Information Systems 56 (2016) 304–325

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Diagnosing behavioral differences between business process models: An approach based on event structures

Abel Armas-Cervantes [*,a], Paolo Baldan [b], Marlon Dumas [a], Luciano Garcia-Bañuelos [a]

[a] Institute of Computer Science, University of Tartu, Estonia
[b] Department of Mathematics, University of Padova, Italy

ARTICLE INFO

Article history:
Received 18 December 2014
Received in revised form
16 September 2015
Accepted 21 September 2015
Available online 19 October 2015

Keywords:
Process model comparison
Asymmetric event structures

ABSTRACT

Companies operating in multiple markets or segments often need to manage multiple variants of the same business process. Such multiplicity may stem for example from distinct products, different types of customers or regulatory differences across countries in which the companies operate. During the management of these processes, analysts need to compare models of multiple process variants in order to identify opportunities for standardization or to understand performance differences across variants. To support this comparison, this paper proposes a technique for diagnosing behavioral differences between process models. Given two process models, it determines if they are behaviorally equivalent, and if not, it describes their differences in terms of behavioral relations – like causal dependencies or conflicts – that hold in one model but not in the other. The technique is based on a translation from process models to event structures, a formalism that describes the behavior as a collection of events (task instances) connected by binary behavioral relations. A naïve version of this translation suffers from two limitations. First, it produces redundant difference statements because an event structure describing a process may contain unnecessary event duplications. Second, this translation is not directly applicable to process models with cycles as the corresponding event structure is infinite. To tackle the first issue, the paper proposes a technique for reducing the number of events in an event structure while preserving the behavior. For the second issue, relying on the theory of complete unfolding prefixes, the paper shows how to construct a finite prefix of the unfolding of a possibly cyclic process model where all possible causes of every activity is represented. Additionally, activities that can occur multiple times in an execution of the process are distinguished from those that can occur at most once. The finite prefix thus enables the diagnosis of behavioral differences in terms of activity repetition and causal relations that hold in one model but not in the other. The method is implemented as a prototype that takes as input process models in the Business Process Model and Notation (BPMN) and produces difference statements in natural language. Differences can also be graphically overlaid on the process models.

© 2015 Elsevier Ltd. All rights reserved.

1. Introduction

Large organizations often need to manage multiple variants of the same business process. For example, an order-

* Corresponding author.
E-mail addresses: abel.armas@ut.ee (A. Armas-Cervantes), baldan@math.unipd.it (P. Baldan), marlon.dumas@ut.ee (M. Dumas), luciano.garcia@ut.ee (L. Garcia-Bañuelos).

http://dx.doi.org/10.1016/j.is.2015.09.009
0306-4379/© 2015 Elsevier Ltd. All rights reserved.

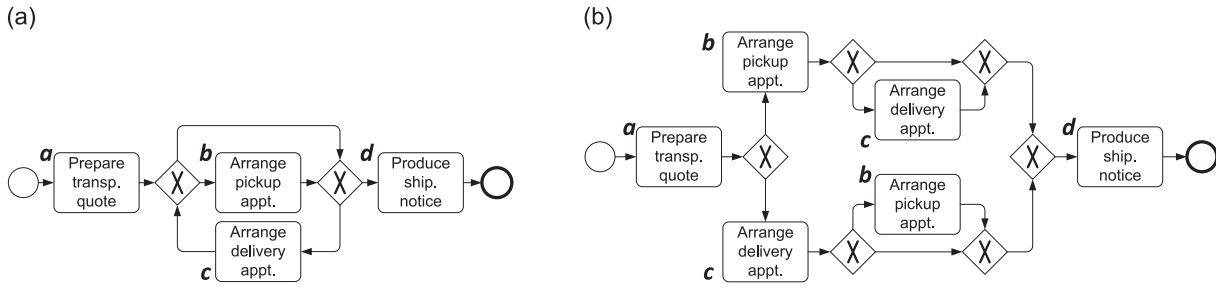

Fig. 1. Variants of business process models (a) M_1 and (b) M_2.

to-cash process may exist in multiple variants, corresponding to different products, different types of customers, different markets in which the company operates, or idiosyncratic choices made by multiple business units over time. During the ongoing management of these processes, analysts need to compare models of multiple process variants [1] in order to identify opportunities for standardization or to understand relative performance differences across variants.

Existing process model comparison methods can be classified into those based on the structure of the models and those based on their behavior. In some cases, a structural comparison – where nodes and/or edges are matched based on the topology of the model – is sufficient to understand the differences between two variants. However, two process models may be structurally different, yet behaviorally equivalent or they may be very similar structurally yet quite different behaviorally, as changes in a few gateways or edges might entail significant behavioral differences.

In this setting, this paper faces the problem of diagnosing behavioral differences between business process models. The paper presents a method to describe differences in terms of binary behavioral relations and activity repetition observed in one process but not in the other. We specifically deal with three elementary types of behavioral relations that, together with repetition, have been postulated as basic control-flow workflow patterns [2], namely causal precedence (corresponding to "sequence" in a process model), conflict (exclusive branches in a process model), and concurrency (parallel branches in a process model). For example, consider the models in BPMN notation in Fig. 1, describing an order fulfillment process, as presented in [3]. We aim at describing their differences via statements of the form: "*In* M_1, *there is a state after* Prepare transportation quote where Arrange delivery appointment can occur before Produce shipment notice *or* Arrange delivery appointment *can be skipped*, *whereas in the matching state in* M_2, Arrange delivery appointment *always occurs before* Produce shipment notice", and "*In* M_1 *activity* Arrange delivery appointment *occurs* 0,1 *or more times*, *whereas in* M_2 *it occurs at most once*".

Throughout the paper we assume that the input process models are given as Petri nets. This design choice enables the application of the presented comparison technique to any process modeling language with a mapping to this formalism. For example, a transformation of a large subset of BPMN to Petri nets can be found in [4].[1] In addition to providing a language-neutral representation, the use of Petri nets allows us to reuse a large body of existing theoretical results, for example the theory of unfolding [6,7].

Given that we focus on describing differences in terms of causality, conflict and concurrency relations, we adopt *event structures* [7] as an abstract representation of processes that explicitly recognizes these three types of relations. Event structures are a well-established model of concurrency where computations are represented as collections of events (activity occurrences) endowed with behavioral relations expressing dependencies between events. Various types of event structures have been introduced in the literature, comprising different binary behavioral relations, such as *prime event structures* [7] (PESs), where events are related by causal dependency and symmetric conflict, and *asymmetric event structures* [8] (AESs), where a form of asymmetric conflict between events is taken as primitive. A representation based on AESs can be more compact than one based on PESs. In fact, in the latter occurrences of the same activity in different contexts are necessarily represented as distinct events, leading to event duplication that is sometimes avoidable in AESs. For the purpose of comparison, more compact representations are desirable as they lead to more concise diagnoses of behavioral relations that exist in one process and not in the other. For this reason, the paper uses AESs as a basis for process model comparison.

In a prior work [9], we proposed a method for behavior-preserving reduction of AESs based on a *quotient operation*, which merges events corresponding to occurrences of the same activity in different contexts while preserving the overall behavior. However, the work in [9] shows that in some cases multiple non-isomorphic "minimal" AESs exist.

In this setting, the contribution of the paper is threefold: (i) we extend our previous work [9], by proposing a deterministic order on the quotient of an AES that leads to a uniquely determined minimal representation of a given acyclic process model; (ii) we propose a method for calculating an error-correcting (partial) synchronized product of two event structures from which differences can be diagnosed at the level of binary behavioral relations that hold in a state of a process model but not in the matching state of the other model; (iii) for cyclic process models, we rely on

[1] This transformation does not cover some BPMN constructs such as OR-joins, which cannot be straightforwardly translated into Petri nets [5].

the theory of unfolding prefixes for determining, for a given process model, a finite structure describing all possible causal dependencies between activities and providing useful information on which activities are repeated and which are not.

This paper is a revised and extended version of the conference paper [10]. The main addition is the definition of the partial synchronized product of two AESs and its application to diagnosing behavioral differences. In [10], the diagnostic is derived from an error-correcting graph matching over the folded AESs which, as explained later, can produce counterintuitive results as it does not take into account the semantics of the relations in the AESs. In particular, it disregards the order of occurrence of activities in the process models. Another addition is a refined description of the technique for identifying and verbalizing differences, which is only sketched in the conference version.

The paper is structured as follows. In Section 2 we discuss some related work. In Section 3 we provide the basics notions for Petri nets and (asymmetric) event structures, as needed in the rest of the paper. In Section 4 we present a technique for determining a canonical AES for an acyclic process model. Then in Section 5 we present the notion of partial synchronized product and discuss how this product allows us to identify behavioral differences between process models. In Section 6 we suggest how the technique can be extended to the analysis of cyclic models. In Section 7 we illustrate a method for verbalizing the behavioral differences and present an evaluation of the corresponding tool on several case studies. Finally, in Section 8 we summarize the contributions and discuss future work.

2. Related work

Approaches for process model comparison can be divided into those based on node label similarity, process structure similarity and behavioral similarity [1]. We remark that node label similarity plays an important role in the alignment of nodes (e.g., tasks) of the process models being compared. In this paper we focus on behavioral similarity, assuming that such an alignment is already determined, i.e., for each node label in one model we are given the corresponding ("equivalent") node label in the other model.

Many equivalence notions have been defined for concurrent systems [11], ranging from trace equivalence (processes are equivalent if they have the same set of traces) to bisimulation equivalence, taking into account also the branching structure of computations, to finer equivalences which consider some concurrency features (processes are equivalent if they have same sets of runs taking into account also concurrency between computational steps). Few methods have been proposed to diagnose differences between processes based on the latter kind of equivalences. The paper [12] presents a technique for deriving equations in a process algebra which characterize the differences between two *labeled transition systems* (LTSs). On one hand, the use of a process algebra rather than a graphical language can make the feedback more difficult to grasp for end users (process analysts, in our context). On the other hand, the technique relies on interleaving bisimulation equivalence and does not take into account the concurrent structure of the process (a process model with concurrency and its "interleaved" version are equivalent). In [13], a method for assessing the difference of LTSs in terms of "edit" operations is presented. However, for the analysts it can be difficult to use such feedback on the LTSs to understand the "reasons" of the behavioral differences, e.g., in terms of behavioral relations that hold in one model and not in the other. Additionally, also in this case concurrency is not taken into account. Analogous remarks apply to [14] that presents a method for diagnosing differences between process models using standard automata theory. The idea here is to identify discrepancies between processes classified according to a predefined taxonomy of possible differences. As pointed out by the author, the set of differences is not guaranteed to be complete, and thus the technique might not report differences between inequivalent processes.

Behavioral profiles (BP) [15] and *causal behavior profiles* [16] are approaches for representing processes using binary relations. They abstract a process using a $n \times n$ matrix, where n is the number of tasks in the process. Each cell contains one out of three relations: *strict order*, *exclusive order* or *interleaving*; plus an additional *co-occurrence* relation in the case of causal behavioral profiles. Both techniques are incomplete as they mishandle several types of constructs, e.g., task skipping (silent transitions), duplicate tasks, and cycles. In this case, two processes can have identical BPs despite not being equated by any classical behavioral equivalence.

Alpha relations [17] are another representation of processes using binary behavioral relations (direct causality, conflict and concurrency), proposed in the context of process mining. Alpha causality however is not transitive (i.e., causality has a localized scope) making alpha relations unsuitable for behavior comparison [18]. Moreover, alpha relations cannot capture so-called "short loops" and hidden tasks (including task skipping). *Relation sets* [19] are a generalization of alpha relations. Instead of one matrix, the authors use k matrices (with a variable k). In each matrix, causality is computed with a different look-ahead. It is shown that 1-look-ahead matrices induce trace equivalence for a restricted family of Petri nets. The authors claim that using k matrices improves accuracy. But it is unclear how a human-readable diagnostic of behavioral differences could be extracted from two sets of k matrices and it is unclear to what notion of equivalence would this diagnostic correspond.

3. Preliminaries

In this section we recall some basics of *Petri nets*, we define their *branching processes* and we introduce *event structures*. These notions will be used throughout the rest of the paper.

3.1. Petri nets

Definition 1 (*Petri net, net system*). A *(Petri) net* is a tuple $N=(P,T,F)$ where P and T are disjoint sets of *places* and *transitions*, respectively, and $F\subseteq(P\times T)\cup(T\times P)$ is the flow relation. A *marking* $M\colon P\to\mathbb{N}_0$ is a function that associates each place $p\in P$ with a natural number (the number of *tokens* in the place). A *net system* $\mathcal{N}=(N,M_0)$ is a Petri net $N=(P,T,F)$ with a fixed *initial marking* M_0.

Hereafter the components of a net system $\mathcal{N}$ will be implicitly named P, T, F and M, possibly with superscripts.

Places and transitions are conjointly referred to as *nodes*. We write ${}^\bullet y=\{x\in P\cup T|(x,y)\in F\}$ and $y^\bullet=\{z\in P\cup T|(y,z)\in F\}$ to denote the *preset* and the *postset* of node y, respectively. By F^+ and F^* we denote the irreflexive and the reflexive transitive closure of F, respectively.

The operational semantics of a net system is defined in terms of markings and transition firings. A marking M *enables* a transition t, denoted as $(N,M)[t\rangle$, if $M(p)>0$ for all $p\in{}^\bullet t$. In this case, the *firing* of t leads to a new marking M', with $M'(p)=M(p)-1$ if $p\in{}^\bullet t\setminus t^\bullet$, $M'(p)=M(p)+1$ if $p\in t^\bullet\setminus{}^\bullet t$, and $M'(p)=M(p)$ otherwise. This is denoted as $M\xrightarrow{t}M'$. A marking M' is said to be *reachable* from M if there exists a sequence of transitions $\sigma=t_1t_2\ldots t_n$ such that $M\xrightarrow{t_1}M_1\xrightarrow{t_2}\ldots\xrightarrow{t_n}M'$. The set of all the markings reachable from a marking M is denoted $[M\rangle$. A marking M is *coverable* if there exist a reachable marking M' such that $M'(p)\geq M(p)$ for every $p\in P$. A marking M is *n-bounded* if every place $p\in P$ contains up to $n\in\mathbb{N}_0$ tokens at M, i.e., $M(p)\leq n$. The net system $\mathcal{N}$ is *n-bounded* if all reachable markings are *n-bounded*. It is called *safe* when it is 1-*bounded*.

Remark. In the paper we restrict to *safe* net systems and we will often identify a safe marking M with the set $\{p\in P|M(p)=1\}$.

Our Petri nets (and net systems) will be labeled, i.e., they will be associated with a function $\lambda\colon T\to\Lambda\cup\{\tau\}$ where Λ is a fixed set of labels and $\tau\notin\Lambda$. A transition t will be called *visible* if $\lambda(x)\neq\tau$, and *silent* otherwise. An example of a labeled net system is provided in Fig. 2. Visible transitions carry their label inside the corresponding rectangle. Silent transitions are marked by a τ located outside the transition.

3.2. Deterministic and branching processes

The partial order semantics of a net system can be formulated in terms of *branching processes* [6],[2] which represent in a tree-like structure several (possibly all) runs of the original net system. A branching process consists of a (branching) occurrence net [7] namely an acyclic net without merging places, i.e., where $|p^\bullet|\leq 1$ for all $p\in P$. Instead, branching places, namely places with several outgoing transitions, representing nondeterministic choices, are allowed. The behavior of an occurrence net can be described in terms of three relations between nodes: *causality*, *concurrency* and *conflict*. Before providing the definition of (branching) occurrence nets and branching process, we introduce such behavioral relations.

