ITC 2/48

Journal of Information Technology and Control
Vol. 48 / No. 2 / 2019
pp. 278-298
DOI 10.5755/j01.itc.48.2.21724

Model Checking Based Approach for Compliance Checking

Received 2018/09/27 | Accepted after revision 2019/05/11

http://dx.doi.org/10.5755/j01.itc.48.2.21724

Model Checking Based Approach for Compliance Checking

Fabio Martinelli

Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche, Pisa, Italy, e-mail: fabio.martinelli@iit.cnr.it

Francesco Mercaldo

Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche, Pisa, Italy Department of Bioscience and Territory, University of Molise, Pesche (IS), Italy, e-mail: francesco.mercaldo@iit.cnr.it, francesco.mercaldo@unimol.it

Vittoria Nardone

Department of Engineering, University of Sannio, Benevento, Italy, e-mail: vnardone@unisannio.it

Albina Orlando

Istituto per le Applicazioni del Calcolo "M. Picone", Consiglio Nazionale delle Ricerche, Napoli, Italy, e-mail: a.orlando@iac.cnr.it

Antonella Santone

Department of Bioscience and Territory, University of Molise, Pesche (IS), Italy, e-mail: antonella.santone@unimol.it

Gigliola Vaglini

Department of Information Engineering, University of Pisa, Pisa, Italy, e-mail: gigliola.vaglini@unipi.it

Corresponding author: vnardone@unisannio.it

Process mining is the set of techniques to retrieve a process model starting from available logging data. The discovered process model has to be analyzed to verify whether it respects the defined properties, i.e., the so-called *compliance checking*. Our aim is to use a model checking based approach to verify compliance. First, we propose an integrated-tool approach using existing tools as ProM (a framework supporting process mining techniques) and CADP (a formal verification environment). More precisely, the execution traces from a software system are extracted. Then, using the "Mine Transition System" plugin in ProM, we obtain a labelled transition system, that can be easily used to verify formal properties through CADP. However, this choice presents the "state explosion" problem, i.e., models discovered through the classical process mining techniques tend to be large and complex. In order to solve this problem, another custom-made approach is shown, which accomplishes a pre-processing on the traces to obtain abstract traces, where abstraction is based on the set of temporal logic formulae specifying the system properties. Then, from the set of abstracted traces, we discover a system described in Lotos, a process algebra specification language; in this way we do not build an operational model for the system, but we produce only a language description from which a model checking environment will automatically obtain the reduced corresponding transition system. Real systems have been used as case studies to evaluate the proposed methodologies.

KEYWORDS: Model discovery; process mining; model checking; compliance checking.

1. Introduction

The general idea of process mining techniques is to discover real processes by extracting knowledge from event logs readily available in information systems. These techniques assume that it is possible to record events. Each event *e* has a set of properties, i.e., resource information (*e* was executed by John), activity (*e* corresponds to a particular procedure), and various data elements. Events are ordered (i.e., no explicit time-stamp is needed) and each event belongs to a particular class (i.e., an activity name). An event refers to a process instance and each process instance is described by a sequence of events referred to as a trace. If we consider activity names, then the trace corresponds to a sequence of such names. An event log is a multi-set of traces, i.e., a collection of traces where some traces may appear multiple times.

Process mining has some disadvantages. One of them is that discovered models tend to be large and complex, especially on flexible scenarios where process execution involves multiple alternatives. In fact, trying to consider every possible process behavior, we can obtain highly complex and incomprehensible models; two typical categories of complex process models are called “lasagna” and “spaghetti” processes because of their intertwined appearance. The reduction of complexity is a major challenge and subject to recent research; abstraction techniques can be usefully employed to obtain simpler processes. Other problems are caused by the (essentially, the high quantity of) additional constraints that have to be imposed on the system to guarantee the needed properties. How such constraints can be included in the model may be very hard to be defined. Compliance checking [25] is an important part of the process mining methodology and it is a relatively novel field of research in that context. Compliance refers to the adherence of the discovered process to internal or external rules and then deals with verification issues. External rules primarily include laws and regulations but can also reflect industry standards or other external requirements. Internal rules include management directives, policies and standards. Moreover, compliance checking is a strong requirement in the context of internal or external audits.

In the light of the above, our aim is to verify compliance through the model checking technique. According to process mining technique, our idea is to build the formal model of system starting from its execution traces. In particular, we have developed two different kinds of approaches: the first one reusing existing tools, named integrated-tool approach, and the second one, called custom-made approach, aims to discover a process using process algebra language. This custom-made approach aims to fix the main limitation of the integrated-tool approach.

More precisely, first we propose the integrated-tool approach using existing tools as ProM[1] (a framework that support a variety of process mining techniques) and CADP [10, 4, 27] (a formal verification environment). In this case, the execution traces from a software system are extracted. Then, using the “Mine Transition System” plugin in ProM, we obtain a labelled transition system, that can be easily used to verify formal properties through CADP, as shown in [19]. However, we demonstrated that this choice presents the well-known “state explosion” problem: the models discovered through classical process mining techniques are large and complex.

In order to solve this problem, another custom-made approach has been proposed, where instead of using ProM, we have defined an algorithm producing ab-

1 http://www.promtools.org/doku.php

stracted model from execution traces. Compliance checking is performed through the model checking of the logic formulae representing the internal or external rules imposed to the system on its model discovered in the form of an abstract Lotos process.

More precisely, we consider that the log refers to the behavior of a distributed system and we can process it to produce distinct traces, each one regarding the behavior of one device in the system (they have the same resource information); moreover sub-traces representing simple loops (we suppose the use of known methods, for example the α^{+}-algorithm [9]) can be individuated. The description of the possible synchronization points, taken from the event properties (i.e., the activity) are used to express the communication among devices. To represent constraints that express particular system requirements, we can use the same formalism and define new traces, not included in the log, but containing existing names connected to activities of the devices. Finally, to address the problem of coping with the high complexity of models obtained by means of automatic process mining, we accomplish a further pre-processing on the traces to obtain abstract traces. Abstraction is seen as an effective approach to represent readable models, showing aggregated activities and hiding irrelevant details.

In our custom-made approach, abstraction is based on the set of temporal logic formulae specifying the system properties. These formulae can be seen also as the declarative representation of the internal and external rules different from the operational ones given by the traces representing the system constraints. From the set of abstracted traces, we discover a system described in Lotos [10], a process algebra specification language; in this way we do not build an operational model for the system, like as a Petri net or a transition system, but we produce only a language description from which the model checking environment will automatically obtain the corresponding transition system. Finally, compliance is established through the model checking of the formulae expressing the compliance rules on the discovered system: any discovered process satisfying the formulae is compliant with the given rules. In this way, we can use existing very efficient model checking environments to establish the compliance of the discovered process without introducing additional concepts or ad hoc model checkers.

A very preliminary work has been presented as poster at ICSOFT 2016 [28].

The remainder of this paper is organized as follows. Section 2 presents a short description of the theoretical background of the work; Section 3 depicts an integrated-tool approach able to construct formal models starting from execution traces of a program. Section 4 improves the integrated-tool approach introducing and describing a custom-made approach able to build reduced formal models, also a simple working example is presented. Moreover, the section illustrates the technique for model discovery and gives the results of the model checking of some properties. Section 5 shows the experimental results achieved during the evaluation of the custom-made approach. Section 6 elicits the limitations of our approach. Section 7 discusses some related work and Section 8 presents the conclusions.

2. Background

Some basic concepts of process algebra specifications and model checking of temporal logic formulae are recalled in this section.

2.1. Basic Lotos

Let us now recall the main concepts of Basic Lotos [10], which is widely used in the specification of concurrent and distributed systems. A Basic Lotos program is defined as:

process *ProcName* := P

where E*nv*

endproc,

where *P* is a process, process *ProcName* := P is a *process declaration* and is a E*nv process environment*, i.e., a set of process declarations. A process is the composition, by means of a set of operators, of a finite set $E = \{i, a, b, \ldots\}$ of atomic *events* (or *actions*). Each occurrence of an action in E represents an event of the system. An occurrence of an event $a \in E - \{i\}$ represents a communication on the gate a. Event *i* does not correspond to a communication and it is called the *unobservable event*. The operational semantics of a process *P* is a labeled transition system, denoted as $S(P)$, i.e., an automaton whose states correspond to processes (the initial state corresponds to P) and whose transitions (arcs) are labeled by events in E. Reader unfamiliar with Lotos process syntax can refer to [11].

2.2. Model Checking Selective Mu-Calculus Formulae

In the model checking framework [6], systems are modelled as transition systems and requirements are expressed as formulae in a temporal logic. Model checkers accept two inputs, a transition system and a temporal formula, and return "true" if the system satisfies the formula and "false" otherwise. We consider formulae expressed in the *selective mu-calculus* temporal logic [2]. The basic characteristic of the selective mu-calculus is that the actions relevant for checking a formula are those ones explicitly mentioned in the modal operators used in the formula itself.

$$\phi ::= tt \mid ff \mid Z \mid \phi \vee \phi \mid \phi \wedge \phi \mid \left[K\right]_R \phi \mid \langle K \rangle_R \phi \mid \nu Z.\phi \mid \mu Z.\phi \tag{1}$$

The syntax of the selective mu-calculus is the following: where K, R are sets of events in E, while Z ranges over a set of variable names; $\mu Z.\phi$ is the least fix-point of the recursive equation $Z=\phi$, while $\nu Z.\phi$ is the greatest one.

The selective modal operators $\langle K\rangle_R\phi$ and $[K]_R\phi$ substitute the standard modal operators $\langle K\rangle\phi$ and $[K]\phi$:

_ $[K]_R\phi$ is satisfied by a state which, for every performance of a sequence of actions not belonging to $R\cup K$, followed by an action in$K$, evolves to a state obeying ϕ.

_ $\langle K\rangle_R\phi$ is satisfied by a state which can evolve to a state obeying ϕ by performing a sequence of actions not belonging to $R\cup K$, followed by an action in K.

