values it may assume, the frequencies at which it changes, the temporal variation in relation to other data, or other constraints that are directly connected to the reference domain. A violation of these conditions could be the signal of different menaces that threat the system, as well as: attempts of a tampering or a cyber attack, a failure in the system operation, a bug in the applications which manage the life cycle of data. To detect such violations is not straightforward as processes could be unknown or hard to extract. In this paper we propose an approach to detect data anomalies. We represent data user behaviours in terms of labelled transition systems and through the model checking techniques we demonstrate the proposed modeling can be exploited to successfully detect data anomalies.

Keywords: anomalies detection, data life cycle, model checking, database, data warehouse, big data

1. Introduction

Data fields within databases usually vary according to specific processes which reflect the steps accomplished by the users. These processes are typically tacit, but they are reflected in the data variations. These variations, for instance, can refer to:

- groups of data fields which vary contextually: for instance, data fields *a*, *b*, and *c* vary every time together;
- temporal intervals in which the data field varies; for instance, data field *a* varies once in a month;
- users which usually have access to the data field and how; for instance, an user writes every two hours on data field *a*.

* Francesco Mercaldo, Antonella Santone
E-mail address: francesco.mercaldo@iit.cnr.it, antonella.santone@unimol.it

1877-0509
10.1016/j.procs.2019.09.297

When a data field variation violates the correspondent process, it can be the case of a "data anomaly"[6]. In fact, data usually evolves according to specific processes. An anomaly, i.e., an illegal or unexpected alteration of data field, could be determined by several likely causes:

1. tampering of the database;
2. a cyber attack to the system of which the database is a part;
3. a failure in the operating system of the infrastructure;
4. a bug in the applications which update the database.

Thus, the detection of a data anomaly could lead to: reveal an improper behaviour of a clerk who has access to data (1); discover an intrusion or a cyber attack (2); verify the quality of software and hardware infrastructure (3,4). The problem is that the data variations could be not made explicit and consequently they cannot be verified, because they are not known by the organization responsible of the database. This happens for several reasons:

- the data variation can be the result of several different processes;
- the data variation could derive from not formalized processes;
- the data variation could be due to incidental or circumstances' factors, as well as: the behaviour of a specific class of users, data are available according to specific constraints, different configurations of the systems may influence the order in which updates are made.

Data fields can be modified by recurrent or by non-deterministic processes. Unlikely recurrent processes, the non-deterministic processes have not the characteristic of occurring always in the same way, since they are the result of stochastic events. An example of a non-deterministic process is the update of a bank account: deposits and withdraws of money can happen in non-deterministic moments, so it is not possible to derive a specific pattern concerning the frequency of variation for the account data field. For their intrinsically unpredictable nature, non-deterministic processes must not be used for deriving data evolution, as this would entail false positives proliferation. Conversely, recurrent processes must be used for defining data life cycle. Recurrent processes may be explicit or latent: explicit processes are known, and could be completely or partially formalized, while latent processes are not known to the organization that owns/uses the database.

A process can remain latent for different reasons:

- it can be the result of actions performed by different applications sharing the same data. Thus, for the responsible of the data source, it is hard to have the comprehensive picture of the correct order in which these applications change a specific data field;
- the applications that modify the data field could belong to third part organizations.

In general, a database administrator may have a partial view of the processes that change the data fields, with missing or incomplete information on:

- which applications modify which data field;
- what is the order of the different application execution when changing the data field;
- what is the frequency of execution of each process that modifies a data field.

In this paper we propose a data-driven approach aimed to extract the data variation directly by observing the database while modifications occur. We focus on recurrent processes that take place at regular time intervals and, each time they are enacted, modify the same sets of data fields. Our aim is to capture these recurring characteristics of data fields variation by mining the variations happening on the database using formal methods. The use of a formal

system model is important to explicitly specify the dependency between the processes and the various stages of data. The data life cycle for each user is modelled through automata. The data anomalies are detected through the model checking technique.

The paper proceeds as follows: next section introduces preliminary background notion about the model checking technique used in the proposed method, Section 3 describes the proposed approach to detect data anomaly, Section 4 presents a case study, current state of the art literature is discusses in Section 5 and, finally, conclusion and future work are drawn in the last section.

2. Background

In this section we provide preliminary notions related to the formal methods technique exploited by the proposed method (i.e., the model checking). To apply model checking we need: (i) Formal Model, (ii) Temporal Logic and (iii) Formal Verification Environment.

Formal Model: Specification is used to represent the system under analysis in a formal way. A language with a mathematically defined syntax and semantics is used. We represent the system behaviour as an automaton consisting of a set of node and a set of labeled edge connecting the nodes. The system state is represented by a node, while the transition of the system by a state to the next one is represented by a labeled edge.