Definition 2 (*Behavioral relations*). Let N be a Petri net and let $x,y\in P\cup T$ be two nodes in N. Then

- x is a *cause* of y, denoted $x<_N y$, if $(x,y)\in F^+$. By $\leq_N$ we denote the *reflexive causal* relation F^*.
- x and y are in *conflict*, denoted $x\#_N y$, if $x,y\in T$, $x\neq y$ are distinct transitions such that ${}^\bullet x\cap{}^\bullet y\neq\emptyset$ (*direct conflict*) or there are $x',y'\in T$ such that $x'\#_N y'$ and $x'\leq_N x$, $y'\leq_N y$ (*inherited conflict*).
- x and y are *concurrent*, denoted as $x\|_N y$, if neither $x<_N y$, nor $y<_N x$, nor $x\#_N y$.

When it is clear from the context, the subscript N will be omitted from the behavioral relations.

Definition 3 (*Occurrence net*). A net $N=(P,T,F)$ is an *(branching) occurrence net* when it satisfies

1. $|{}^\bullet p|\leq 1$ for every $p\in P$;
2. $<_N$ is acyclic (hence the causal relation $\leq_N$ is a partial order);
3. the set of causes $\lfloor x\rfloor=\{y\in T|y\leq_N x\}$ is finite for every $x\in P\cup T$;
4. $\#_N$ is irreflexive, i.e., $\neg(x\#_N x)$ for any $x\in P\cup T$.

In an occurrence net, coverability and reachability can be characterized in terms of the behavioral relations. More precisely, say that a set of places $X\subseteq P$ is a *co-set* if $X^2\subseteq\|$, namely for all $p,p'\in X$ it holds $p\|p'$. A *cut* is a maximal co-set with respect to set inclusion. It can be shown that for occurrence nets satisfying suitable finitariness conditions, co-sets are exactly the coverable markings and cuts are the reachable markings. In particular, this holds for occurrence nets which have a finite initial marking and where any reachable marking enables finitely many transitions. Note that these conditions are satisfied by any occurrence net underlying a branching process of a finite Petri net.

We can now introduce unfolding and branching processes of a net system [6,7]. Given a function $f\colon X\to Y$ and a subset $X'\subseteq X$, we write $f(X')$ as a shorthand for $\{f(x)|x\in X'\}$.

Definition 4 (*Unfolding, branching process*). Let $\mathcal{N}=(P,T,F,M_0)$ be a net system. The *unfolding* of $\mathcal{N}$ is the tuple $Unf(\mathcal{N})=(B,E,G,\rho)$ consisting of an occurrence net (B,E,G) and a function $\rho\colon B\cup E\to P\cup T$, generated by the inductive rules in Fig. 3.

We call *branching process* of $\mathcal{N}$ any prefix of the unfolding, i.e., any tuple $\beta=(B',E',G',\rho')$ such that $B'\subseteq B$, $E'\subseteq E$, for any $e\in E'$, $\lfloor e\rfloor\subseteq E'$ and ${}^\bullet e,e^\bullet\subseteq B'$, and G', ρ' are the obvious restriction of G and ρ.

Places and transitions in the unfolding are often referred to as *conditions* and *events*, respectively. According to rule (I), the set $\mathrm{Min}(\beta)$ of minimal elements of $B\cup E$ with respect to causality corresponds to the set of places in the initial marking of $\mathcal{N}$, i.e., $\rho(\mathrm{Min}(\beta))=M_0$. Whenever a co-set in β corresponds through ρ to the preset of some transition t of the original net (meaning that such pre-set is coverable),

[2] Note that in this section, the term *process* refers to a control-flow abstraction of a business process based on a partial order semantics.

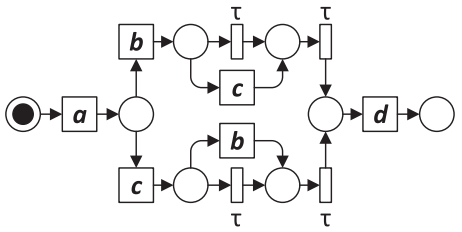

Fig. 2. The net $\mathcal{N}_2$ corresponding to model M_2 in Fig. 1(b).

$$\frac{p \in M_0}{b := \langle \varnothing, p\rangle \in B \quad \rho(b) := p}\ (\mathrm{I}) \qquad \frac{t \in T \quad X \subseteq B \quad X^2 \subseteq \| \quad \rho(X) = {}^\bullet t}{e := \langle X, t\rangle \in E \quad {}^\bullet e := X \quad \rho(e) := t}\ (\mathrm{T1})$$

$$\frac{e = \langle X, t\rangle \in E \quad t^\bullet = \{p_1, \ldots, p_n\}}{b_i := \langle e, p_i\rangle \in B \quad e^\bullet := \{b_1, \ldots, b_n\} \quad \rho(b_i) := p_i}\ (\mathrm{T2})$$

Fig. 3. Branching process, inductive rules.

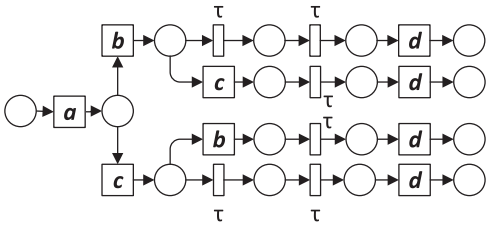

Fig. 4. The unfolding $\mathcal{U}(\mathcal{N}_2)$.

according to rule (T1) the branching process can be extended with an event e representing the corresponding firing of t. Rule (T2) generates the postset of e. As an example, the unfolding of the net system in Fig. 2 is provided in Fig. 4.

Intuitively the unfolding $Unf(\mathcal{N})$ represents any possible behavior of the net system, while a branching processes represents, in general, just some subset of the possible computations.

In a branching process, different occurrences of the same transition are represented as distinct events and the postsets of different events are disjoint. As a result, some nodes in the net system are represented more than once in the branching process. For example, the unfolding in Fig. 4 contains multiple events d which are all instances of a single transition in the net system of Fig. 2.

Partially ordered runs of a net system can be expressed in terms of configurations of a branching process. Roughly a configuration is a set of events that can occur together in a single run. Hence it does not contain events in conflict and for any event it includes all of its causes. Below we formally introduce the notion of a configuration in a branching processes, which will be used in Section 6 to define the complete prefixes of the unfolding.

Definition 5 (*Configuration*). Let $\beta = (B, E, G, \rho)$ be a branching process.

- A *configuration* C of β is a set of events, $C \subseteq E$, which is (i) causally closed, i.e., if $e \in C$ then $\lfloor e \rfloor \subseteq C$, and (ii) conflict free, i.e., for all $e, e' \in C$, $\neg(e \#_\beta e')$. We denote by $Conf(\beta)$ the set of configurations of the branching process β and by $MaxConf(\beta)$ the subset of configurations maximal with respect to set inclusion.
- For an event $e \in E$ of β, its set of causes $\lfloor e \rfloor$ is a configuration called the *local configuration* of e.

Given a branching process β of a net system $\mathcal{N}$, the target cut for a configuration $C \in Conf(\beta)$ is defined as $Cut(C) = (Min(\beta) \cup \bigcup_{e \in C} e^\bullet) \setminus (\bigcup_{e \in C} {}^\bullet e)$. Its image $\rho(Cut(C))$ in $\mathcal{N}$ is a reachable marking that we denote by $Mark(C)$.

As a reference behavioral equivalence in this paper, we use a variation of *pomset equivalence*, which ignores invisible events and is sensible to termination [20,21]. Roughly speaking, it equates systems which can execute the same visible activities, with identical relations of causal dependency and concurrency.

A pomset is a tuple $\langle X, \leq_X, \lambda_X\rangle$, where X is a set of events, $\leq_X$ is a partial order and λ_X is the labeling function. An *isomorphism* between two pomsets X and Y is an isomorphism between the underlying sets, which respects labels and order, i.e., a bijection $f: X \to Y$ such that, $\lambda_X = \lambda_Y \circ f$, and $e <_X e'$ iff $f(e) <_Y f(e')$ for all $e, e' \in X$.

A configuration C can be seen as a pomset with the order and labeling which are the restriction of those of the corresponding net. For this reason we will use C to refer to the configuration or to the corresponding pomset, interchangeably. For a configuration C, we denote by $C^\Lambda = \{e \in C | \lambda(e) \neq \tau\}$ the subset of visible events in C or the corresponding pomset, which is called the *visible pomset* underlying C. Moreover, we denote by $Conf(\beta)^\Lambda$ the set of visible pomsets underlying the configurations of a branching process β, i.e., $Conf(\beta)^\Lambda = \{C^\Lambda : C \in Conf(\beta)\}$. An analogous notation $MaxConf(\beta)^\Lambda$ is used for the maximal configurations.

Definition 6 (*Completed pomset equivalence*). We say that two net systems $\mathcal{N}$ and $\mathcal{N}'$ are *completed (visible) pomset equivalent*, denoted $\mathcal{N} \equiv_{cp} \mathcal{N}'$, whenever $MaxConf(Unf(\mathcal{N}))^\Lambda = MaxConf(Unf(\mathcal{N}'))^\Lambda$.

In words, $\mathcal{N} \equiv_{cp} \mathcal{N}'$ when $\mathcal{N}$ and $\mathcal{N}'$ have maximal configurations, corresponding to completed computations (namely computation which cannot be further extended, because they are either terminated or infinite), which induce the same visible pomsets. This means that the concurrent structure of such computations (causal dependencies and parallelism between visible events) is exactly the same.

When comparing process models, the differences can involve visible and silent events. Commonly, differences determined only by silent events result irrelevant to the user or they are hard to be properly reported. This is the reason why the paper focus on a weak notion of equivalence which abstracts from silent events. We will come back to this point later.

3.3. Event structures

This section introduces two variants of event structures, which are the cornerstones of our comparison technique, *prime event structures* [7] and *asymmetric event structures* [8].

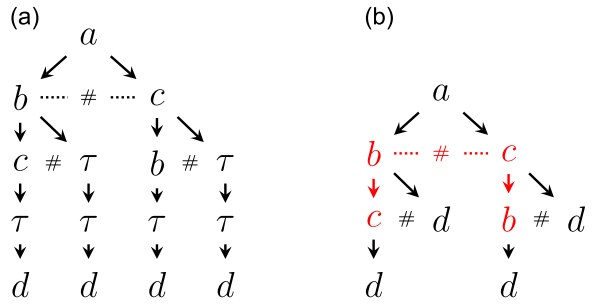

Fig. 5. (a) A PES $\mathbb{P}$ and (b) the corresponding PES without silent events $\mathbb{P}^{\Lambda}$. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

3.3.1. Prime event structures

Prime event structures, introduced in [7], are probably the most widely used event structure model.

Definition 7 (*Prime event structure*). A (labeled) *prime event structure* (PES) is a triple $\mathbb{P}=\langle E, \leq, \#, \lambda\rangle$, where $\leq$ (causality relation) is a partial order on E, $\#$ (conflict relation) is irreflexive, symmetric and hereditary with respect to $\leq$, namely if $e\#e'$ and $e'\leq e''$ then $e\#e''$, for all $e, e', e'' \in E$, and $\lambda: E\to\Lambda\cup\{\tau\}$ is a labeling function. The set of *configurations* $Conf(\mathbb{P})$ and of *maximal configurations* $MaxConf(\mathbb{P})$ of $\mathbb{P}$ are defined exactly as for branching processes.

Given a branching process $\beta=(B,E,G,\rho)$ of a net system $\mathcal{N}$, we can define the corresponding PES in an obvious way, just forgetting the conditions and keeping the events, the relations of causality and conflict over the events, and the labeling function. Fig. 5(a) shows the PES extracted from the unfolding in Fig. 4. Solid arrows represent causality, and annotated dotted lines represent conflict. Two events are concurrent if they are neither in causality nor in conflict relation. For the sake of readability, it is common practice to represent only direct causality, omitting the transitive closure, and direct conflicts, omitting the inherited ones.

As mentioned before, for process comparison we rely on a weak behavioral equivalence, which abstracts from silent events. We can also take a more radical solution which consists in directly removing the silent events from the PES, keeping only the visible events and their dependencies.

Definition 8 (*Pruning silent events in PESs*). For a given PES $\mathbb{P}=\langle E, \leq, \#, \lambda\rangle$, we denote by $\mathbb{P}^{\Lambda}$ the PES having $E'=\{e | e\in E\wedge\lambda(e)\neq\tau\}$ as events, with the natural restrictions of causality, conflict and labeling.

As an example, Fig. 5(b) reports the PES $\mathbb{P}^{\Lambda}$ obtained from the PES $\mathbb{P}$ in the same figure (for the moment ignore the coloring).

This approach can be convenient as it produces a smaller PES which can be further compacted as explained below. However, it must be noted that the transformation is not guaranteed to preserve completed pomset equivalence (while it preserves pomset equivalence). An example can be found in Fig. 6, where $\mathbb{P}_1=\mathbb{P}_0^{\Lambda}$ and $MaxConf(\mathbb{P}_0)=\{\{a,b,c\},\{a,b\}\}$ while $MaxConf(\mathbb{P}_1)=\{a,b,c\}$.

It is easy to see that the problem does not occur if we consider process models where a silent events is never the last event of a computation.

Proposition 1 (*Pruning silent events preserves* $\equiv_{cp}$). *Let* $\mathbb{P}=\langle E, \leq, \#, \lambda\rangle$ *be a PES such that for any* $C\in MaxConf(\mathbb{P})$ *and* $e\in C$ *there is* $e'\in C^{\Lambda}$ *such that* $e\leq e'$. *Then* $\mathbb{P}\equiv_{cp}\mathbb{P}^{\Lambda}$.

Proof. It is immediate to see that, under the hypothesis, for any $C\in MaxConf(\mathbb{P})$ we have $C=\bigcup_{e\in C^{\Lambda}}\lfloor e\rfloor_{\mathbb{P}}\in Conf(\mathbb{P})$ (where $\lfloor e\rfloor_{\mathbb{P}}$ denotes the set of causes of e in $\mathbb{P}$).

Now, if $C\in MaxConf(\mathbb{P})$ in order to conclude that $C^{\Lambda}\in Conf(\mathbb{P}^{\Lambda})$ is maximal, observe that if $C^{\Lambda}\subseteq C'$ for some $C'\in Conf(\mathbb{P}^{\Lambda})$ then, by the above, $C=\bigcup_{e\in C^{\Lambda}}\lfloor e\rfloor_{\mathbb{P}}\subseteq\bigcup_{e\in C'}\lfloor e\rfloor_{\mathbb{P}}$. Since the latter is a configuration in $\mathbb{P}$, by maximality of C we have $C=\bigcup_{e\in C'}\lfloor e\rfloor_{\mathbb{P}}$ hence $C^{\Lambda}=(\bigcup_{e\in C'}\lfloor e\rfloor_{\mathbb{P}})^{\Lambda}=C'$, as desired.