A transition system T satisfies a formula ϕ, written $T\models\phi$, if and only if $p\models\phi$, where p is the initial state of T. Moreover, a process P satisfies ϕ if $S(P)$ satisfies ϕ. The precise and formal definition of satisfaction of selective mu-calculus formulae can be found in [2].

The basic characteristic of the selective mu-calculus is that the actions relevant for checking a formula ϕ are those ones explicitly mentioned in the modal operators used in the formula itself. Thus we define the set $O(\phi)$ of *occurring actions* of a formula ϕ as the union of all sets K and R appearing in the modal operators ($[K]_R\psi$, $\langle K\rangle_R\psi$) occurring in ϕ. A ρ - bisimulation can be defined, formally characterizing the notion of "the same behavior with respect to a set ρ of actions":

two transition systems are ρ - **equivalent** *if a ρ - bisimulation relating their initial states exists.*

The definition of ρ -bisimulation is based on the concept of α - ending path: an α - *ending path* is a sequence of transitions, labelled by events not in ρ, and followed by a transition labelled by the event α in ρ. Two states S_1 and S_2 are ρ - bisimilar if and only if for each α -ending path starting from S_1 and ending into S_1', there exists an α - ending path starting from S_2 and ending into a state ρ - bisimilar to S_1', and vice-versa. If a ρ -bisimulation relating the initial states of two transition systems exists, then the two systems are ρ - equivalent. As conclusion, in [2] the following theorem is proved:

Theorem 1. *Two transition systems are ρ - equivalent if and only if they satisfy the same set of formulae with occurring events in ρ.*

The interesting consequence of the theorem is that a formula of the selective mu-calculus with occurring events in a set ρ can be checked on any transition system ρ - equivalent to the standard one, in particular on the system with the lowest number of states.

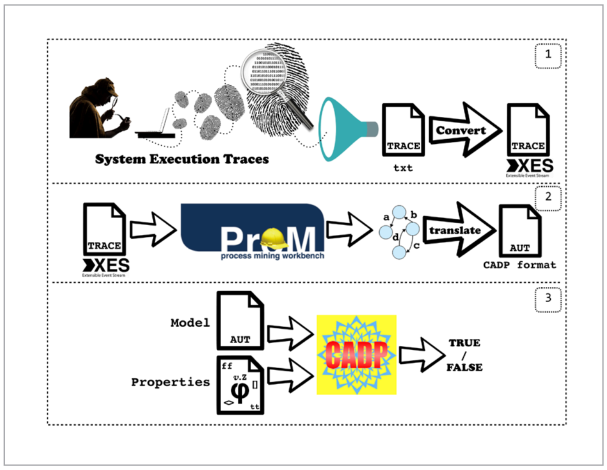
3. Integrated-Tool Approach

In order to link model checking verification closer to real implementation allowing to perform compliance checking an approach integrating existing tools has been proposed in this section.

Figure 1 shows the work-flow of the integrated-tool approach able to model and verify a system starting from its execution traces. It is mainly based on three steps:

_ **First step (see Figure 1 (1)):** It starts from the execution traces of a program obtained from the execution of a software system. Traces are usually stored in text files and they contain both static and dynamic information retrieved during software execution. Static information regards, for instance, class structure in terms of methods and fields. Dynamic information refers to method calls, field access in read or write mode and synchronization

Figure 1
The work-flow of the integrated approach



on objects. Starting from this textual format traces an eXtensible Event Stream format (XES) is generated. XES is an IEEE XML-based standard for event logs. During this conversion process the traces are filtered removing all the unnecessary information.

- **Second step (see Figure 1 (2)):** It creates the model from the XES traces using Process Mining Workbench (ProM)[2]. ProM is an extensible framework that supports a wide variety of process mining techniques in the form of plug-ins. It is an independent platform as it is implemented in Java, and can be downloaded free of charge. From the XES Event Log, using the "Mine Transition System" plugin in ProM developed by H.M.W. Verbeek, a labelled transition system is obtained. The transitions correspond to the events in the log, whereas a state corresponds to a situation in between two events.

- **Third step (see Figure 1 (3)):** It applies the model checking technique. Once the formal model has been retrieved, it is easily used to verify properties using a model checker tool. This step checks the sets of logic properties against the formal model obtained starting from the feature set, as described above. In our approach, the Construction and Analysis of Distributed Processes (CADP) tool [10] is invoked as formal verification environment. In order to apply CADP, the transition system obtained is converted into the input format of CADP, parsing the automaton ProM file. Moreover, the property, written in selective mu-calculus, can be equivalently transformed in the syntax of the logic used by the CADP environment.

3.1. Result Using the Integrated-Tool Approach

In order to evaluate the integrated-tool approach, an example of a real system obtained from the ProM website[3] has been considered. It describes a realistic transaction process within a banking context. In the integrated-tool approach evaluation, the first step has been skipped because the considered real case study has already developed and made available from the repository of the ProM database. The analysed process contains all sort of monetary checks, authority notifications, and logging mechanisms responding to the new degree of responsibility and accountability that current economic environments demand. As stated in [22], "the banking regulation states that serial numbers must be compared with an external database governed by a recognized international authority ("Check Authority Serial Numbers CASN"). In addition, the bank of the case study decided to incorporate two complementary checks to its policy: an internal bank check ("Check Bank Serial Numbers CBSN"), and a check among the databases of the bank consortium this bank belongs to ("Check Inter-Bank Serial Numbers CIBSN"). At a given point, due to technical reasons (i.e., peak hour network congestion, malfunction of the software, deliberated blocking attack, etc.), the external check CASN is no longer performed, contradicting the modeled process, i.e., all the running instances of the process involving cash payment can proceed without the required check".

According with our preliminary approach, we formulate the above anomaly in mu-calculus logic formulae using the following pattern:

The formula φ means that for each action a not preceded by b and c and for each action b not preceded by c, eventually the action c will be performed.

2 http://www.promtools.org/

3 http://data.4tu.nl/repository/uuid:c1d1fdbb-72df-470d-9315-d6f97e1d7c7c

The formula φ expresses the above anomalous situation that the external check CASN is no longer performed.

$$
\begin{aligned}
&\varphi_1 = [CASN]_{\{CIBSN,CBSN\}}[CIBSN]_{\{CBSN\}}\psi_1 \\
&\psi_1 = (\mu X.<->_A tt \wedge [CBSN]_A X) \\
&\varphi_2 = [CASN]_{\{CIBSN,CBSN\}}[CBSN]_{\{CIBSN\}}\psi_2 \\
&\psi_2 = (\mu X.<->_A tt \wedge [CIBSN]_A X) \\
&\varphi_3 = [CIBSN]_{\{CASN,CBSN\}}[CASN]_{\{CBSN\}}\psi_3 \\
&\psi_3 = (\mu X.<->_A tt \wedge [CBSN]_A X) \\
&\varphi_4 = [CIBSN]_{\{CASN,CBSN\}}[CBSN]_{\{CASN\}}\psi_4 \\
&\psi_4 = (\mu X.<->_A tt \wedge [CASN]_A X) \\
&\varphi_5 = [CBSN]_{\{CASN,CIBSN\}}[CASN]_{\{CIBSN\}}\psi_5 \\
&\psi_5 = (\mu X.<->_A tt \wedge [CIBSN]_A X) \\
&\varphi_6 = [CBSN]_{\{CASN,CIBSN\}}[CIBSN]_{\{CASN\}}\psi_6 \\
&\psi_6 = (\mu X.<->_A tt \wedge [CASN]_A X) \\
&\varphi = \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \wedge \varphi_5 \wedge \varphi_6 .
\end{aligned} \tag{2}
$$

The model checker returns "false" when evaluating φ, stating that the anomalous situation is immediately detected, identifying the anomalous subprocess (process cash payment), and eventually taking the necessary countermeasures. The advantage is that it is better to discover the error as soon as possible. It is worth noting that when a property does not hold, the model checking algorithm generates a counter-example, i.e., an execution trace leading to a state in which the property is violated. This ability to generate counter-examples, which can be exploited to pinpoint the cause of an error, is the main advantage of model checking, as compared to other well-known techniques for software verification, as abstract interpretation-based static analysis.

In the used dataset there are six different scenarios: (i) 2000-all-noise; (ii) 2000-all-nonoise; (iii) 2000-scen1; (iv) 2000-scen2; (v) 10000-all-noise; and (vi) 10000-all-nonoise.

The first item of the string is the number of traces in the XES event stream file. "noise" (resp. "nonoise") specifies if the considered traces are (resp. are not) affected by the noise. Furthermore, there are two files used in [22] which present two possible scenarios: *Serial Number Check* and *Receiver Preliminary Profiling*, i.e., "scen1" and "scen2", respectively. The results of the verification of φ formula are: "True" in 2000-all-nonoise, 2000-scen2 and 1000-all-nonoise, "False" in the other cases.

Table 1 shows the results achieved by φ formula. As pointed out from the results, φ is false in some scenarios stating that anomalous traces occur. In particular, anomalous situations are detected in the presence of noise which could be due for different reasons, i.e., deliberate blocking attack, peak hour network congestion or malfunction of the software.

The sizes of models used in the experimental evaluation are shown in Table 2. The size of a model is expressed in terms of states and transitions. As shown in Table 2 the number of states and transitions grows dramatically according to the growing of number of traces. As a simple example the reader can refer the last two rows of Table 2. This means that our preliminary approach suffers of the well-known states explosion problem. To fix this weakness we propose another solution able to directly build a reduced model starting from the execution traces. This model is presented in Section 4.