Whether the automaton exhibit an $s \xrightarrow{a} s'$ edge, this means that the system can evolve from the s state into the s' state through the a action (that ranges over sets of $\mathcal{A}$ actions). The initial state of the automaton is the so-called *initial state*. Considering that it is usually convenient to represent automaton algebraically in the form of processes, we exploit the Calculus of Communication Systems of Milner [17] (CCS), a widespread process algebra.

CCS defines operators to build finite processes, to describe parallel composition, choice between actions and scope restriction but also some notion of recursion aimed to catch infinite behaviour.

A CCS process p is formally defined using the structural operational semantics, i.e., a set of conditional rules describing the transition relation of the automaton corresponding to the behavior expression defining p. The considered automaton is called *standard transition system* of p. Further details can be found in [17].

Temporal Logic: in order to define properties we need a precise notation. We use *mu-calculus* logic [28], which syntax is below reported. We suppose that Z ranges over a set of variables, K and R range over sets of actions $\mathcal{A}$.

$$\phi ::= \mathtt{tt} \mid \mathtt{ff} \mid Z \mid \phi \vee \phi \mid \phi \wedge \phi \mid [K]\,\phi \mid \langle K \rangle\,\phi \mid \nu Z.\phi \mid \mu Z.\phi$$

The satisfaction of a formula ϕ by a state s of a transition system, denoted by $s \models \phi$, is so defined:

- each state satisfies `tt` and no state satisfies `ff`;
- a state satisfies $\phi_1 \vee \phi_2$ ($\phi_1 \wedge \phi_2$) if it satisfies ϕ_1 or (and) ϕ_2.
- $[K]\,\phi$ and $\langle K \rangle\,\phi$ are the modal operators:

  $[K]\,\phi$ is satisfied by a state which, for every performance of an action in K, evolves in a state obeying ϕ.

  $\langle K \rangle\,\phi$ is satisfied by a state which can evolve to a state obeying ϕ by performing an action in K.

In Table 1 is reported the precise definition of the satisfaction of a closed formula φ by a state s (denoted $s \models \varphi$). $\mu Z.\phi$ and $\nu Z.\phi$ are the fixed point formulae, where μZ (νZ) *binds* free occurrences of Z in ϕ. An occurrence of Z is free if it is not within the scope of a binder μZ (νZ). A formula is *closed* if it contains no free variables. $\mu Z.\phi$ is the least fix-point of the recursive equation $Z = \phi$, while $\nu Z.\phi$ is the greatest one. A transition system T satisfies a formula ϕ, denoted $T \models \phi$, if and only if $q \models \phi$, where q is the initial state of T. A CCS process p satisfies ϕ if the standard transition system of p satisfies ϕ.

In the paper we use, in the modal operators, the following abbreviations: $-$ for $\mathcal{A}$ and $-K$ for $\mathcal{A} - K$, where $\mathcal{A}$ is the set of all actions and $K \subseteq \mathcal{A}$.

Formal Verification Environment: once defined the model and the temporal logic properties, we need something enabling us to check whether the model satisfies the defined properties. To this aim formal verification is considered,

$$
\begin{aligned}
&p \not\models \texttt{ff}\\
&p \models \texttt{tt}\\
&p \models \varphi \wedge \psi \text{ iff } p \models \varphi \text{ and } p \models \psi\\
&p \models \varphi \vee \psi \text{ iff } p \models \varphi \text{ or } p \models \psi\\
&p \models [K]\,\varphi \text{ iff } \forall p'.\forall \alpha \in K. p \xrightarrow{\alpha} p' \text{ implies } p' \models \varphi\\
&p \models \langle K\rangle\,\varphi \text{ iff } \exists p'.\exists \alpha \in K. p \xrightarrow{\alpha} p' \text{ and } p' \models \varphi\\
&p \models \nu Z.\varphi \;\text{ iff } p \models \nu Z^n.\varphi \text{ for all } n\\
&p \models \mu Z.\varphi \;\text{ iff } p \models \mu Z^n.\varphi \text{ for some } n
\end{aligned}
$$

where:

- for each n, $\nu Z^n.\varphi$ and $\mu Z^n.\varphi$ are defined as:

$$
\begin{aligned}
&\nu Z^0.\varphi = \texttt{tt} \qquad\qquad \mu Z^0.\varphi = \texttt{ff}\\
&\nu Z^{n+1}.\varphi = \varphi[\nu Z^n.\varphi/Z] \quad \mu Z^{n+1}.\varphi = \varphi[\mu Z^n.\varphi/Z]
\end{aligned}
$$

  where the notation $\varphi[\psi/Z]$ indicates the substitution of ψ for every free occurrence of the variable Z in φ.

Table 1: Satisfaction of a closed formula by a state

a system process exploiting mathematical reasoning to verify if a system (i.e., the model) satisfies some requirement (i.e., the temporal logic properties).

In last years several verification techniques were proposed, in this paper model checking [24] is considered.

