Conformance Checking using Formal Methods

Antonella Santone[1] and Gigliola Vaglini[2]
[1]*Department of Engineering, University of Sannio, Benevento, 82100, Italy*
[2]*Department of Information Engineering, University of Pisa, Pisa, 56122, Italy*

Keywords: Model Discovery, Process Mining, Model Checking, Conformance Checking.

Abstract: Conformance checking is an important process mining task; it aims to detect inconsistencies between the model of a process and its corresponding execution log. This paper proposes an approach in which it is given a declarative description, represented by a set of temporal logic properties, for the process model; the process discovered from the log is described by means of a process algebra, and conformance checking is performed through the model checking of the discovered process against the properties. To discover the process we consider additional information contained in the log and associated with the single events. Moreover, since discovered processes tend, in general, to be very large and complex, we look for a reduced process containing only the parts relevant for the properties satisfaction. In this way we reduce both the space needed for the discovered process and the time complexity for the properties verification.

1 INTRODUCTION

The process mining techniques allow knowledge extraction from events stored by information systems. A classical application of process mining is process discovery. Process discovery techniques take an event log and produce a process description. In the case of distributed systems, the discovered process may be typically described in one of the several possible specification languages, i.e., Petri nets (Reisig and Rozenberg, 1998), Business Process Model and Notation (BPMN)[1], or process algebra specifications (Milner, 1989). The general idea is to discover, besides monitoring and improving, real processes by extracting knowledge from event logs readily available in information systems. In fact, all process mining techniques assume that it is possible to record events. Event logs may store also additional information about events, i.e., the particular device executing the activity or the characterization of synchronization events among devices.

Despite its benefits, 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 model every possible process behavior can result in an information overload that reduces model comprehensibility. Consequently, conformance checking is an important part of the process mining methodology; by this checking it is possible to evaluate the level of discrepancy between the discovered process and the process model.

In this work, we consider that distinct sets of traces, each one regarding the behavior of one device in the system, are retrieved from the log exploiting the additional information on the source of each activity, and that sub-traces representing simple loops can be individuated (using known methods, such as, for example, the α^+-algorithm (de Medeiros et al., 2004)); moreover, the description of the possible synchronization points, again taken from the additional information included in the log, are used to express the communication among devices. Finally, to address the problem of coping with the high complexity of models obtained by means of automatic process mining, we accomplish a pre-processing on the traces guided by the properties required for the system. Abstraction is seen as an effective approach to represent readable models, showing aggregated activities and hiding irrelevant details. In our case, abstraction is characterized by the set of temporal logic formulae specifying the system properties. These formulae are, in some sense, the declarative side of the system. From the resulting traces, we discover a system described by CCS (Milner, 1989), a process algebra specification language. Conformance is established through the model checking of the properties on the discovered process:

[1]http://www.omg.org/spec/BPMN/2.0

Santone, A. and Vaglini, G.
Conformance Checking using Formal Methods.
DOI: 10.5220/0005993402580263
In *Proceedings of the 11th International Joint Conference on Software Technologies (ICSOFT 2016) - Volume 1: ICSOFT-EA*, pages 258-263
ISBN: 978-989-758-194-6

in this way we can use existing model checking environments without introducing additional concepts to establish the conformance of the discovered process.

The remainder of this paper is organised as follows. Section 2 presents a short description of the theoretical background of the work; Section 3 describes the specification and reduction methodology, while Section 4 shows the technique for model discovery. Section 5 discusses some related work and Section 6 presents the conclusions and the future works.

2 BACKGROUND

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

2.1 CCS

We assume that the reader is familiar with the basic concepts of process algebras and CCS. We summarize the most relevant definitions below, and refer to (Milner, 1989) for more details. The CCS syntax we consider is the following:

$$p ::= \alpha.p \mid nil \mid p+p \mid p|p \mid x \mid p[f]$$

Each constant x has a definition of the type $x := p_x$, where p_x is called the *body* of x. α ranges over a set of names of actions $\mathcal{A} \cup \tau$, where $\mathcal{A} = \{a, b, , ...\}$ and τ is the invisible action; *nil* denotes the empty process. The operators to build processes are prefixing ($\alpha.p$), summation ($p+p$), parallel composition ($p|q$), relabeling ($p[f]$). In this paper we use slightly differently the communication operator: in fact, we suppose that the actions causing the synchronization between p and q are equal. Moreover, we use the action named δ only to synchronize processes on termination. All the processes we define have the *well-formedness* property, that is, they synchronize on δ if and only if they terminate. A typical use of δ can be found in the definition (see (Milner, 1989)) of the operators $\|$ and ";" that correspond to the sequentialization and parallel execution of two well-formed processes. The process resulting from their applications are well-formed too.

$$p\|q = p[\delta_1/\delta]|q[\delta_2/\delta]|(\delta_1.\delta_2.DONE + \delta_2.\delta_1.DONE)$$
$$p;q = (p[\delta_1/\delta]|\delta_1.q)$$
$$DONE := \delta.nil$$

DONE is a constant whose body contains only the termination.