Table 1

φ Property results

Traces / Formulae	2000-all-noise	2000-all-nonoise	2000-scen1	2000-scen2	10000-all-noise	10000-all-nonoise
φ	**FALSE**	**TRUE**	**FALSE**	**TRUE**	**FALSE**	**TRUE**

Table 2

φ Model and size

Model \ Size	States	Transition
2000-all-noise	**94803**	**96801**
2000-all-nonoise	**89810**	**91808**
2000-scen1	**88863**	**90861**
2000-scen2	**81792**	**83790**
10000-all-noise	**480361**	**490359**
10000-all-nonoise	**434073**	**444071**

In order to better analyze the results obtained by the φ formula, we defined additional eight formulae able to check every single trace belonging to a specific scenario. In particular, these formulae investigate the cause of φ failure. The specified properties are expressed by the following selective mu-calculus:

$$\begin{aligned}
\varphi_1 &= \langle CASN \rangle_{\varnothing} tt \wedge \langle CIBSN \rangle_{\varnothing} tt \wedge \langle CBSN \rangle_{\varnothing} tt \\
\varphi_2 &= [CASN]_{\varnothing} ff \wedge \langle CIBSN \rangle_{\varnothing} ff \wedge \langle CBSN \rangle_{\varnothing} ff \\
\varphi_3 &= \langle CIBSN \rangle_{\varnothing} tt \wedge \langle CBSN \rangle_{\varnothing} tt \wedge [CASN]_{\varnothing} ff \\
\varphi_4 &= \langle CASN \rangle_{\varnothing} tt \wedge \langle CBSN \rangle_{\varnothing} tt \wedge [CIBSN]_{\varnothing} ff \\
\varphi_5 &= \langle CASN \rangle_{\varnothing} tt \wedge \langle CIBSN \rangle_{\varnothing} tt \wedge [CBSN]_{\varnothing} ff \\
\varphi_6 &= \langle CASN \rangle_{\varnothing} tt \wedge [CIBSN]_{\varnothing} ff \wedge [CBSN]_{\varnothing} ff \\
\varphi_7 &= \langle CIBSN \rangle_{\varnothing} tt \wedge [CASN]_{\varnothing} ff \wedge [CBSN]_{\varnothing} ff \\
\varphi_8 &= \langle CBSN \rangle_{\varnothing} tt \wedge [CASN]_{\varnothing} ff \wedge [CIBSN]_{\varnothing} ff
\end{aligned} \quad (3)$$

Roughly speaking, the formulae have the following meaning:

- φ_1 checks if all the three actions (CASN, CIBSN and CBSN) are performed.
- φ_2 checks if all CASN, CIBSN and CBSN are not performed.
- φ_3 checks if CIBSN and CBSN actions are performed and the CASN action is not performed.
- φ_4 checks if CASN and CBSN actions are performed and the CIBSN action is not performed.
- φ_5 checks if CIBSN and CASN actions are performed and the CBSN action is not performed.
- φ_6 checks if CIBSN and CBSN actions are not performed and the CASN action is performed.
- φ_7 checks if CBSN and CASN actions are not performed and the CIBSN action is performed.
- φ_8 checks if CIBSN and CASN actions are not performed and the CBSN action is performed.

Table 3 shows the results obtained during the verification of the formulae specified above. In particular, Table 3 is organized as follow: the above specified formulae are described in the rows, while the scenarios in the columns. Each single model represents a single realistic banking transaction trace. The first four sets have 2000 different traces, so 2000 formal models. The last two have 10000 traces corresponding to 10000 different formal models. The last row is the total number of the analyzed traces resulting true to the formulae. This value is obtained by adding to each other the values in the corresponding column. The table shows the num-

Table 3

Detailed properties

Formulae	Traces					
	all-noise	all-nonoise	scen1	scen2	all-noise	all-nonoise
φ_1	531	708	327	701	2478	3326
φ_2	1293	1292	1348	1299	6678	6674
φ_3	67	0	325	0	249	0
φ_4	50	0	0	0	259	0
φ_5	49	0	0	0	260	0
φ_6	5	0	0	0	28	0
φ_7	3	0	0	0	18	0
φ_8	2	0	0	0	30	0
# of Traces	2000	2000	2000	2000	10000	10000

ber of true achieved by every type of analyzed model. As the results shown and according to the "True" values of the φ formula, the files with no noise and the files of second scenario have all the traces of transactions correct, i.e., whenever a client executed a payment in cash, the three required actions have been performed. This result is highlighted by positive values of φ_1 and φ_2 and the values equal to zero achieved by the other formulae. In the "False" cases the anomalous situations are caused by several reasons. In the "scen1" scenario bad and unsafe transactions occur because only the action CASN has not been performed. Finally, in the scenarios affected by noise the causes of failure occur because one or two required actions are not performed during a payment in cash.

4. Custom-Made Approach

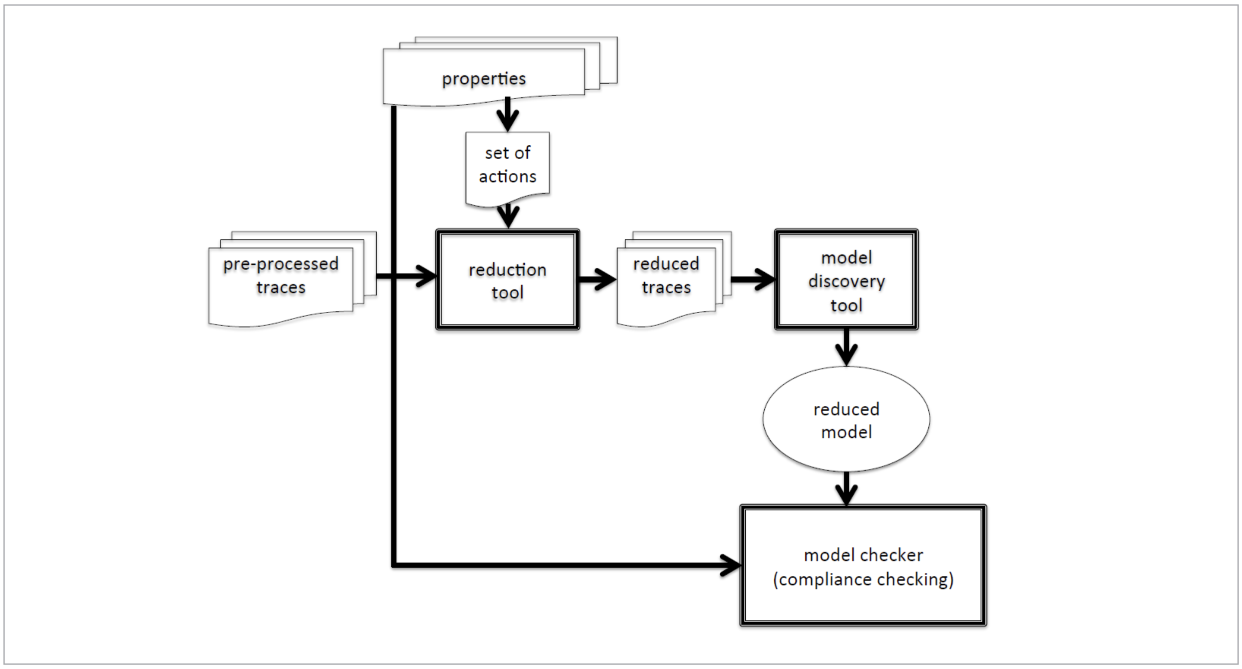
As stated in Section 3 the integrated-tool approach attempt suffers of the state explosion problem. In order to address this limitation we have developed another approach able to fix the state explosion problem.

The basic steps of the new methodology that we are going to propose are summarized in Figure 2. In this section, we assume that the log is pre-processed so that the traces are rearranged to obtain separate traces for each device in a distributed system exploiting the source of each activity. Further, we give each activity a different name; moreover simple loops are solved using α^+-algorithm; after this pre-processing, we consider that it is now possible to describe the traces derived from the log by means of the simple language defined in the next subsection. Successively, names corresponding to activities performing equal communication among devices are given a new equal name and, finally, system constraints may be included in the specification by means of new traces. Property-driven reductions can be performed on the resulting traces to obtain an abstract model of the system in the process algebra Lotos. This model will be model checked against the required properties to verify its compliance.

4.1. Trace-Based System Specification

A specification as we will use in the following can be derived from the log of a physical system or from the instrumentation of a software system. The language we assume be used to define the traces obtained through the log pre-processing is the following.

Figure 2
The methodology



Definition 1 (Trace). Let $E = \{e_1, e_2, ...\}$ be a set of activity names, a trace of such names can be built up by the following syntax:

$$t ::= e \mid t.t \mid \langle t \rangle^* \mid \lambda \quad (4)$$

where $e \in E$ and λ is the empty sequence.

The operator "." represents trace concatenation: usually it is omitted. The operator "" represents the iteration of a trace and it turns out that $\langle\langle t\rangle^*\rangle^*$ is equivalent to $\langle t\rangle^*$. Moreover, $\langle\lambda\rangle^*$ is equivalent to λ. The following definitions are of interest.

Definition 2 (Alphabet, Branching names). Let T be a set of traces:

_ αT is the alphabet of T, and

_ $Be(T)$ is the set of pairs defined as follows:

$$Be(T)=\{(e,e')|t_1=s.e.t \in T, t_2=s.e'.t' \in T, e \neq e' \; and \; s,t,t' \in E'\}$$

For example, given $T = \{a.b.c.d.e, a.b.g.h\}$,

_ $\alpha T = \{a, b, c, d, e, g, h\}$;

_ $Be(T) = \{(c, g)\}$.

After having obtained from the log the set T of traces of activity names, our aim is to obtain from T the model of the distributed system as Lotos processes composed in parallel. The first step of our method is:

1 **Individuation of the traces of each component of the distributed system in isolation (the layout)**

We give the following definition.

Definition 3 (Layout Specification). Given a set of traces of activity names derived from a log, the Layout Specification of a distributed system is $LS = \{T_1, ..., T_n\}$, where each T_i has a distinct alphabet αT_i, and each activity in T_i has the same source, different from that of each other T_j.

The second step consists in the representation in the traces of communications performed among devices; the activity definitions in the log allows the individuation of corresponding communication activities. This step is called:

2 **Specification of the synchronization between components (the flow)**

Definition 4 (Flow Specification. Renaming function $\rightsquigarrow$). Consider the Layout Specification LS and the set $C = \{c_1, ..., c_m\}$, where $c_j = \{c_{j_1}, ..., c_{j_k}\}$, $1 \leq j \leq m$, where the names in the same tuple individuate corresponding communication activities in the log.

