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*

Abstract

We address the problem of detecting and diagnosing behavioral differences between business process models. We rely on a translation from process models into asymmetric event structures (AESs), a formalism for the abstract representation of concurrent processes in terms of events and behavioral relations between events. A naïve version of this translation suffers from two limitations. First, it produces redundant difference diagnostic statements because an AES may contain unnecessary event duplications. Second, it is not applicable to process models with cycles. In order to tackle the first limitation, we rely on a technique for reducing duplication of events in AESs while preserving the behavior. For the second limitation, we propose a method for constructing a finite unfolding prefix and a corresponding AES, which captures all the possible causal dependencies between activities in the given process model. For comparison purposes, exploiting the AESs extracted from the process models, we build a sort of partial synchronized product, easing the identification of behavioral differences which can be possibly expressed in terms of behavioral relations and of repetition behaviors.

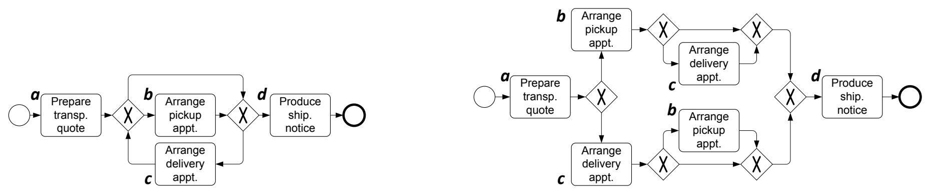
Keywords: Process model comparison, Asymmetric event structures

Email addresses: `abel.armas@ut.ee` (Abel Armas-Cervantes), `baldan@math.unipd.it` (Paolo Baldan), `marlon.dumas@ut.ee` (Marlon Dumas), `luciano.garcia@ut.ee` (Luciano Garcia-Bañuelos)

1. Introduction

Comparing models of business process variants is a basic operation when managing collections of process models [1]. In some cases, syntactic matching of nodes or edges are sufficient to understand differences between two variants. However, two variants may be syntactically different and still be behaviorally equivalent or they may be very similar syntactically but quite different behaviorally, as changes in a few gateways or edges may entail significant behavioral differences.

This paper presents a technique to compare business processes in terms of behavioral relations between tasks. The technique diagnoses differences in the form of binary behavioral relations (e.g., causality and conflict) that hold in one model but not in the other. For example, given the models in Figure 1[1] 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*"*. The diagnosis also considers cyclic behavior, e.g. *"In M_1 activity* Arrange delivery appointment *occurs 0,1 or more times, whereas in M_2 it occurs at most once"*.



(a) M_1 (b) M_2

Figure 1: Variants of business process models

The key idea of the proposal is to compare abstract representations of the input process models based on binary behavioral relations. Then existing behavioral differences between the process models can be expressed as binary relations that hold in one process and not in the other. To this end, process models are abstracted to a well-known model of concurrency, namely *event structures* [3], where computations are represented via events (activity

[1]Based on an order fulfillment process presented in [2].

occurrences) and binary behavioral relations between events. Clearly, if two process models have isomorphic event structures, then they are behaviorally equivalent, since they represent the same behavioral relations between the corresponding events. There are various types of event structures comprising different types of relations, such as *prime event structures* [3] (PESs) and *asymmetric event structures* [4] (AESs). For the purpose of comparison, more compact representations are desirable as they lead to more concise diagnosis of relations that exist in one process and not in the other. In this respect, AESs are more compact than PESs in the sense that the occurrence of the same activity in different contexts (determined by the possibility of task skipping) are necessarily represented as distinct events in PESs, leading to a duplication of events which is possibly avoidable in AESs. In a prior work [5], we proposed a method for behavior-preserving folding (reduction) of AESs based on a so-called *folding operation* which, roughly speaking, merges events corresponding to occurrences of the same activity in different contexts while preserving the behavior. However, the work in [5] shows that in some cases multiple non-isomorphic "minimal" AESs exist that represent the same behavior.

In this setting, the contributions of the paper are threefold: (i) we extend our previous work [5], by proposing a deterministic order on the folding of an AES that leads to a uniquely determined minimal representation of the behavior of a given process model; (ii) we rely on the theory of unfolding prefixes for determining, for a given process model with cycles, a finite structure describing all possible causal dependencies between activities; this gives also information on which activities are repeated and which are not; (iii) 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 repetition or binary behavioral relations that exist in a state of a process model but not in the matching state of the other model.

For the sake of presentation, we assume that the input process models are represented as Petri nets. Transformations from other process modeling notations (e.g., BPMN) to Petri nets are defined elsewhere [6].

This paper is a revised and extended version of [7]. With respect to the conference version, the main extension is the definition of the partial synchronized product of two AESs and its application to diagnosing behavioral differences. In the conference version, the difference diagnostic was derived from an error-correcting graph matching over the folded AESs which, as explained later, can produce scarcely intuitive results as it does not take into account the semantics of the relations in the AESs. In particular, the diagnostic method based on error-correcting graph matching disregards the order of occurrence of activities in the process models. A secondary extension is a refined specification of the method for identifying and verbalizing differences, which was only incompletely sketched in the conference version.

The paper is structured as follows, Section 2 discusses related work. Section 3 provides definitions of notions used in the rest of the paper. The methods for calculating AESs (both for acyclic and cyclic models) are presented in Section 4. Next, Section 5 presents the notion of partial synchronized product and how this product allows us to identify and verbalize the differences between a pair of process models. Finally Section 6 summarizes the contributions and discusses 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) across the process models being compared. In this paper we focus on behavioral similarity, assuming that such an alignment is given, i.e., for each node label in one model we are given the corresponding ("equivalent") node label in the other model.

There are many equivalence notions for concurrent systems [8], ranging from trace equivalence (processes are equivalent if they have the same set of traces) to bisimulation equivalence, to finer equivalences which preserve some concurrency features of computations (two models are equivalent if they have same sets of runs taking into account concurrency between events). Few methods have been proposed to diagnose differences between processes based on these notions of equivalence. The paper [9] presents a technique to derive equations in a process algebra characterizing the differences between two *labeled transition systems* (LTSs). The use of a process algebra makes the feedback difficult to grasp for end users (process analysts in our context) and the technique relies on a notion of equivalence that does not take into account the concurrent structure in the process (a process model with concurrency and its sequential simulation are equivalent). In [10], a method for assessing dissimilarity of LTSs in terms of "edit" operations is presented. However, such feedback on LTSs does not tell the analyst what relations exist in one model that do not exist in the other. Also, it is based on a notion of equivalence that again does take concurrency into account. The same remarks apply to [11], which presents a method for diagnosing differences between pairs of process models using standard automata theory. In addition, in [11] the set of reported differences is not guaranteed to be complete.

Behavioral profiles (BP) [12] and *causal behavior profiles* [13] are two approaches to represent 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 behaviorally equivalent in any standard sense.

Alpha relations [14] 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 [15]. Moreover, alpha relations cannot capture so-called "short loops" and hidden tasks (including task skipping). *Relation sets* [16] 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

This section introduces some fundamental notions on *Petri nets*, *branching processes* and *event structures* that will be used in subsequent parts of the paper.

3.1. Petri nets

Definition 1 (Petri net, net system). A tuple $N = (P, T, F)$ is a *Petri net*, where P is a set of *places*, T is a set of *transitions*, with $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs. A *marking* $M : P \to \mathbb{N}_0$ is a function that

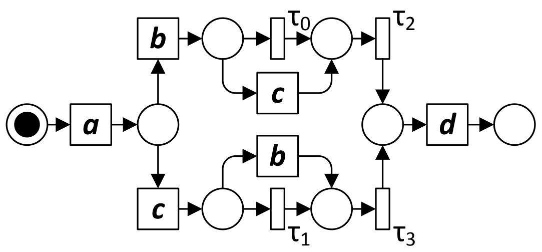


Figure 2: The net $\mathcal{N}_2$ corresponding to model M_2 in Figure 1b

associates each place $p \in P$ with a natural number (viz., place tokens). A *net system* $\mathcal{N} = (N, M_0)$ is a Petri net $N = (P, T, F)$ with an *initial marking* M_0.

Hereafter the components of a net system $\mathcal{N}$ will be implicitly named P, T, F and M_0, possibly with superscripts.

Places and transitions are conjointly referred to as *nodes*. We write ${}^\bullet y = \{x \in P \cup T \mid (x, y) \in F\}$ and $y^\bullet = \{z \in P \cup T \mid (y, z) \in F\}$ to denote the *preset* and *postset* of node y, respectively. By F^+ and F^* we denote the irreflexive and 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 $\forall p \in {}^\bullet t : M(p) > 0$. Moreover, the occurrence 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. We use $M \xrightarrow{t} M'$ to denote the occurrence of t. The marking M_n is said to be reachable from M if there exists a sequence of transitions $\sigma = t_1 t_2 \ldots t_n$ such that $M \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} M_n$. 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 of a net is *safe* if $M(p) \leq 1$ for every place p. A net system $\mathcal{N}$ is said to be *safe* if all its reachable markings are *safe*. In the following we restrict ourselves to *safe* net systems and we will often identify a safe marking M with the set $\{p \in P \mid M(p) = 1\}$.

