Synchronizing Process Model and Event Abstraction for Grounded Process Intelligence (Extended Version)

Janik-Vasily Benzin[1], Gyunam Park[2], and Stefanie Rinderle-Ma[1]

[1] Technical University of Munich, TUM School of Computation, Information and Technology, Garching, Germany {janik.benzin,stefanie.rinderle-ma}@tum.de
[2] Process Mining Group, Fraunhofer FIT, Aachen, Germany
gyunam.park@fit.fraunhofer.de

Abstract. Model abstraction (MA) and event abstraction (EA) are means to reduce complexity of (discovered) models and event data. Imagine a process intelligence project that aims to analyze a model discovered from event data which is further abstracted, possibly multiple times, to reach optimality goals, e.g., reducing model size. So far, after discovering the model, there is no technique that enables the synchronized abstraction of the underlying event log. This results in loosing the grounding in the real-world behavior contained in the log and, in turn, restricts analysis insights. Hence, in this work, we provide the formal basis for synchronized model and event abstraction, i.e., we prove that abstracting a process model by MA and discovering a process model from an abstracted event log yields an equivalent process model. We prove the feasibility of our approach based on behavioral profile abstraction as non-order preserving MA technique, resulting in a novel EA technique.

Keywords: Event Abstraction · Model Abstraction · Complexity · Synchronization

1 Introduction

Discovering a process model M from an event log L is a key step in analyzing the actual process behavior recorded by information systems [15]. However, events are often logged at a low granularity level, leading to the discovery of complex and uninterpretable process models that do not match stakeholders' expectations. *Event abstraction* (EA) techniques [18,50] have been proposed to address this challenge by lifting the granularity of events to satisfy an *abstraction goal* that formalizes stakeholder's expectations. While empirical studies have evaluated whether EA techniques can satisfy abstraction goals like model complexity reduction [41], existing EA techniques neither provide formal guarantees on the reduction of model complexity nor on satisfying any other goal. Hence, current EA techniques cannot be applied to find the optimal abstraction. Without optimality, downstream process intelligence tasks such as process enhancement [15], business process simulation (BPS) [19], and predictive process monitoring (PPM) [49] are uncertain to meet the stakeholder's expectations.

Model abstraction (MA) techniques [37,20,32], by contrast, ensure satisfying an abstraction goal such as reducing model complexity through solving an optimization

problem [20,32]. Specifically, the optimization objective is the abstraction goal and decision variables are the application sequence of abstraction operations and their parameters. Hence, the question arises **how the optimality guarantees of MA can be utilized for EA and, in turn, the knowledge of real-world process behavior stored in the logs can be exploited for process intelligence tasks on abstracted models, i.e., how to synchronize MA and EA (RQ).**

For illustrating **RQ** and its effects on process intelligence, consider the scenario depicted in Fig. 1: A bank runs trading processes for i) derivative and ii) fixed income products logged by two information systems. The bank wants to understand the common business process underlying the trading of i) and ii) through simulation and prediction tasks.

The bank starts with discovering a process model M (cf. Fig. 1③) from log L merged from the two information systems (cf. Fig. 1①), i.e., $L \xrightarrow{pd} M$ using data-aware discovery, e.g., [19] after [16], as in this case also data objects are of interest. Assume now that in order to reduce the obvious complexity of M, behavioral profile abstraction (BPA) ma_{bpa} [37] abstracts M into M_a (cf. Fig. 1 ⑤) and subsequently, in order to generalize data objects to streamline the underlying processes, abstracts M_a into M_{aa} (cf. Fig. 1 ⑦). While M_a and M_{aa} fulfill the optimization goals, they have lost their grounding in an event log. Thus, the bank cannot proceed due to missing abstracted event logs L_a (cf. Fig. 1 ④) and L_{aa} (cf. Fig. 1 ⑥) that match the granularity of M_a and M_{aa} respectively and contain the actual observed behavior with rich event attributes.

One solution would be to generate logs by playing out the abstracted models. However, this cannot reproduce the actual behavior stored in the log w.r.t., e.g., number of traces, frequency of paths, or data values. To overcome this information loss and to maintain flexibility of process intelligence tasks, we propose synchronizing MA with EA techniques to mirror the abstraction applied to M on L. The core idea of synchronization is to prove that abstracting a model via ma and discovering a model from an log abstracted by ea yields equivalent models M_a (annotated by question mark in Fig. 1).