Vice versa, if $C^{\Lambda}\in MaxConf(\mathbb{P})$, in order to conclude that $C\in Conf(\mathbb{P})$ is maximal, observe that if $C\subseteq C_1$ for some $C_1\in Conf(\mathbb{P})$ then $C^{\Lambda}\subseteq C_1^{\Lambda}$. By maximality of C^{Λ} this means that $C^{\Lambda}=C_1^{\Lambda}$. Therefore $C=\bigcup_{e\in C^{\Lambda}}\lfloor e\rfloor_{\mathbb{P}}=\bigcup_{e\in C_1^{\Lambda}}\lfloor e\rfloor_{\mathbb{P}}=C_1$, as desired.□

The rest of the paper is compatible with both approaches, namely, after the translation of a process model to a PES one can either consider the underlying visible PES (if the transformation is known to preserve the behavior) or keep the silent events and ignore them in the later stage of the comparison.

3.3.2. Asymmetric event structures

We now turn our attention to asymmetric event structures, where symmetric conflict is replaced by a possibly non-symmetric relation. Below, for a relation $r\subseteq X\times Y$ and $X'\subseteq X$ we denote by $r_{X'}$ the restriction of r to X', namely $r\cap(X'\times Y)$.

Definition 9 (*Asymmetric event structure*). A (labeled) *asymmetric event structure* (AES) is a tuple $\mathbb{A}=\langle E, \leq, \nearrow, \lambda\rangle$, where E is a set of events, $\leq$ (*causality*) and $\nearrow$ (*asymmetric conflict*) are binary relations on E, and $\lambda: E\to\Lambda\cup\{\tau\}$ is a labeling function, such that $\leq$ is a partial order and $\lfloor e\rfloor=\{e'\in E | e'\leq e\}$ is finite for all $e\in E$, and $\nearrow$ satisfies, for all $e, e', e''\in E$ (1) if $e<e'$ then $e\nearrow e'$; (2) if $e\nearrow e'$ and $e'<e''$ then $e\nearrow e''$; (3) $\nearrow_{\lfloor e\rfloor}$ is acyclic; (4) if $\nearrow_{\lfloor e\rfloor\cup\lfloor e'\rfloor}$ is cyclic then $e\nearrow e'$.

Graphically, causality is still represented by a solid arrow and asymmetric conflict with a dashed arrow. Intuitively, for $e\nearrow e'$ there are two interpretations:

(i) the occurrence of e' *prevents* a subsequent occurrence of e, or
(ii) e *precedes* e' in all computations where both events occur.

By (ii), asymmetric conflict can be seen as a weak form of causality. This is why condition (1) above requires (strict) causality to be included in asymmetric conflict. Asymmetric conflict is inherited along causality, as expressed by (2) if $e\nearrow e'<e''$ then $e\nearrow e''$, since e has to occur necessarily before e'' when they occur in the same computation. Concerning conditions (3) and (4), observe that events forming a cycle of asymmetric conflict cannot appear in the same run, since each event in the cycle should occur before itself in the run. More generally, this holds for events including a cycle of asymmetric conflict in their causes. In this view, condition (3) corresponds to irreflexiveness of conflict in PES, while

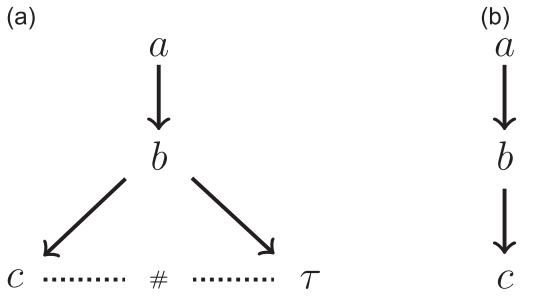

Fig. 6. (a) PES $\mathbb{P}_0$ and (b) its restriction to observable behavior $\mathbb{P}_1$.

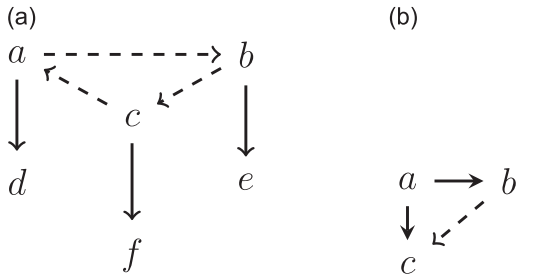

Fig. 7. (a) Inheritance of asymmetric conflict $\mathbb{A}_0$ and (b) Extension relation on configurations $\mathbb{A}_1$.

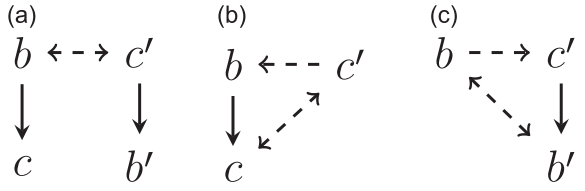

Fig. 8. Equivalent AESs. (a) $\mathbb{A}_2$, (b) $\mathbb{A}_3$ and (c) $\mathbb{A}'_3$.

condition (4) requires that binary symmetric conflict are represented by asymmetric conflict in both directions. For instance, for the AES $\mathbb{A}_0$ in Fig. 7(a) events $\{a,b,c\}$ form a cycle of asymmetric conflict: any computation includes at most two of them. Hence events $\{d,e,f\}$, including $\{a,b,c\}$ in their causes, can never occur together in a computation. Note that, by (3) we need to have $a \nearrow e$, $b \nearrow f$ and $c \nearrow d$. Moreover, by (4) d,e,f are all in asymmetric conflict. All these asymmetric conflicts do not appear in the pictures, since in the graphical representation only direct causality and non-inherited asymmetric conflict are depicted. Moreover causality takes precedence over asymmetric conflict.

Similar to what done for PESs, two events are said *concurrent* when they are neither in causal nor in asymmetric conflict relation.

A *configuration* of an AES $\mathbb{A}$ is a set of events $C \subseteq E$ such that (i) for any $e \in C$, $\lfloor e \rfloor \subseteq C$ (causal closedness); (ii) $\nearrow_C$ is acyclic (conflict freeness). The set of configurations of $\mathbb{A}$ is denoted by $Conf(\mathbb{A})$. Also configurations of AESs will be identified with pomsets by taking as order on events the transitive closure of asymmetric conflict, namely a configuration $C \in Conf(\mathbb{A})$ will be seen as a pomset $\langle C, \nearrow_C^*, \lambda_C \rangle$.

In the case of PESs, given two configurations C, C' such that $C \subseteq C'$, we have that C can be extended by executing the events in $C' \setminus C$ in any order compatible with causality. Hence subset inclusion can be interpreted as an *extension* relation for configurations. This is no longer true for AESs. For instance, consider the AES $\mathbb{A}_1$ in Fig. 7(b). Note that $\{a,b\}$ can evolve to $\{a,b,c\}$, while $\{a,c\}$ cannot because $b \nearrow c$ and thus b cannot occur after c. Given two configurations $C, C' \in Conf(\mathbb{A})$, we say that C' *extends* C, written $C \sqsubseteq C'$, if $C \subseteq C'$ and for all $e \in C_1,\ e' \in C_2 \setminus C_1$, $\neg(e' \nearrow e)$.

Any PES $\mathbb{P} = \langle E, \leq, \#, \lambda \rangle$ can be seen as a special AES by defining asymmetric conflict as $\nearrow = \#$. AESs can provide a more compact representation of a given set of pomsets than PESs. As an example, consider the AESs in Fig. 8 (where events are named by their label, possibly with a superscript). The AES $\mathbb{A}_2$ can be seen as the direct translation of a PES (it represents the colored events and relations in Fig. 5 (b)), and hence it includes event duplications. It is not difficult to see that $\mathbb{A}_3$ and $\mathbb{A}'_3$ are smaller, visible-pomset equivalent versions of $\mathbb{A}_2$. It can be seen that both $\mathbb{A}_3$ and $\mathbb{A}'_3$ are minimal, namely there is no AES representation for the same behavior with a smaller number of events.

4. Canonical representation of acyclic process models

In the previous section we have seen how a business process model, described as a Petri net, can be mapped to a PES, which in turn can be seen as an AES. Here we face the problem of producing a canonical reduced version of a given AES. We exploit some previous results in [9] concerning the possibility of merging distinct events in an AES without altering the behavior. In order to ensure some form of canonicity of the resulting reduced AES we leverage the notion of canonical labeling of graphs. In this way, the reduced AES is uniquely determined by the original model: starting from two isomorphic PESs and repeatedly applying the behavior preserving reduction operation, the resulting minimal AESs are isomorphic.

We restrict to acyclic process models which – via the unfolding construction – generate finite event structures. We will discuss later in Section 6 how cyclic models can be treated.

A basic notion in the reduction technique for AESs in [9] is that of "quotient" of an AES with respect to a set of events. We report it below, suitably adapted it to the needs of this paper.

Definition 10 (*Quotient of an AES*). Let $\mathbb{A} = \langle E, \leq, \nearrow, \lambda \rangle$ be an AES and X be a set of events. The *quotient* of $\mathbb{A}$ with respect to X, denoted $\mathbb{A}_{/X}$, is the structure obtained from $\mathbb{A}$ by replacing the set of events X with a single event e_X, such that $\lambda(e_X)$ is the label of all the events in X, the causes of e_X are $\lfloor e_X \rfloor = \bigcap_{x \in X} \lfloor X \rfloor$ the common causes of events in X and

- $e_X < e$ if $x < e$ for some $x \in X$;
- $e_X \nearrow e$ if $x \nearrow e$ for all $x \in X$;
- $e \nearrow e_X$ if $e \nearrow x$ for all $x \in X$.

The quotient is called *behavior preserving* when $\mathbb{A}_{/X}$ is an AES and $\mathbb{A} \equiv_{cp} \mathbb{A}_{/X}$.

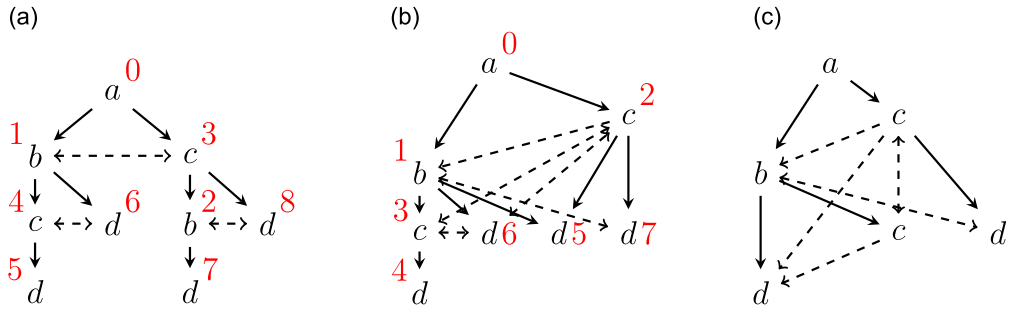

Fig. 9. Canonical labeling and quotient. (a) $\mathbb{A}_4$, (b) $f(\mathbb{A}_4)_{/\{b,b\}}$ and (c) $f^+(\mathbb{A}_4)$.

The paper [9] provides sufficient conditions on the set of events X ensuring that the quotient with respect to X is behavior preserving (actually, in the paper the quotient is shown to preserve history preserving bisimilarity, which is finer than completed pomset equivalence). Roughly, the events in X should represent different occurrences of the same activity in different contexts a fact which is formalized by requiring the events in X to have the same label, be in conflict and have (essentially) the same asymmetric conflicts with the other events. Space limitations prevent from reporting full technical details. Consistently with the terminology of [9], a set of events X inducing a behavior preserving quotient will be referred to as a *combinable* set of events. At every iteration a combinable set of events is chosen for merging, until one reaches a "minimal" AES, where no further quotient step is possible.

Unfortunately, different choices of the sets of events to be merged can lead to different minimal representations. For instance, the AESs $\mathbb{A}_3$ and $\mathbb{A}_3'$ in Fig. 8 can be obtained from $\mathbb{A}_2$ by choosing the combinable sets of events b, b' or c, c', respectively. In both cases, no further reduction is possible and thus they provide minimal representations of the same AES. In this regard, it is necessary to define a deterministic order on the set of combinable events. Unfortunately, features of the sets of events such as size or labels of the events are not enough, since it is possible to have different combinable sets with the same number of events and same labels.

In order to address this problem, we leverage some concepts from graph theory. Specifically we rely on the concept of canonical labeling of a graph [22] that originates as an approach to deciding graph isomorphism. Let $Canon(G)$ be a function that maps a graph G to a *canonical label* in the sense that, given graphs G and H, we have $Canon(G) = Canon(H)$ iff H and G are isomorphic. If we use the string representation of the adjacency matrix of a graph, then a canonical label for a graph G can be determined by computing all permutations of its adjacency matrix and selecting the largest (some authors take the smallest) with respect to lexicographical order. Clearly, this approach is computationally expensive, but state-of-the-art software implement several practical heuristics to compute canonical labels.

Formally, let $G = (V, A)$ be a graph, where V is the set of vertices and A the set of arcs. Moreover, let $M(G)$ be the adjacency matrix of G, in some fixed linear representation. For any order of the set of vertices, represented as a numbering $\gamma: V \to \{0, 1, \ldots |V|\}$, we get a corresponding string $M(G)^\gamma$. Then the canonical label of G is the string induced by an order $\hat{\gamma}$, such that $M(G)^\gamma \leq_{lex} M(G)^{\hat{\gamma}}$ holds for every possible order γ. The order $\hat{\gamma}$ is referred to as the canonical order.

In our implementation, we use *nauty* [23] for computing the graph canonical label and the corresponding order $\hat{\gamma}$ on the vertices which is mostly of interest for us. Nauty and other similar tools work on graphs with unlabeled edges, while AESs can be naturally seen as graphs with labeled edges. The problem is easily overcome by using some isomorphism-preserving transformation of edge-labeled into edge-unlabeled graphs (we used the one in [24]).

The canonical order $\hat{\gamma}$ on the vertices of the graph associated to an AES can be easily used to establish a total order on the possible quotients, thus yielding a minimal and canonical AES for a PES. For a combinable set of events X, we denote by $X^{\hat{\gamma}}$ the ordered string of numbers corresponding to the events in X.

Definition 11 (*Deterministic quotient*). Let $\mathbb{A}$ be an AES, and $\hat{\gamma}: E \to \mathbb{N}_0$ be the canonical order of events given by nauty. Let $X, Y \subseteq E$ be combinable sets of events. Then the precedence of X over Y in a deterministic quotient is defined by the following conditions, listed in decreasing order of relevance:

(i) $\lambda(e) >_{lex} \lambda(e')$ where $e' \in Y$ and $e \in X$, or
(ii) $\lambda(e) =_{lex} \lambda(e') \wedge |X| > |Y|$, or
(iii) $\lambda(e) =_{lex} \lambda(e') \wedge |X| = |Y| \wedge X^{\hat{\gamma}} >_{lex} Y^{\hat{\gamma}}$.

Whenever, applying quotient steps according to such order, we reach an AES where no further reduction steps are possible, this is denoted by $f^+(\mathbb{A})$ and referred to as *minimal canonical quotient*.

Since each quotient step replace two or more events with a single one, starting from a finite AES the length of the sequence of quotient steps will be necessarily bounded (by the number of events), hence a minimal canonical quotient is reached. The fact that the order on quotient steps given in Definition 11 is clearly total, and thus the sequence of steps is essentially deterministic, ensures that the reduction of a finite AES will produce a uniquely determined result.