The renaming function S is such that $\forall j, s \in [1..m]$, $S(c_j) = e_j$, and $e_j \notin \alpha LS$, moreover, for $j \neq s$ it is $S(c_j) \neq S(c_s)$. The Flow Specification is $FS = LS \rightsquigarrow S(C)$, which is the result of the renaming. From now on we will use $S(C)$ for $\{S(c_1), ..., S(c_m)\}$.

S renames all elements of C using the same new name for all elements of the same tuple of C; obviously the new name given to each tuple is different from those chosen for the other tuples and from all other names in the traces. Table 5 shows an example of renaming function.

The third (possibly not present) step consists in the definition of specific requirements for the system. They are expressed as traces that are built using the alphabet of the Flow specification. This step is:

3 **Construction of the traces that model constraints (the control)**

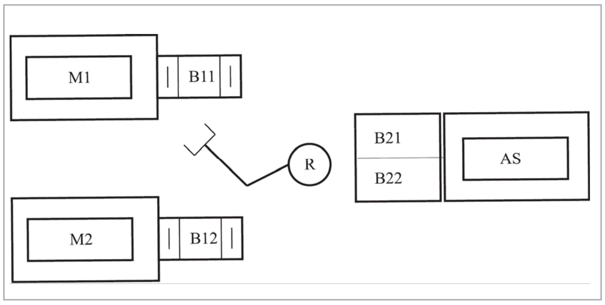
Definition 5 (Control specification). Consider the Flow specification FS, $CS = \{t_1, ..., t_c\}$ is a unique set of finite traces on αFS (also called *control traces*) with a unique activity source different from any other in FS. CS is the Control Specification of the system. CS is a set of traces not retrieved from the log; note that, all events in the Control Specification result in communication events. A wide class of constraints can be expressed by means of such kind of traces, also binding the behavior of several system components. Obviously a Control Specification expresses system requirements that are due, but that the system does not necessarily respect; actually, imposing such constraints can cause deadlocks in the system. The trace-based System Specification (SS for short) is defined as follows.

Definition 6 (System Specification). A System Specification SS can be either a Flow Specification only, $SS = (LS \rightsquigarrow S(C))$, or a Flow Specification plus a Control Specification in the same language, $SS = (LS \rightsquigarrow S(C)) \cup CS$.

4.2. The Working Example

The example we use to describe the approach is a simple automated manufacturing system (called *MS* in the following) depicted in Figure 3. The layout consists of two machines M1 and M2, one robot R, and an assembly station AS. The machines and the assembly station are cells provided with buffering areas of limited storage capacity. Obviously, this very simple example permits a log containing *perfect information*.

Figure 3
Plant of the system



The machine M1 (M2) performs an operation op1 (op2) on a raw part of type t1 (t2). After the operation op1 (op2) is performed, the part is available in the output buffering area *B*11 (*B*12) and is picked up by the robot R to be moved into the input buffering area *B*21 (*B*22) of the assembly station. The finished product must be assembled from two parts of type t1 and one part of type t2; the assembly station must wait for the robot having moved a second part of type t2 in its input buffering area after it is set free from the station itself. To guarantee this result, the control may specify that the non-deterministic behavior of the robot (on choice between a part of type t1 and t2, when both are available) must be restricted to always move two parts of type t1 for each part of type t2. This control specification supplies a bit of further information with respect to the abstract requirement on the correct assembling of the finished products: it also constrains the free behavior of the system to perform a subset of the acceptable computations.

The traces of the components of the working example are very simple and self-explaining and we just report them in Table 4 (machines and buffers are indexed).

We have considered buffers of one position only, but the specification can be easily extended.

Table 5 shows the renaming function *S* which defines the flow of the parts; the events in *αLS* that are not present in the table are assumed unchanged and represent the internal behavior of each component. The derived Flow Specification, FS, is shown in Table 6.

The constraints to be imposed over the system can be expressed by the Control Specification, *CS*, in Table 6. The trace *C*1 requires that at least one occurrence of the event *part*2*_load* happens after two occurrences of the event *part*1*_mov*, whereas *C*2 requires that at

Table 4
Layout Specification, LS

Machines *Mi*, **with** $i \in [1,2]$
$M_i = \{\langle Mi\_start.Mi\_op.Mi\_end\rangle^*\}$
Buffering areas *Bij*, **with** $i, j \in [1, 2]$
$Bij = \{\langle Bij\_in.Bij\_out\rangle^*\}$
Robot *R*
$R = \{\langle R\_init.R\_start1.R\_op.R\_end1\rangle^*,$ $\langle R\_init.R\_start2.R\_op2.R\_end2\rangle^*\}$
Assembly Station *AS*
$AS = \{\langle AS\_start.AS\_load1.AS\_load2.AS\_op.AS\_end\rangle^*,$ $\langle AS\_start.AS\_load1.AS\_load2.AS\_op.AS\_end\rangle\}$

Table 5
Renaming function, *S*

$\forall i \in [1..n]$:
$S(Mi\_end, B1i\_in) = parti\_aval$
$(B1i\_out, R\_starti) = parti\_mov$
$S(R\_endi, B2i\_in) = parti\_load$
$S(B2i\_out, AS\_loadi) = parti\_work$

Table 6
Trace-based System Specification, *SS*

Flow Specification (FS)	
Machines *Mi*, **with** $i \in [1,2]$	
$Mi = \{\langle Mi\_start.Mi\_op.parti\_aval\rangle^*\}$	
Machines' buffering areas *B*1*i*, **with** $i \in [1,2]$	
$B1i = \{\langle parti\_aval.parti\_mov\rangle^*\}$	
Assembly station's buffering area *B*2*i*, **with** $i \in [1,2]$	
$B1i = \{\langle parti\_load.parti\_work\rangle^*\}$	
Robot *R*	
R	$\{\langle R\_init.part1\_mov.R\_op1.part1\_\text{load}\rangle^*,$
	$\langle R\_init.part2\_mov.R\_op2.part2\_load\rangle^*\}$
Assembly Station *AS*	
$AS =$	$\{\langle AS\_start.part1\_work.part2\_work.AS\_op.AS\_end\rangle^*,$
	$\langle AS\_start.part2\_work.part1\_work.AS\_op.AS\_end\rangle^*\}$
Control Specification (CS)	
C=	$\{\langle part1\_mov.part1\_mov.part2\_load\rangle^*,$
	$\langle part2\_mov.part1\_mov.part1\_load\rangle^*\}$

least two occurrences of the event *part*1*_load* happen after one occurrence of the event *part*2*_mov*. The property expressed by the formula ϕ in the following section represents the logic version of the behavior imposed on the system by the control traces; it is worth noting that the specification of the assembly station does not appear to be consistent with this requirement; in fact, the event *AS_load*1 (representing the input of the part of type t1) does not appear two times in the traces of AS.

4.3. Formula-Based Reduction of the System Specification

The approach we present is based on the reduction of the system specification: the function below removes events from a generic trace.

Definition 7 ($del_I(t)$). Given a trace t on the alphabet E and the set $I \subseteq$ E, we define the function del_I: $E^* \to E^*$ as follows:

$$del_I(t) = \begin{cases} e & if\ \ t = e\ and\ e \in I \\ \lambda & if\ \ t = e\ and\ e \not\in I \\ del_I(t').del_I(t'') & if\ \ t = t'.t'' \\ < del_I(t') >^* & if\ \ t = < t' >^* \\ \lambda & if\ \ t = \lambda \end{cases} .$$

The function del_I can be extended to any set of traces T as follows:

$$del_I(T) = \{del_I(t) \mid t \in T. \qquad (5)$$

According to the syntax of a trace given in Definition 4.1, if the trace t is of the form $''e''$ two cases may occur: 1) e represents an interesting activity: we cannot delete e 2) e does not represent an interesting activity: we delete e and the empty trace λ is returned; If the trace t is of the form $t'.t''$, we apply the delete function on t' and on t'', while if the trace t is of the form $\langle t' \rangle^*$, we apply the delete function on t', keeping the recursion. The function terminates when the empty trace is encountered. After having applied del_I on the sets of traces belonging to a system specification, we expect that the new set of traces describes a behavior equivalent to the old one with respect to I. Consequently, there are several problems to be taken into account: when a synchronization event does not belong to I, its elimination can avoid the possible deadlock of the system. Also the elimination of one branching name could avoid the feasibility of alternative behaviours of the system. Then, the problem is: what is a suitable set I that can be used to reduce a system specification without altering the behavior of the system? We use as a guide to build I the property φ to be verified, since its satisfaction must be preserved by the reduction.

Definition 8 ($I(SS, \varphi)$). Consider the system specification $SS = (LS \rightsquigarrow S(C)) \cup CS$ and the selective mu-calculus formula φ, the set

$$I(SS, \varphi) = \mathcal{O}(\varphi) \cup \mathcal{B}e\big(LS \rightsquigarrow S(C)\big) \cup S(C)$$

is the set of names of activities that cannot be cancelled from SS.

Consider again the specification of Table 4, interesting properties to prove are: "the assembly station cannot produce the final result after obtaining only one piece from M1 and only one piece from $M2$":

$$\begin{aligned}\emptyset = {} & [M1\_op]_{\{M2\_op\}}[M2\_op]_{\{M1\_op\}} \\ & [AS\_op]_{\{M1\_op,M2\_op\}}ff \wedge \\ & [M2\_op]_{\{M1\_op\}}[M1\_op]_{\{M2\_op\}} \\ & [AS\_op]_{\{M1\_op,M2\_op\}}ff \end{aligned} \qquad (6)$$

"it is possible that the assembly station provides the final product"

$$\psi = \langle AS\_op \rangle_{\emptyset} tt \qquad (7)$$

"the system is deadlock-free"

$$\chi = \nu Z. \langle \mathrm{E} \rangle_{\mathrm{E}} tt \wedge [\mathrm{E}]_{\mathrm{E}} Z. \qquad (8)$$

In the following we shall consider only the property ϕ, then:

$$I(SS, \phi) = \mathrm{I} = \mathcal{O}(\phi) \cup \{parti\_mov, parti\_load, \\ parti\_work, i \in [1{,}2]\},$$

where

$$\mathcal{O}(\phi) = \{M1\_op, M2\_op, AS\_op\};$$

$del_I(SS)$ obtains the reduced traces shown in Table 7.