MA techniques can be distinguished into order-preserving and non-order-preserving techniques. In this work, we opt to study non-order-preserving techniques such as [37] as they pose the more general, harder problem. Moreover, MA techniques like [37] enable both, unrestricted abstraction of control flow with data flow abstraction by clustering activities according to similar data flow or semantical control flow abstraction by clustering activities that are semantically-related according to domain-specific *part-of* relations between activities [33]. Overall, the contributions of this work include conditions under which MA and EA can be synchronized for non-order-preserving MA and an EA technique that maintains observed distributions in the abstracted log. As these contributions are conceptual and formal, the evaluation is the formal synchronization proof.

Section 2 introduces background and related work. Sect. 3 establishes the theoretical foundation for synchronization by formalizing the problem and presenting our general approach. Sect. 4 presents our concrete BPA-based synchronization method, including the adaptation of the BPA technique and the design of the corresponding synchronized EA technique. Sect. 5 lays the theoretical foundations needed for proving synchronization correctness, and Sect. 6 proves synchronization correctness to estab-

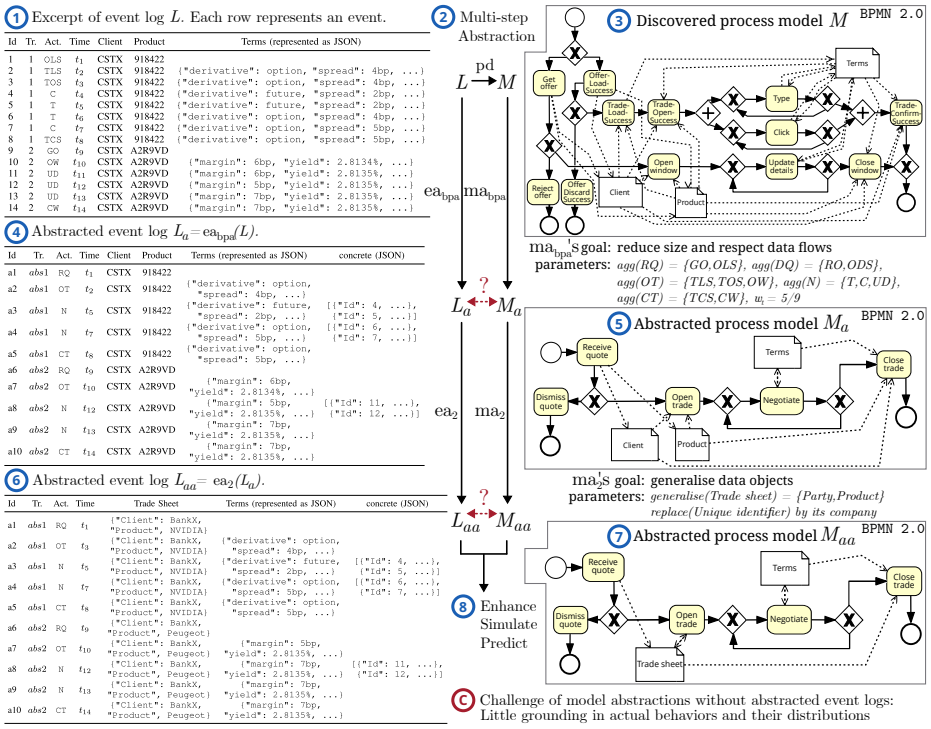

Fig. 1: Process Intelligence in the Financial Domain (Example)

lish the equivalence of MA's and EA's resulting process model. Sect. 7 demonstrates the impact of synchronization on the illustrative example from the financial domain. Sect. 8 concludes the paper.

2 Background and Related Work

Model representation, abstraction, and existing techniques: As logical representation of process models, we opt for process trees due to their *block-structuredness*, often advocated in MA literature [26,37,20] and due to its favorable properties like *soundness* [16]. Let $\mathcal{A}$ be the set of all possible activity names and the silent activity $\tau \notin \mathcal{A}$. Then process trees are recursively defined by

- $M = v$ for $v \in \mathcal{A}$ or $M = \tau$ are process trees (referred to as leaves), and
- $M = \oplus(M_1,...,M_n)$ with n process trees $M_1,...,M_n$ and operators $\oplus \in \{\times, \rightarrow, \wedge, \circlearrowleft\}$ is a process tree referred to as a $\oplus$-node.