Proposition 2 (*Canonical quotient of AES*). *Let $\mathbb{A}_1$ and $\mathbb{A}_2$ be isomorphic finite AESs. Then the deterministic quotient of $\mathbb{A}_1$ and $\mathbb{A}_2$ produces a canonical AES, such that $f^+(\mathbb{A}_1)$ is isomorphic to $f^+(\mathbb{A}_2)$.*

Fig. 9 illustrates the canonical quotient of $\mathbb{A}_4$, which corresponds to the PES $\mathbb{P}$ in Fig. 5, and the order $\hat{\gamma}$ as assigned by nauty. The combinable sets of events in $\mathbb{A}_4$ are $\{\{b(1), b(2)\}, \{c(3), c(4)\}, \{d(5), d(6)\}, \{d(7), d(8)\}\}$. According to Definition 11, the set $\{b(1), b(2)\}$ takes precedence over the others (by condition (i)). The AES produced by taking the quotient with respect to $\{b(1), b(2)\}$ is depicted in Fig. 9(b). Note that a fresh event b is added, replacing the set $\{b(1), b(2)\}$, and the order $\hat{\gamma}$ is recalculated for the new AES. The minimal and canonical AES $f^+(\mathbb{A}_4)$ is reported in Fig. 9 (c). In this case, in order to preserve the behavior, we need to keep two events with label c and two with label d.

5. Comparison of acyclic process models

When comparing process models, differences may concern the nature of the involved activities and the way such activities are related. The presence of different activities reduces, at the level of event structures, to the presence of events with different labels, which is easy to detect and describe. Instead, properly diagnosing and reporting differences in the way common activities (i.e., events carrying the same label in both process models) are related in the processes is a more complex problem.

Since an AES can be seen as a labeled graph, the comparison of AESs can be approached by approximate graph matching techniques. This is, in fact, the technique used in [10]. Clearly, if two AESs are diagnosed as isomorphic then they are behaviorally equivalent, hence no difference diagnostic needs to be reported. Moreover, if an error-correcting graph matching is used, the same algorithm would gather information about the differences on event occurrences (process tasks) and mismatching behavioral relations. Given the intuitive interpretation of behavioral relations used by AESs, the verbalization of such differences is straightforward. Unfortunately, a conventional approximate graph matching would not take into account the order induced by the behavioral relations of an AES, as illustrated by the optimal graph matching shown in Fig. 10 between the displayed AESs.

The numbers in brackets in Fig. 10 indicate the optimal matching as computed by an error-correcting graph matching technique on the AESs. Note that the graph matching technique associates c_1''' with c_2'. However, c_1''' causally depends on c_1'', which remains unmatched, and thus the latter would seem a more reasonable matching for c_2'. This suggests the unsuitability of a purely "syntactical" approach and the opportunity of devising a new approach for computing an optimal matching between the events of different AESs, taking into account also the semantics of the relation, i.e., the ordering they induced on events. Clearly, the technique should not only diagnose the similarities but also keep track of the differences found on the input AESs, in the same spirit of the error-correcting graph matching techniques.

In the remainder of this section we present some basic bricks of our approach for comparing process models working on the corresponding AESs: *matching behavior* and *identifying differences*. Here we restrict to acyclic models and thus to finite AESs. A generalization to cyclic models will be discussed in Section 6.

5.1. Matching behavior

The first challenge is to determine how similar the behavior of two given AESs is. As mentioned above, we consider *completed pomset equivalence* as the reference for behavioral equivalence for AESs. If two AESs exhibit different behavior due to differences in the set of event labels or in the underlying behavioral relations, then such differences will emerge in their maximal computations and thus they would not be completed pomset equivalent. In this case, we are interested in finding the best (or at least a good) approximated behavioral matching between the AESs in order to provide to the user an explanation of the behavioral differences.

We start by introducing the concept of partial match between two configurations, which is intended to represent a sort of approximated isomorphism between the corresponding visible-pomsets. Note that the definitions in this section are based on the notion of configuration and extension. Hence, although formulated on AESs, they can be easily adapted to other types of event structures. Below, given a partial function f, we write $f(x) = \perp$ to indicate that f is undefined on x.

Definition 12 (*Partial match*). Let $\mathbb{A}_1$ and $\mathbb{A}_2$ be AESs and let $C_i \in Conf(\mathbb{A}_i)$, for $i \in \{1, 2\}$ be configurations. A *partial match* between C_1 and C_2 is a partial injective function $\xi: C_1 \nrightarrow C_2$, such that for all $e_1, e_1' \in E_1$, with $\xi(e_1), \xi(e_1') \neq \perp$, the following holds:

1. $\lambda_1(e_1) = \lambda_2(\xi(e_1))$;
2. $e_1 \leq_1 e_1'$ iff $\xi(e_1) \leq_2 \xi(e_1')$.

In words, a partial match is a function ξ that establishes a correspondence between events of the two pomsets, respecting labeling and order. It is partial and not necessarily surjective, meaning that some events in C_1 might not correspond to any event in C_2, and vice versa.

A partial match between configurations can be obtained by starting from the initial (empty) configurations and extending, step by step, such configurations by means of the following operations:

1. *matching* (both configurations synchronously evolve by executing events with the same label and causal relations with the past), and
2. *hiding* (only one configuration evolves by executing an event while the other stays idle, hence the event

executed is in a sense "hidden" since it is not included in the match).

Matching and hiding operations can be expressed by the inductive rules in Fig. 11 that, when applied to a partial match $\xi: C_1 \nrightarrow C_2$, produce another partial match involving larger configurations. In the rules, given a partial match $\xi: C_1 \nrightarrow C_2$, we write $\xi[e_1 \to e_2]: (C_1 \cup \{e_1\}) \nrightarrow (C_2 \cup \{e_2\})$ to denote the partial match obtained from ξ, by defining $\xi[e_1 \to e_2](e_1) = e_2$ and $\xi[e_1 \to e_2](e_3) = \xi(e_3)$ for $e_3 \in C_1 \setminus \{e_1\}$. Finally, we write $C \xrightarrow{e}_{\lambda(e)} C \cup \{e\}$ to denote that $C \cup \{e\} \in Conf(\mathbb{A})$, for a configuration $C \in Conf(\mathbb{A})$ and an event $e \notin C$. Note that, in case silent events have not been removed from the AESs during their extraction from the process, the hiding operations are used also to ignore silent events.

We aim at defining a technique for determining partial matches between configurations that minimize the number of hiding operations applied to visible events. Clearly, a partial match built by applying the (match) rule to visible events and the (hide_x) rule to silent events is an isomorphism between the underlying visible pomsets. Therefore, whenever it is possible to establish a correspondence between the maximal configurations of two AESs by using only such rules, we can conclude that the AESs are behaviorally equivalent. In general, the matches between maximal configurations that minimize the number of hidings on visible events intuitively capture the best approximate correspondence between the corresponding visible pomsets (common behavior), highlighting the differences in the form of hiding operations.

Definition 13 (*Optimal partial match*). Let $\mathbb{A}_1$ and $\mathbb{A}_2$ be AESs and let $\xi: C_1 \nrightarrow C_2$ a partial match. The *cost* of ξ is defined as

$$g(\xi) = |C_1^{\Lambda}| + |C_2^{\Lambda}| - |\xi| \cdot 2. \quad (1)$$

A partial match $\xi: C_1 \nrightarrow C_2$ is called *optimal* if it minimizes the cost, namely

$$g(\xi) = \min\{g(\xi') | \xi': C_1 \nrightarrow C_2\}.$$

The cost $g(\xi)$ intuitively expresses the "quality" of the partial match ξ. In fact, notice that $g(\xi)$ is the number of visible events in C_1 and C_2 which does not have a matching event in the other configurations. Hence $g(\xi)$ counts the number of hide operations on visible events needed to construct the partial match. As observed above, when $g(\xi) = 0$, the partial match ξ is actually an isomorphism between the visible pomsets underlying C_1 and C_2. Note that, for any two configurations C_1 and C_2, there is always a trivial partial match (the worst one) consisting of the empty function $\varnothing: C_1 \nrightarrow C_2$.

The partial matches between configurations of two AESs can be collected in what we call a partial synchronized product.

Definition 14 (*Partial synchronized product*). Let $\mathbb{A}_1$ and $\mathbb{A}_2$ be AESs. The *partial synchronized product* is the graph $G = \langle S, T \rangle$ where:

- S is the set of partial matches $\xi: C_1 \nrightarrow C_2$, for $C_i \in Conf(\mathbb{A}_i)$ $(i \in \{1,2\})$;
- T is the set of transitions $(\xi: C_1 \nrightarrow C_2) \xrightarrow{op} (\xi': C_1' \nrightarrow C_2')$ defined by the rules in Fig. 11.

The partial synchronized product is inductively generated by the rules in Fig. 11, starting from an "initial" node $(\varnothing: \varnothing \nrightarrow \varnothing)$ corresponding to the unique partial match for the empty configurations. Note that rules (hide_l) and (hide_r) can increase the cost by one (when the hidden event is visible) or leave it unchanged (when the hidden event is silent), while rule (match) always leaves the cost unchanged. Therefore, whenever $(\xi: C_1 \nrightarrow C_2) \xrightarrow{op} (\xi': C_1' \nrightarrow C_2')$, then $g(\xi) \leq g(\xi')$. This fact, when searching for optimal partial matches allows for some pruning in the generation of the partial synchronized product.

The partial synchronized product obviously contains all optimal matches (as it contains all partial matches). However, the size of a partial synchronized product is exponential, making its full exploration computationally unfeasible.

We adopt a branch an bound approach, more specifically an adaptation of the well-known A^* algorithm [25], in order to build some informative part of the partial synchronized product. The A^* algorithm requires two cost functions: one to evaluate the cost from the root of the state space to a given node, referred to as the function g or past-cost function, and a heuristic function to estimate the distance to the goal state, referred to as the function h or future-cost function. For a partial match $\xi: C_1 \nrightarrow C_2$, we use the cost $g(\xi)$, defined in (1), as the past-cost function, while the future-cost function $h(\xi)$ is defined as

$$h(\xi) = |(\lambda(E_1') \cup \lambda(E_2')) \setminus (\lambda(E_1') \cap \lambda(E_2'))| \quad (2)$$

where for $i \in \{1,2\}$, the set $E_i' = \{e \in E_i | \nexists e' \in C_i: e \nearrow e'\}$ contains the events which can possibly executed in the future of the current configuration C_i. Note that h counts the number of activities (event labels) which are in the future of a configuration and not in the future of the other. It is intended to provide an estimate of the number of events to be hidden in the construction of a partial match of maximal configurations including C_1 an C_2. For this, it optimistically assumes that events with the same label will indeed contribute to a one-to-one match between the two configurations (and unmatched events with the same label are counted only once). It can be seen that this function is admissible in the sense required in [26] for the use of the algorithm A^*.

The pseudo-code for the search algorithm is presented in Algorithm 1. It refers to a function η which is the sum of the past- and future-cost functions, namely $\eta(\xi) = g(\xi) + h(\xi)$ for any partial match $\xi: C_1 \nrightarrow C_2$.

Note that the *A** algorithm is tightly coupled with the

Algorithm
Input: $\mathbb{A}_1 = \langle E_1, \leq_1, \nearrow_1, \lambda_1 \rangle$ and $\mathbb{A}_2 = \langle E_2, \leq_2, \nearrow_2, \lambda_2 \rangle$
Output: Multiset of partial matches for the maximal configurations
`// Initialization`
foreach $C \in MaxConf(\mathbb{A}_1) \cup MaxConf(\mathbb{A}_2)$ **do**
  $GW(C) = \infty$
  $MATCHES[C] = \varnothing$
end
$\xi_0 = \varnothing : \varnothing \nrightarrow \varnothing$
OPEN $\leftarrow \{\xi_0\}$
while $OPEN \neq \varnothing$ **do**
  Choose any $(\xi : C_1 \nrightarrow C_2) \in$ OPEN, with minimum $\eta(\xi)$
    OPEN $\leftarrow$ OPEN $\setminus \{\xi\}$
    `// Pruning`
  if `isCandidate`$(C_1, \xi, \mathbb{A}_1) \vee$ `isCandidate`$(C_2, \xi, \mathbb{A}_2)$ **then**
    `// Best match`
    if $C_1 \in MaxConf(\mathbb{A}_1) \wedge C_2 \in MaxConf(\mathbb{A}_2)$ **then**
      `updateMatches`(C_1, ξ)
      `updateMatches`(C_2, ξ)
    end
    foreach $C_1 \xrightarrow{e_1} C_1', C_2 \xrightarrow{e_2} C_2'$, *s.t.* $\lambda_1(e_1) = \lambda_2(e_2)$ **do**
      if $\xi[e_1 \mapsto e_2]$ *is a partial match (Def. 12)* **then**
        OPEN $\leftarrow$ OPEN $\cup \{(\xi[e_1 \mapsto e_2] : C_1' \nrightarrow C_2')\}$ ▷ MATCH
      end
    end
    foreach $C_1 \xrightarrow{e_1} C_1'$ **do**
      OPEN $\leftarrow$ OPEN $\cup \{(\xi : C_1' \nrightarrow C_2)\}$ ▷ HIDE e_1
    end
    foreach $C_2 \xrightarrow{e_2} C_2'$ **do**
      OPEN $\leftarrow$ OPEN $\cup \{(\xi : C_1 \nrightarrow C_2')\}$ ▷ HIDE e_2
    end
  end
end
return $MATCHES$

Procedure `isCandidate`$(C, \xi, \mathbb{A})$
  return $\exists M \in MaxConf(\mathbb{A}) : C \subset M \wedge \eta(\xi) \leq GW(M)$

Procedure `updateMatches`(C, ξ)
  if $\eta(\xi) \leq GW(C)$ **then**
    $MATCHES[C] \leftarrow MATCHES[C] \cup \{\xi\}$
    $GW[C] \leftarrow \eta(\xi)$
  end

semantics of the underlying AESs in the sense that the match and hide operations are based on the possible extensions of the configurations. Fig. 12 shows two AESs and a part of their partial synchronized product, which contains the optimal matches for the maximal configurations. Observe that, in the partial synchronized product, the fact that two operations can be applied independently leads to diamonds-like shapes in the graph. E.g., in Fig. 12, starting from $\xi = [a_1 \mapsto a_2] : \{a_1\} \nrightarrow \{a_2\}$, we can apply rule (match) to the events labeled b, thus producing the partial match $\xi_1 = \xi[b_1 \mapsto b_2] : \{a_1, b_1\} \nrightarrow \{a_2, b_2\}$, and then apply ($\text{hide}_\text{I}$) to the c-labeled event c_1, thus getting $\xi_1 : \{a_1, b_1, c_1\} \nrightarrow \{a_2, b_2\}$, or vice versa, apply first (hide_I) and then(match).

Algorithm 1. Computing partial matches.

5.2. Identifying differences

The partial synchronized product is a rich structure that represents all possible partial matches (obtained through match and hide operations). Let us assume that some partial matches between maximal configurations have been chosen, possibly optimal or simply good, when determined with some heuristic approach.

We now discuss how the information provided by such matches can be used in order to explain the behavioral differences between two AES models. A first possibility consists in simply verbalizing the hide operations. Note that, differently from the "syntactical" approach based on graph matching algorithms, in this way we capture how early a discrepancy can arise during the execution of the processes. More specifically, the closer a hide operation is to the "initial" node ($\emptyset: \emptyset \nrightarrow \emptyset$), the sooner the discrepancy can occur. For instance, in Fig. 12, consider the hide operations $(\xi = [a_1 \mapsto a_2]: \{a_1\} \nrightarrow \{a_2\}) \xrightarrow{hide(c_1,\_)} (\xi: \{a_1, c_1\} \nrightarrow \{a_2\})$ and $(\xi' = \xi[b_1 \mapsto b_2]: \{a_1, b_1\} \nrightarrow \{a_2, b_2\}) \xrightarrow{hide(c_1,\_)} (\xi': \{a_1, b_1, c_1\} \nrightarrow \{a_2, b_2\})$. If expressed just mentioning the hidden events, they would lead to the same behavioral diagnostic, namely: "*In model* 1, *there is a state where c can occur*, *whereas in the matching state in model* 2, *it cannot*". This explanation of the difference is not very informative. The reason is that we completely ignore the states where the difference emerges, hereafter often referred as the *context* of the difference. On the other hand, if we take fully into account the context and provide a diagnostic statement per each hide operation, then the output is largely redundant and, again, difficult to interpret.