Our Petri nets (and net systems) will be labeled, i.e., they will be associated with a function $\lambda : T \to \Lambda \cup \{\tau\}$ where Λ is a fixed set of labels. A transition t will be called *visible* if $\lambda(x) \neq \tau$, otherwise x is *silent*. An example of a labeled net system is shown in Figure 2, where the label of visible transitions is inside the corresponding rectangle. Silent transitions are unlabeled (and they possibly have a name τ_i, located outside the corresponding rectangle).

3.2. Deterministic and branching processes

The partial order semantics of a net system can be formulated in terms of runs or, more precisely, prefixes of runs that are referred to as *deterministic processes* [17].[2] A process can be represented as an acyclic net with neither branching nor merging places, i.e., $\forall p \in P : |{}^{\bullet}p| \leq 1 \wedge |p^{\bullet}| \leq 1$. Alternatively, several (possibly all) runs can be accommodated in a single tree-like structure, called *branching process* [3], which can contain branching places and explicitly represents three behavior relations: *causality*, *concurrency* and *conflict* defined as follows.

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

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

We next provide a formal definition of branching process. With a slight abuse of notation, given a function $f : X \to Y$ and a subset $X' \subseteq X$, we write $f(X')$ as a shorthand for $\{f(x) \mid x \in X'\}$.

Definition 3 (branching process). Let $\mathcal{N} = (P, T, F, M_0)$ be a net system. A *branching process* $\beta = (B, E, G, \rho)$ of $\mathcal{N}$ is a net (B, E, G) generated by the inductive rules in Figure 3. The rules also define a function $\rho : B \cup E \to P \cup T$ that maps each node in β to a node in $\mathcal{N}$.

For a branching process $\beta = (B, E, G, \rho)$, places in B and transitions in E are often referred to as conditions and events, respectively. The set $Min(\beta)$ 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., $\rho(Min(\beta)) = M_0$. A *co-set* is a set of conditions $B' \subseteq B$ such that for all $b, b' \in B'$ it holds $b \parallel b'$. A *cut* is a maximal co-set w.r.t. set inclusion.

[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.

$$\frac{p \in M_0}{b := \langle \varnothing, p\rangle \in B \quad \rho(b) := p} \qquad \frac{t \in T \quad B' \subseteq B \quad \rho(B') = {}^\bullet t}{e := \langle B', t\rangle \in E \quad {}^\bullet e := B' \quad \rho(e) := t}$$

$$\frac{e = \langle B', t\rangle \in E \quad t^\bullet = \{p_1, \dots, p_n\}}{b_i := \langle e, p_i\rangle \in B \quad e^\bullet := \{b_1, \dots, b_n\} \quad \rho(b_i) := p_i}$$

Figure 3: Branching process, inductive rules

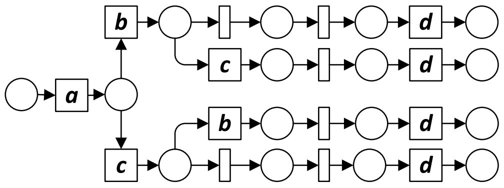


Figure 4: The unfolding $\mathcal{U}(\mathcal{N}_2)$

There exists a largest branching process, representing any possible behavior of the net system $\mathcal{N}$, called the *unfolding* of $\mathcal{N}$ and denoted by $\mathit{Unf}(\mathcal{N})$ [18, 3].

A branching process does not contain merging conditions. As a result, some nodes in the net system need to be represented more than once in the branching process. For example, the branching process in Figure 4 contains multiple instances of b, c and d, and all the events d come from a single transition in the net system in Figure 2.

Definition 4 (configuration and deterministic process). 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., $\forall e' \in E, e \in C : e' \leq_\beta e \Rightarrow e' \in C$, and (ii) conflict free, i.e., $\forall e, e' \in C$, $\neg(e \#_\beta e')$. We denote by $\mathit{Conf}(\beta)$ the set of configurations of the branching process β and by $\mathit{MaxConf}(\beta)$ the subset of maximal configurations w.r.t. set inclusion.
- The *local configuration* of an event $e \in E$ is its set of causes $\lfloor e \rfloor = \{e' \mid e' \leq e\}$. The set of strict causes of an event $e \in E$ is $\lfloor e) = \lfloor e \rfloor \setminus \{e\}$.
- A *deterministic process* $\pi = (B_\pi, E_\pi, G_\pi, \rho)$ is the net induced by a configuration C, where $B_\pi = \bigcup_{c \in C} ({}^\bullet c \cup c^\bullet)$, $E_\pi = C$, and $G_\pi = G \cap (B_\pi \times E_\pi \cup E_\pi \times B_\pi)$.

For a condition $b \in B$ we will write $\lfloor b \rfloor$ as a shorthand for $\lfloor {}^\bullet b \rfloor$.

Given a branching process β of a net system $\mathcal{N}$, the target cut for a configuration $C \in \mathit{Conf}(\beta)$ is defined as $\mathit{Cut}(C) = (\mathit{Min}(\beta) \cup \bigcup_{c \in C} c^\bullet) \backslash (\bigcup_{c \in C} {}^\bullet c)$. Its image in $\mathcal{N}$, $\rho(\mathit{Cut}(C))$, is a reachable marking in $\mathcal{N}$ denoted by $\mathit{Mark}(C)$. Let C and C' be configurations of β, such that $C \subset C'$, and let π and π' be their corresponding deterministic branching processes. If $X = C' \setminus C$, then we write $\pi' = \pi \oplus X$ and we say that π' is an *extension* of π.

Throughout this paper, we use *visible-pomset equivalence* [19] as the notion of behavioral equivalence. 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* of 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 $e <_X e' \Leftrightarrow f(e) <_Y f(e')$ 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 refer as C to the configuration and its corresponding pomset interchangeably. For a configuration C, we denote by $C^\Lambda = \{e \in C \mid \lambda(e) \neq \tau\}$ the subset of visible events in C or the corresponding pomset, which is called the *visible pomset* underlying C. Moreover, we denote by $\mathit{Conf}(\beta)^\Lambda$ the set of visible pomsets underlying the configurations of a branching process β, i.e., $\mathit{Conf}(\beta)^\Lambda = \{C^\Lambda : C \in \mathit{Conf}(\beta)\}$. Armed with the concepts above, we can now formally define visible-pomset equivalence [19].

Definition 5 (visible-pomset equivalence). Let $\mathit{Unf}(\mathcal{N})$ and $\mathit{Unf}(\mathcal{N}')$ be the unfoldings of the net systems $\mathcal{N}$ and $\mathcal{N}'$. Then $\mathcal{N}$ *visible-pomset approximates* $\mathcal{N}'$, written $\mathcal{N} \sqsubseteq_{pt} \mathcal{N}'$ when for every visible-pomset $X^\Lambda \in \mathit{Conf}(\mathit{Unf}(\mathcal{N}))^\Lambda$ there is an isomorphic visible-pomset $Y^\Lambda \in \mathit{Conf}(\mathit{Unf}(\mathcal{N}'))^\Lambda$. We way that $\mathcal{N}$ and $\mathcal{N}'$ are *visible pomset equivalent*, denoted $\mathcal{N} \equiv_{vp} \mathcal{N}'$, if each is $\sqsubseteq_{vp}$ to the other.

3.3. Event structures

This section introduces two variants of event structures, which are the cornerstones of our comparison technique, *prime event structures* [3] and *asymmetric event structures* [4].

Definition 6 (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 w.r.t. $\leq$, namely if $e\#e' \wedge e' \leq e'' \Rightarrow e\#e''$ for all $e, e', e'' \in E$. Finally, $\lambda : E \to \Lambda$ is the labeling function.

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 visible events, and causality, conflict and labeling on events. Figure 5 shows the PES with all the observable behavior of the net system $\mathcal{N}_2$ from Figure 2. Solid arrows represent causality, and annotated dotted lines represent conflict. 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.

$$\begin{tikzcd}
 & & a \arrow[dll] \arrow[drr] & & & & \\
b \arrow[d] \arrow[drr] \arrow[rr, dotted, no head] & & \# \arrow[rr, dotted, no head] & & c \arrow[d] \arrow[drr] & & \\
c \arrow[d] \arrow[r, phantom, "\#"] & {} & d & & b \arrow[d] \arrow[r, phantom, "\#"] & {} & d \\
d & & & & d & &
\end{tikzcd}$$

Figure 5: PES $\mathbb{P}$

The configurations of a PES are defined exactly as for branching process. We will denote the set of configurations of a PES $\mathbb{P}$ by $\mathit{Conf}(\mathbb{P})$.

We now turn our attention to *Asymmetric Event Structures* (AESs).

Definition 7 (asymmetric event structure). An AES is a triple $\mathbb{A} = \langle E, \leq, \nearrow, \lambda \rangle$, where E represents the set of events, $\leq$ is the causality relation, $\nearrow$ is the asymmetric conflict relation and $\lambda : E \to \Lambda \cup \{\tau\}$ is the labeling function. Moreover, for all $e, e', e'' \in E$ the following holds: (1) $\lfloor e \rfloor = \{e' \mid e' \leq e\}$ is finite, (2) $e < e' \Rightarrow e \nearrow e'$, (3) if $e \nearrow e'$ and $e' < e''$ then $e \nearrow e''$, (4) $\nearrow|_{\lfloor e \rfloor}$ is acyclic, (5) if $\nearrow|_{\lfloor e \rfloor \cup \lfloor e' \rfloor}$ is cyclic then $e \nearrow e'$. We will write $\Psi_{\mathbb{A}}$ to refer to the pair $(<, \nearrow)$ of behavior relations of $\mathbb{A}$.