Let $\mathcal{M}$ be the set of all models. Then, a **model abstraction (MA)** is a partial function $\text{ma}: \mathcal{M} \nrightarrow \mathcal{M}$ that maps a process model M to another (abstracted) process model M_a where it is assumed that the complexity of M_a is reduced compared to M. We call ma is *applicable* to M when $M \in dom(\text{ma})$. The most common complexity

metric used in MA techniques is the size of the process model [35,20] (cf. Tab. 1). In Fig. 1, size $|M|=28<|M_{aa|}=12$ if we count all nodes in the models. For process trees, the size of $|M|$ is defined as the sum of $\oplus$-nodes and leaves. We refer to the activities in a process tree M with A_M, e.g., $A_{M_{aa}}=\{\texttt{RQ},\texttt{OT},\texttt{N},\texttt{CT}\}$ in Fig. 1 ⑦.

In Tab. 1, we report the 19 MA techniques that are not covered by the MA survey [37] from 2012. In addition to the year of publication, we report the following nine properties for each technique. The *abstraction goal* (goal in Tab. 1) specifies the target of the MA technique. The type specifies whether the abstraction operators are only *order-preserving* (OP), only *non-order-preserving* (NOP), or both. The *model* reports the modeling language in which process models are represented. The *process perspective* (Persp.) reports what perspective is represented in the model and, subsequently, abstracted by the technique. For the perspective, we abbreviate the *control-flow* persp. by `C`, the *data* persp. by `D`, and the *organizational* persp. by `O`.

Ref.	Year	Goal	Type	Model	Persp.	Op.	Abs. Obj.	Optim.	pd	ea
[25]	2023	Efficient verification	OP	BPMN	C,D	E	SESE, data objects	✓	✗	✗
[43]	2023	Quick overview	OP	BPMN	C	A,E	SESE	✗	✗	✗
[39,40] [38]	2022	Quick overview	OP	BPMN Collab.	C,D,O	A,E,G	SESE, data objects, lanes, messages	✗	✗	✗
[3]	2020	Quick overview	OP	WoMan [8]	C	A	SESE	✗	WIND [8]	✗
[44]	2019	Quick overview	OP	BPMN	C	A	SESE	✗	✗	✗
[28]	2019	Quick overview	OP	BPMN	C,D	A,G	SESE, data objects	✓	MARBLE [27]	✗
[45]	2018	Quick overview	OP	BPMN	C	A	SESE	✗	✗	✗
[32,31]	2018	Efficient prediction	OP	GSPN	C,P	A	SESE	✓	IM [17]	✗
[5]	2015	Quick overview, balanced discovery	NOP	PN	C	E	*P,F*	✗	ILP [47]	✗
[10]	2015	Configurable	(N)OP	BPMN	C	A,E	SESE, Flows	✗	✗	✗
[20,21]	2015	Configurable	OP	BPMN	C,D,O	A,E,G	SESE, data objects, resources	✓	✗	✗
[14]	2014	Privacy concerns	OP	PT	C	E	SESE	✗	✗	✗
[7,6]	2013	Balanced discovery	OP	PN	C	E	*P,F*	✗	All	✗
[13,29] [30]	2013	Custom views	(N)OP	BPMN	C,D,O	A,E,G	SESE, data objects, activities, resources	✗	✗	✗
[12,11]	2013	Change propagation	(N)OP	BPMN	C,D,O	A,E,G	SESE, data objects, activities, resources	✗	✗	✗
[37,34] [36,33]	2012	Large repository	NOP	BPMN	C,D	A	Activities	✗	✗	✗
[23]	2012	Quick overview	OP	BPMN	C,D	A	SESE, data object	✗	✗	✗
[46]	2011	Large repository, quick overview	(N)OP	BPMN	C	A,E	SESE, flows	✗	✗	✗
[48]	2011	Quick overview	OP	CCS	C	A,E	SESE	✗	✗	✗

Table 1: Model Abstraction Techniques

Next, the abstraction *operators* (Op.) show whether the technique applies *elimination* (E), *aggregation* (A), *generalisation* (G), or a combination of the three. The *abstraction objects* (Abs. Obj.) report what model elements in the domain of the abstraction operators, i.e., what model elements can be deleted, aggregated, or generalised. Obviously, the serialization of arbitrary BPMN process models into their *refined process tree structure* [42] with single-entry single-exit (SESE) process fragments is prevalent among MA techniques.