A more abstract explanation of the differences, e.g., in terms of behavioral relations that hold in one process and not in the other, can be convenient and easier to understand for the user. We next present an approach that aims at gathering the differences in the behavioral relations of two AESs models which, in some sense, explain the dis-

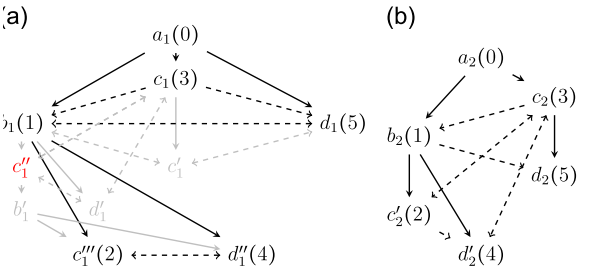

Fig. 10. Event mappings computed with an error-correcting graph matching technique. (a) $\mathbb{A}_5$ and (b) $\mathbb{A}_6$.

$$\frac{C_1 \xrightarrow{e_1}_a C_1' \quad C_2 \xrightarrow{e_2}_a C_2' \quad \xi' = \xi[e_1 \mapsto e_2] : C_1' \nrightarrow C_2' \text{ partial match}}{(\xi : C_1 \nrightarrow C_2) \xrightarrow{match(e_1,e_2)} (\xi' : C_1' \nrightarrow C_2')} \text{ (match)}$$

$$\frac{C_1 \xrightarrow{e_1}_a C_1'}{(\xi : C_1 \nrightarrow C_2) \xrightarrow{hide(e_1,\_)} (\xi : C_1' \nrightarrow C_2)} \text{ (hide}_\text{l}\text{)}$$

$$\frac{C_2 \xrightarrow{e_2}_a C_2'}{(\xi : C_1 \nrightarrow C_2) \xrightarrow{hide(\_,e_2)} (\xi' : C_1 \nrightarrow C_2')} \text{ (hide}_\text{r}\text{)}$$

Fig. 11. Partial matching operations.

(a)

	(a_1, a_2)	(b_1, b_2)
$(c_1, \quad)$	$(<, \quad)$	$(\|\|, \quad)$

(b)

	(a_1, a_2)	(b_1, b_2)
(c_1, c_2)	$(<, <)$	$(\|\|, \#)$

Fig. 13. Matrix for (a) partial match $(\{a_1, b_1, c_1\}, \xi, \{a_2, b_2\})$ and (b) extended partial match $(\{a_1, b_1, c_1\}, \zeta, \{a_2, b_2\})$.

$$\frac{e_1 \in C_1 \quad \zeta(e_1) = \perp = \zeta^{-1}(e_2) \quad \lambda_1(e_1) = \lambda_2(e_2)}{\zeta[e_1 \mapsto e_2]} \text{ (Ext}_\text{l}\text{)}$$

Fig. 14. Extended match rules.

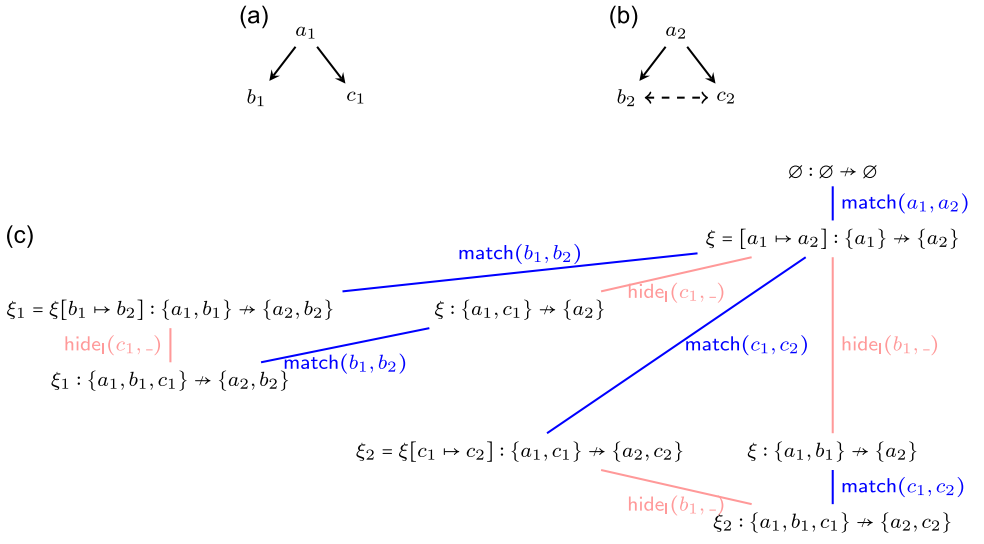

Fig. 12. AESs (a) $\mathbb{A}_a$ (b) $\mathbb{A}_b$ and (c) their partial synchronized product with the optimal matches.

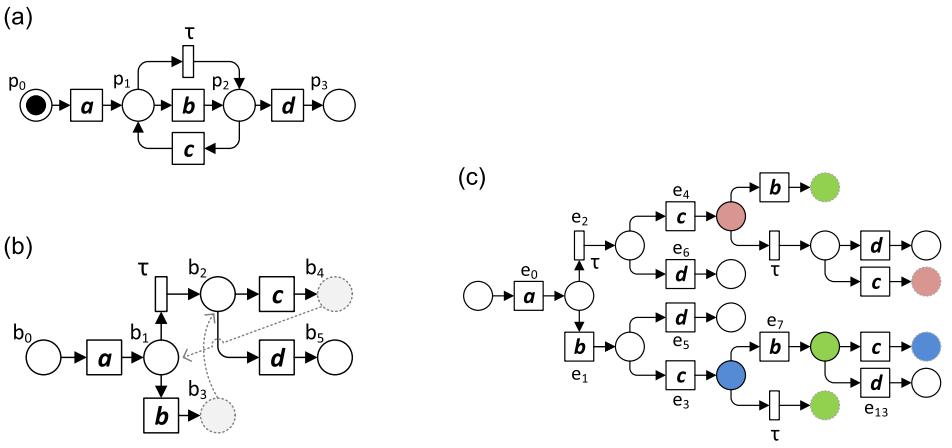

Fig. 15. (a) Petri net $\mathcal{N}_1$ and two different unfoldings (b) β_1 and (c) β_2. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

crepancies emerging (in the form of hide operations) in the partial synchronized product. For instance, the behavioral difference between the AESs in Fig. 12 could be expressed with the single diagnostic statement "*In model* 1, *b and c are in parallel, whereas in model* 2, *b and c are mutually exclusive*".

As a first step, observe that we can view a partial match $\xi: C_1 \nrightarrow C_2$, with $\xi = [a_1 \mapsto a_2, b_1 \mapsto b_2]$, as a matrix of behavioral relations, where columns represent matched events in ξ and the rows represent the hidden (unmatched) events. For instance, the partial match $\xi: \{a_1, b_1, c_1\} \nrightarrow \{a_2, b_2\}$ (Fig. 12) leads to the matrix in Fig. 13(a). The only unmatched event $c_1 \in C_1$ is represented by the row $(c_1, \_)$ that reports that $a_1 < c_1$ and $b_1 \parallel c_1$.

The overall idea in order to diagnose the differences in terms of behavioral relations is the following. Given a partial match $\xi: C_1 \nrightarrow C_2$ between maximal configurations, we extend it as far as possible to the unmatched events by weakening the requirement that the match should respect the order in the pomsets (hence two events e_1 and e_2 can be paired when they have the same label, even if their dependencies with the previously matched events differ) and possibly choosing events outside the configurations. For instance, in Fig. 13(a), the reason why c_1 is unmatched is that there is no event in C_2, with the same label and that relates with a_2 and b_2, in the same way as c_1 relates with a_1 and b_1. In this case, we could match c_1 with c_2, despite the fact that $c_2 \notin C_2$ is in conflict – rather than concurrent – with b_1 (see Fig. 13(b)). Following some heuristics, we will try to match events whose dependencies with past events are as similar as possible.

Definition 15 (*Extended partial match*). Let $\mathbb{A}_1 = \langle E_1, \leq_1, \nearrow_1, \lambda_1 \rangle$ and $\mathbb{A}_2 = \langle E_2, \leq_2, \nearrow_2, \lambda_2 \rangle$ be AESs and let $\xi: C_1 \nrightarrow C_2$ a partial match between configurations C_1 and C_2. A *extended partial match* for ξ is an injective partial function $\zeta: E_1 \nrightarrow E_2$ such that (i) $\xi \subseteq \zeta$, (ii) for any $e_1 \in C_1$ such that $\zeta(e_1) \neq \perp$ it holds $\lambda_2(\zeta(e_1)) = \lambda_1(e_1)$ and (iii) for any $e_1 \in E_1$, if $\zeta(e_1) \neq \perp$ then either $e_1 \in C_1$ or $\zeta(e_1) \in C_2$.

In words, an extension for a partial match ξ is any label-preserving partial function extending ξ. Condition (iii) says that extensions are only allowed when they permit to match some previously unmatched event in C_1 or in C_2. Note that also events outside the configurations can be involved. When we need to match an event in one configuration with an event outside the other, e.g., if for $e_1 \in C_1$ we have $\zeta(e_1) \notin C_2$, it means that $\zeta(e_1)$ – the natural candidate for matching e_1 – cannot be included in C_2 due to some conflict. Hence this situation witnesses some behavioral difference concerning conflicts between activities.

As in the case of partial matches, we introduce some measure of the "quality" of an extension. Intuitively, we try to minimize the number of dependencies on which the matched events differ.

Definition 16 (*Cost of an extended partial match*). Let $\mathbb{A}_1$ and $\mathbb{A}_2$ be AESs and let $\zeta: E_1 \nrightarrow E_2$ be an extended partial match, between configurations C_1 and C_2. The *cost* of ζ is defined as

$$K(\zeta) = |\{(e_1, rel, e_1') : rel \in \{\nearrow^+, <\} \wedge \zeta(e_1) \neq \perp \wedge \zeta(e_1') \neq \perp \wedge \neg(e_1 \, rel \, e_1' \iff \zeta(e_1) \, rel \, \zeta(e_1'))\}|$$

Roughly, the cost $K(\zeta)$ measures the number of events, which are paired in ζ, despite having mismatching relations. Actually, we also count the number of relations on which the events differ. For instance, if we have events a, b with $a < b$ (hence $a \nearrow b$) and it is not the case that $\zeta(a) < \zeta(b)$, then we have two possibilities: either $\zeta(a) \nearrow \zeta(b)$ or not. In the first case, the mismatch increments the cost by 1, in the second by 2.

We are interested in maximal extensions of a partial match, namely extension where all pairs of events with the same labels have been matched, which minimize the cost. If the explicit computation of a maximal extension with least cost is computationally too expensive, one can use a local search criteria, i.e., start from a partial match and add a single pair of events each time, by applying rule (Ext_l) in Fig. 14, or its dual (Ext_r) that is not reported, minimizing the cost at each step.

Consider for example the partial match $\xi: \{a_1, b_1, c_1\} \nrightarrow \{a_2, b_2\}$ (Fig. 12). The corresponding optimal

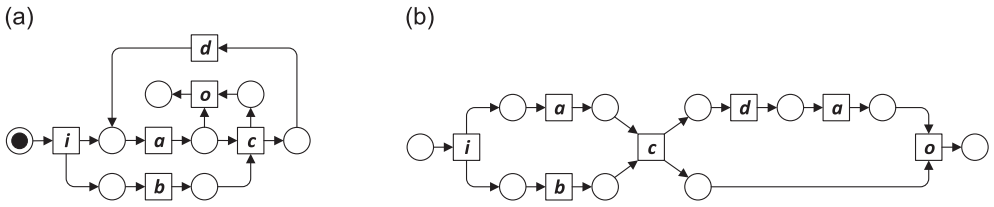

Fig. 16. "Cyclic" net system $\mathcal{N}_3$ and its unfolding $\mathcal{U}(\mathcal{N}_3)$.

maximal extension is $\zeta = \xi[c_1 \mapsto c_2]: \{a_1, b_1, c_1\} \to \{a_2, b_2\}$, shown in Fig. 13(b). The cost is $K(\zeta) = 2$ since $c_2 \# b_2$, hence $c_2 \nearrow b_2$ while it is not the case that $c_1 \nearrow b_1$, and $b_1 < c_1$ while it is not the case that $b_2 < c_2$. In this case, finding ζ is very simple since event c_1 can be only matched with c_2.

In general, there can be several optimal partial matches for a maximal configuration and also several optimal extensions for each partial match, possibly leading to different explanations. In the absence of any other intuitive criteria for distinguishing optimal matches and extensions, we select any of the possible optimal solutions for generating a verbalization of the differences. A description of the verbalization phase, along with a corresponding tool, will be given in Section 7.

6. Finite representation of cyclic process models

A fundamental problem with cyclic process models is that their unfolding is typically infinite. The seminal work in [27], later developed by many authors (see, e.g., [28] and references therein), introduced sophisticated strategies for truncating the unfolding at a finite level, thus obtaining what is called a *complete unfolding prefix (CP)* that provides a representation of any reachable state. The authors in [29] introduced a framework where a canonical unfolding prefix, complete with respect to a chosen property, not limited to reachability, can be constructed. Our own work relies on such a framework for determining a finite fragment of the unfolding providing information about causal dependencies and multiplicity of activities.

6.1. Causal dependencies between activities

Consider the net system $\mathcal{N}_1$ and its unfolding prefix β_1 in Fig. 15, which can be shown to be marking complete, i.e., each marking reachable in $\mathcal{N}_1$ is represented by some cut in β_1. The procedure for computing a marking-complete unfolding prefix consists in applying the inductive rules in Fig. 3, starting from the initial marking and stopping at so-called *cut-off* events, namely events that do not provide new "information" concerning reachability. In our example, it is possible to stop at the cut-off events b and c, generating the conditions b_2 and b_4, respectively, since any other addition to the prefix would duplicate information about reachability and executability already represented. Although this prefix includes a representation of all reachable markings and all executable transitions, it does not include the information that we require to diagnose the behavioral differences of business processes. For instance, the fact that c causally precedes b and d is not explicitly represented in this prefix. In order to obtain a larger prefix that represents explicitly all the causal relations between activities, we will use a stronger cut-off condition. In the case of the net system $\mathcal{N}_1$ in Fig. 15(a), an unfolding prefix complete with respect to this stronger notion is β_2 in Fig. 15(c). The colors of the places are used to indicate the conditions from which the behavior will start repeating, thus suggesting why maximal events are cut-offs (the notion formally introduced below in Definition 17).