AESs have two relations: causality, with the same interpretation as in PES, and asymmetric conflict, which is an asymmetric version of the conflict in PES. Graphically, causality is represented by a solid arrow and asymmetric conflict with a dashed arrow. Intuitively, for $a \nearrow b$ there are two interpretations: (i) the occurrence of b *prevents* the occurrence of a, or (ii) a *precedes* b in all computations where both events occur. By (ii), asymmetric conflict can be seen as a weak form of causality. Similarly to what done for PESs, two events are said *concurrent* when they are neither in causal nor in asymmetric conflict relation. In the graphical representation only non transitive relations are depicted, either causality or asymmetric conflict, and causality takes precedence over asymmetric conflict.

Definition 7 expresses different properties of the asymmetric conflict relation. Specifically, as mentioned in the intuition above, asymmetric conflict is a weak form of causality and then $\nearrow$ includes $<$, see point (2) in Definition 7. Asymmetric conflict is inherited along causality, point (3) in Definition 7, and if $a \nearrow b < c$ then $a \nearrow c$, since a has to occur necessarily before c when they occur in the same computation. Cycles of asymmetric conflict define conflict over events, i.e., events forming a cycle of $\nearrow$ cannot appear in the same computation since they have to occur before themselves, see points (3) and (4). The notion of conflict over sets of events $\#X$ in AESs is defined by the following rules

$$\frac{e_0 \nearrow e_1 \nearrow \ldots \nearrow e_n \nearrow e_0}{\#\{e_0, \ldots, e_n\}} \qquad\qquad \frac{\#(X \cup \{e\}) \;\; e \leq e'}{\#(X \cup \{e'\})}$$

The first rule captures the fact that events in a cycle of asymmetric conflict cannot occur in the same computation. The second rule expresses inheritance of conflict with respect to causality: if events in the set $X \cup \{e\}$ cannot occur in the same computation and $e \leq e'$, then also events in $X \cup \{e'\}$ cannot occur in the same computation. The reason is that the presence of e' requires the prior occurrence of e. Figure 6 shows an example where $\#\{a, b, c\}$ by the first rule of conflict over sets and, by the second rule, applied three times, we deduce $\#\{a', b', c'\}$. Note that the second rule is essential: in fact, by Definition 7(2) we have that $c \nearrow a'$, $a \nearrow b'$ and $b \nearrow c'$, but events a', b', c' are not in a cycle of asymmetric conflict, hence the first rule would be insufficient to prove $\#\{a', b', c'\}$.

$$\begin{tikzcd}
a \arrow[rr, dashed] \arrow[dd] & & b \arrow[dl, dashed] \arrow[dd] \\
& c \arrow[ul, dashed] \arrow[dd] & \\
a' & & b' \\
& c' &
\end{tikzcd}$$

Figure 6: Inheritance of conflict along causality in AESs.

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 free). The set of configurations of $\mathbb{A}$ is denoted by $\mathit{Conf}(\mathbb{A})$. Also configurations of AESs will be identified with pomsets taking as order on events the transitive closure of asymmetric conflict, namely a configuration $C \in \mathit{Conf}(\mathbb{A})$ will be associated with a pomset $\langle C, \nearrow_C^*, \lambda_C \rangle$, where $\nearrow_C$ and λ_C are the restriction of the asymmetric conflict relation and of the labeling to events in C.

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 *configuration extension* relation. This is no longer true for AESs. For instance, consider the AESs in Figure 7. Note that $\{a, b\}$ can evolve to $\{a, b, c\}$, while $\{a, c\}$ cannot because the occurrence of c prevents that of b. Given, if $C, C' \in Conf(\mathbb{A})$ configurations, we say that C' extends C, written $C \sqsubseteq C'$, if $C \subseteq C'$ and for all $e \in C_1$, $e' \in C_2 \setminus C_1$, $\neg(e' \nearrow e)$.

Clearly any PES can be seen as a special AES where the conflict relation is replaced with asymmetric conflict relations in both directions. In general, as already mentioned, AESs are more expressive than PESs and they can provide a more compact representation of a given set of configurations. As an example, consider the AESs in Figure 8. The AES $\mathbb{A}_1$ can be seen as the direct translation of a PES, hence including event duplications. Instead, $\mathbb{A}_2$ and $\mathbb{A}_3$ are smaller, visible-pomset equivalent versions of $\mathbb{A}_1$. In some sense, both $\mathbb{A}_2$ and $\mathbb{A}_3$ are minimal, namely there is no smaller AES representation for the same behavior.

$$\begin{tikzcd}
a \arrow[r] \arrow[d] & b \arrow[dl, dashed] \\
c &
\end{tikzcd}$$

Figure 7: $\mathbb{A}_0$

4. Canonical folding of process models

This section consists of two parts. The first part addresses the problem of finding a way of producing a canonical reduced version of a given AES, by leveraging the notion of canonical labeling of graphs. The second part extends the method to support the comparison of process models that possibly include cycles.

$$\begin{tikzcd}
b \arrow[r, dashed, leftrightarrow] \arrow[d] & c' \arrow[d] \\
c & b'
\end{tikzcd}$$

(a) $\mathbb{A}_1$

$$\begin{tikzcd}
b \arrow[d] & c' \arrow[l, dashed] \arrow[dl, dashed, leftrightarrow] \\
c &
\end{tikzcd}$$

(b) $\mathbb{A}_2$

$$\begin{tikzcd}
b \arrow[r, dashed] \arrow[dr, dashed, leftrightarrow] & c' \arrow[d] \\
& b'
\end{tikzcd}$$

(c) $\mathbb{A}_3$

Figure 8: Equivalent AESs

4.1. Canonical representation of acyclic process models

In order to exploit the reduced event structure model for comparison purposes the result of the reduction should uniquely determined from the original model. In other words, starting from two isomorphic PESs and repeatedly applying the behavior preserving folding operation, the resulting minimal AESs should be isomorphic.

Space limitations prevents from reporting the details of the reduction technique for AESs introduced in [5]. Briefly, given a business process model, one consider the underlying PES. This can be seen as a special AES, from which the reduction starts. The basic idea is that of identifying sets of events which intuitively represent different instances of the same activity, with the property that such events can be merged without modifying the behavior represented by the AES. A set X of events with this property in an AES $\mathbb{A}$ is called a set of *combinable events* and merging operation is called *folding*. The AES resulting from $\mathbb{A}$ by folding the set of events X is denoted $\mathbb{A}_{/X}$. In [5] it is shown that the folding preserves visible-pomset equivalence (actually it preserves history preserving bisimilarity, which is a stricter behavioral equivalence). At every iteration a combinable set of events is chosen for folding, until one reaches a "minimal" AES, where no further folding steps are possible.

Unfortunately, different choices of the sets of events to be folded can lead to different minimal representations. For instance, the AESs $\mathbb{A}_2$ and $\mathbb{A}_3$ in Figure 8 can be obtained from $\mathbb{A}_1$ by folding events b, b' or c, c', respectively. They are not further foldable and thus they provide minimal representations of the same AES.

In order to address this problem, we leverage some concepts from graph theory. More specifically we rely on the concept of canonical labeling of a graph [20], 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) lexicographical exemplar among them. 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 $\gamma : V \to \{0, 1, ...|V|\}$, we get a corresponding string $M(G)^{\gamma}$. Then the canonical label of G is the string induced by an order $\hat{\gamma}$, s.t., $M(G)^{\gamma} \leq_{lex} M(G)^{\hat{\gamma}}$ holds for every possible order γ. The order $\hat{\gamma}$ is referred to as the canonical order.

In our implementation, we use **nauty** (`http://pallini.di.uniroma1.it/`) 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 [21]).

The canonical order on the vertices of the graph associated to an AES can be easily used to establish a total order on the folding that yields 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 8 (deterministic folding). 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 folding is defined by the following conditions, listed in decreasing relevance:

(i) $\lambda(e) >_{lex} \lambda(e')$ where $e' \in Y$ and $e \in X$, or

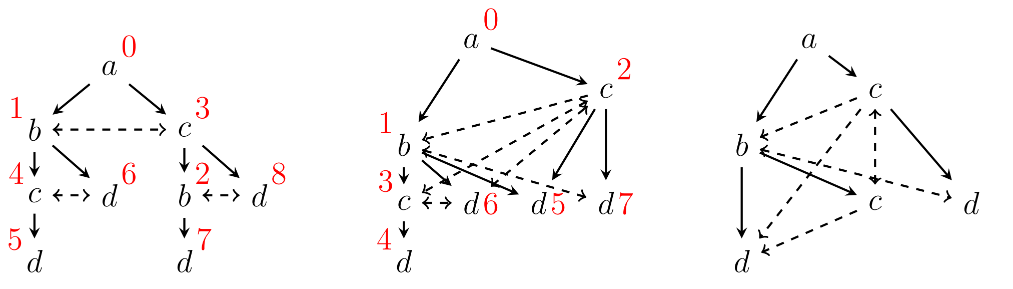
(ii) $\lambda(e) =_{lex} \lambda(e') \wedge |X| > |Y|$, or

(iii) $\lambda(e) =_{lex} \lambda(e') \wedge |X| = |Y| \wedge X^{\hat{\gamma}} >_{lex} Y^{\hat{\gamma}}$.