2.2 Model Checking Selective Mu-calculus Formulae

In the model checking framework (Clarke and Lerda, 2007), systems requirements are expressed as formulae in a temporal logic. Model checkers accept two inputs, a system model 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 (Barbuti et al., 1999), which is different from mu-calculus (Stirling, 1989) in the definition of the modal operators. The syntax of the selective mu-calculus is the following:

$$\phi ::= \mathtt{tt} \mid \mathtt{ff} \mid Z \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid [K]_R\phi \mid \langle K\rangle_R\phi \mid \nu Z.\phi \mid \mu Z.\phi$$

where K, R are sets of actions in $\mathcal{A}$, while Z ranges over a set of variable names; $\mu Z.\phi$ is the least fixpoint 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 process which, after the performance of a sequence of actions not belonging to $R \cup K$, followed by an action in K, can transform into a process obeying ϕ. $\langle K\rangle_R\phi$ is satisfied by a process which is able to transform into a process obeying ϕ after the performance of a sequence of actions not belonging to $R \cup K$, followed by an action in K.

The precise and formal definition of satisfaction of selective mu-calculus formulae can be found in (Barbuti et al., 1999).

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.* In (Barbuti et al., 1999) it has been proved that: "two processes are ρ-equivalent if and only if they satisfy the same set of formulae with occurring actions in ρ". The interesting consequence of the theorem is that a formula of the selective mu-calculus with occurring actions in ρ can be checked on any process ρ-equivalent to the standard one.

3 THE SPECIFICATION AND REDUCTION METHODOLOGY

In this section, we assume that the log is rearranged so that separate traces for each device in a distributed system can be obtained exploiting the source of each activity. Further, we give each activity a different name; moreover simple loops are solved as (de Medeiros et al., 2004); at the end, 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. Property-driven reductions can be performed on the resulting traces to obtain an abstract CCS system. This system specification will be model checked against the required properties to state its conformance.

3.1 Trace-based System Specification

As stated in (de Medeiros et al., 2004), *"We assume perfect information: (i) the log must be complete (i.e., if a task can follow another task directly, the log contains an example of this behavior) and (ii) the log is noise free (i.e., everything that is registered in the log is correct). This is not a real limitation because we are primarily interested in formal properties of our algorithms"*. The language we assume to define the traces obtained through the log pre-processing is the following.

Definition 1 (Trace)**.** *Let $\mathcal{A} = \{e_1, e_2, \ldots\}$ be a set of activity names, a trace of such names can be built up by the following syntax: $t ::= e \mid t.t \mid < t >^* \mid \lambda$ where $e \in \mathcal{A}$ 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 $<< t >^* >^*$ is the same as $< t >^*$ and $< \lambda >^*$ is the same as 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*
- *$\mathcal{B}e(T)$ is the set of pairs (e, e') defined as follows:*
  $\mathcal{B}e(T) = \{(e, e') \mid t_1 = s.e.t \in T,\ t_2 = s.e'.t' \in T,\ e \neq e' \text{ and } s, t, t' \in \mathcal{A}^*\}$
  The names in each pair in $\mathcal{B}e(T)$ represent the point in which two traces begin to differ.

After having obtained from the log the set T of traces of activity names, our aim is to derive from T the description of the distributed system through CCS processes composed in parallel. The first step of the 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 $LS = \{T_1, \ldots, T_n\}$, 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. Thus $\alpha LS = \alpha T_1 \cup \ldots \cup \alpha T_n$.*

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 $C = \{c_1, \ldots, c_m\}$, where $c_j = (e_{j_1}, \ldots, e_{j_k}), 1 \leq j \leq m$ and 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 $FS = LS \rightsquigarrow S(C)$. From now on we will use $S(C)$ for $\{S(c_1), \ldots, S(c_m)\}$.*

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. It is possible to have a third step too of the method, when some system requirements are included in the specification. This kind of requirements should be expressible as constraints on the order and the number of communication activities, and thus by means of traces on αFS. From now on we call System Specification, *SS*, any Flow specification.