In the model checking technique the properties are formulated in temporal logic: each property is evaluated against the system (i.e., the automaton-based model). The model checker accepts as input a model and a property, it returns "true" whether the system satisfies the formula and "false" otherwise. The performed check is an exhaustive state space search that is guaranteed to terminate since the model is finite.

As model checker in this paper we consider TAPAs [4], a tool for specifying and analyzing concurrent systems.

3. The Data Anomaly Detection Method

The proposed method is data-driven: it works with homogeneous data (stored in relational databases or data warehouses) but also with heterogeneous ones (i.e., big data).

Our proposal for data anomaly detection comprises two main phases: the *Automata generation* depicted in Figure 1 and the *Model Verification* shown in Figure 2.

The *Automata generation* (Figure 1) builds an automaton for each user. In figure, as an example, a set of four users (*user A*, *user B*, *user C* and *user D*) has been considered. Each user is able to perform *read*, *update* and *write* operation on data. Each user operation is stored through *log*: from the log are gathered a series of information enabling the proposed method to build an automaton for each user. In the considered example (*automaton A*, *automaton B*, *automaton C* and *automaton D*, representing the user behaviours, have been generated. Furthermore a *cluster automaton* is built, with the responsibility to guarantee the synchronization between the other automata.

To build the automata a log with following information is considered:

- *timestamp*: date and time when the transaction took place;
- *user*: user performing the transaction;
- *operation*: type of transaction performed;

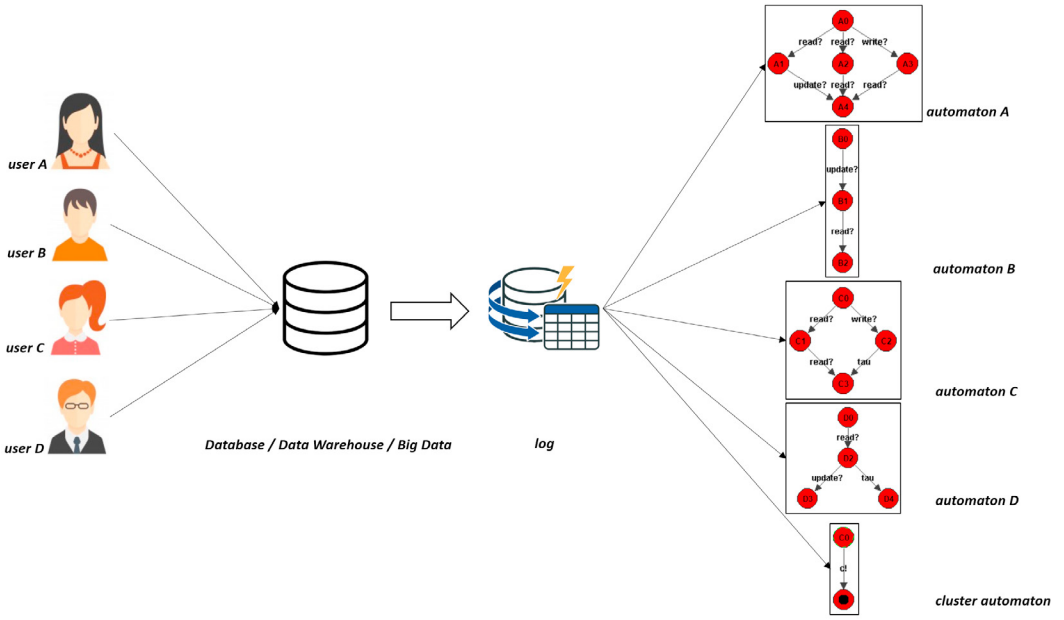

Fig. 1: Automata generation.

Table 2: Temporal logic formula to detect data anomalies.

$$\varphi = \nu X.[action\_A]\ \texttt{ff}\ \wedge\ [-action\_A, action\_B]\ X$$

- *table*: table name;
- *field*: analysed field.

Through the timestamp, a series of time window fixed clusters is defined: data operations falling in each temporal cluster are retrieved from the log. The *cluster automaton* is able to ensure synchronization between the several clusters (and between the relative actions of the user automata).

We consider as data anomaly a variation from the expected behaviour. The legitimate behaviour can be described in terms of logic temporal properties. For this reason, once built the automata (in the *Automata Generation* phase) representing the user behaviour, the *Model Verification* (Figure 2) is focused on the data anomaly detection.

In this phase a CSS model invoking in parallel all the automata is built (i.e., *model* in Figure 2) with a set of restricted action to guarantee the synchronization. Thus, the *model checker* verifies whether the built model satisfies a set of properties expressed in mu-calculus: whether the model checker outputs *true* the model is compliant with respect to the legitimate behaviour, otherwise (i.e., the model checker outputs *false*) the property under analysis is not verified on the model and this is symptomatic that a deviation from the legitimate behaviour is occurred.

4. The Case Study

In this section we present a case study aimed to demonstrate how the proposed method can be successfully exploited to detect anomalies in the data alteration processes.