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

Figure 9 illustrates the canonical folding of $\mathbb{A}_4$, which corresponds to the PES $\mathbb{P}$ in Figure 5. $\mathbb{A}_4$ shows the order $\hat{\gamma}$ assigned by nauty. The combinable sets of events in $\mathbb{A}_4$ are $\{\{b(1), b(2)\}, \{c(3), c(4)\}, \{d(5), d(6)\}, \{d(7), d(8)\}\}$, and from Definition 8 we know that $\{b(1), b(2)\}$ takes precedence over the others. The folding of $\{b(1), b(2)\}$ is depicted in Figure 9b. 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. Finally, Figure 9c depicts the minimal and canonical AES. In this particular case, it was necessary to keep two events with label c and two with label d to preserve the behavior.

The fact that the order on folding steps given in Definition 8 is clearly total, and thus folding is essentially deterministic, ensures that the reduction of an AES will produce a uniquely determined result.



(a) $\mathbb{A}_4$ (b) $f(\mathbb{A}_4)_{/\{b,b\}}$ (c) $f^+(\mathbb{A}_4)$

Figure 9: Canonical labeling and folding

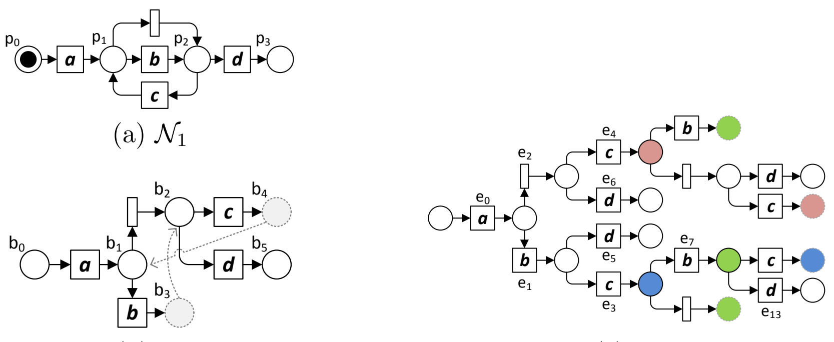
Proposition 1 (canonical folding of AES)**.** *Let $\mathbb{A}_1$ and $\mathbb{A}_2$ be isomorphic AESs. Then the deterministic folding of $\mathbb{A}_1$ and $\mathbb{A}_2$ produces a canonical AES, such that $f^+(\mathbb{A}_1)$ is isomorphic to $f^+(\mathbb{A}_2)$.*

4.2. Finite representation of cyclic process models

A fundamental problem with cyclic process models is that their branching processes, in the presence of cyclic behavior, are infinite. The seminal work in [22], later developed by many authors (see, e.g., [23] and citations therein) introduced sophisticated strategies for truncating the unfolding to a finite level, while keeping a representation of any reachable state, thus getting what is referred to as the *complete unfolding prefix (CP)*. In particular, the authors in [24] introduced a framework where a canonical unfolding prefix, complete with respect to a suitable property, not limited to reachability, can be constructed. Our own work relies on such a framework.

In the following we restrict ourselves to Petri nets without duplicate tasks (namely, the labeling of the visible events is assumed to be injective). Consider the net system $\mathcal{N}_1$ and the complete unfolding prefix β_1 presented in Figure 10. Note that both b_1 and b_4 correspond to the place p_1 in $\mathcal{N}_1$. To compute a marking complete unfolding prefix, we start applying the inductive rules in Figure 3. In this case, it is possible to stop unfolding once we reach b_2 and b_4 roughly because any addition to the prefix would duplicate information already represented. Events b and c are called *cutoff events*. 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. For this reason, we will use a stronger cutoff condition, leading to a larger prefix of the unfolding that makes explicit all the causal relations between activities. In the case of the net system $\mathcal{N}_1$ in Figure 10a the required unfolding prefix is β_2 (Figure 10c).

Formally, we resort to the notion of cutting context introduced in [24]. A cutting context is a tuple $\Theta = (\approx, \triangleleft, \mathcal{C})$ where $\approx$ is an equivalence relation over configurations, $\triangleleft$ is a total order over configurations, and $\mathcal{C}$ is the set of configurations used at the time of the computation of the unfolding prefix – e.g., the cutting context used in McMillan [22] is $\Theta_{McMillan} = (\approx_{mark}, \triangleleft_{size}, \mathcal{C}_{loc})$, where $\approx_{mark}$ equates two configurations when they produce the same marking, $\triangleleft_{size}$ is the total order induced by the size of configurations, and $\mathcal{C}_{loc} = \{\lfloor e \rfloor \mid e \in E\}$ is the set of local configurations. As already mentioned, the complete unfolding prefix β_1 is computed by using McMillan's cutting context. In fact, 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 $||\lfloor a \rfloor|| < ||\lfloor c \rfloor||$, then event c is a cutoff event. The cutting context in [25], denoted $\Theta_{ERV} = (\approx_{mark}, \triangleleft_{slf}, \mathcal{C}_{loc})$, differs from that in [22] for the definition of the partial order $\triangleleft_{slf}$, which is refined by considering action labels thus leading to more cut-offs and smaller prefixes (see [25] for details). For our purposes, consider a cutting context which is a modification of Θ_{ERV} with a refined equivalence relation over configurations taking into account also the labels of the events that produced the current marking. Roughly speaking, each token stores also the labels of the events in its history.



(a) $\mathcal{N}_1$

(b) β_1

(c) β_2

Figure 10: Petri net and two different unfoldings

Definition 9 ($\approx_{Pred}$)**.** Let $\beta = (B, E, G, \rho)$ be a branching process of a labeled Petri net system with a net $N = (P, T, F, \lambda)$. A pair of configurations $C_1, C_2 \in \mathit{Conf}(\beta)$ are equivalent, represented as $C_1 \approx_{Pred} C_2$, iff $\mathit{eMark}(C_1) = \mathit{eMark}(C_2)$, where

$$\mathit{eMark}(C) = \{\langle \rho(b), \rho_\tau(\lfloor b \rfloor)\rangle \mid b \in \mathit{Cut}(C)\}.$$

where for $X \subseteq E$, we define $\rho_\tau(X) = \{\rho(x) \mid x \in X \ \wedge \ \lambda(x) \neq \tau\}$, namely the set of non-silent transitions which are images of event X.

We rely on the cutting context $\Theta_{Pred} = (\approx_{Pred}, \lhd_{slf}, \mathcal{C}_{loc})$. According to the theory in [24] once we have proved that the equivalence $\approx_{Pred}$ and the adequate order $\lhd_{slf}$ are preserved by finite configuration extensions, we immediately have an algorithm for constructing a canonical, finite prefix of the unfolding, complete with respect to the equivalence $\approx_{Pred}$. The latter means that for any configuration C in the full unfolding there will be a configuration in the finite prefix such that $C \approx_{Pred} C'$.

Since our cutting context is a slight variation of that in [25], we can rely on their work for the proof.

Proposition 2 (equivalence is preserved by extension)**.** *Let $\beta = (B, E, G, \rho)$ be the branching process of a net system $\mathcal{N}$, with a net $N = (P, T, F)$, and $C, C' \in \mathit{Conf}(\beta)$ be a pair of configurations, s.t. that $C \approx_{Pred} C'$. Therefore, for every suffix V of C, there exists a finite suffix V' of C' s.t.:*

$$C' \oplus V' \approx_{Pred} C \oplus V$$

Proof. Let C, C' be configurations such that $C \approx_{Pred} C'$ 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 $\mathit{Mark}(C)[t\rangle$.

According to Definition 9, $\mathit{eMark}(C) = \mathit{eMark}(C')$, which in turn implies that $\mathit{Mark}(C) = \mathit{Mark}(C')$. Hence $\mathit{Mark}(C')[t\rangle$, which implies the existence of an extension $V' = \{e'\}$ of C', where $\rho(e') = t$.

Clearly $\mathit{Mark}(C \oplus \{e\}) = \mathit{Mark}(C' \oplus \{e'\})$. So, the fact that $\mathit{eMark}(C \oplus \{e\}) = \mathit{eMark}(C' \oplus \{e'\})$ is quite immediate. Take any condition $s' \in \mathit{Cut}(C' \oplus \{e'\})$. 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

$\rho(\lfloor s' \rfloor) = \{\rho(e)\} \cup \bigcup_{s'' \in {}^\bullet e'} \rho(\lfloor s'' \rfloor)$
$= \{\rho(e)\} \cup \bigcup_{s'' \in {}^\bullet e} \rho(\lfloor s'' \rfloor)$ [since $\rho(e) = t = \rho(e')$ and $C \approx_{Pred} C'$]
$= \rho(\lfloor s \rfloor)$

Therefore $\langle \rho(s), \rho(\lfloor s \rfloor) \rangle = \langle \rho(s'), \rho(\lfloor s' \rfloor) \rangle$.