3.2 Formula-based Reduction of the System Specification

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

Definition 5 ($del_I(t)$)**.** *Given a trace t on the alphabet $\mathcal{A}$ and $I \subseteq \mathcal{A}$, we define the function $del_I : \mathcal{A}^* \rightarrow \mathcal{A}^*$ as following:*

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

where λ is the empty trace.

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

$$del_I(T) = \{del_I(t) \mid t \in T\}$$

According to the syntax of a trace given in Definition 1, if the trace t is of the form "e" two cases may occur: 1) e is not the name of an interesting activity: we delete e; 2) e is the name of an interesting activity: we cannot 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 recursively on t'', while if the trace t is of the form $< t' >^*$, we apply recursively 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 the activity names in I. Consequently, there are several problems to be taken into account: when the name given to a synchronization activity 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 behaviors of the system. Thus, to preserve the truth value of a formula, the names of all communications and all branching names, must be preserved by the reduction.

Definition 6 ($I(SS,\varphi)$)**.** *Consider a system specification $SS = LS \rightsquigarrow S(C) = \{T_1, \cdots, T_n\}$ and a selective mu-calculus formula φ. The activity names of interest in the system are*

$$I(SS,\varphi) = O(\varphi) \cup \mathcal{B}e(T_1 \cup \cdots \cup T_n) \cup S(C)$$

4 MODEL DISCOVERY

Now we define a general syntactic transformation function $\mathcal{T}$ which transforms a trace-based system specification into a CCS process by means of the auxiliary functions defined below. For the purpose of mathematical reasoning it is often convenient and simpler to represent a transition system algebraically in the form of CCS processes, especially because all formal verification environments can easily compute, through the operational semantics, the corresponding transition system.

Let T be a set of traces.

- $\mathcal{F}irst(T) = \{e \mid e.t \in T \text{ or } < e.t >^* .t' \in T\}$
- $\mathcal{R}est(T) = \{t \mid e.t \in T\} \cup \{t_1 . < e.t_1 >^* .t_2 \mid < e.t_1 >^* .t_2 \in T\}$
- $\mathcal{C}ont(T) = \{t_2 \mid < t_1 >^* .t_2 \in T\}$

$\mathcal{F}irst(T)$ returns the set of all the first names of the traces in T, while $\mathcal{R}est(T)$ says as a trace may go on after its first activity has been performed; $\mathcal{C}ont(T)$ describes what happens when a loop is skipped. For example, the trace $< e.t >^* .t'$ describes a behavior that becomes $t. < e.t >^* .t'$ after the execution of e, while it becomes t' when the loop terminates.

Definition 7 (Function $\mathcal{S}plit$)**.** *Consider a set of traces T, $\mathcal{S}plit(T) = T_1, \ldots, T_k$ where each subset T_i is such that*

- $T_i = \{t_{i_1}, \ldots, t_{i_n} \mid n \geq 1, t_{i_1}, \ldots, t_{i_n} \in T_i, \mathcal{F}irst(\{t_{i_1}\}) = \ldots = \mathcal{F}irst(\{t_{i_n}\})\}$, $\forall i \in [1..k]$;
- $T = T_1 \cup \cdots \cup T_k$;
- $T_i \cap T_j = \emptyset$ *and* $\mathcal{F}irst(T_i) \neq \mathcal{F}irst(T_j)$, $\forall i, j \in [1..k]$ *and* $i \neq j$.

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

Definition 8 ($\mathcal{T}$)**.** *Consider the system specification $SS = ((LS \rightsquigarrow S(C))) = \{T_1, \cdots, T_n\}$ the CCS processes $x_1 := \mathcal{T}(T_1), \ldots, x_n := \mathcal{T}(T_n)$ can be obtained by applying the transformation function $\mathcal{T}$ defined below to each T_i, $1 \geq i \leq n$: $\mathcal{T}(T_i) =$*

$$\begin{cases} \mathcal{T}'(t, DONE) & \text{if } T_i = t \\ (F_1; R_1 + C_1) + \cdots + (F_r; R_r + C_r) & \text{otherwise} \end{cases}$$

with

$$\begin{cases} \mathcal{S}plit(T_i) = T_{i_1}, \ldots, T_{i_r}, r \geq 1 & \\ F_j := \mathcal{F}irst(T_{i_j}); DONE & j \in [1..r] \\ R_j := \mathcal{T}(\mathcal{R}est(T_{i_j})) & j \in [1..r] \\ C_j := \mathcal{T}(\mathcal{C}ont(T_{i_j})) & j \in [1..r] \end{cases}$$

and $\mathcal{T}'(t, C) =$

$$\begin{cases} C & \text{if } t =<> \\ NC, \text{ where} & \\ NC := \mathcal{T}'(t_1, NC) + \mathcal{T}'(t_2, C) & \text{if } t =< t_1 >^* .t_2 \\ e.\ \mathcal{T}'(t', C) & \text{if } t = e.t' \end{cases}$$

where NC is a new constant.