Since the reduction is formula-based, we will prove, in the following section, that the complete and reduced systems satisfy the same set of formulae.

Table 7
Reduced traces for checking the formula ϕ, SS_ϕ

Machines Mi, **with** $i \in [1,2]$	
$R\_Mi =$	$\{\langle Mi\_op.parti\_avl\rangle^*\}$
REDUCTION: 1 event	
Machines' buffering areas $B1i$, **with** $i \in [1,2]$	
$R\_B1i =$	$\{\langle parti\_avl.parti\_mov\rangle^*\}$
REDUCTION: 0 event	
Assembly station's buffering area $B2i$, **with** $i \in [1,2]$	
$R\_B2i =$	$\{\langle parti\_load.parti\_work\rangle^*\}$
REDUCTION: 0 event	
Robot R	
$R\_R=$	$(\{\langle part1\_mov.part1\_load\rangle^*, \langle part2\_mov.part2\_load\rangle^*\})$
REDUCTION: 4 events (2 events for each trace)	
Assembly Station AS	
$R\_AS =$	$(\{\langle part1\_work.part2\_work.AS\_op^*, \langle part2\_work.part1\_work.AS\_op^*\})$
REDUCTION: 4 events (2 events for each trace)	
Control Traces	
$C =$	$(\{\langle part1\_mov.part1\_mov.part2\_load^*, \langle part2\_mov.part1\_load.part1\_load\rangle^*\})$
REDUCTION: 0 event	

It is worth noting that ψ can be checked on the same reduced system as ϕ, but a better reduction could be made; on the contrary, χ can be more efficiently checked on the system reduced on the basis of ϕ, for example, since deadlock-freeness is preserved by the reduction while $I(SS,\chi)= E$.

4.4. Model Discovery

Now we define a general syntactic transformation function T which transforms a trace-based system specification into a Lotos program by means of the auxiliary functions defined below. The model described by the Lotos program is simpler and more compact than one directly given as a transition system; moreover, all model checking environments can easily obtain the transition system corresponding to the Lotos program.

Let t be a trace and T a set of traces. $First(t)$ and $Rest(t)$ are inductively defined as follows::

$$
\begin{aligned}
First(\lambda) &= \lambda\\
First(e) &= e\\
First(t_1.t_2) &= First(t_1)\\
First(\langle t\rangle^*) &= First(t)
\end{aligned}
$$

Moreover, it holds that $First(T) = \{First(t) | t \in T\}$.

$$
\begin{aligned}
Rest(\lambda) &= \lambda\\
Rest(e) &= \lambda\\
Rest(t_1.t_2) &= Rest(t_1).t_2\\
Rest(\langle t\rangle^*) &= Rest(t).\langle t\rangle^*
\end{aligned}
$$

Moreover, it holds that $Rest(T) = \{Rest(t) | t \in T\}$.

$$Cont(T) = \{t_2 \,|\langle t_1\rangle^*.t_2 \in T\}$$

$First(T)$ returns the set of all the first names of the traces in T, while $Rest(T)$ defines how a trace may go on after its first activity has been performed; $Cont(T)$ describes what happens when a loop is skipped. For example, the trace $\langle e.t\rangle^*.t'$ describes a behavior that becomes $t.\langle e.t\rangle^*.t'$ after the execution of e , while it becomes t' when the loop terminates..

Definition 9 (Function $Split$). Consider a set of traces T, $Split(T) = T_1,...,T_k$ where each subset T_i is such that

- $T_i = \{t_{i_1}, ..., t_{i_n} |n \geq 1, t_{i_1}, ..., t_{i_n})\in T_i$, $First(\{t_{i_1}\}) = \cdots = First(\{t_{i_n}\}), \forall i \in [1...k]$;
- $T = T_1 \cup ... \cup T_k$;
- $T_i \cap T_j = 0$ and $First(T_i) \neq First(T_j), \forall i, j \in [1...k]$ and $i \neq j$.

Intuitively, $Split(T)$ divides T in $k \geq 1$ distinct sub-sets such that all traces having the same first event are put in the same sub-set. For example, let

$$T = \{\langle a, b, c, d\rangle, \langle a, d\rangle, \langle c, h\rangle,\langle c, k\rangle,\langle b\rangle\}.$$

$Split(T)$ produces the following three sub-sets:

$$
\begin{aligned}
T_1 &= \{\langle a, b, c, d\rangle, \langle a, d\rangle\};\\
T_2 &= \{\langle c, h\rangle,\langle c, k\rangle\};\\
T_3 &= \{\langle b\rangle\}.
\end{aligned}
$$

Now we are ready to define the syntactic transformation function $\mathcal{T}$.

Definition 10 ($\mathcal{T}$). Consider the system specification $SS = ((LS \rightsquigarrow S(C))\cup CS) = \{T_1, ..., T_n\}$ the Lotos processes $x_1 \coloneqq \mathcal{T}(T_1), \dots, x_n \coloneqq \mathcal{T}(T_n)$ can be obtained by applying the transformation function $\mathcal{T}$ defined below to each subset T_i, $1 \leq i \leq n$, of SS:

$$
\begin{cases}
\mathcal{T}'(t, exit) & \text{if } \mathrm{T_i} = t\\
(F_1 \gg R_1 \;\; [\;\;]C_1)[\;\;] & \\
\dots & \\
\;\;[\;\;](F_r \gg R_r[\;\;]C_r & otherwise
\end{cases}
$$

with

$$
\begin{array}{llll}
Split(T_i) = & T_{i_1}, \dots, T_{i_r}, r \geq 1 & \\
F_j & := First\left(T_{i_j}\right) \text{exit}; & j \in [1..r] \\
R_j & := \mathcal{T}\left(Rest\left(T_{i_j}\right)\right) & j \in [1..r] \\
C_j & := \mathcal{T}\left(Cont\left(T_{i_j}\right)\right) & j \in [1..r]
\end{array}
$$

and

$$
\mathcal{T}(t,C) = \begin{cases} C & if \quad t = \langle\ \rangle \\ NC & \\ where & \\ NC := \mathcal{T}'(t_1, NC)[\ ] & \\ \mathcal{T}'(t_2, C) & if \quad t = \langle t_1 \rangle^*.t_2 \\ e; \mathcal{T}'(t', C) & if \quad t = e.t' \end{cases}
$$

where *NC* is a new constant. As a first simple example, consider the set of traces of Figure 4, $T = \{b.e, b.f\}$, and the LTS of the Lotos process P resulting from the application of $\mathcal{T}(T)$, where P is:

$$process \quad P := b; \quad (e; \quad exit \quad [\ \ ] \quad f; \quad exit) \quad endproc.$$

Now consider the following two traces with loops, i.e.,

$$T' = \{\langle a.b \rangle^*, \langle a.c \rangle^*.d\}.$$

Figure 4
$S(\mathcal{T}(P))$

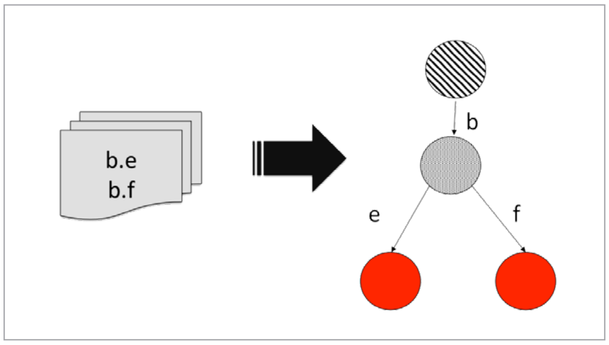


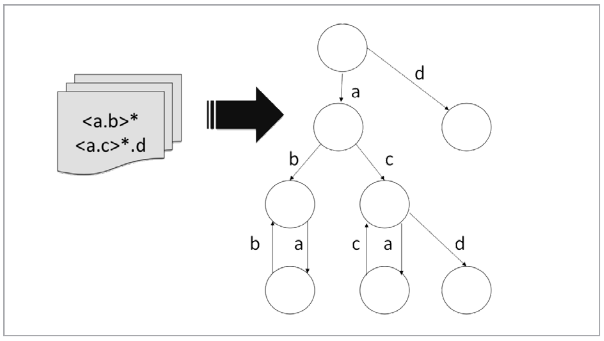
Figure 5 shows the LTS of the Lotos process P' resulting from the application of $\mathcal{T}(T')$, where P' is:

```
process P' :=  a;  (b;  X  [ ]c;  Y)[ ]d;  exit,
where
    process X :=  a;  b;  X  endproc
    process Y :=  a;  c;  Y  [ ]d;  exit  endproc
endproc
```

In the cases above it is $Split(T) = T$ and $Split(T') = T'$. Now, let us consider the following set of traces:

$$T = \{\langle a.\langle b.c \rangle^*.d \rangle^*, a.b.f, \langle a.e \rangle^*\}$$

Figure 5
$S(\mathcal{T}(P'))$



The application of $\mathcal{T}(T)$ as long as possible produces the following Lotos processes. Obviously the tool implementing the transformation functions above performs some optimizations of their behaviour; for example, the introduction of constants is avoided, if an existing one has the same declaration part or it is composed of one event; moreover, the branches containing only an *exit* command are eliminated. The optimized program is the following:

$$
\begin{array}{l}
x := a; \quad (b; (c; z[\ \ ]f; exit) \quad [\ \ ]d; \quad w[\ \ ]e; y \quad) \\
y := a; e; y \\
z := b; c; z \quad [\ \ ]d; w \\
w := a; z.
\end{array}
$$

Finally, the complete Lotos program corresponding to a System Specification is obtained as follows

Definition 11. Consider the system specification $SS = ((LS \rightsquigarrow S(C)) \cup CS) = \{T_1, ..., T_n\}$, the corresponding Lotos process is:

```
process x := (x_1|[S_1]|(x_2|[S_2]|...
             (x_{n−1}|[S_{n −1}|]x_n) ... )),
where
   process x_1 := k_1  endproc
   ...
   process x_n := k_n  endproc
endproc
```

with $k\_1 = \mathcal{T}(T_1)$, and $k\_n = \mathcal{T}(T_n)$.