- $s' \in Cut(C') \setminus {}^\bullet e'$
  In this case, if we take the only condition $s \in Cut(C) \setminus {}^\bullet e$ such that $\rho(s') = \rho(s)$, since $C \approx_{Pred} C'$, we immediately get that $\langle \rho(s), \rho(\lfloor s \rfloor) \rangle = \langle \rho(s'), \rho(\lfloor s' \rfloor) \rangle$.

Therefore we conclude that $eMark(C') \subseteq eMark(C)$. However, since the argument is perfectly symmetric, we can deduce equality. □

The following proposition shows that the canonical unfolding prefix constructed with Θ_{Pred} contains witnesses for all the causal relations that would be exhibited in the (possibly infinite) unfolding of a business process with cycles.

Proposition 3 (causal dependencies in the prefix)**.** *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 the CP unfolding based on the cutting context $\Theta_{Pred} = (\approx_{Pred}, \lhd_{slf}, \mathcal{C}_{loc})$. Then β_Θ is "complete with respect to the causal dependencies", i.e., for any pair of events $e_1, e_2 \in E : e_1 < e_2$ then*

$$\exists e_1', e_2' \in E' : e_1' < e_2', \textit{ where } \rho(e_1) = \rho_\Theta(e_1') \textit{ and } \rho(e_2) = \rho_\Theta(e_2').$$

Proof. Let $e_1, e_2 \in E$ be events of the unfolding such that $e_1 < e_2$. This means $e_1 \in \lfloor e_2 \rfloor$. Consider the configuration $C = \lfloor e_2 \rfloor$. By completeness there is a configuration C' in the prefix such that $eMark(C) = eMark(C')$. 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 $\rho(e_1) \in \rho(\lfloor s \rfloor)$. 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 $eMark(C) = eMark(C')$, it holds that $\langle \rho(s'), \rho(\lfloor s' \rfloor \rangle = \langle \rho(s), \rho(\lfloor s \rfloor \rangle$. 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. □

4.2.1. Multiplicity of activities

We now show how to identify the multiplicity of each activity (i.e., labeled transition in the original net) given the canonical unfolding prefix induced by Θ_{Pred}. Specifically, we seek to determine if a labeled transition t in a net system can be executed at most once or possibly more than once. Restricting to the class of free-choice sound workflow nets, which have been observed to be sufficiently expressive in most cases [26], we can also show that activities that can occur more than once in a computation can actually occur any number of times, hence they are part of some cyclic behavior.

Since we deal with safe nets, we observe that if a transition can occur twice in a configuration, the corresponding events must be causally related.

Proposition 4 (repetition)**.** *Let β be a branching process of a net system $\mathcal{N}$. Let $C \in Conf(\mathcal{N})$ be a configuration such that there exists $e, e' \in C$, $e \neq e'$ 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 any place in ${}^\bullet t$, contradicting the assumption that $\mathcal{N}$ is safe. □

The above observation motivates the interest for the following definition in the study of repetitive behaviors.

Definition 10 (self-preceding transitions). Let $\beta = (B, E, G, \rho)$ be the unfolding prefix induced by Θ_{Pred} for a net $N = (P, T, F, \lambda)$. The set of *self-preceding transitions* of N is defined as $\mathcal{R} = \{\rho(e_1) \mid \exists C \in Conf(\beta).\ e_1, e_2 \in C \ \wedge\ \rho(e_1) = \rho(e_2) \wedge e_1 < e_2\}$.

Note that the possibility of reducing the repetition to a causal dependency ensures that the finite prefix (which contains full information about causal dependencies) will be also sufficient to identify repeated events. As an example it can be checked that $C = \{e_0, e_2, e_6\}$, $C' = \{e_0, e_1, e_5\}$ and $C'' = \{e_0, e_1, e_3, e_7, e_{13}\}$ are configurations in the unfolding prefix β_2 from Figure 10c. Activity b is part of repetitive behavior as C'' includes two (causal dependent) occurrences of b. This holds despite the fact that there are (maximal) configurations including only a single occurrence of b (like C) or none (like C').

Definition 10 tells us which transitions in the original net system may occur more than once. By the same token, we can determine which transitions occur at least once. The latter correspond to events that occur in the intersection of all completed configurations.

Definition 11 (necessary transitions). Let $\beta = (B, E, G, \rho)$ be the unfolding prefix induced by Θ_{Pred} for a net system $\mathcal{N}$. The *necessary transitions* $\mathcal{K}$ of $\mathcal{N}$ is defined as $\mathcal{K} = \varrho(\bigcap \mathit{MaxConf}(\beta))$.

Based on the above definitions of $\mathcal{R}$ and $\mathcal{K}$, we classify transitions in a net system into three disjoint categories: those fired "0 or more times" (denoted as "$\star$"); "1 or more times" (denoted as "+"); and at most once "0..1". Formally:

Definition 12 (multiplicity of a transition). Let $\beta = (B, E, G, \rho)$ be the unfolding prefix induced by Θ_{Pred} for a net system $\mathcal{N}$, the multiplicity of a labeled transition is defined as:

- $0..1 = \{e \in E \mid \rho(e) \notin \mathcal{R}\}$
- $+ = \{e \in E \mid \rho(e) \in \mathcal{R} \cap \mathcal{K}\}$
- $\star = \{e \in E \mid \rho(e) \in \mathcal{R} \ \wedge \ e \notin +\}$

In the case a transition may be fired zero or more times ($\star$), the above definition does not tell us whether the transition can be repeated an unbounded number of times or a bounded number of times. Below, we refine the notion of multiplicity for a class of workflow nets, for which we show that when an activity is classified as $\star$, it means it can be fired any number of times.

4.2.2. Multiplicity of transitions in free-choice workflow nets

Transitions which, according to Definition 10 are marked as repetitive, namely either "+" or "*" can surely occur more than once in a computation, but still they could occur at most a bounded number of times. We next show that if we focus on the class of *sound free-choice workflow nets* [26], a transition which is marked as "+" or "*", may fire any number of times, namely it is part of a cyclic behavior.

Workflow nets [26] 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 13 (WF-net, WF-system). A Petri net $N = (P, T, F)$ is a *workflow net* (*WF-net*) if it includes a distinguished *source* place $i \in P$, with

${}^{\bullet}i = \varnothing$, a distinguished *sink* place $o \in P$, with $o^{\bullet} = \varnothing$, 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 [27] 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 when a token reaches the sink place. 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 $M'[t\rangle$. It is $\mathcal{N}$ is *bounded* when the set $[N, M_0\rangle$ is finite.

Definition 14 (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 [28, 29] are a well-behaved family of nets, where several properties, which are hard to check for general Petri nets, admit efficient verification technique.

Definition 15 (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 $p_1{}^{\bullet} \cap p_2{}^{\bullet} = \varnothing$ or $p_1{}^{\bullet} = p_2{}^{\bullet}$.

In words, in a free-choice net whenever two places share a transition it their postsets, they share all transitions in their postsets. 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.

Armed with the definitions above, we show that for the class of (safe) free-choice sound WF-nets, the self-preceding transitions captured by the proposed cutting context $\Theta_{Pred} = (\approx_{Pred}, \triangleleft_{slf}, \mathcal{C}_{loc})$, namely those transitions marked as "*" or "+" according to Definition 10 represent unbounded repetitive behavior.

We first need a preliminary technical result.

Lemma 5 (sequences of firings). *Let $\mathcal{N}$ be a free-choice sound WF-net. Let $t_0, \ldots, t_n$ be transitions such that $t_i{}^{\bullet} \cap {}^{\bullet}t_{i+1} \neq \varnothing$ for any $i \in \{0, \ldots, n-1\}$ 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 $M[t_0\sigma_0 t_1\sigma_1 \ldots \sigma_{n-1} t_n\rangle$.*

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 $\sigma_0, \dots, \sigma_n$ such that $M[t_0\sigma_0 t_1\sigma_1 \dots \sigma_{n-1}t_n\rangle M_n$. 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

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

as desired. □

We can now easily conclude with the desired result.

Proposition 6. *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 the prefix β_Θ, 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 $e = e_0 < e_1 < \dots < e_n = e'$ such that $e_i{}^\bullet \cap {}^\bullet e_{i+1} \neq \varnothing$ for any $i \in \{0, \dots, n-1\}$. 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 $i \in \{0, \dots, n-1\}$.

Now, given any marking M such that $M[t\rangle$, we can simply apply Lemma 5, to deduce that there are $\sigma_i \in T^*$, $i \in \{0, \dots, n-1\}$, such that

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

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

The insights we got from this section are used later in the verbalization of differences involving repeated tasks (Section 5.3).

5. Comparison of process models

When comparing process models, differences can 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 are 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 process 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 approach used in [7]. Clearly, if two AESs are diagnosed as isomorphic, it seems sensible to conclude that they are behaviorally equivalent. Moreover, if an error-correcting graph matching is used, the same algorithm would gather the information about the differences on event occurrences (process activities) and mismatching behavior relations. Given the intuitive interpretation of behavior relations used by AESs, the verbalization of 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 matching shown in Figure 11, between the folded AESs of the running example (cf. Figure 1).