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

Definition 9. *Consider the system specification $SS = ((LS \rightsquigarrow S(C))) = \{T_1, \cdots, T_n\}$, the corresponding CCS process is:*

$$x := (x_1 \| x_2 \| \cdots \| x_n)$$
$$\text{where}, \quad x_i := \mathcal{T}(T_i) \quad \forall i \in [1..n]$$

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

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

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

The corollary below expresses important consequences on the system requirements of the reduction method.

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

$$\mathcal{T}(SS) \models \psi \text{ iff } \mathcal{T}(del_\rho(SS)) \models \psi$$

Example.
Let now consider the following Layout Specification *LS* and the renaming function *S*:

$$\begin{array}{lcl} LS & = & \{T_1, T_2\}, \text{ where} \\ T_1 & = & \{< a.b >^*, < a.c >^* .d\} \\ T_2 & = & \{< e.f >^*\} \\ S(C) & = & S(c_1) = k, \text{ where} \\ c_1 & = & (a, e) \end{array}$$

The Flow Specification $FS = LS \rightsquigarrow S(C)$ is:

$$\begin{array}{lcl} FS & = & \{T_1', T_2'\} \\ T_1' & = & \{< k.b >^*, < k.c >^* .d\} \\ T_2' & = & \{< k.f >^*\} \end{array}$$

Let $\varphi = \langle k \rangle_\emptyset \langle d \rangle_\emptyset \, \mathtt{tt}$ be the property that must be verified. Applying our methodology we obtain the following reduced Flow Specification *RFS*:

$$\begin{array}{lcl} RFS & = & \{RT_1', RT_2'\} \\ RT_1 & = & \{< k. >^*, < k >^* .d\} \\ RT_2 & = & \{< k >^*\} \end{array}$$

The discovered CCS model obtaining using *RFS* is:

$$\begin{array}{lcl} x & := & (x_1 \| x_2) \\ x_1 & := & k.x_1 + d.DONE \\ x_2 & := & k.x_2 \end{array}$$

5 RELATED WORK

Conformance testing, i.e., checking whether a given model and a given event log fit together, can be a seen as a very specific form of log-based verification: the discovered process model (i.e., a Petri net) is used to verify whether the log satisfies some behavioral properties expressed by several possible formalisms. The first comprehensive approach to conformance analysis was proposed in (Rozinat and van der Aalst, 2008) by Rozinat and van der Aalst. Successively, in (de Leoni et al., 2015) van der Aalst and other authors present a novel log preprocessing and conformance checking approach tailored towards declarative models. They formulated conformance checking problems as an optimization problem and they suggest that traces containing undesirable behaviors could be corrected. Recently, declarative approaches have received increasing interest and suggest a basically different way of describing business processes too (van der Aalst et al., 2009). While imperative models specify exactly how things have to be done, declarative approaches only focus on the logic that governs the interplay of actions in the process by describing the activities that can be performed, as well as constraints prohibiting undesired behavior. However, declarative process mining techniques may produce models with a high quantity of constraints, which may be incomprehensible for humans, as showed by Bose et al. (Bose et al., 2013). The combination of constraints in a declarative process model might generate new hidden dependencies, which are complex and difficult to be identified (Haisjackl et al., 2013); moreover, the increasing number of restrictions negatively impacts on the model quality (Rosa et al., 2011). The work of van der Aalst et al. (van der Aalst et al., 2005) follows another road and is closely related to this paper: given an event log and a temporal property, they verify whether the observed behavior matches the (un)expected/(un)desirable behavior through a custom-made LTL Checker.

In this paper, we analyse conformance checking using a formal verification methodology: first we derive from the log a process described by a process algebra, then we verify conformance between the discovered process and any process model respecting the given properties by the model checking of the discovered process with respect to those properties. We can exploit existing efficient model checking frameworks to perform this task, for example, (Francesca et al., 2011; De Francesco et al., 2016; De Francesco et al., 2014; Barbuti et al., 2005). To cope with time and space complexity (a typical problem of the model checking context), we discover an abstract process by a preprocessing of the traces derived from the log, so directly building a reduced process, without generating before the LTS, for example, corresponding to the complete log as in (Rubin et al., 2006); the reduction is performed on the basis of the temporal logic formulae to be verified and the method is completely automatable since it is based on the syntactic structure of the formulae.