Moreover, each $S_i, \forall i \in [1...n-1]$ is the set $\alpha T_i \cap (\alpha T_{i+1} \cup ... \cup \alpha T_n)$.

The correctness of our method is stated by the following theorem.

Theorem 2. *Let SS be a System Specification, ψ a selective formula and $\rho = I(SS,\psi)$:*

$$S\left(\mathcal{T}\left(del_\rho(SS)\right)\right) \text{ and } S(\mathcal{T}(SS) \text{ are } \rho-equivalent$$

Proof. First of all we need some notations and definitions. We define a transition relation which ignores a given set of actions: given a transition system $T = (S, A, \longrightarrow, p)$ and a set of actions $\rho \subseteq A$, we define the relation $\longrightarrow_\rho \subseteq S \times \rho \times S$ such that, for each $\alpha \in \rho, \rho \in S$:

By $p \overset{\alpha}{\rightarrow}_\rho q$ we express the fact that it is possible to pass from p to q by performing a (possibly empty) sequence of actions not belonging to ρ and then the action α in ρ. Note that $\longrightarrow_A = \longrightarrow$.

The notions of ρ -bisimulation and ρ -equivalence between transition systems are given as follows. Informally, two transition systems are ρ -equivalent iff they behave in the same way with respect to the actions in ρ.

Let $T_1 = (S_1, A, \longrightarrow, p_1)$ and $T_2 = (S_2, A, \longrightarrow, p_2)$ be transition systems and $\rho \subseteq A$.

- A ρ *-bisimulation*, B, is a binary relation on $S_1 \times S_2$ such that rBq implies:
  (i) $r \overset{\alpha}{\rightarrow}_\rho r'$ implies $q \overset{\alpha}{\rightarrow}_\rho q'$ with $r'Bq'$; and
  (ii) $q \overset{\alpha}{\rightarrow}_\rho q'$ implies $r \overset{\alpha}{\rightarrow}_\rho r'$ with $r'Bq'$
- T_1 and T_2 are ρ *-equivalent* ($T_1 \approx_\rho T_2$) iff there exists a ρ -bisimulation B containing the pair (p_1, p_2).

For each $1 \le i \le n$, let q_i be the Lotos process obtained by T_i and let p_i be the Lotos process obtained by T_i after deleting the actions not in ρ (i.e., $del_\rho(T_i)$). We show that $B = \{(p_i, q_i)\}$ is a ρ -bisimulation.

The proof is carried out by structural induction on Lotos processes.

We consider the case $p_i = \alpha; p_i'$. All the other cases can be proved similarly. If α belongs to ρ, it means that all the traces in T_i start with α, so $p_i \overset{\alpha}{\rightarrow}_\rho p_i'$ and q_i is equal to $\alpha; q_i'$, thus $q_i \overset{\alpha}{\rightarrow}_\rho q_i'$ and by inductive hypothesis $(p_i', q_i') \in B$. If α does not belong to ρ, $p_i \overset{\beta}{\rightarrow} p_i'$. Since α is not a branching action, then q_i cannot be of the form $\alpha.r + s$, so it holds that $q_i \overset{\beta}{\rightarrow} q_i'$ and by inductive hypothesis $(p_i', q_i') \in B$. To complete the proof also the actions that q can perform have been considered.

The corollaries below express important consequences on the system requirements of the reduction method.

$$p \overset{\alpha}{\longrightarrow}_\rho q \equiv p \overset{\delta\alpha}{\longrightarrow} q, where \;\; \delta \in (A-\rho)^*$$

Corollary 1. *Let SS be a System Specification, ψ a selective formula and $\rho = (SS, \psi)$:*

$$S(\mathcal{T}(SS)) \vDash \psi \;\; iff \;\; S(\mathcal{T}\left(del_\rho(SS)\right)) \vDash \psi$$

Proof. The proof follows immediately by Theorems 1 and 2.

Corollary 2. *Let SS be a System Specification, $\chi = \nu Z.\langle E\rangle_E\, tt \wedge [E]_E\, Z$, ψ any selective formula, and $\rho = I(SS, \psi)$:*

$$S(\mathcal{T}(SS)) \vDash \chi \;\; iff \;\; S(\mathcal{T}\left(del_\rho(SS)\right)) \vDash \chi$$

Proof. The proof follows immediately by Theorem 2 and by Definition 8, since $Be(SS) = Be(del_\rho(SS))$.

For the experiment we used the CADP [10] tool. CADP is a popular toolbox that supports high-level descriptions written in Lotos. In Table 8, $MS = \mathcal{T}(SS)$ is the program obtained from the specification in Table 6 (which is defined in the following section and can be obtained from the tool used to implement the methodology), while $S(MS)$ represents the transition system for this process; the table compares the size of $S(MS)$ with the size of the transition systems $S(MS\_\phi)$: the reduction we perform is significant, in terms of both states and transitions, and thus it implies a corresponding reduction of the verification time.

Table 8
Reduction results

S(*MS*)		*S*(*MS_ϕ*)		
states	transitions	states	transitions	state space reduction %
62208	230688	9504	30384	86.3 %

Table 9

Verification time for the property ϕ

standard transition system	reduced transition system	time reduction %
203.2s	18.8s	90.7 %

Table 9 shows the time employed by the CADP to check ϕ on both specifications and the time reduction obtained with our methodology for this purpose.

5 Experimental Evaluation Using the Custom-Made Approach

The methodology presented in the previous section has been exploited for the implementation of a prototype, whose architecture is shown in Figure 6. The prototype is implemented in Java, for portability and reusability. Moreover, OpenXES for managing event log data has been used. The parsed traces are translated into Lotos processes following the methodology described in the previous section; the part of the tool performing the trace reduction can be skipped and, in this case, the produced Lotos program corresponds to the complete model representing the system traces. In any case, the produced Lotos process can be applied to the CADP formal verification environment to check the properties.

The aim is to evaluate the performance of the approach presented in this paper. Experiments were executed on a 32 bit, 2.5 GHz Intel Core i7 CPU equipped with 2 GB of RAM and running Ubuntu 15.10 Linux.

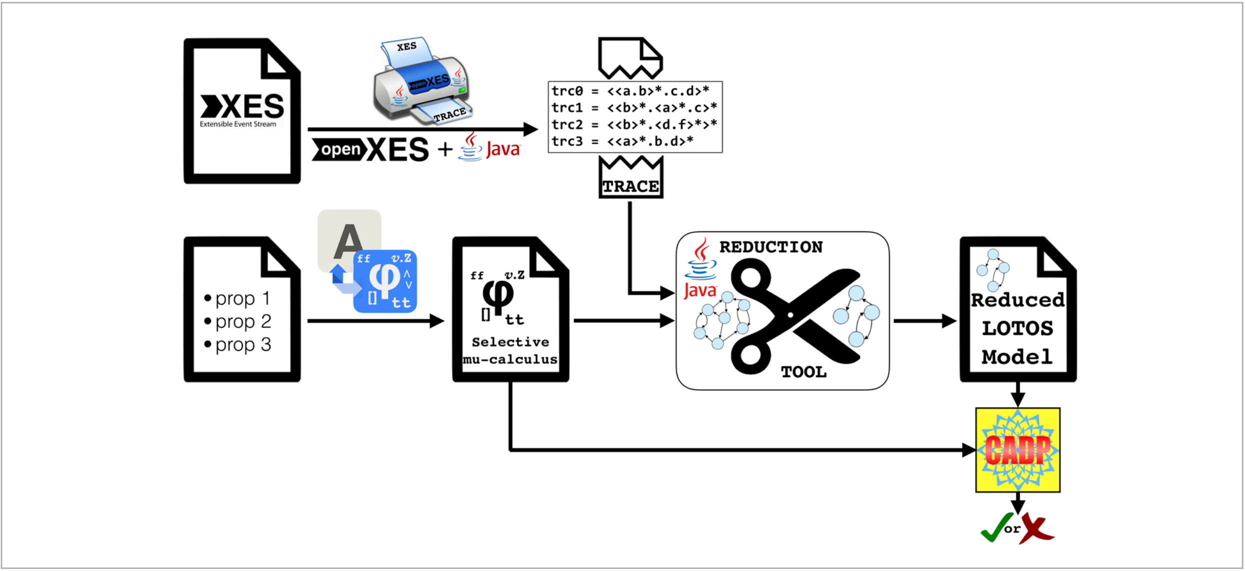
First of all we consider again the example of Section 3. Table 10 shows the state space reduction obtained using the custom-made approach. In particular, the table shows:

- in the column “custom-made approach”, sub-column “time”, the time to produce the Lotos specification *RP* starting from the reduced traces is reported. Moreover, the table shows also the number of states (sub-column “states”) and of the transitions (sub-column “trans”) of the automaton corresponding to the Lotos process *RP* obtained using our approach. Note that the reduction has been applied according to the property that we have to prove for each case study. In this case we consider the φ property explained in Section 3.
- in the column “reduction”, the state space reduction is reported.