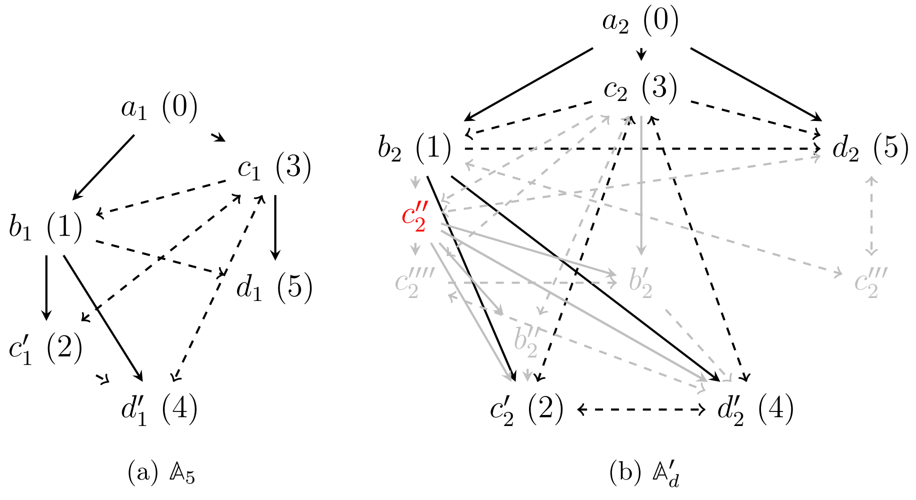


Figure 11: Event mappings computed with an error-correcting graph matching technique

The numbers inside the parenthesis in Figure 11 correspond to the optimal matching computed by an error-correcting graph matching technique on the folded AESs. Since the occurrence of c_2' causally depends on the occurrence of c_2'', it seems more natural to consider c_2'' be a better matching for c_1' than the matching identified by the graph matching technique. It is clear, therefore, that we need an new approach to compute the pairwise optimal matching of AESs, that is, one taking into account the order induced by the behavior relations on AES. Moreover, it is also clear that we need not only to diagnose the similarities but also to keep track of the differences found on the input pair of AES, in the same spirit of the error-correcting graph matching techniques.

One additional concern is to provide a systematic approach to produce intuitive diagnostic describing the differences found while comparing a pair of AESs. Therefore, in the remainder of this section we present the elements of our approach to compare AESs: *matching behavior*, *identifying differences* and *verbalizing differences.*

5.1. Matching behavior

The first challenge is to determine the behavior similarity between a pair of AESs. As mentioned above, we take *visible-pomset equivalence* as the reference for behavioral equivalence between AESs. Two processes are deemed visible-pomset equivalent when they have isomorphic sets of visible-pomsets. So, if two AESs exhibit different behavior (due to differences in the set of events or in the underlying behavioral relations), it is clear that their corresponding visible-pomsets would differ as well. Hence, we are interested in finding the best (or at least a good) approximated behavioral matching between AESs

We start by introducing the concept of partial match between two configurations, which is intended to capture the idea of an approximated isomorphism between the corresponding visible-pomsets. Note that the definitions in this section are formulated around the notion of configuration and extension, thus the definitions can be easily extrapolated to other types of event structures besides AES. For a partial function f, the notation $f(x) = \perp$ indicates that f is undefined on x.

Definition 16 (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:

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

$$\frac{C_1 \xrightarrow{e_1}_a C_1'}{(C_1, \xi, C_2) \xrightarrow{hide\ e_1} (C_1', \xi, C_2)} \, hide\ e_1$$

$$\frac{C_2 \xrightarrow{e_2}_a C_2'}{(C_1, \xi, C_2) \xrightarrow{hide\ e_2} (C_1, \xi', C_2')} \, hide\ e_2$$

Figure 12: Partial matching operations

1. $\lambda_2(\xi(e_1)) = \lambda_1(e_1)$
2. $e_1 \le_1 e_1'$ iff $\xi(e_1) \le_2 \xi(e_1')$

In words, a partial match is a function ξ that establishes a correspondence between events of the two pomsets, respecting both labeling and order. Note that *match* is a partial and non surjective function, meaning that some events in C_1 may not have a mapping to any event in C_2, and vice versa.

We now notice that a partial match between configurations can be thought as the result of applying two operations over "growing" pomsets

1. *matching* of events (both pomsets synchronously evolve a pair of events that have the same label), and
2. *hiding* of an event (only one pomset evolves with a single event while the other remains the same).

Matching and hiding operations can be expressed as inductive rules, as shown if Figure 12, that applied to a partial match ξ between C_1 and C_2 produce another partial match involving larger configurations. Since the same partial match can be associated with different pairs of configurations, we write (C_1, ξ, C_2) to refer to ξ seen as a partial match between C_1 and C_2. Above, we write $\xi[e_1 \to e_2] : C_1 \cup \{e_1\}) \to (C_2 \cup \{e_2\})$ to denote $\xi[e_1 \to e_2](e_1) = e_2$ and $\xi[e_1 \to e_2](e_3) = \xi(e_3)$ for $e_3 \in C_1 \backslash \{e_1\}$. Finally, we write $C \xrightarrow{e}_{\lambda(e)} C \cup \{e\}$ to denote $C \cup \{e\} \in \mathit{Conf}(\mathbb{A})$, for a configuration $C \in \mathit{Conf}(\mathbb{A})$ and an event $e \notin C$.

Starting from the above concepts, we aim at defining a technique that allows to optimize the matching of pomsets or, equivalently, to minimize the number of hiding operations in a partial match. Clearly, whenever it is possible to establish a mapping between the pomsets of two AESs using only *matching* operations, the AESs will be equivalent. Conversely, when the AESs are not equivalent then the optimal match of their pomsets –the one with the minimum number of hidings– would capture both the largest approximate visible-pomset (or common behavior) and the corresponding differences, in the form of hiding operations.

Let $\mathbb{A}_1$ and $\mathbb{A}_2$ be AESs. The "quality" of a partial match $s = (C_1, \xi, C_2)$ is captured by a value $g(s)$

$$g(s) = |C_1| + |C_2| - |\xi| \cdot 2. \tag{1}$$

The function $g(s)$ above is aimed at quantifying the "quality" of the matchings between a pair of pomsets. When $g(s) = 0$ ξ is a visible-pomset isomorphism between pomsets (or configurations) C_1 and C_2. When $g(s) > 0$ the partial match ξ required one or more hiding operations. This case can be interpreted as an approximate (or non complete) visible-pomset isomorphism of pomsets C_1 and C_2.

Note that given two AESs $\mathbb{A}_1$ and $\mathbb{A}_2$, for any two configurations C_1 and C_2 there is always a partial match. However, only a subset of the possible partial matches would have a minimum cost. The partial matches with minimum cost are said *optimal* and formally defined as follows.

Definition 17 (optimal 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 $s = (C_1, \xi, C_2)$, where $\xi : C_1 \nrightarrow C_2$, is called *optimal* when

$$g(s) = \min\{g(s) \mid \xi' : C_1 \nrightarrow C_2\}$$

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

Definition 18 (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 triples (C_1, ξ, C_2), where $\xi : C_1 \nrightarrow C_2$ is a partial match;
- T is the set of transitions $(C_1, \xi, C_2) \xrightarrow{op} (C_1', \xi', C_2')$ defined by the rules in Figure 12.

It is immediate to see that the partial synchronized product is inductively built starting from an "initial" node $(\varnothing, \varnothing, \varnothing)$ corresponding to the unique partial matching for the empty configurations, and then expanding the graph by using the rules in Figure 12.

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 construction computationally unfeasible.

We adopt a branch an bound approach, more specifically an adaptation of the well-known A^* algorithm [33], in order to build an informative part of the partial synchronized product. As usual, the A^* algorithm requires two cost functions: one to evaluate the cost from the root of the state space to a given path, 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.

Given a partial partial match $s = (C_1, \xi, C_2)$, the past-cost function $g(s)$ is that defined in Equation 1. The future cost function $h(s)$ is shown in Equation 2, where $E_i' = \{e \in E_i \mid \nexists e' \in C_i : e \nearrow e'\}$ for $i \in \{1, 2\}$.

$$h(s) = |(\lambda(E_1') \cup \lambda(E_2')) \setminus (\lambda(E_1') \cap \lambda(E_2'))| \tag{2}$$

Intuitively, h provides a measure of the number of events to be hidden in the future of C_1 and C_2. It optimistically assumes that events with the same label will indeed contribute to a one-to-one match between the two configurations. It can be seen that this estimate is admissible in the sense required in [30] for the use of the algorithm A^*.

The function for the A^* algorithm is then $\eta(s) = g(s) + h(s)$ for any partial match $s = (C_1, \xi, C_2)$. The pseudo-code for the search algorithm is presented in Algorithm 1 and uses function η.

In this context, the A^* algorithm is tightly coupled with the semantics of the underlying AESs, because the match and hide operations are based on the possible extensions of the configurations. In other words, the nodes expanded by the A^* algorithm from a partial match represent extensions in both configurations in the case of match, or extension in only one configuration in the case of hide.