Table 2 shows the temporal logic formula aimed to detect data anomalies.

The φ property is describing following behavior: "it is not possible to perform the *action_A* whether *action_B* is not previously done". This property will be considered in the case study to detect the data anomalies.

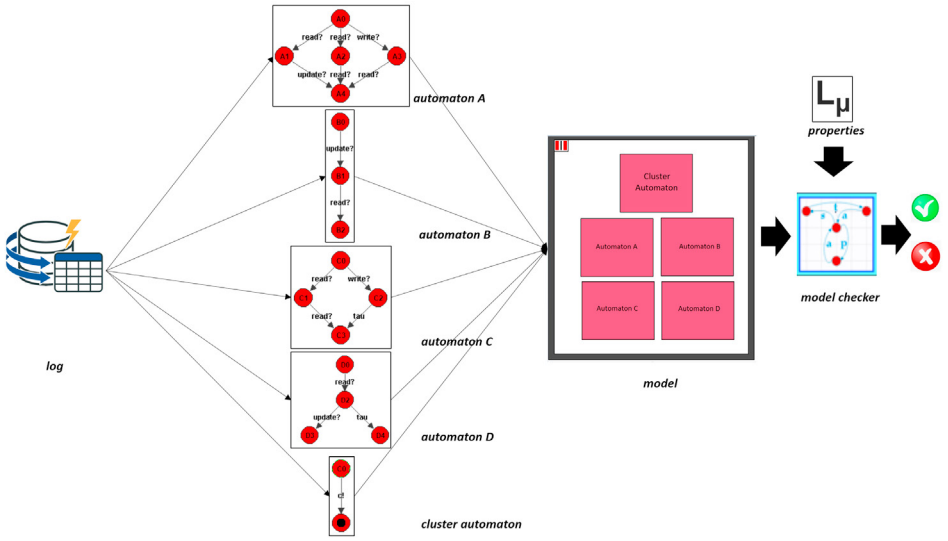

Fig. 2: Model Verification.

Figure 3 shows the model we considered in the case study: the left side of the figure shows the process system, while the right one shows the automata. To guarantee the synchonization between the automata the action we consider as restricted the actions on the *cluster automata*.

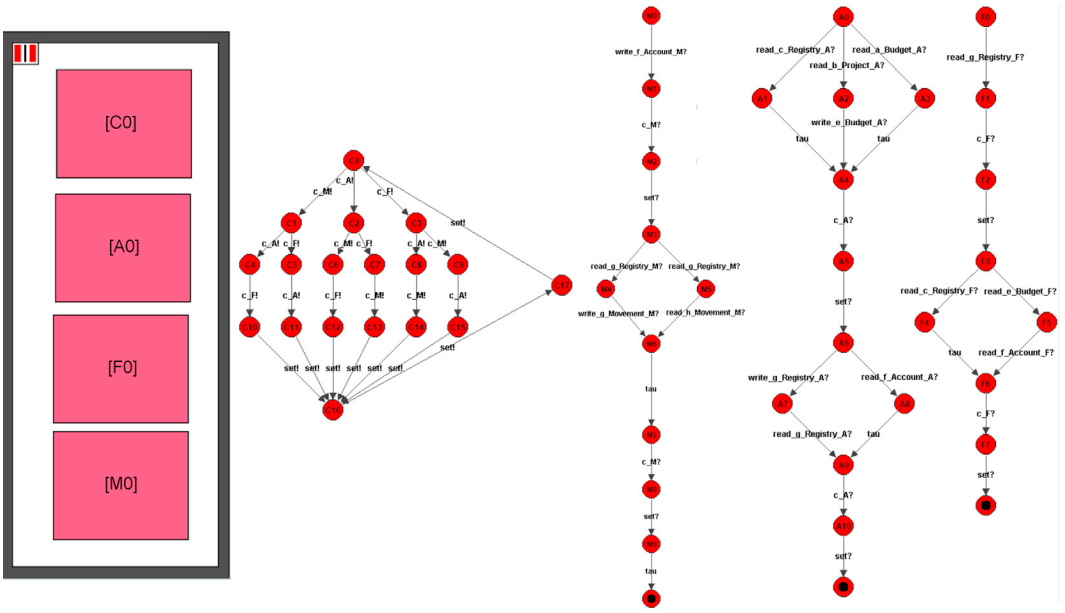

Fig. 3: Process system and automata case study.

As shown in Figure 3 the system is composed by the parallel of $A0$ (i.e., the *Antonella* user), $M0$ (i.e., the *Madalina* user), $F0$ (i.e., the *Francesco* user) and $C0$ (i.e., the *cluster automaton*) processes with the restriction on the c_A, c_M, c_F actions representing the clusters for *Antonella*, *Madalina* and *Francesco* plus an additional *set* restricted action aimed to guarantee that the various automata are properly synchronized. In the case study two temporal clusters and three users have been considered.