As the results show, our custom-made approach is able to achieve a state space reduction that is greater than or equal to 96% in all of considered cases. It

Figure 6

The Architecture of our Prototype



```
trc0 = <<a.b>*.c.d>*
trc1 = <<b>*.<a>*.c>*
trc2 = <<b>*.<d.f>*>*
trc3 = <<a>*.b.d>*
```

Table 10
Results for the reduced bank example - state space

Case study	custom-made approach			reduction
	time (s)	states	trans.	state space reduction %
2000-all-noise	0.219	3264	4495	96%
2000-all-nonoise	0.326	787	1081	99%
2000-scen1	0.275	998	1377	98%
2000-scen2	0.258	754	1074	99%
10000-all-noise	1.148	10659	14487	97%
10000-all-nonoise	1.487	902	1203	99%

should be underlined that this reduction of the state space has also led to a considerable reduction of verification time. For example, for "10000-all-nonoise" the custom-made approach has employed 134.080 sec, while the integrated-tool ones has employed 31686.372 sec with a reduction equal to 99%.

To better complete evaluation, also other samples of real systems were selected. Moreover, we report also the property that we have checked on each case study.

Repair Telephones: this example is taken from the ProM website[4] and considers a process to repair telephones in a company. The company can fix 3 different types of phones.

Property: "It is not possible to archive the case if the user has not been informed". This property is expressed by the following selective mu-calculus:

$$\varphi = [Archive_{Repair}]_{\{Inform\_User\}} ff. \tag{9}$$

The property is true on both the reduced and the complete models.

University of Pisa: the careers of the students of the Bachelor Degree in Computer Engineering have been examined to evaluate the level of compliance of these students with a set of rules that, if followed, lead to the best performance in time and results. The rules specifying the correct behavior are expressed as constraints on the order in which the marks for some courses can be obtained. For example the following constraint can be set

4 http://www.processmining.org/logs/start

Property: "It is possible to obtain marks for Computer Architecture first and then for Digital Logic Design". This property is expressed by the following selective mu-calculus formula:

$$\varphi = \langle computer\_architecture\rangle_{\emptyset}\langle digital\_logic\_design\rangle_{\emptyset} tt. \tag{10}$$

Property φ is false in a correct behavior and it is false on both the reduced and the complete models, since Digital Logic Design is a bridging course for Computer Architecture.

WABO: this example is taken from the ProM website[5]. This data originates from the CoSeLoG project executed under NWO project number 638.001.211. This event log contains the records of the execution of the receiving phase of the building permit application process in an anonymous municipality.

Property: "activity "T10 Determine necessity to stop indication" which determines whether the process should be stopped cannot be performed if activity "T06 Determine necessity of stop advice" has not been performed".

This property is expressed by the following selective mu-calculus:

$$\varphi = [a]_{\{b\}} ff. \tag{11}$$

where

$a = T10\_Determine\_necessity\_to\_stop\_indication$

$b = T06\_Determine\_necessity\_of\_stop\_advice$

The property is true in both in the reduced and in the standard models.

Table 11 has the same structure of Table 10. It shows the state space reduction obtained using our custom-made approach. In these case we also report the data related to the standard model obtained using the integrated-tool approach.

In all the experiments we obtain good results (for example, for the WABO case study, 99% of reduction is obtained). Table 11 shows that the time required for the translation to a Lotos specification of the distributed systems is very low and is even lower when compared with the high efficiency of a general purpose model checker verifying the compliance rules.

5 http://www.processmining.org/logs/start

Table 11
Results for the analysed data sets - state space

Case study	custom-made			integrated-tool			reduction
	time (s)	states	trans.	time (s)	states	trans.	state space reduction %
Repair Telephones	0.016	4	5	1.43	389	464	98 %
University of Pisa	0.01	142	372	0.262	480	820	60 %
WABO	0.034	17	26	2.445	474	588	99 %

6. Limitations

Analyzing data gathered from a running program provides a definite image of a software system, especially when the source code is not provided. Thus, to perform dynamic analysis, the data are collected through instrumentation and execution of a system into one or more execution traces. In order to produce an accurate picture of the system many execution traces have to be collected and this lead to retrieve a large amount of data. This leads to a time-consuming activity, typical of a dynamic analysis. In fact, the main disadvantage of this kind of approach is related to the large amount of data that is collected at run-time. Furthermore, to cope with the space complexity problem, these collected data have to be pre-processed in order to generate models with a reduced size acceptable by formal verification tools. For this purpose, we have used the selective mu-calculus logic, which induces an abstraction using a set of actions O. Obviously on this model it is possible only to verify properties with occurring actions in O. Thus, if from one side our approach provides a solution able to solve the scalability problem, from the other one we are able to only verify behaviours represented in the abstracted model.

Considering that the model is built from traces, another weakness is represented by the impossibility to exactly localize at source-code level the vulnerability. This leads to more time required to fix the problem in the source code. Thus, the proposed method suffers from typical weaknesses of dynamic approaches i.e., the vulnerability can be found only after it happened.

Another issue is related to the temporal logic formulae that are generated with the help of domain experts, for this reason the proposed approach is not fully automatic.

Generally speaking, we consider a dynamic approach because dynamic analysis is able of exposing a subtle flaw or vulnerability too complicated for static analysis alone to reveal and can also be the more expedient method of testing [21]. From the other side, dynamic analysis is able to find defects only in the part of the trace that is actually collected.

7. Related Work

Process mining from execution traces is an interesting and challenging research problem in many areas of computer science. In the information systems context, this is referred to workflow mining, aiming at retrieving business process models from the analysis of event logs recorded in one or more information systems used to support those processes. Quite a lot of research has been done in this setting ([12, 5, 24] to name just a few). The reached results focused on different problems, such as log analysis through clustering, data cleaning from noise, or recognition of particular workflow patterns. Most of them build the workflow models by using the Petri Net formalism and apply analysis techniques on such models.

Model Checking is a verification technique to establish whether a system model complies with a specification described in a formal language. Typically, system models are given by non-deterministic or probabilistic automata. Several works aiming to verify the system development have been proposed. For example, authors in [31] propose ConTEA, a tool integrating the UPPAAL model checker with ConData model based test generator and Conformance and

Fault Injection (ConFI) methodology. The main aim of this work is to improve the quality of the formal model and robustness of the system under analysis. This purpose is reached using a single state machine derived using both ConFI and UPPAAL. Differently, the focus of the proposed method is on the abstraction more than on the quality and robustness of the built models. Boucherit et al. in [3] verify both model and implementation of a software system using an hybrid approach combining property base testing and model checking technique. They propose an approach based on Petri Nets and illustrate its functioning through a simple example related to an access control system. A Formal Quality of Service Assurances Method which relies on stochastic Markov models is proposed in [15] with the aim to facilitate the decision-making process. They consider probabilistic model checking with a set of user-related metric to automatically generate a probabilistic model. The focus of this study is related to the deployment phase, when the software engineer has to select an appropriate cloud offer and deploy the application or one of its parts, such as a microservice. While the probabilistic model checking can not provide counterexample, in the non probabilistic setting counterexamples represent one of the key reasons for the success of model checking [16]. They provide, in the case where model checking shows a property to be false, evidence of this violation, typically in the form of a trace through the model. The method we proposed, considering the model checking technique, offers the counterexample showing the reason why a trace is marked as false from the model checker.

According with the above works, the models used in model checking are manually constructed. However, such model-construction can be extremely time-consuming, or even infeasible in the case of insufficient documentation for an existing system, thus there is an increasing interest in model learning (or specification mining) for formal verification. For example, in [18] a learning algorithm has been proposed for probabilistic systems. More precisely, in [18] AAlergia has been presented which is a state merging algorithm that exclusively learns deterministic models. Given a sample of traces, the algorithm generates a Deterministic Labeled Markov Chain (DLMC) model. A limit of this approach is that the algorithm might not converge to a good model in general. Only with a sufficiently big sample set of traces it is ensured that a given property will hold on the original and the learned model with the same probability. Researchers in [1, 20] propose a fuzzy model aimed to provide security for sensitive data/information in web applications exploiting fuzzy classifiers. Their model is automatically developed to detect vulnerability in web applications and state threat or penetration level. As discussed in literature, when classification models built with machine learning algorithms fail in instance prediction, it is not easy for the analysis to detect and understand what happened [23]. Differently, the adoption of the model checking techniques with the adoption of the counterexample provides to the analyst a reasoning technique to deep understand the model and its behaviour.

Authors in [8, 17] propose TLA+, a specification language for concurrent and reactive systems combining the temporal logic TLA with the full first-order logic and set theory. Proving TLA+ properties needs theorem proof system requiring human expertise in the proof checkers. On the contrary, our method uses a model-checking based verification tool that allows to automatically check the properties, saving a lot of efforts.

On the other hand, learning models is also useful in the compliance checking field, which has a growing importance for the businsess process management and auditing communities. It refers to the adherence of the discovered process to internal or external rules and then deals with verification issues. Many efforts have been taken in the research of business process compliance checking. The first comprehensive compliance checking approach based on Petri net patterns and alignments was proposed in [25] by Ramezani, Fahland, and van der Aalst. Colored Petri Nets have been used in [11] to perform a backward compliance checking to verify whether executions of business processes are complying with certain normative constraints. They principally focused on the formal theories of normative systems. In [14] compliance checking has been conducted using an abstract process model and abstract compliance rules. In this way state explosion arising from control flow and data dimension is avoided.

In this paper, we derive an abstract model, defined in a process algebra, from traces obtained from the execution log; process mining techniques are supposed to be used to obtain a model for which compliance checking can be performed to find commonalities and discrepancies between the modeled behavior and the observed behavior [30]. In our case compliance rules can be represented both directly as other traces to be included in the model and as temporal logic formulae; in this last case, we check compliance using a formal verification methodology, i.e., model checking. In this way we establish compliance in a formal environment without introducing additional concepts. Moreover, our aim is to use an existing model checker such as CADP [10], which is a mature verification tool with modern designs, with expressive input languages and efficient analysis methods, and not to design a custom-made model checker. In fact, the most widely used model checkers are by now extremely sophisticated programs that have been crafted over many years by experts in the specific techniques employed by the tool: any re-implementation of similar tools could likely yield worse performance. We exploit CADP [10] since it is a popular toolbox maintained, regularly improved, and used in many industrial projects, as a verification framework. Another important advantage of using CADP is that, when a property does not hold, the model checking algorithm generates a counter-example, i.e., an execution trace leading to a state in which the property is violated. This ability to generate counter-examples can be exploited to pinpoint the cause of an error and possibly correct it. By automating a tedious task that must otherwise be done manually, our approach reduces the cost associated with analysing process models for compliance. A typical problem of the model checking context is present also in our context, since the process discovered from the log can be much too complex; then we use abstraction techniques to discover abstract processes after pre-processing the traces derived from the log, so directly building a reduced model; the method is completely automatable since it is based on the syntactic structure of the formulae to be verified.