As mentioned above, we resort to the notion of cutting context in [29]. A cutting context is a tuple $\Theta = (\approx, \lhd, \mathcal{C})$ where $\approx$ is an equivalence relation over configurations, $\lhd$ is a total order over configurations, and $\mathcal{C}$ is the set of configurations considered for checking the cut-off condition. For example, the cutting context used in [27] is $\Theta_{McM} = (\approx_{mark}, \lhd_{size}, \mathcal{C}_{loc})$, where $\approx_{mark}$ equates two configurations when they produce the same marking, $\lhd_{size}$ is the total order induced by the size of configurations, and $\mathcal{C}_{loc} = \{\lfloor e \rfloor | e \in E\}$ is the set of local configurations. This is the cutting context producing the unfolding prefix β_1, complete for reachability. For instance, if we consider the local configurations $\lfloor c \rfloor = \{a, \tau, c\}$ and $\lfloor a \rfloor = \{a\}$, then one can easily check that $Mark(\lfloor a \rfloor) = Mark(\lfloor c \rfloor) = \{p_1\}$. Moreover, since $|\lfloor a \rfloor| < |\lfloor c \rfloor|$, event c is a cut-off. The cutting context in [30], denoted $\Theta_{ERV} = (\approx_{mark}, \lhd_{slf}, \mathcal{C}_{loc})$, differs from Θ_{McM} for the definition of the partial order $\lhd_{slf}$, which is refined by including action labels in the comparison. This leads to more cut-offs and smaller prefixes (see [30] for details). For our purposes, we consider a cutting context which is a modification of Θ_{ERV} with a refined equivalence relation over configurations taking into account also some information about the history of the current state, namely the labels of the events that produced the current marking. Roughly speaking, each token stores also the labels of the events in its history.

Definition 17 (*h-marking,* $\approx_h$). Let $\mathcal{N} = \langle N, m_0 \rangle$ be a net system, where $N = (P, T, F, \lambda)$ and let $Unf(\mathcal{N}) = (B, E, G, \rho)$ be its unfolding. For a configuration $C \in Conf(Unf(\mathcal{N}))$, we define the *history marking* as

$$hMark(C) = \{\langle \rho(b), \rho(\lfloor b \rfloor^{\Lambda}) \rangle | b \in Cut(C)\}.$$

The configurations $C_1, C_2 \in Conf(Unf(\mathcal{N}))$ are deemed equivalent, written $C_1 \approx_h C_2$, if $hMark(C_1) = hMark(C_2)$. An unfolding prefix $\beta = (B', E', G', \rho')$ is called *h-complete* (complete for $\approx_h$) when for any configuration $C \in Conf(Unf(\mathcal{N}))$ there exists $C' \in Conf(\beta)$ such that $C \approx_h C'$.

We rely on the cutting context $\Theta_h = (\approx_h, \lhd_{slf}, \mathcal{C}_{loc})$. According to the theory in [29], once we have proved that the equivalence $\approx_h$ and the adequate order $\lhd_{slf}$ are preserved by finite configuration extensions, we immediately have an algorithm for constructing a canonical, finite h-complete prefix of the unfolding.

Since our cutting context is a slight variation of that in [30], we can rely on their work for the proof. We only need to prove that $\approx_h$ is preserved by extension. Recall that given a configuration $C \in Conf(Unf(\mathcal{N}))$, a finite set of events V is called a *suffix* of C if $C \cap V = \emptyset$ and $C \cup V \in Unf(\mathcal{N})$.

Proposition 3 ($\approx_h$ *is preserved by extension*). *Let $\mathcal{N} = \langle N, m_0\rangle$ be a net system, where $N = (P, T, F, \lambda)$ and let $Unf(\mathcal{N}) = (B, E, G, \rho)$ be its unfolding. Let $C, C' \in Conf(\beta)$ be configurations such that $C \approx_h C'$. For every suffix V of C, there exists a suffix V' of C' such that*

$$C' \cup V' \approx_h C \cup V.$$

Proof. Let C, C' be configurations such that $C \approx_h C'$ and let V be a suffix of C. We can assume that V consists of a single event, namely $V = \{e\}$. The general case easily follows by an inductive argument. This means that there is a transition t in N such that $\rho(e) = t$ and $Mark(C)[t\rangle$.

According to Definition 17, $hMark(C) = hMark(C')$, which in turn implies that $Mark(C) = Mark(C')$. Hence $Mark(C')[t\rangle$, which implies the existence of an extension $V' = \{e'\}$ of C', where $\rho(e') = t$.

Clearly $Mark(C \cup \{e\}) = Mark(C' \cup \{e'\})$. Moreover, also the fact that $hMark(C \cup \{e\}) = hMark(C' \cup \{e'\})$ is quite immediate. In fact, take any condition $s' \in Cut(C' \cup \{e'\})$. There are two possibilities:

- $s' \in e'^{\bullet}$
  We have that $\lfloor s'\rfloor = \{e\} \cup \bigcup_{s'' \in {}^{\bullet}e'} \lfloor s''\rfloor$. Consider the only condition $s \in e^{\bullet}$ such that $\rho(s') = \rho(s)$. We have that
  $$\rho(\lfloor s'\rfloor^{\Lambda}) = \{\rho(e) | \rho(e) \neq \tau\} \cup \bigcup_{s'' \in {}^{\bullet}e'} \rho(\lfloor s''\rfloor^{\Lambda}) =$$
  $$\{\rho(e) | e \neq \tau\} \cup \bigcup_{s'' \in {}^{\bullet}e} \rho(\lfloor s''\rfloor^{\Lambda}) = \rho(\lfloor s\rfloor^{\Lambda})$$
  where the second equality is motivated by the fact that $\rho(e) = t = \rho(e')$ and $C \approx_h C'$. Therefore $\langle \rho(s), \rho(\lfloor s\rfloor)\rangle = \langle \rho(s'), \rho(\lfloor s'\rfloor)\rangle$.
- $s' \in Cut(C') \backslash {}^{\bullet}e'$
  In this case, if we take the only condition $s \in Cut(C) \backslash {}^{\bullet}e$ such that $\rho(s') = \rho(s)$, since $C \approx_h C'$, we immediately get that $\langle \rho(s), \rho(\lfloor s\rfloor^{\Lambda})\rangle = \langle \rho(s'), \rho(\lfloor s'\rfloor^{\Lambda})\rangle$.

Therefore we conclude that $hMark(C') \subseteq hMark(C)$. Since the argument is perfectly symmetric, we can also deduce the converse inclusion, and thus equality.□

We next prove that an h-complete prefix contains witnesses for all the causal dependencies that would arise in the (possibly infinite) unfolding of a business process with cycles.

Proposition 4 (*Completeness for causal dependencies*). *Let $\mathcal{N}$ be a net system, let $Unf(\mathcal{N}) = (B, E, G, \rho)$ be its unfolding and let $\beta_{\Theta} = (B', E', G', \rho')$ be an h-complete prefix. For any pair of events $e_1, e_2 \in E^{\Lambda}$, if $e_1 < e_2$ then there are $e'_1, e'_2 \in E'$ such that $\rho(e_1) = \rho'(e'_1)$, $\rho(e_2) = \rho'(e'_2)$ and $e'_1 < e'_2$.*

Proof. Let $e_1, e_2 \in E^{\Lambda}$ be events of the unfolding such that $e_1 < e_2$. This means $e_1 \in \lfloor e_2\rfloor$. Consider the configuration $C = \lfloor e_2\rfloor \backslash \{e_2\}$. By completeness there is a configuration C' in the prefix such that $hMark(C) = hMark(C')$. Certainly $Mark(C') = Mark(C)$ enables $\rho(e_2)$ hence C' admits an extension with event e'_2 such that $\rho(e'_2) = \rho(e_2)$. Moreover, since $e_1 < e_2$ there is a condition $s \in {}^{\bullet}e_2 \cap Cut(C)$ such that $e_1 < s$ and thus $\rho(e_1) \in \rho(\lfloor s\rfloor^{\Lambda})$. If we take the only condition $s' \in Cut(C')$ such that $\rho(s) = \rho(s')$, we have that $s' \in {}^{\bullet}e'_2$ and, since $hMark(C) = hMark(C')$, it holds that $\langle \rho(s'), \rho(\lfloor s'\rfloor^{\Lambda})\rangle = \langle \rho(s), \rho(\lfloor s\rfloor^{\Lambda})\rangle$. This means that there is $e'_1 \in \lfloor s'\rfloor$ such that $\rho(e'_1) = \rho(e_1)$. Note that $e'_1 \in \lfloor s'\rfloor$ means $e'_1 < s'$, whence $e'_1 < e'_2$, as desired.□

6.2. Multiplicity of transitions

We now show how a h-complete unfolding prefix can be used in order to deduce information about the multiplicity of each transition in the original net, namely for understanding whether a transition can be executed at most once or possibly more than once. This, in turn, gives information about multiplicity of activities that correspond to transition labels.

As a first step, we observe that, since we deal with safe nets, if a transition occurs occur twice in a configuration, then the corresponding events must be causally related.

Proposition 5 (*Repetition*). *Let $\mathcal{N}$ be a net system and let $C \in Conf(Unf(\mathcal{N}))$ be a configuration such that there exist $e, e' \in C$, $e \neq e'$ and $\rho(e) = \rho(e') = t$. Then either $e < e'$ or $e' < e$.*

Proof. Observe that $e \# e'$ cannot hold, otherwise C would not be a configuration. If we had neither $e < e'$ nor $e' < e$, then e and e' would be concurrent. As a consequence also ${}^{\bullet}e \cup {}^{\bullet}e'$ would be concurrent. Therefore, the corresponding marking in $\mathcal{N}$ would be coverable and it would have two tokens in all places of ${}^{\bullet}t$, contradicting the assumption that $\mathcal{N}$ is safe.□

The above result motivates the interest for the following notion in the study of repetitive behaviors.

Definition 18 (*Self-preceding transitions*). Let $\mathcal{N}$ be a net system and let $Unf(\mathcal{N})$ be its unfolding. We define the set of *self-preceding transitions* of $\mathcal{N}$ as $\mathcal{R}_{\mathcal{N}} = \{\rho(e_1) | \exists\, C \in Conf(Unf(\mathcal{N}))\ .\ e_1, e_2 \in C \wedge \rho(e_1) = \rho(e_2) \wedge e_1 < e_2\}$.

According to Proposition 5, if some transition t of a safe net occurs twice in some computation, then it is necessarily classified as a self-preceding transition in the sense above. Note that this does not mean that it will be repeated in all computations, but just that there is at least a computation where the transition occurs two (or more) times.

In the unfolding we can also single out the transitions that necessarily occur at least once. These are the transitions that occur in the intersection of all maximal configurations.

Definition 19 (*Necessary transitions*). Let $\mathcal{N}$ be a net system. The set of *necessary transitions* of $\mathcal{N}$ is defined as $\mathcal{K}_{\mathcal{N}} = \bigcap \rho(MaxConf(Unf(\mathcal{N})))$.

The possibility of reducing the repetition to a causal dependency ensures that an h-complete prefix will be also sufficient to identify possibly repeated and necessary events.

Proposition 6 (*Necessary and self-preceding transitions in the prefix*). *Let $\mathcal{N}$ be a net system and let $\beta = (B, E, G, \rho)$ be a h-complete prefix of its unfolding. Then*

- $\mathcal{R}_{\mathcal{N}} = \{\rho(e_1) | \exists C \in Conf(\beta). e_1, e_2 \in C \wedge \rho(e_1) = \rho(e_2) \wedge e_1 < e_2\}$
- $\mathcal{K}_{\mathcal{N}} = \bigcap_{\varrho}(MaxConf(Unf(\beta)))$

For instance, consider the h-complete unfolding prefix β_2 in Fig. 15(c). Activity b can be repeated in a computation and indeed we can find a configuration $C = \{e_0, e_1, e_3, e_7, e_{13}\}$ that includes two (causal dependent) occurrences of b. Note also that b does not repeat in all computations. E.g., the maximal configurations $C' = \{e_0, e_2, e_6\}$ include a single occurrence of b, and in $C'' = \{e_0, e_1, e_5\}$ we have that b does not occur at all.

We can thus classify transitions in a net system into three disjoint categories, according to their repetitive behavior.

Definition 20 (*Multiplicity of a transition*). Let $\mathcal{N}$ be a net system. The multiplicity of a transition t in $\mathcal{N}$ is defined as:

- 0..1 (fires at most once) if $t \notin \mathcal{R}_{\mathcal{N}}$;
- $+$ (fires one or more times) if $t \notin \mathcal{R}_{\mathcal{N}} \cap \mathcal{K}_{\mathcal{N}}$;
- $*$ (fires 0 or more times) if $t \notin \mathcal{R}_{\mathcal{N}} \setminus \mathcal{K}_{\mathcal{N}}$.

When a transition is classified as "fires one or more times" or "fires zero or more times" ("$+$" or "$*$"), it just means that there are computations where the transition fires at least twice, but we are not sure that the transition can be repeated an unbounded or a bounded number of times. E.g., consider the net system and its unfolding in Fig. 16. The multiplicity of a is "$+$", but a can occur at most twice in a computation.

Observe that if we are interested in the multiplicity of activities, namely transition labels, rather than of transition themselves (this makes a difference if the labeling is not injective), we need to adapt the definitions above. More precisely, the sets of labels corresponding to $\mathcal{R}_{\mathcal{N}}$ and $\mathcal{K}_{\mathcal{N}}$ above are

- $\Lambda\mathcal{R}_{\mathcal{N}} = \{a \in \Lambda | \exists C \in MaxConf(\beta). |(\lambda \circ \rho)^{-1}(a) \cap C| \geq 2\}$, namely labels that can occur more than once in a computation are those that occur more than once in a maximal configuration of a h-complete prefix (this is, in general, a superset of $\lambda(\mathcal{R}_{\mathcal{N}})$);
- $\Lambda\mathcal{K}_{\mathcal{N}} = \bigcap \lambda(\varrho(MaxConf(\beta)))$, namely labels that necessary appear in a computation are those that occur in any maximal configuration of a h-complete prefix.

6.3. Multiplicity of transitions: the case of free-choice workflow nets

In this section we show that if we restrict to the class of free-choice sound workflow nets, that have been observed to be sufficiently expressive in most situations [31], we can give more precise indications on the repetitive behavior of transitions.

More specifically, we have just seen that transitions which, according to Definition 18, are marked as repetitive, namely either "$+$" or "$*$" can surely occur more than once in a computation, but still they could occur a bounded number of times. Here we show that for sound free-choice workflow nets, a transition which is marked as "$+$" or "$*$", might fire any number of times, namely it is part of a cyclic behavior.

Workflow nets [31] are a class of nets with one single source and sink place such that every transition is on a path from the source to the sink.

Definition 21 (*WF-net, WF-system*). A Petri net $N = (P, T, F, \lambda)$ is a *workflow net* (*WF-net*) if it includes a distinguished *source* place $i \in P$, with $\bullet i = \emptyset$, a distinguished *sink* place $o \in P$, with $o^\bullet = \emptyset$, and the *short-circuit* net $N^* = (P, T \cup \{t^*\}, F \cup \{(o, t^*), (t^*, i)\})$, where $t^* \notin T$, is strongly connected. A net system $\mathcal{N} = (N, M_0)$, where N is a WF-net and $M_0 = \{i\}$, is a *WF-net system*.

Soundness [32] is a commonly adopted criterion of correctness for WF-nets. A sound WF-net system guarantees that any of its executions always ends with a token in the sink place and no other token is left in the net. Recall that a net system $\mathcal{N} = (N, M_0)$ is *live*, if for every reachable marking $M \in [N, M_0\rangle$ and $t \in T$, there exists a marking $M' \in [N, M\rangle$, such that $M'[t\rangle$.