Moreover, we generally label the actions with the following syntax: operation_field_Table_User, where *operation* ∈ { *read*, *update*, *write* }, *field* is the considered field belonging to a certain *Table* and *user* represents the user performing the transaction.

From the general formula for data anomaly detection depicted in Table 2, we formulate following properties, as shows in Figure 4.

Anomaly_1	ν X. ([read_g_Registry_A?]false ∧ [-(read_g_Registry_A?, write_f_Account_M?)]X)	Yes	0.0 s
Anomaly_2	ν X. ([read_f_Account_F?]false ∧ [-(read_f_Account_F?, write_f_Account_M?)]X)	Yes	0.0 s
Anomaly_3	ν X. ([read_c_Registry_A]false ∧ [-(read_c_Registry_A?, write_g_Registry_A?)]X)	Yes	0.005 s
Anomaly_4	ν X. ([write_g_Registry_A?]false ∧ [-(write_g_Registry_A?, read_g_Registry_F?)]X)	Yes	0.0 s
Anomaly_5	ν X. ([read_f_Account_F?]false ∧ [-(read_f_Account_F?, write_f_Account_M?)]X)	Yes	0.0 s
Legitimate_1	ν X. ([read_b_Project_A?]false ∧ [-(read_b_Project_A?, write_g_Registry_A?)]X)	No	0.0 s
Legitimate_2	ν X. ([write_e_Budget_A?]false ∧ [-(write_e_Budget_A?, write_g_Registry_A?)]X)	No	0.006 s
Legitimate_3	ν X. ([write_e_Budget_A?]false ∧ [-(write_e_Budget_A?, read_e_Budget_F?)]X)	No	0.003 s
Legitimate_4	ν X. ([read_c_Registry_A?]false ∧ [-(read_h_Movement_M?, read_c_Registry_A?)]X)	No	0.0 s
Legitimate_5	ν X. ([write_e_Budget_A?]false ∧ [-(write_e_Budget_A?, write_g_Movement_M?)]X)	No	0.005 s

Fig. 4: Data anomaly detection temporal logic properties.

The formulae shown in Figure 4 are related to following (anomalous and legitimate) behaviours:

- *anomaly 1*: it is not possible that *Madalina* performs a *write* operation on the *f* field on the "Account" table whether *Antonella* did not *read* the *b* field of the "Registry" table previously: this represents an anomaly, in fact from the model in Figure 3 *Madalina* performs a *write* operation on the *f* field on the "Account" table before the *read* operation of *Antonella* on the *b* field of the "Registry" table for this reason the *model checking* outputs *true*;

- *anomaly 2*: it is not possible that *Madalina* performs a *write* operation on the *f* field on the "Account" table whether *Francesco* did not *read* the *f* field of the same table is not previously done: this represents an anomaly, in fact *Madalina* performs a *write* operation on the *f* field on the "Account" table before the *read* operation of *Francesco* on the *f* field of the same table. For this reason the *model checking* outputs *true*;

- *anomaly 3*: it is not possible that *Antonella* performs a *read* operation on the *c* field on the "Registry" table whether the same user did not perform previously a *write* operation on the *g* field of the "Registry" table. This represent an anomaly, in fact the *read* operation of *Antonella* in the model of Figure 3 occurs before her *write* operation;

- *anomaly 4*: it is not possible that *Antonella* performs a *read* operation on the *c* field on the "Registry" table whether *Francesco* did not perform a *read* operation on the *g* field of the "Registry" table previously. This is an anomaly: in fact *Francesco* perform the *read* operation on this field before the *read* operation of the *Antonella* user;

- *anomaly 5*: it is not possible that *Madalina* performs a *write* operation on the *f* field on the "Account" table whether *Francesco* did not perform a *read* on the *f* field of the "Account" table previously. This is an anomaly, as detected by the *model checker*: in fact the *write* operation of *Madalina* occurs before the *read* operation of *Francesco*;

- *legitimate 1*: it is not possible that *Antonella* performs a *read* operation on the *b* field on the "Project" table whether the same user did not previously performed a *write* operation on the *g* field of the "Registry" table. This represents a legitimate behavior, as shown from the model in Figure 3 and for this reason the model checker outputs *false*;

- *legitimate 2*: it is not possible that *Antonella* performs a *write* operation on the *e* field on the "Budget" table whether the same user did not perform a *write* operation on the *g* field of the "Registry" table previously. This represents a legitimate behavior, as shown from the model in Figure 3 and for this reason the model checker outputs *false*;

- *legitimate 3*: it is not possible that *Antonella* performs a *write* operation on the *e* field on the "Budget" table whether *Francesco* did not previously perform a *read* operation on the *e* field of the "Budget" table. This represents a legitimate behavior, as shown from the model in Figure 3 and for this reason the model checker outputs *false*;

- *legitimate 4*: it is not possible that *Antonella* performs a *read* operation on the *c* field on the "Registry" table whether *Madalina* user did not *read* the *h* field of the "Movement" table previously. This represents a legitimate behavior, as shown from the model in Figure 3 and for this reason the *model checker* outputs *false*;

- *legitimate 5*: it is not possible that the *Antonella* user performs a *write* operation on the *e* field on the "Budget" table whether *Madalina* did not perform a *write* operation on the *g* field of the "Movement" table previously. This represents a legitimate behavior, as shown from the model in Figure 3 and for this reason the model checker outputs *false*.