The model checking approach has been already used in [13] where the authors propose to map BPMN models directly to finite state machines (i.e., Kripke structures) and to express the compliance rules in a graphical language for better understandability. Subsequently, these are translated into linear temporal logic formulae to be integrated.

8. Conclusion

This paper presents an approach to compliance checking through model checking. The approach aims at discovering a process described by means of a process algebra language, while the compliance rules are defined through temporal logic properties.

The main characteristics of the method are:

- the traces obtained from the logs are pre-processed to obtain a suitable representation of a distributed system;
- pre-processed traces are reduced on the basis of the logic properties; so it is avoided the state explosion arising when starting from big sets of traces;
- reduction rules directly apply to the traces, without building the corresponding LTSs before; the rules are based on the traces syntax and are completely automatable, no semantic information is required and an easy implementation is possible;
- compliance between the discovered process and the rules is performed through model checking of a set of temporal logic properties defining the rules; existing model checking environments can be used and ad-hoc checkers do not need; the semantics of the system is obtained as a transition system automatically generated by the model checking environment;
- the proof method can easily include compositional techniques too, like as, for example, [7, 8, 26], so proving the properties for single devices it is possible to obtain the proof of the properties for the whole system.

In this paper, we present a formal approach and presuppose perfect information. However, real logs are rarely complete and/or noise free. We are considering how to modify the truth value of each formula on the basis of the fact that such value is true on all behaviours except some with very low probability of occurrence, or this value is false on all behaviors except those with very low probability of occurrence, or it is false because of the incompleteness of the behavior.

Acknowledgment

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. Alhassan, J. K., Misra, S., Umar, A., Maskeliūnas, R., Damaševi ius, R., Adewumi, A. A Fuzzy Classifier-Based Penetration Testing for Web Applications. In International Conference on Information Theoretic Security, Springer, 2018, 95-104. https://doi.org/10.1007/978-3-319-73450-7_10

2. Barbuti, R., De Francesco, N., Santone, A., Vaglini, G. Selective Mu-Calculus and Formula-Based Equivalence of Transition Systems. Journal of Computing System Sciences, 1999, 59(3), 537-556. https://doi.org/10.1006/jcss.1999.1660

3. Boucherit, A., Castro, L. M., Khababa, A., Hasan, O. Towards the Formal Development of Software Based Systems: Access Control System as a Case Study. Information Technology and Control, 2018, 47(3), 393-405. https://doi.org/10.5755/j01.itc.47.3.20330

4. Ceccarelli, M., Cerulo, L., Santone, A. De Novo Reconstruction of Gene Regulatory Networks from Time Series Data, an Approach Based on Formal Methods. Methods, 2014, 69(3), 298-305. https://doi.org/10.1016/j.ymeth.2014.06.005

5. Chan, N. N., Yongsiriwit, K., Gaaloul, W., Mendling, J. Mining Event Logs to Assist the Development of Executable Process Variants. In Advanced Information Systems Engineering - 26th International Conference, CAiSE 2014, Thessaloniki, Greece, June 16-20, 2014, 548-563. https://doi.org/10.1007/978-3-319-07881-6_37

6. Clarke, E. M., Lerda, F. Model Checking: Software and Beyond. Journal of Universal Computer Science, 2007, 13(5), 639-649.

7. Clarke, E. M., Long, D. E., McMillan, K. L. Compositional Model Checking. In IEEE Computer Society, LICS, 1989, 353-362.

8. Cousineau, D., Doligez, D., Lamport, L., Merz, S., Ricketts, D., Vanzetto, H. Tla+ Proofs. In International Symposium on Formal Methods, Springer, 2012, 147-154. https://doi.org/10.1007/978-3-642-32759-9_14

9. de Medeiros, A. K. A., van Dongen, B. F., van der Aalst, W. M. P., Weijters, A. J. M. M.. Process Mining: Extending the Ĩ-Algorithm to Mine Short Loops. In Eindhoven University of Technology, Eindhoven, 2004.

10. Garavel, H., Lang, F., Mateescu, R., Serwe, W. CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes. STTT, 2013, 15(2), 89-107. https://doi.org/10.1007/s10009-012-0244-z

11. Jiang, J., Aldewereld, H., Dignum,V., Tan, Y-H. Compliance Checking of Organizational Interactions. ACMTransactions on Management Information System, 2015, 5(4), 23:1-23:24. https://doi.org/10.1145/2629630

12. Kalenkova, A., Lomazova, I. A., van der Aalst, W. M. P. Process Model Discovery: A Method Based on Transition System Decomposition. In Application and Theory of Petri Nets and Concurrency - 35th International Conference, PETRI NETS 2014, Tunis, Tunisia, June 23-27, 2014, 71-90. https://doi.org/10.1007/978-3-319-07734-5_5

13. Kherbouche, O. M., Ahmad, A., Basson, H. Formal Approach for Compliance Rules Checking in Business Process Models, 2013. https://doi.org/10.1109/ICET.2013.6743500

14. Knuplesch, D., Ly, L. T., Rinderle-Ma, S., Pfeifer, H., Dadam, P. On Enabling Data-Aware Compliance Checking of Business Process Models. In Conceptual Modeling - ER 2010, 29th International Conference on Conceptual Modeling, Vancouver, BC, Canada, November 1-4, 2010, 6412, LNCS, 332-346. https://doi.org/10.1007/978-3-642-16373-9_24

15. Kochovski, P., Drobintsev, P. D., Stankovski, V. Formal Quality of Service Assurances, Ranking and Verification of Cloud Deployment Options with a Probabilistic Model Checking Method. Information and Software Technology, 2019, 109, 14-25. https://doi.org/10.1016/j.infsof.2019.01.003

16. Kwiatkowska, M., Norman, G., Parker, D. Advances and Challenges of Probabilistic Model Checking. In 2010 48th IEEE Annual Allerton Conference on Communication, Control, and Computing (Allerton), 2010, 1691-1698. https://doi.org/10.1109/ALLERTON.2010.5707120

17. Lamport, L. Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman Publishing Co., Inc., 2002.

18. Mao, H., Chen, Y., Jaeger, M., Nielsen, T. D., Larsen, K. G., Nielsen, B. Learning Probabilistic Automata for Model Checking. In 8th International IEEE Computer Society Conference on Quantitative Evaluation of Systems, QEST 2011, Aachen, Germany, 5-8 September, 2011, 111-120. https://doi.org/10.1109/QEST.2011.21

19. Martinelli, F., Mercaldo, F., Nardone, V., Orlando, A., Santone, A, Vaglini, G. Safety Critical Systems Formal Verification Using Execution Traces. In Proceedings of

the 27th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2018. https://doi.org/10.1109/WETICE.2018.00054

20. Martinelli, F., Mercaldo, F., Nardone, V., Santone, A.. Car Hacking Identification Through Fuzzy Logic Algorithms. IEEE International Conference on Fuzzy Systems, 2017. https://doi.org/10.1109/FUZZ-IEEE.2017.8015464
21. Moser, A., Kruegel, C., Kirda, E. Limits of Static Analysis for Malware Detection. In IEEE 23rd Annual Computer Security Applications Conference (ACSAC 2007), 2007, 421-430. https://doi.org/10.1109/ACSAC.2007.4413008
22. Munoz-Gama, J. Conformance Checking and Diagnosis in Process Mining - Comparing Observed and Modeled Processes. Springer Lecture Notes in Business Information Processing, 270, 2016. https://doi.org/10.1007/978-3-319-49451-7
23. Parnas, D. L. The Real Risks of Artificial Intelligence. Communications of the ACM, 2017, 60(10), 27-31. https://doi.org/10.1145/3132724
24. Partington, A., Wynn, M. T., Suriadi, S., Ouyang, C, Karnon, J. Process Mining for Clinical Processes: A Comparative Analysis of four Australian Hospitals. ACM Transactions on Management Information Systems, 2015, 5(4),19:1-19:18. https://doi.org/10.1145/2629446
25. Ramezani, E., Fahland, D., van der Aalst, W. M. P. Where Did I Misbehave? Diagnostic Information in Compliance Checking. In Business Process Management - 10th International Conference, BPM 2012, Tallinn, Estonia, September 3-6, Springer, 2012, 748, LNCS, 262-278. https://doi.org/10.1007/978-3-642-32885-5_21
26. Santone, A. Automatic Verification of Concurrent Systems Using a Formula-Based Compositional Approach. Acta Informatica, 2002, 38(8), 531-564. https://doi.org/10.1007/s00236-002-0084-5
27. Santone, A., Vaglini, G. Abstract Reduction in Directed Model Checking CCS Processes. Acta Informatica, 2012, 49(5), 313-341. https://doi.org/10.1007/s00236-012-0161-3
28. Santone, A., Vaglini, G. Conformance Checking Using Formal Methods. In Proceedings of the 11th International Joint Conference on Software Technologies (ICSOFT 2016), 2016, 1: ICSOFT-EA, 258-263. https://doi.org/10.5220/0005993402580263
29. Santone, A., Vaglini, G., Villani, M. L. Incremental Construction of Systems: An Efficient Characterization of the lacking sub-system. Science of Computer Programming, 2013, 78(9), 1346-1367. https://doi.org/10.1016/j.scico.2012.07.015
30. van der Aalst, W. M. P. Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer, 2011.
31. Villani, E., Pontes, R. P., Coracini, G. K., Ambrosio, A. M. Integrating Model Checking and Model Based Testing for Industrial Software Development. Computers in Industry, 2019, 104, 88-102. https://doi.org/10.1016/j.compind.2018.08.003