6 CONCLUSION AND FUTURE WORK

An approach to conformance checking through model checking has been presented. The approach aims at the discovering of a process described in the process algebra framework, while the system model is defined through temporal logic properties. The main characteristics of the method are: (i) conformance between the discovered process and the system model performed by model checking of the temporal logic properties; (ii) pre-processed traces are reduced on the basis of the system properties; so it is avoided the state explosion arising when starting from big sets of traces; (iii) transformation rules directly apply to traces, without building any process representation before. As future work we planning to implement our approach and to use LOTOS (Bolognesi and Brinksma, 1987) specification language, instead of CCS, in order to use more mature tools.

REFERENCES

Barbuti, R., De Francesco, N., Santone, A., and Vaglini, G. (1999). Selective mu-calculus and formula-based equivalence of transition systems. *J. Comput. Syst. Sci.*, 59(3):537–556.

Barbuti, R., De Francesco, N., Santone, A., and Vaglini, G. (2005). Reduced models for efficient CCS verification. *Formal Methods in System Design*, 26(3):319–350.

Bolognesi, T. and Brinksma, E. (1987). Introduction to the iso specification language lotos. *Computer Networks*, 14:25–59.

Bose, R. P. J. C., Maggi, F. M., and van der Aalst, W. M. P. (2013). Enhancing declare maps based on event correlations. In *BPM 2013, Beijing, China, August 26-30, 2013. Proceedings*, pages 97–112.

Clarke, E. M. and Lerda, F. (2007). Model checking: Software and beyond. *J. UCS*, 13(5):639–649.

De Francesco, N., Lettieri, G., Santone, A., and Vaglini, G. (2014). Grease: A tool for efficient "nonequivalence" checking. *ACM Trans. Softw. Eng. Methodol.*, 23(3):24.

De Francesco, N., Lettieri, G., Santone, A., and Vaglini, G. (2016). Heuristic search for equivalence checking. *Software and System Modeling*, 15(2):513–530.

de Leoni, M., Maggi, F. M., and van der Aalst, W. M. P. (2015). An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. *Inf. Syst.*, 47:258–277.

de Medeiros, A. K. A., van Dongen, B. F., van der Aalst, W. M. P., and Weijters, A. J. M. M. (2004). Process mining: Extending the α-algorithm to mine short loops. In *Eindhoven University of Technology, Eindhoven*.

Francesca, G., Santone, A., Vaglini, G., and Villani, M. L. (2011). Ant colony optimization for deadlock detection in concurrent systems. In *COMPSAC*, pages 108–117.

Haisjackl, C., Zugal, S., Soffer, P., Hadar, I., Reichert, M., Pinggera, J., and Weber, B. (2013). Making sense of declarative process models: Common strategies and typical pitfalls. In *BMMDS/EMMSAD 2013, Valencia, Spain, June 17-18, 2013. Proceedings*, pages 2–17.

Milner, R. (1989). *Communication and concurrency*. PHI Series in computer science. Prentice Hall.

Reisig, W. and Rozenberg, G., editors (1998). *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*, volume 1491 of *Lecture Notes in Computer Science*. Springer.

Rosa, M. L., Wohed, P., Mendling, J., ter Hofstede, A. H. M., Reijers, H. A., and van der Aalst, W. M. P. (2011). Managing process model complexity via abstract syntax modifications. *IEEE Trans. Industrial Informatics*, 7(4):614–629.

Rozinat, A. and van der Aalst, W. M. P. (2008). Conformance checking of processes based on monitoring real behavior. *Inf. Syst.*, 33(1):64–95.

Rubin, V., Dongen, B. F. V., Kindler, E., and Günther, C. W. (2006). Process mining: A two-step approach using transition systems and regions. Technical report, BPM Center Report BPM-06-30, BPM Center.

Stirling, C. (1989). An introduction to modal and temporal logics for ccs. In *Concurrency: Theory, Language, And Architecture*, pages 2–20.

van der Aalst, W. M. P., de Beer, H. T., and van Dongen, B. F. (2005). Process mining and verification of properties: An approach based on temporal logic. In *OTM Conferences 2005, Agia Napa, Cyprus, October 31 - November 4, 2005, Proceedings, Part I*, pages 130–147.

van der Aalst, W. M. P., Pesic, M., and Schonenberg, H. (2009). Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113.