The anomalous and legitimate properties (in the second column of Figure 4) are verified by the TAPAs model checker on the model in Figure 3 with a *true* output, as shows by the third column in Figure 4 (i.e., *Yes*). Furthermore, the last column in Figure 4 shows the time in second employed by the TAPAs model checker to verify each property, ranging from 0.0 to 0.06 seconds. Experiments were performed on a Microsoft Windows 10 machine equipped with following configuration: 8th Generation Intel Core i7 2.0 GHz CPU, NVIDIA GeForce MX150 graphic card, 16 GB RAM memory and 500 GB Hard Disk.

It is important to observe that the extended number of possible legitimate behaviours (and of the data anomalies) would make the analysis impossible without the support of automatic formal verification environments. Whether there is any possibility of a data anomaly, the model checker will be able to find it. In a parallel system where concurrency must be managed (as the one shown in the model depicted in Figure 3), formal analysis can explore all possible interleaving and event orderings. This level of coverage is not possible to achieve through testing, from the other side formal methods have the property of completeness i.e., they cover all aspects of the system under analysis.

5. Related Work

Anomaly detection has been studied in several contexts, such as intrusion detection [5,8,35], fraud detection [10,21,32],sensor networks [9,19,29,33], and so on.

Chandola *et al.* [6], conducted a survey covering most of the applications and techniques proposed for anomaly detection in the above mentioned fields.

Available techniques can be classified based on the approach they adopt to detect the anomalies. Our approach belongs to the classification based techniques, thus, in this section, we mainly focus on this research area. However, many other approaches are available that detect anomalies using nearest neighbor algorithms [3], clustering [11], and statistical anomalies [2].

In this paper, we focus on anomalies in data fields with a special focus on the database, data warehouse, and big data contexts. In this field, related work is mainly focused on finding anomalies caused by schema and data updated concurrently [26], as well as to the detection of outliers [31,34], noise [30].

In the data warehouse context, built-in operations such as drill-down, roll-up, and selection can be used to explore data cubes and identify anomalies. A different approach has been proposed by Sarawagi *et al.* [25] that use discovery-driven exploration paradigm to mine OLAP (i.e., On-Line Analytical Processing) systems and identify exceptions within data cubes. A systematic semi-automatic approach to monitor Key Performance Indicators has been proposed by Mat *et al.* [15] to enable managers to identify deviations in their strategic plan by analyzing the data warehouse. Son *et al.* [27] propose three simple methods using moving average and 3-sigma techniques to detect anomalies in log data.

Big data represent a recent challenge for anomaly detection techniques, because of the huge amount of data, the data heterogeneity, and the velocity of data production. Rettig *et al.* [23] use Kafka queues [12] and Spark Streaming to detect of anomalies over high velocity streams of events. Machine learning, in particular, has been widely used to detect anomalies in streaming datasets [1] using Hoeffding tree algorithm [18], Convolutional Neural Networks [14], or randomized algorithms such as rPS and gPS [7]. Lighari and Hussain [13] combine rule based and clustering analysis for security analysis [16] of big data-set.

Most of the above mentioned works, aim a detecting outliers at an individual instance level (a.k.a. point anomalies) and looking for values that deviate from the underlying data distribution [20]. However, in our case, we are also interested in detecting contextual anomalies, in which a specific transaction may be considered normal in a given context (e.g., for a particular time period or a given user) as well as collective anomalies, in which a subset of instances occur together as a collection [6]. In this case, multi-dimensional statistical tests have been proposed to detect user behavioural anomalies [22].

A comparison of the proposed approach is a very tricky task due the several factors that influence the effectiveness of each proposal. A first evaluation dimension is the ability to deal with point anomalies, contextual anomalies and/or collective anomalies. Despite many approaches try to solve the problem reducing contexts and collections to instances, and successively applying point anomaly techniques to the resulting data-set, the obtained results are not optimized. Another evaluation dimension concerns the technique computational complexity. Nearest neighbor and clustering based approaches do not scale for complex and large data sets, because they rely on the computation of distance measurement between normal and anomalous instances. On the other hand, rule based approaches are highly dependant on the preliminary construction of the rules. The distribution of anomalies within the data set is another factor that may influence the results of a classification techniques. Sparse and rare anomalies are easily identified by nearest neighbor and clustering algorithms, but dense or clustered anomalies may be identified as normal data especially for unsupervised approaches. In this case, our approach is independent of the data, since rules are constructed appropriately.