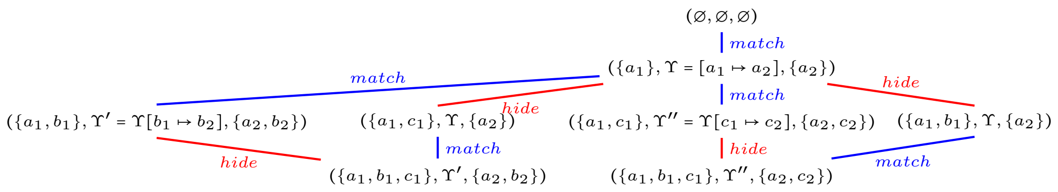
Figure 13 shows two AESs and a part of their partial synchronized product, which contains the optimal matches for the maximal configurations. Observe that, in the partial synchronized product, the fact that a pair of operations can be applied independently is captured by diamonds-like shapes in the graph. E.g., in Figure 13, after $(\{a_1\}, \xi = [a_1 \mapsto a_2], \{a_2\})$, it is possible to match the events with label b $(\{a_1, b_1\}, \xi' = \xi[b_1 \mapsto b_2], \{a_2, b_2\})$ and then hide the c-labeled event $(\{a_1, b_1, c_1\}, \xi', \{a_2, b_2\})$, or vice versa.

$$\begin{tikzcd}
 & a_1 \arrow[dl] \arrow[dr] & \\
b_1 & & c_1
\end{tikzcd}$$

(a) $\mathbb{A}_a$

$$\begin{tikzcd}
 & a_2 \arrow[dl] \arrow[dr] & \\
b_2 \arrow[rr, dashed, leftrightarrow] & & c_2
\end{tikzcd}$$

(b) $\mathbb{A}_b$



$(\varnothing, \varnothing, \varnothing)$
match
$(\{a_1\}, \Upsilon = [a_1 \mapsto a_2], \{a_2\})$
match
hide
match
hide
$(\{a_1, b_1\}, \Upsilon' = \Upsilon[b_1 \mapsto b_2], \{a_2, b_2\})$
$(\{a_1, c_1\}, \Upsilon, \{a_2\})$
hide
$(\{a_1, c_1\}, \Upsilon'' = \Upsilon[c_1 \mapsto c_2], \{a_2, c_2\})$
$(\{a_1, b_1\}, \Upsilon, \{a_2\})$
match
hide
hide
match
$(\{a_1, b_1, c_1\}, \Upsilon', \{a_2, b_2\})$
$(\{a_1, b_1, c_1\}, \Upsilon'', \{a_2, c_2\})$

(c) partial synchronized product for $\mathbb{A}_a$ and $\mathbb{A}_c$

Figure 13: AESs and their partial synchronized product with the optimal matches

5.2. Identifying differences

The partial synchronized product is a rich structure that represents the hide and match operations, which lead to some partial matches (possibly optimal or simply good, when determined with some heuristic approach).

In order to explain the behavioral differences, a possibility consists in simply verbalizing the hide operations. Note that differently from a purely syntactical approach this will capture how early a discrepancy can arise during the execution of the processes. More specifically, the closer a hide operation is from the "initial" node $(\varnothing, \varnothing, \varnothing)$, the sooner the discrepancy can occur.

Note that the partial synchronized product explicitly represents the state (partial match) where a discrepancy occurs, hereinafter called the *context*. Then, a hide operation be expressed as an event that occurs in one model but not in the other. For instance, in Figure 13 there is a node representing the hide operation $(\{a_1\}, \xi = [a_1 \mapsto a_2], \{a_2\}) \xrightarrow{hide\ c_1} (\{a_1, c_1\}, \xi, \{a_2\})$ and another representing the hide operation $(\{a_1, b_1\}, \xi' = \xi[b_1 \mapsto b_2], \{a_2, b_2\}) \xrightarrow{hide\ c_1} (\{a_1, b_1, c_1\}, \xi', \{a_2, b_2\})$. The behavioral difference represented by these two nodes is the same, namely: *"In model 1, there*

Algorithm 1 Optimal matches

```
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 optimal matches for the maximal configurations
    // Initialization
    foreach $C \in Conf(\mathbb{A}_1) \cup Conf(\mathbb{A}_2)$ do
        $GW(C) = \infty$
        $MATCHES[C] = \varnothing$
    end
    $s_0 = \langle \varnothing, \varnothing, \varnothing \rangle$
    OPEN $\leftarrow \{s_0\}$
    while $OPEN \neq \varnothing$ do
        Choose any $s = \langle C_1, \xi, C_2 \rangle \in$ OPEN, with minimum $\eta(s)$
            OPEN $\leftarrow$ OPEN $\setminus \{s\}$
            // Pruning
        if isCandidate($C_1, s, \mathbb{A}_1$) $\vee$ isCandidate($C_2, s, \mathbb{A}_2$) then
            // Best match
            if $C_1 \in MaxConf(\mathbb{A}_1) \wedge C_2 \in MaxConf(\mathbb{A}_2)$ then
                updateMatches($C_1, s$)
                updateMatches($C_2, s$)
            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
                OPEN $\leftarrow$ OPEN $\cup \{\langle C_1', \xi[e_1 \mapsto e_2], C_2' \rangle\}$      $\triangleright$ MATCH
            end
            foreach $C_1 \xrightarrow{e_1} C_1'$ do
                OPEN $\leftarrow$ OPEN $\cup$ $\{\langle C_1', \xi, C_2 \rangle\}$      $\triangleright$ HIDE $e_1$
            end
            foreach $C_2 \xrightarrow{e_2} C_2'$ do
                OPEN $\leftarrow$ OPEN $\cup$ $\{\langle C_1, \xi, C_2' \rangle\}$      $\triangleright$ HIDE $e_2$
            end
        end
    end
    return $MATCHES$
Procedure isCandidate($C, s, \mathbb{A}$)
    return $\exists M \in MaxConf(\mathbb{A}) : C \subset M \wedge \eta(s) \leq GW(M)$
Procedure updateMatches($C, s$)
    if $\eta(s) \leq GW(C)$ then
        $MATCHES[C] \leftarrow MATCHES[C] \cup \{s\}$
        $GW[C] \leftarrow \eta(s)$
    end
```

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

(a)

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

(b)

Figure 14: (a) Matrix representations for (a) partial match $(\{a_1, b_1, c_1\}, \xi, \{a_2, b_2\})$ and (b) extended partial match $(\{a_1, b_1, c_1\}, \zeta, \{a_2, b_2\})$

is a state where c *always occurs, whereas in the matching state in model 2, it cannot occur"*. These two differences however differ in terms of the state where the difference is observed. In the first case, the state in question is the one reached immediately after we execute activity a, whereas in the second case, it is the state reached immediately after we execute activity b. We can therefore see that if we map each hide operation in the partial synchronized product into a difference diagnostic statement, the resulting statements can be largely redundant and difficult to interpret.

For this reason 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 more convenient and understandable for the user. Thus, we next present an approach that aims at gathering the differences in the behavioral relations which are capable of explaining discrepancies represented as hide operations in the partial synchronized product.

In this approach, the behavioral difference between the AESs in Figure 13 can all be expressed using one single diagnostic statement, that is: *"In model 1,* b *and* c *are in parallel, whereas in model 2,* b *and* c *are mutually exclusive"*.

To implement this latter approach, we have to select the set of behavioral relations that best helps with the verbalization of the discrepancies captured by a given hide operation. To this end, we observe that a partial matching (C_1, ξ, C_2) can be seen as a matrix of behavioral relations, denoted as Ψ_ξ. In this alternative representation, the columns represent the matched events in ξ and the rows represent the hidden (unmatched) events. Note that Ψ_ξ represents a partially filled matrix. For instance, the matrix representation of the matching $(\{a_1, b_1, c_1\}, \xi, \{a_2, b_2\})$ (Figure 13) is displayed in Figure 14a.

The overall idea in order to diagnose the differences in terms of behavioral relations is the following. Given a partial match ξ the idea is to extend it as far as possible to the unmatched events, renouncing to the requirement that the match should respect the order in the pomsets, but still trying, following some heuristic to match events, which have the same label and dependencies as similar as possible. Also events outside the configurations can be involved.

Definition 19 (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 $\zeta(e_1) \neq \bot$ it holds $\lambda_2(\zeta(e_1)) = \lambda_1(e_1)$ and (iii) for any $e_1 \in E_1$ if $\zeta(e_1) \neq \bot$ then $e_1 \in C_1$ 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.

Then we introduce some measure of the "quality" of an extension. Roughly, we try to minimize the number of dependencies on which the matched events differ, still matching any possible pair of events with the same label.

Definition 20 (cost of extensions). Let $\zeta : E_1 \nrightarrow E_2$ be an extension of the partial match ξ, between configurations C_1 and C_2. The cost of an extension is defined as

$$K(\zeta) = |\{((e_1, e_2), rel, (e_1', e_2')) : \ rel \in \{\nearrow^*, <\} \ \wedge \ \zeta(e_1) = e_2 \ \wedge \ \zeta(e_1') = e_2' \ \wedge \\ \neg(e_1 \ rel \ e_1' \iff e_2 \ rel \ e_2')\}|$$

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 (thus applying the rule in Figure 15, where $e_1 \in C_1$ or $e_2 \in C_2$, minimizing the cost at each step).

$$\frac{\zeta(e_1) = \bot = \zeta^{-1}(e_2) \quad \lambda_1(e_1) = \lambda_2(e_2)}{\zeta[e_1 \mapsto e_2]} \ \textit{synthetic match } e_1, e_2$$

Figure 15: Synthetic matching operation

Consider for example the optimal matching $(\{a_1, b_1, c_1\}, \xi, \{a_2, b_2\})$ (Figure 13). The corresponding optimal maximal extension is shown in Figure 14b, i.e., $(\{a_1, b_1, c_1\}, \zeta = \xi[c_1 \mapsto c_2], \{a_2, b_2\})$. This example is very simple because there is only one possibility to match the event c_1 in $\mathbb{A}_b$ (with event c_2).