Definition 22 (*Soundness*). A WF-net system $\mathcal{N} = (N, M_0)$ is *sound* when the net system (N^*, M_0), where N^* is the short-circuit net of N, is live and bounded.

Free-choice Petri nets [33,34] are a well-behaved family of nets, where several properties, which are hard to check for general Petri nets, admit efficient verification techniques.

Definition 23 (*Free-choice Petri net*). A Petri net N is free-choice if for any pair of places $p_1, p_2 \in P$ then either $p_1^\bullet \cap p_2^\bullet = \emptyset$ or $p_1^\bullet = p_2^\bullet$.

In words, in a free-choice net whenever two places share a transition it their postsets, they have the same postset. As mentioned before, free-choice WF-nets represent a good compromise between expressiveness and analyzability. In particular, parallelism, sequential routing, conditional routing and iteration can be modeled without violating the free-choice property.

We next show that for the class of (safe) free-choice sound WF-nets, the self-preceding transitions, namely those transitions marked as "$*$" or "$+$" according to Definition 18 represent unbounded repetitive behavior.

We first need a preliminary technical result.

Lemma 7 (*Sequences of firings*). *Let $\mathcal{N}$ be a free-choice sound WF-net system. Let $t_0, \ldots, t_n$ be transitions such that $t_i^\bullet \cap \bullet t_{i+1} \neq \emptyset$ for $i \in \{0, \ldots, n-1\}$ and let M be a marking such that $M[t_0\rangle$. Then there are sequences of transitions $\sigma_i \in T^*$, $i \in \{0, \ldots, n-1\}$, such that $M[t_0\sigma_0 t_1 \sigma_1 \ldots \sigma_{n-1} t_n\rangle$.*

$$\varnothing : \varnothing \nrightarrow \varnothing$$
$$\big| \ \mathsf{match}(a_1, a_2)$$
$$\xi = [a_1 \mapsto a_2] : \{a_1\} \nrightarrow \{a_2\}$$
$$\big| \ \mathsf{hide}_{\mathsf{r}}(\_, c_2)$$
$$\xi : \{a_1\} \nrightarrow \{a_2, c_2\}$$
$$\big| \ \mathsf{match}(d_1, d_2)$$
$$\xi[d_1 \mapsto d_2]\{a_1, d_1\} \nrightarrow \{a_2, c_2, d_2\}$$

Fig. 17. Partial synchronized product for the optimal matching of a pair of configurations in the AESs in Fig. 10.

Table 1
BIT process library.

Library	Number of models	Number of elements		
		Min	Max	Avg
A	152	3	33	12.3
B3	184	3	37	9.1
C	16	8	36	17.3

Proof. The proof is by induction on n. The base case $n=0$ is trivial. Let us assume the result for n and prove it for $n+1$. By inductive hypothesis there are $\sigma_0, \ldots, \sigma_n$ such that $M[t_0\sigma_0 t_1\sigma_1 \ldots \sigma_{n-1} t_n\rangle M_n$. Moreover, by hypothesis, there is at least one place $p \in t_n^\bullet \cap {}^\bullet t_{n+1}$ and we know that $p \in M_n$. Since $\mathcal{N}$ is a sound WF-net, from marking M_n there is a firing sequence which leads to a marking consisting of one token only in the sink place

$M_n[\sigma\rangle\{o\}$.

Since $p \in {}^\bullet t_{n+1}$, surely $p \neq o$. Hence the token in p is consumed by some transition in σ, namely $\sigma = \sigma' t \sigma''$ with $p \in {}^\bullet t$.

Since $\mathcal{N}$ is free-choice, and ${}^\bullet t \cap {}^\bullet t_{n+1} \supseteq \{p\} \neq \varnothing$ we deduce ${}^\bullet t = {}^\bullet t_{n+1}$. Therefore, since $M_n[\sigma' t\rangle$ we also have $M_n[\sigma' t_{n+1}\rangle$. Therefore

$M[t_0\sigma_0 t_1\sigma_1 \ldots \sigma_{n-1} t_n \sigma' t_{n+1}\rangle$

as desired.□

We can now easily conclude with the desired result.

Proposition 8 (*Repetitive behavior*). *Let $\mathcal{N}$ be a free-choice sound WF-net and let t be a transition marked as repetitive ("$*$" or "$+$"). Then there are firings sequences in which transition t fires any number of times.*

Proof. Let t be a transition marked as repetitive ("$*$" or "$+$"). This means that there are events e, e' in $Unf(\mathcal{N})$, such that $\rho(e) = \rho(e') = t \wedge e < e'$. We show that for any marking M, such that $M[t\rangle$, there is a sequence $\sigma \in T^*$ such that $M[t\sigma t\rangle$. From this the result immediately follows.

Since $e < e'$, there must be a causal chain of $e = e_0 < e_1 < \cdots < e_n = e'$ such that $e_i^\bullet \cap {}^\bullet e_{i+1} \neq \varnothing$ for any $i \in \{0, \ldots, n-1\}$. Therefore, if we consider the image through ρ in $\mathcal{N}$, we get corresponding sequence of transitions $\rho(e_0) = t_0 = t$, $\rho(e_1) = t_1$, …, $\rho(e_n) = t_n = t$, with $t_i^\bullet \cap {}^\bullet t_{i+1} \neq \varnothing$ for $i \in \{0, \ldots, n-1\}$.

Table 2
Sizes of PESs and AESs for the BIT process library.

Library	Events PES			Events AES		
	Min	Max	Avg	Min	Max	Avg
A	3	203	18.27	3	203	16.77
B3	3	72	9.08	3	72	8.53
C	6	420	57.93	4	259	36.27

Now, given any marking M such that $M[t\rangle$, we can simply apply Lemma 7, to deduce that there are $\sigma_i \in T^*$, $i \in \{0, \ldots, n-1\}$, such that

$M[t_0\sigma_0 t_1\sigma_1 \ldots \sigma_{n-1} t_n\rangle$.

recalling that $t = t_0 = t_n$ and denoting $\sigma = \sigma_0 t_1\sigma_1 \ldots \sigma_{n-1}$, we get that as desired.□

Again, the theory can be adapted if labeling is not injective and we are interested in the repetition of labels (tasks) rather than transitions. In this case we can distinguish between labels that can occur more than one time in a computation (the class "$*$" defined as before) and the subclass which can occur an unbounded number of times in a computation, defined as $\lambda(\mathcal{R}_\mathcal{N})$.

7. Verbalizing differences

On the basis of the theory developed in the previous two sections, we outline an approach for producing intuitive diagnostics describing the differences found in the comparison of AESs models.

We propose to verbalize each discrepancy by means of a statement consisting of two parts: a description of the *context* where the discrepancy occurs and a description of the *difference* itself.

Recall that we call context, the state in the execution of the process models where a given discrepancy occurs. A full representation of the context consists of a partially ordered set of events (activity executions) leading to the point where the discrepancy is observed. In the case of visual feedback, this can be represented by animating the process model in order to show to the user an execution path leading to the context under consideration. On the other hand, when verbalizing a context in textual form, listing all the events in an execution path leading to a given context is arguably less readable. It might be convenient to report only a partial description of the context, consisting of the last event (i.e., last activity) executed before the state of interest is reached. In the examples given below we opt for this latter (highly abbreviated) verbalization approach for the context. The problem of accurate abbreviation of execution paths leading to a given state in a process model is further studied in [35].

The difference itself is described by referring to either a behavioral relation between events that holds in one model and not in the other, or by stating that the multiplicity of an activity in one model differs from the multiplicity of the same activity in the other model. To this end, a behavioral relation between activity a and b is verbalized as follows:

Table 3
Execution times for computing the minimal canonical AESs of the BIT process library.

Library	Computation time (s)								
	PES			Canonical labeling			Minimal canonical quotient AES		
	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
A	0	2.39	0.06	0.01	0.56	0.02	0	5.93	0.07
B3	0	0.32	0.02	0.01	0.10	0.01	0	1.19	0
C	0.01	52.61	3.67	0.01	2.92	0.36	0	535.61	36.76

Table 4
Land development application process.

Model	Number of BPMN elements
SA 1	37
SA 2	47
SA 3	36
WA 1	28
WA 2	50
WA 3	31

Table 5
Size of event structures of the land development application dataset.

Model	Events	
	PES	**AES**
SA 1	13	13
SA 2	80	80
SA 3	52	30
WA 1	14	14
WA 2	80	80
WA 3	46	23

Table 6
Size of event structures, PESs and AESs, for the land development application process.

Model	Computation time (sec)		
	PES	Canonical numbering	Minimal canonical folded AES
SA 1	0.25	0.05	0.01
SA 2	1.34	0.14	0.44
SA 3	1.19	0.12	0.22
WA 1	0.09	0.03	0
WA 2	0.95	0.08	1.33
WA 3	0.32	0.05	0.54

- Causality ($<$): "*a always occurs before b*".
- Asymmetric conflict ($\nearrow$): "*a can occur before b or a can be skipped*".
- Conflict (#): "*a and b are mutually exclusive*".
- Concurrency ($\|$): "*a and b are parallel*".

The multiplicity of an activity is verbalized as follows:

- 0..1: "*occurs at most once*",
- $+$: "*occurs at least once*", and
- $*$: "*occurs* 0,1 *or more times*".

Table 7
Comparison results. Average time and number of differences.

Model 1	**Model 2**	**Avg. time (s)**		**Differences**	
		PES	AES	PES	AES
SA 1	WA 1	0.16	0.29	23	23
SA 2	WA 2	2.79	5.17	6	6
SA 3	WA 3	98.56	145.52	104	80

Whereas, for safe and sound free-choice workflow nets, the multiplicity of an activity is verbalized as follows:

- $+$: "*occurs any number of times, but at least once*", and
- $*$: "*occurs any number of times*".

Based on the above verbalizations of context, behavioral relations and multiplicity, we use the following templates to verbalize a given discrepancy between two models $M1$ and $M2$:

1. Case of unmatched event: "*In M1, there is a state after* $<\_context\_>$ *where* $<\_activity\_>$ *can occur, whereas it cannot occur in the matching state in M2*";
2. Case of mismatching relations. "*In M1, there is a state after* $<\_context\_>$ *where* $<\_verbalization\ of\ relation\ 1\_>$, *whereas in the matching state in M2,* $<\_verbalization\ for\ relation\ 2\_>$";
3. Case of mismatching multiplicity: "*In M1,* $<\_activity\_>$ $<\_verbalization\ of\ multiplicity\ in\ M1\_>$, *whereas in M2, it* $<\_verbalization\ of\ activity\ multiplicity\ in\ M2\_>$.

For illustration, Fig. 17 shows a fragment of the partial synchronized product including an optimal partial match for the configurations $\{a_1, d_1\}$ and $\{a_2, c_2, d_2\}$ of the AESs of the running example (models and corresponding AESs displayed in Figs. 1 and 10, respectively). The following are the resulting verbalizations of the differences captured by the extended partial match $\zeta = \xi[c_1 \mapsto c_2]: \{a_1, d_1\} \to \{a_2, c_2, d_2\}$ of Fig. 17:

- $c, d = (\nearrow, <)$: *In M1, there is a state after a where c can occur before d or c can be skipped, whereas in the matching state in M2, c always occurs before d*
- $b(*, 0..1)$: *In M1, b occurs any number of times; whereas in M2, it occurs at most once*
- $c(*, 0..1)$: *In M1, c occurs any number of times; whereas in M2, it occurs at most once*

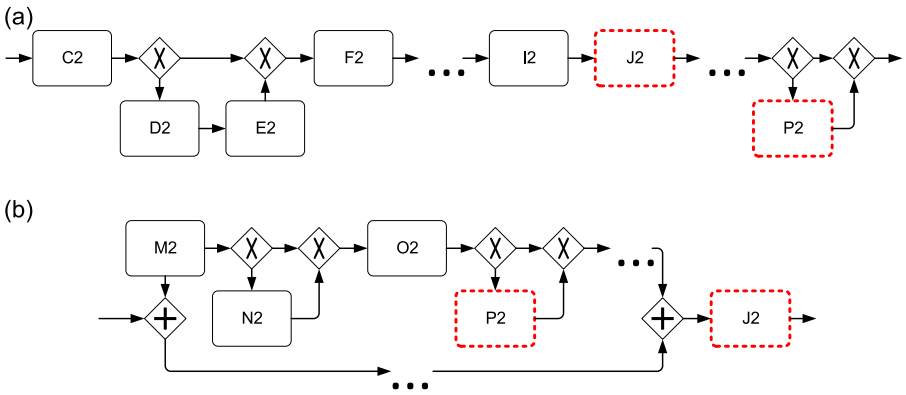

Fig. 18. Snippet of the process models (a) SA 3 and (b) WA 3.

In the case of tasks with repetitive behavior, one event is randomly chosen and the feedback is generated with respect to this event (note that the feedback from other instances would be the same).

There are some cases where differences cannot be expressed as mismatching behavioral relations. For instance, consider the optimal partial match $\xi[a_1 \mapsto a_2, b_1 \mapsto b_2, c_1'' \mapsto c_2', d_1'' \mapsto d_2']$ for the pair of configurations $\{a_1, b_1, c_1'', b_1', d_1''\}$ and $\{a_2, b_2, c_2', d_2'\}$ of $\mathbb{A}_5$ and $\mathbb{A}_6$ (Fig. 10), respectively. In this case the event b_1' had to be hidden, nevertheless there is no other event with label b in $\mathbb{A}_6$ that can be used for extending the partial match ξ, thus we say that b_1' is an *unmatched event*. In this case, we produce the following feedback and include the set of direct causally preceding events to give a context to the feedback. For the running example, the feedback would be:

- $b, =(<,)$: *In M*1, *there is a state after b, where b can occur, whereas it cannot in the matching state in M*2

This latter verbalization illustrates one type of confusion that can arise due to the abbreviation of the context: it is unclear after which occurrence of b is it the case that b always occurs again in $M1$.

7.1. Evaluation

We implemented the ideas presented in the previous sections in a research prototype, called BP-Diff [36]. The tool takes as input pairs of process models expressed in the standard BPMN notation and produces diagnostics of the differences found, both in visual and textual form. The tool is publicly available as a Software-as-a-Service at http://diffbp-bpdiff.rhcloud.com/ and its source code can be found at https://code.google.com/p/fdes/.

7.1.1. Performance

In order to assess the scalability of our method, we measured the performance of BPDiff with respect to one of its more critical steps, namely the computation of canonical reduced AESs. To this end, we used the BIT process library (release 2009), which is a collection of real-life process models from financial services, telecommunications and other domains [37]. From this collection of models we selected the subset of sound models (not all of the models in the repository exhibit this property). The final dataset consists of 352 models, with an average size of 12.4 elements (where events, tasks and gateways counts as elements). More details are given in Table 1.

We computed the canonical reduced AES for each model in the collection five times and averaged the execution time. The tests were run on a laptop computer running Mac OS X with a 2 GHz Intel Core i7 with 4 GB of main memory. The tool is implemented in Java and we used an Oracle Java Virtual Machine 1.7 with 1 GB of maximum heap size.

Three models (two from Library A and one from Library C) were discarded because the computation of the canonical labeling required too much time (more than 8 min). This was due to the large size of the corresponding PESs, which consisted of 566, 779 and 1630 events. Table 2 summarizes the sizes of the resulting PESs and AESs for the remaining process models (150 in Library A, 184 in Library B3 and 15 in Library C). The greatest reduction of the AESs was observed in the process models from the library C, where the average size of the event structures was reduced from 57.93 to 36.27 from the PESs to AESs, accounting for a reduction of 37%.