The main difference between the cited works and our method is the adoption of the formal methods to detect data anomalies.

6. Conclusion and Future Work

In this paper a method aimed to mine data anomalies is proposed. We represent the user behaviour in terms of automaton and we exploit the model checking technique to detect whether a deviation from the expected behaviour occurred. We plan to exhaustively evaluate the proposed method using a real-world dataset related, for instance, to a financial domain. Furthermore, a research direction could be focused on the semantic meaning of change, i.e. when the change of data is not consistent with its semantic identity. Furthermore, we will consider the adoption of timed automata to better model the data variation time-window.

Acknowledgments

This work has been partially supported by H2020 EU-funded projects SPARTA contract 830892 and C3ISP and EIT-Digital Project HII and PRIN “Governing Adaptive and Unplanned Systems of Systems” and the EU project CyberSure 734815.

References

1. Amen, B., Grigoris, A., 2018. A theoretical study of anomaly detection in big data distributed static and stream analytics, in: 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 1177–1182. doi:10.1109/HPCC/SmartCity/DSS.2018.00198.
2. Badriyah, T., Rahmaniah, L., Syarif, I., 2018. Nearest neighbour and statistics method based for detecting fraud in auto insurance, in: 2018 International Conference on Applied Engineering (ICAE), pp. 1–5. doi:10.1109/INCAE.2018.8579155.
3. Byers, S., Raftery., A.E., 1998. Nearest-neighbor clutter removal for estimating features in spatial point processes. Journal of the American Statistical Association 93, 577–584. URL: http://www.jstor.org/stable/2670109.
4. Calzolai, F., De Nicola, R., Loreti, M., Tiezzi, F., 2008. Tapas: A tool for the analysis of process algebras, in: Transactions on Petri Nets and Other Models of Concurrency I. Springer, pp. 54–70.
5. Casas, P., Soro, F., Vanerio, J., Settanni, G., D’Alconzo, A., 2017. Network security and anomaly detection with big-dama, a big data analytics framework, in: 2017 IEEE 6th International Conference on Cloud Networking (CloudNet), pp. 1–7. doi:10.1109/CloudNet.2017.8071525.
6. Chandola, V., Banerjee, A., Kumar, V., 2009. Anomaly detection: A survey. ACM Comput. Surv. 41, 15:1–15:58. URL: http://doi.acm.org/10.1145/1541880.1541882, doi:10.1145/1541880.1541882.