The partial synchronized product may contain more than one optimal match for a maximal configuration (and also more than one extended partial match), each of which leads to the same number of differences. In the absence of any other intuitive criteria for distinguishing optimal matches, we select any such matching to generate a verbalization of differences. The following section describes the verbalization step.

5.3. Verbalizing differences

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.

The context describes the configuration (herein called "state") in the execution of the process model where a given discrepancy occurs. A full representation of the context consists of a partially ordered set of events (activity executions) leading to a given state where the discrepancy is observed. In the case of visual feedback, this can be visually represented by animating the process model in order to show to the user an execution path leading to the state in question. On the other hand, listing all the events in an execution path leading to a given state is arguably less readable in textual form. Instead, when verbalizing a context in textual form, it may be more convenient to refer only to a partial description of the context, consisting only of the last event (i.e., last activity) executed before the configuration in question 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 (configuration) in a process model is further studied in [31].

The difference itself is described by referring to either a behavioral relation in one model that is not present 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:

- Causality ($<$): "a *always occurs before activity* 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:

$$\begin{tikzcd}
(\varnothing,\varnothing,\varnothing) \arrow[d, dash, "match"] \\
(\{a_1\}, \Upsilon = [a_1 \mapsto a_2], \{a_2\}) \arrow[d, dash, "match"] \\
(\{a_1\}, \Upsilon, \{a_2, c_2\}) \arrow[d, dash, "match"] \\
(\{a_1, d_1\}, \Upsilon, \{a_2, c_2, d_2\})
\end{tikzcd}$$

Figure 16: Partial synchronized product for the optimal matching of a pair of configurations in the AESs in Fig. 11

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

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_ > *always occurs, 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, Figure 16 shows the partial synchronized product representation of the optimal matching for the configurations: $(\{a_1, d_1\}, \{a_2, c_2, d_2\})$, they are configurations of the AES of the running example (Figure 11). It is worth mentioning that the runs $\{a, b, d\}, \{a, c, d\}, \{a, b, c, d\}, \{a, c, b, d\}$ can be performed by both process models. Therefore, the corresponding pomsets will be mapped only with match operations.

The following are the resulting verbalizations of the differences captured in Figure 16:

- 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*

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). In the last step, we need to produce the feedback for the set of unmatched events. In this case, we also include the set of direct causally preceding events to give a context in the feedback. For the running example, the feedback would be:

- b, = (<,): *In M1, there is a state after* b, *where* b *always occurs, whereas it cannot occur in the matching state in M2*

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

6. Conclusion

This article presented a method for comparing business process models based on binary behavioral relations, specifically those supported by AES. The proposed method involves four sub-methods that constitute distinct contributions of the article:

1. A method to calculate a canonically reduced AES from an acyclic Petri net.

2. A method to compute a finite representation of repetitive behavior that preserves all causal dependencies of each transition in a Petri net with cycles (i.e., each activity in a cyclic process model).

3. A method for constructing a partial “error-correcting” synchronized product of two event structures.

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 are present 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.

The presented methods are implemented in a prototype tool, namely BP-Diff [32]. BP-Diff takes as input a pair of process models captured in the BPMN notation and produces a visual and/or textual diagnostic of their differences. A command-line version of the tool that provides textual diagnostic is available at `https://code.google.com/p/fdes/`. A software-as-a-service version that provides both visual and textual diagnostic is available at `http://diffbp-bpdiff.rhcloud.com/`.

A direction for future research is the optimization of the proposed method in order to address scalability concerns. The method involves an unfolding step to calculate an AES, followed by a folding 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 may 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 a pair of process models. Another direction for future work is an 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.

[1] R. Dijkman, M. Dumas, B. van Dongen, R. Käärik, J. Mendling, Similarity of business process models: Metrics and evaluation, Inf. Sys. 36 (2) (2011) 498–516.

[2] M. L. Rosa, S. Clemens, A. H. M. ter Hofstede, N. Russell, Appendix A. The Order Fulfillment Process Model, in: Modern Business Process Automation 2010, Springer.

[3] M. Nielsen, G. D. Plotkin, G. Winskel, Petri Nets, Event Structures and Domains, Part I, Theoretical Computer Science 13 (1981) 85–108.

[4] P. Baldan, A. Corradini, U. Montanari, Contextual Petri Nets, Asymmetric Event Structures, and Processes, Information and Computation 2001 171 1–49.

[5] A. Armas-Cervantes, P. Baldan, L. García-Bañuelos, Reduction of event structures under history preserving bisimulation, CoRR abs/1403.7181. URL `http://arxiv.org/abs/1403.7181`

[6] R. Dijkman, M. Dumas, C. Ouyang, Semantics and analysis of business process models in BPMN, Information and Software Technology 50 (12) (2008) 1281–1294.

[7] A. Armas-Cervantes, P. Baldan, M. Dumas, L. García-Bañuelos, Behavioral Comparison of Process Models Based on Canonically Reduced Event Structures, in: Prof. of BPM, Springer, 2014, pp. 267–282.

[8] R. van Glabbeek, U. Goltz, Refinement of actions and equivalence notions for concurrent systems, Acta Informatica 37 (2001) 229–327.

[9] R. Cleaveland, On automatically explaining bisimulation inequivalence, in: CAV, LNCS 531, Springer, 1991, pp. 364–372.

[10] O. Sokolsky, S. Kannan, I. Lee, Simulation-Based Graph Similarity, in: TACAS, LNCS 3920, Springer, 2006, pp. 426–440.

[11] R. Dijkman, Diagnosing Differences between Business Process Models, in: BPM, Vol. 5240 of LNCS 5240, Springer, 2008, pp. 261–277.

[12] M. Weidlich, J. Mendling, M. Weske, Efficient Consistency Measurement Based on Behavioral Profiles of Process Models, IEEE TSE 37 (3) (2011) 410–429.

[13] M. Weidlich, A. Polyvyanyy, J. Mendling, M. Weske, Causal Behavioural Profiles, Fundamenta Informaticae 113 (3-4) (2011) 399–435.

[14] 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.

[15] E. Badouel, On the α-Reconstructibility of Workflow Nets, in: S. Haddad, L. Pomello (Eds.), ATPN, LNCS 7347, Springer, 2012.

[16] M. Weidlich, J. van der Werf, On Profiles and Footprints-Relational Semantics for Petri Nets, in: ATPN, LNCS 7347, Springer, 2012, pp. 148–167.

[17] U. Goltz, W. Reisig, The non-sequential behaviour of Petri nets, Information and Control 57 (2/3) (1983) 125–147.

[18] J. Engelfriet, Branching processes of Petri nets, Acta Informatica 28 (1991) 575–591.

[19] R. van Glabbeek, U. Goltz, Equivalence notions for concurrent systems and refinement of actions., Acta Informatica 379 (1989) 237–248.

[20] B. D. McKay, Practical graph isomorphism, Department of Computer Science, Vanderbilt University, 1981.

[21] G. Kant, Using canonical forms for isomorphism reduction in graph-based model checking, Technical report, CTIT University of Twente, Enschede (July 2010).

[22] K. L. McMillan, D. K. Probst, A Technique of State Space Search Based on Unfolding, Formal Methods in System Design 6 (1) (1995) 45–65.

[23] J. Esparza, K. Heljanko, Unfoldings - A Partial order Approach to Model Checking, EACTS Monographs in Theoretical Computer Science, Springer, 2008.

[24] V. Khomenko, M. Koutny, W. Vogler, Canonical prefixes of Petri net unfoldings, Acta Informatica 40 (2) (2003) 95–118.

[25] J. Esparza, S. Römer, W. Vogler, An Improvement of McMillan's Unfolding Algorithm, Formal Methods in System Design 30 (2) (2002) 285–310.

[26] W. M. P. van der Aalst, Verification of workflow nets, in: ICATPN, Springer, 1997, pp. 407–426.

[27] W. van der Aalst, Workflow verification: Finding control-flow errors using petri-net-based techniques, in: BPM, Vol. 1806 of LNCS, Springer, 2000, pp. 161–183.

[28] E. Best, Structure theory of petri nets: the free choice hiatus, in: Petri Nets: Central Models and Their Properties, Vol. 254 of LNCS, Springer, 1987, pp. 168–205.

[29] J. Desel, J. Esparza, Free Choice Petri Nets, Cambridge University Press, New York, NY, USA, 1995.

[30] R. Dechter, J. Pearl, Generalized best-first search strategies and the optimality of A*, J. ACM 32 (1985) 505–536.

[31] N. Lohmann, D. Fahland, Where did I go wrong? - explaining errors in business process models., in: BPM, 2014, pp. 283–300.

[32] 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.

[33] P. E. Hart, N. J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, in: IEEE Transactions on Systems, Science, and Cybernetics, 1968.