The minimum, maximum and average execution times for this experiment are reported in Table 3. To better understand the source of overhead, the execution times are split for each of the major phases in the method, namely the computation of the PES (including the computation of the unfolding prefix of the net system), the computation of the canonical labeling of events in the PES, and the computation of the corresponding minimal canonical quotient AES.

The largest execution times were observed on the computation of AESs for Library C. The overhead can be associated with a pair of process models that have a particularly complex topology and that resulted in large PESs (420 events). This in turn induced also a high overhead in the computation of the canonical labeling and in the quotient of the AESs. In spite of the above, the average execution time remains reasonable, of the order of seconds for most of the cases.

7.1.2. Comparison: performance and size of the diagnostics

We conducted a second set of experiments to assess the size of the diagnostics reported to users. To this end, we selected a collection of process models for handling land development applications used by two Australian states,

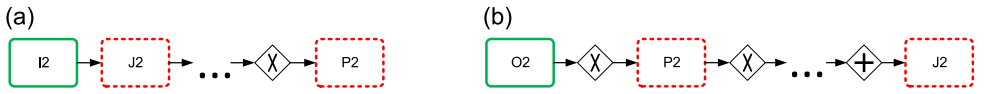

Fig. 19. (a) SA_3^1 and (b) WA_3^2.

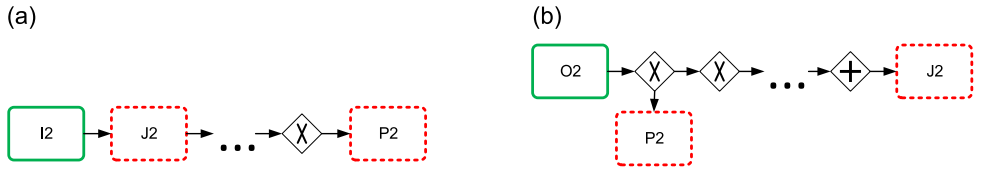

Fig. 20. (a) SA_3^1 and (b) WA_3^3.

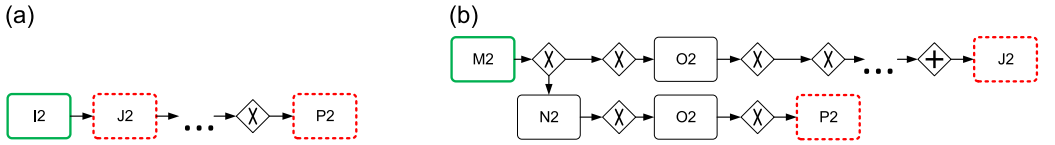

Fig. 21. (a) SA_3^1 and (b) WA_3^4.

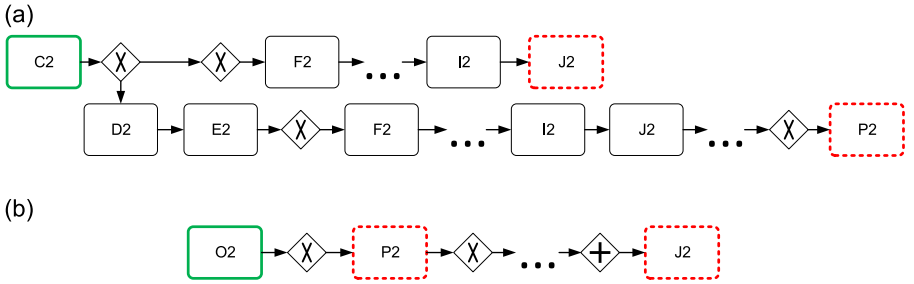

Fig. 22. (a) SA_3^2 and (b) WA_3^2.

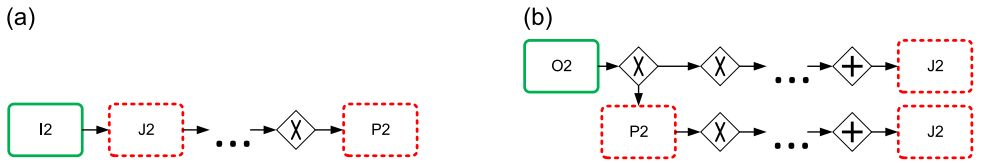

Fig. 23. (a) SA_3^1 and (b) WA_3^1.

namely South Australia (SA) and Western Australia (WA). The collection consists of 3 pairs process models in BPMN notation, each pair corresponding to subprocesses of the whole land development application process from each state. The models use uniform naming conventions, in a way that we can consider that nodes with the same label as referring to the same concrete task. Table 4 presents the size of models in the final dataset.

Table 5 presents the size of the event structures associated to the models in the land development dataset. In this case, the size of the AES remained the same for the first two pairs of models. However, we observe a reduction for the AESs associated to the third case. The execution times for computing the canonical quotient AESs are shown in Table 6.

The execution times for comparison are reported in Table 7. This includes the computation of the partial synchronized product for every pair of models.

The diagnostic of differences found when comparing the third pair of models comprised 104 statements when using PES, which was reduced to 80 statements when using AES. A more detailed analysis of the diagnostics showed that 14 statements generated when comparing the PESs where summarized by 4 statements generated using AESs. Moreover, 10 statements generated from the comparison of PESs were not longer required because the corresponding events were merged in the AESs. All the remaining diagnostic statements were the same for both types of event structures.

To exemplify the kind of visual and textual diagnostic produced by our technique, consider the excerpts of the

process models SA 3 and WA 3, which are presented in Fig. 18. Concretely, we consider the differences involving tasks J2 and P2, which have red dotted borders in the picture. The tool generates a total of 4 statements for explaining this difference, when using PESs. Such differences, both in textual and visual form, are presented below.

1. "*In model* 1, *there is a state after the execution of* I2 *where* J2 *precedes* P2; *whereas in model* 2, *there is a state after the execution of* O2 *where* P2 *precedes* J2" (Fig. 19),
2. "*In model* 1, *there is a state after the execution of* I2 *where* J2 *precedes* P2; *whereas in model* 2, *there is a state after the execution of* O2 *where* J2 *and* P2 *are mutually exclusive*" (Fig. 20),
3. "*In model* 1, *there is a state after the execution of* I2 *where* J2 *precedes* P2; *whereas in model* 2, *there is a state after the execution of* M2 *where* J2 *and* P2 *are mutually exclusive*" (Fig. 21), *and*
4. "*In model* 1, *there is a state after the execution of C2 where* J2 *and* P2 *are mutually exclusive*; *whereas in model* 2, *there is a state after the execution of* O2 *where* P2 *precedes* J2" (Fig. 22).

Conversely, only one statement is produced when using AESs, which would replace the four statements presented above. The statement is:

- "*In model* 1, *there is a state after the execution of* I2 *where* J2 *precedes* P2; *whereas in model* 2, *there is a state after the execution of* O2 *where* P2 *can occur before* J2, *or* P2 *can be skipped*" (Fig. 23).

This gain in compactness clearly stems from the expressive power introduced by the presence of asymmetric conflict in AESs, which allows the technique to merge some duplicated tasks.

8. Conclusion

This paper presented a method for comparing business process models. The behavior of process models is abstracted to asymmetric event structures, which are then compared highlighting differences in the corresponding behavioral relations. The proposed method involves four sub-methods that constitute distinct contributions of the paper:

1. A method for obtaining a canonically reduced AES from an acyclic Petri net.
2. A method for constructing a partial "error-correcting" synchronized product of two finite AESs.
3. A method to compute a finite representation of the behavior of a possibly cyclic Petri net that preserves all causal dependencies and the information on the repetition of activities.
4. A method that, given the AESs extracted from two process models, verbalizes their behavioral differences in terms of activity repetition and binary behavioral relations that hold in one process model but not in the other. This verbalization can be complemented with a visualization of the partial configurations (states) of the input process models where the differences occur.

A direction for future research is the optimization of the proposed method in order to address scalability concerns. For cyclic models, the method involves an unfolding step to calculate an AES, followed by a quotient step to reduce the AES, a calculation of a synchronized product of two event structures, and a traversal of the synchronized product in order to identify behavioral differences and to recast them in terms of behavioral relations. In order to enhance scalability, it might be possible to partially merge some of these steps or to perform some steps incrementally, only inasmuch as needed in order to detect representative differences between process models. Another direction for future work is an extensive empirical usability evaluation of the proposed method, which would provide input to fine-tune the visualizations and templates used to provide the difference diagnostic.

Acknowledgments

This research was supported by the European Social Fund via the Doctoral Studies and Internationalisation Programme (DoRa) and by an institutional grant of the Estonian Research Council and by the project *static ANalysis of COncurrent and REactive software systems (ANCORE)* funded by the University of Padova.

References

[1] R. Dijkman, M. Dumas, B. van Dongen, R. Käärik, J. Mendling, Similarity of business process models: metrics and evaluation, Inf. Syst. 36 (2) (2011) 498–516.
[2] W. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, A.P. Barros, Workflow patterns, in: Distributed and Parallel Databases, 2003, pp. 5–51.
[3] M.L. Rosa, S. Clemens, A.H.M. ter Hofstede, N. Russell, Appendix A, The order fulfillment process model, in: Modern Business Process Automation, Springer-Verlag, Berlin, Heidelberg, 2010.
[4] R. Dijkman, M. Dumas, C. Ouyang, Semantics and analysis of business process models in BPMN, Inf. Softw. Technol. 50 (12) (2008) 1281–1294.
[5] C. Favre, D. Fahland, H. Völzer, The relationship between workflow graphs and free-choice workflow nets, Inf. Syst. 47 (2015) 197–219.
[6] J. Engelfriet, Branching processes of Petri Nets, Acta Inf. 28 (1991) 575–591.
[7] M. Nielsen, G.D. Plotkin, G. Winskel, Petri nets, event structures and domains, Part I, Theoret. Comput. Sci. 13 (1981) 85–108.
[8] P. Baldan, A. Corradini, U. Montanari, Contextual petri nets, asymmetric event structures, and processes, Inf. Comput. 171 (2001) 1–49.
[9] A. Armas-Cervantes, P. Baldan, L. García-Bañuelos, Reduction of Event Structures Under History Preserving Bisimulation, CoRR abs/1403.7181, arxiv.org/abs/1403.7181.
[10] A. Armas-Cervantes, P. Baldan, M. Dumas, L. García-Bañuelos, Behavioral comparison of process models based on canonically reduced event structures, in: BPM, Lecture Notes in Computer Science, vol. 8659, Springer International Publishing, Switzerland, 2014, pp. 267–282.
[11] R. van Glabbeek, U. Goltz, Refinement of actions and equivalence notions for concurrent systems, Acta Inf. 37 (2001) 229–327.
[12] R. Cleaveland, On automatically explaining bisimulation inequivalence, in: CAV, Lecture Notes in Computer Science, vol. 531, Berlin, Heidelberg, 1991, pp. 364–372.

[13] O. Sokolsky, S. Kannan, I. Lee, Simulation-based graph similarity, in: TACAS, Lecture Notes in Computer Science, vol. 3920, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 426–440.
[14] R. Dijkman, Diagnosing differences between business process models, in: BPM, Lecture Notes in Computer Science, vol. 5240, Springer, Berlin, Heidelberg, 2008, pp. 261–277.
[15] M. Weidlich, J. Mendling, M. Weske, Efficient consistency measurement based on behavioral profiles of process models, IEEE TSE 37 (3) (2011) 410–429.
[16] M. Weidlich, A. Polyvyanyy, J. Mendling, M. Weske, Causal behavioural profiles, Fund. Inf. 113 (3–4) (2011) 399–435.
[17] W.M.P. van der Aalst, T. Weijters, L. Maruster, Workflow mining: discovering process models from event logs, IEEE TKDE 16 (9) (2004) 1128–1142.
[18] E. Badouel, On the α-reconstructibility of workflow nets, in: ATPN, Lecture Notes in Computer Science, vol. 7347, Springer, Berlin, Heidelberg, 2012.
[19] M. Weidlich, J. van der Werf, On profiles and footprints-relational semantics for petri nets, in: ATPN, Lecture Notes in Computer Science, vol. 7347, Springer, Berlin, Heidelberg 2012, pp. 148–167.
[20] R. van Glabbeek, U. Goltz, Equivalence notions for concurrent systems and refinement of actions, Acta Inf. 379 (1989) 237–248.
[21] U. Goltz, A. Rensink, Finite Petri nets as models for recursive causal behaviour, Theoret. Comput. Sci. 124 (1994) 169–179.
[22] B.D. McKay, Practical Graph Isomorphism, Department of Computer Science, Vanderbilt University, 1981.
[23] B.D. McKay, A. Piperno, Practical graph isomorphism, {II}, J. Symb. Comput. 60 (2014) 94–112.
[24] G. Kant, Using Canonical Forms for Isomorphism Reduction in Graph-Based Model Checking, Technical Report, CTIT University of Twente, Enschede, July 2010.
[25] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Syst. Sci. Cybern. SSC-4 (2) (1968) 100–107.
[26] R. Dechter, J. Pearl, Generalized Best-first Search Strategies and the Optimality of A*, J. ACM 32 (1985) 505–536.
[27] K.L. McMillan, D.K. Probst, A technique of state space search based on unfolding, Formal Methods Syst.Des. 6 (1) (1995) 45–65.
[28] J. Esparza, K. Heljanko, Unfoldings—A Partial order Approach to Model Checking, EACTS Monographs in Theoretical Computer Science, Springer-Verlag Berlin, Heidelberg, 2008.
[29] V. Khomenko, M. Koutny, W. Vogler, Canonical prefixes of Petri Net unfoldings, Acta Inf. 40 (2) (2003) 95–118.
[30] J. Esparza, S. Römer, W. Vogler, An improvement of McMillan's unfolding algorithm, Formal Methods Syst. Des. 30 (2) (2002) 285–310.
[31] W.M.P. van der Aalst, Verification of workflow nets, in: ICATPN, Springer, Berlin, Heidelberg, 1997, pp. 407–426.
[32] W. van der Aalst, Workflow verification: finding control-flow errors using petri-net-based techniques, in: BPM, Lecture Notes in Computer Science, vol. 1806, Springer-Verlag, London, UK, 2000, pp. 161–183.
[33] E. Best, Structure theory of Petri Nets: the free choice hiatus, in: Petri Nets: Central Models and Their Properties, Lecture Notes in Computer Science, vol. 254, Springer, Berlin, Heidelberg, 1987, pp. 168–205.
[34] J. Desel, J. Esparza, Free Choice Petri Nets, Cambridge University Press, New York, NY, USA, 1995.
[35] N. Lohmann, D. Fahland, did I go wrong?—Explaining errors in business process models, in: BPM, Lecture Notes in Computer Science, vol. 8659, Springer International Publishing, Switzerland, 2014, pp. 283–300.
[36] A. Armas-Cervantes, P. Baldan, M. Dumas, L. García-Bañuelos, BP-Diff: A Tool for Behavioral Comparison of Business Process Models, in: Proceedings of the BPM Demo Session 2014, 2014.
[37] D. Fahland, C. Favre, B. Jobstmann, J. Koehler, N. Lohmann, H. Völzer, K. Wolf, Instantaneous soundness checking of industrial business process models, in: BPM, Lecture Notes in Computer Science, vol. 5709, Springer, Berlin, Heidelberg, 2009, pp. 278–293.