7. Chen, Z., Yu, X., Ling, Y., Song, B., Quan, W., Hu, X., Yan, E., 2018. Correlated anomaly detection from large streaming data, in: 2018 IEEE International Conference on Big Data (Big Data), pp. 982–992. doi:10.1109/BigData.2018.8622004.
8. Fan, W., Miller, M., Stolfo, S.J., , Chan, P.K., 2001. Using artificial anomalies to detect unknown and known network intrusions, in: Proceedings 2001 IEEE International Conference on Data Mining, pp. 123–130.
9. Hayes, M.A., Capretz, M.A.M., 2014. Contextual anomaly detection in big sensor data, in: 2014 IEEE International Congress on Big Data, pp. 64–71. doi:10.1109/BigData.Congress.2014.19.
10. Hollmén, J., Tresp, V., 1999. Call-based fraud detection in mobile communication networks using a hierarchical regime-switching model, in: Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II, MIT Press, Cambridge, MA, USA. pp. 889–895. URL: http://dl.acm.org/citation.cfm?id=340534.340832.
11. Ji, L., Yang, Y., Yan, L., 2011. An anomaly detection algorithm based on clustering, in: 2011 International Conference on Computer Science and Service System (CSSS), pp. 1059–1062. doi:10.1109/CSSS.2011.5974574.
12. Kreps, J., 2011. Kafka : a distributed messaging system for log processing, in: 6th Workshop on Networking meetsDatabases (NetDB 2011).
13. Lighari, S.N., Hussain, D.M.A., 2017. Hybrid model of rule based and clustering analysis for big data security, in: 2017 First International Conference on Latest trends in Electrical Engineering and Computing Technologies (INTELLECT), pp. 1–5.
14. Lu, S., Wei, X., Li, Y., Wang, L., 2018. Detecting anomaly in big data system logs using convolutional neural network, in: 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech), pp. 151–158. doi:10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00037.
15. Maté, A., Zoumpatianos, K., Palpanas, T., Trujillo, J., Mylopoulos, J., Koci, E., 2014. A systematic approach for dynamic targeted monitoring of kpis, in: CASCON.
16. Mercaldo, F., Nardone, V., Santone, A., Visaggio, C., 2016. Hey malware, i can find you!, pp. 261–262. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84983803047&doi=10.1109%2fWETICE.2016.67&partnerID=40&md5=21b88831ed64d663142b799da7ac03b9, doi:10.1109/WETICE.2016.67. cited By 10.
17. Milner, R., 1989. Communication and concurrency. PHI Series in computer science, Prentice Hall.
18. Muallem, A., Shetty, S., Pan, J., Zhao, J., Biswal, B., 2017. Hoeffding tree algorithms for anomaly detection in streaming datasets: A survey 8, 339–361. doi:10.4236/jis.2017.84022.
19. Onal, A.C., Berat Sezer, O., Ozbayoglu, M., Dogdu, E., 2017. Weather data analysis and sensor fault detection using an extended iot framework with semantics, big data, and machine learning, in: 2017 IEEE International Conference on Big Data (Big Data), pp. 2037–2046. doi:10.1109/BigData.2017.8258150.
20. Park, C.H., 2018. Anomaly pattern detection on data streams, in: 2018 IEEE International Conference on Big Data and Smart Computing (BigComp), pp. 689–692. doi:10.1109/BigComp.2018.00127.
21. Phua, C., Alahakoon, D., Lee, V., 2004. Minority report in fraud detection: Classification of skewed data. SIGKDD Explor. Newsl. 6, 50–59. URL: http://doi.acm.org/10.1145/1007730.1007738, doi:10.1145/1007730.1007738.
22. Prarthana, T.S., Gangadhar, N.D., 2017. User behaviour anomaly detection in multidimensional data, in: 2017 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), pp. 3–10. doi:10.1109/CCEM.2017.19.
23. Rettig, L., Khayati, M., Cudr-Mauroux, P., Pirkowski, M., 2015. Online anomaly detection over big data streams, in: 2015 IEEE International Conference on Big Data (Big Data), pp. 1113–1122. doi:10.1109/BigData.2015.7363865.
24. Santone, A., Vaglini, G., 2012. Abstract reduction in directed model checking ccs processes. Acta Informatica 49, 313–341. doi:10.1007/s00236-012-0161-3. cited By 15.
25. Sarawagi, S., Agrawal, R., Megiddo, N., 1998. Discovery-driven exploration of olap data cubes, in: EDBT.
26. Shu, Z., Li, S., Zuo, Y., Zhou, X., Tang, Y., 2005. Correction strategy for view maintenance anomaly after schema and data updating concurrently, in: Proceedings of the Ninth International Conference on Computer Supported Cooperative Work in Design, 2005., pp. 1046–1051 Vol. 2. doi:10.1109/CSCWD.2005.194333.
27. Son, S., Gil, M.S., Moon, Y.S., 2017. Anomaly detection for big log data using a hadoop ecosystem, in: 2017 IEEE International Conference on Big Data and Smart Computing (BigComp), pp. 377–380. doi:10.1109/BIGCOMP.2017.7881697.
28. Stirling, C., 1989. An introduction to modal and temporal logics for ccs, in: Concurrency: Theory, Language, And Architecture, pp. 2–20.
29. Subramaniam, S., Palpanas, T., Papadopoulos, D., Kalogeraki, V., Gunopulos, D., 2006. Online outlier detection in sensor data using non-parametric models, in: Proceedings of the 32Nd International Conference on Very Large Data Bases, VLDB Endowment. pp. 187–198. URL: http://dl.acm.org/citation.cfm?id=1182635.1164145.
30. Sukhobok, D., Nikolov, N., Roman, D., 2017. Tabular data anomaly patterns, in: 2017 International Conference on Big Data Innovations and Applications (Innovate-Data), pp. 25–34.
31. Sun, P., Chawla, S., Arunasalam, B., 2006. Mining for Outliers in Sequential Databases. pp. 94–105. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611972764.9, doi:10.1137/1.9781611972764.9, arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9781611972764.9.
32. Taniguchi, M., Haft, M., Hollmen, J., Tresp, V., 1998. Fraud detection in communication networks using neural and probabilistic methods, in: Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98 (Cat. No.98CH36181), pp. 1241–1244 vol.2. doi:10.1109/ICASSP.1998.675496.
33. Van Phuong, T., Hung, L.X., Cho, S.J., Lee, Y.K., Lee, S., 2006. An anomaly detection algorithm for detecting attacks in wireless sensor networks, in: Proceedings of the 4th IEEE International Conference on Intelligence and Security Informatics, Springer-Verlag, Berlin, Heidelberg. pp. 735–736. URL: http://dx.doi.org/10.1007/11760146_111, doi:10.1007/11760146_111.
34. Wang, Z., Huang, X., Song, Y., Xiao, J., 2017. An outlier detection algorithm based on the degree of sharpness and its applications on traffic big data preprocessing, in: 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)(, pp. 478–482. doi:10.1109/ICBDA.2017.8078867.
35. Warrender, C., Forrest, S., Pearlmutter, B., 1999. Detecting intrusions using system calls: alternative data models, in: Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No.99CB36344), pp. 133–145. doi:10.1109/SECPRI.1999.766910.