For the last three properties, we report whether the MA technique has the property or not. A MA technique applies optimization (Optim.) iff the abstraction goal is translated into an optimization objective and a solver is proposed that finds an optimal operator sequence and each operator's respective parameters that must be applied on a model to satisfy the abstraction goal. A MA techniques is formulated on discovered process models (pd) iff it takes an event log L as input, discovers a process model M through process discovery technique pd, and then abstracts M. Additionally, we report the process discovery technique that is considered in the MA technique. Lastly, a MA technique synchronizes its abstraction operators (ea) iff for each operator a corresponding EA technique is defined that also abstracts the event log.

Roughly half of MA techniques focus solely on the "quick overview" abstraction goal, i.e., aim to reduce the model size. Further selected goals in descending order are as follows. The goal "large repository" (2 times) is at par with "configurable" and "balanced discovery". While the former aims to manage a large repository of process models by only storing the fine-grained models and generating the coarse-grained models through MA, the latter two depend on the user through parametrization and the process discovery quality dimensions respectively. For example, [10] proposes a process querying language that is capable of abstracting behavior before returning the result. The remaining five abstraction goals are each only once targeted. Interestingly, only two MA techniques target a goal, "efficient verification" and "efficient prediction", that considers process intelligence tasks beyond understanding and visualisation.

The majority of 19 techniques propose order-preserving abstraction operators with 6/19 MA techniques considering operators that are non-order-preserving. MA techniques are commonly proposed for BPMN models with 7 exceptions: One technique is proposed for the declarative model language of the workflow management (WoMan) framework [8], one technique is proposed for generalised stochastic Petri nets (GSPN), two techniques are proposed for Petri nets (PN), and one technique each for process trees (PT) and models in the *calculus for communication systems* [24]. Clearly, the control-flow perspective is always represented and the main target of any abstraction operator (cf. abstraction objects). 15/19 MA techniques

In the following, we discuss the five MA techniques that consider an event log L in more detail. MA technique [7] abstracts the discovered process model M by filtering arcs in the *unfolding* of M and by applying three structural simplifications on the refolded M. Likewise, MA technique [5] abstract the discovered process model M by either projecting M into less complex model classes like *series-parallel* Petri nets or by removing infrequently-enabled arcs detected through replay from M. In both techniques, the mapping between activities in the model and events in the event log does not change, because both techniques neither abstract activities, nor events. MA technique [32] optimizes the application sequence of five order-preserving abstraction operators towards reducing the model size while controlling the information loss for efficiently predicting process performance. MA technique [28] applies a sequence of abstraction operators on a discovered process model M; MA technique [3] applies pattern mining to find order-preserving clusters of activities in M to be abstracted. All three techniques [32,28,3] face the challenges of MA without abstracted event logs (cf. Sect. 1): little grounding in actual behaviors and lack of flexibility for applying

downstream process intelligence tasks. Besides, none of the three MA techniques considers non-order-preserving abstraction.

Log representation and abstraction: An event log L is a multiset of traces, i.e., $L=[\sigma_1,...]\in\mathbb{B}(\mathcal{A}^*)$. We denote the set of activities that occur in an event log with A_L and write $e\in\sigma$ iff e occurs in σ, i.e., $\exists i\in\{1,...,|\sigma|\}:\sigma[i]=e$ with $\sigma[i]$ retrieving the ith event. There are 10 distinct activities (Act.) in A_L for L in Fig. 1 ①. We omit further rich event attributes like the "Terms" in L in our trace conceptualization, because they are irrelevant for the proof of synchronization. **Event abstraction (EA)** is defined as a partial function $\text{ea}:\mathbb{B}(\mathcal{A}^*)\nrightarrow\mathbb{B}(\mathcal{A}^*)$ such that the number of traces and events do not increase, i.e., $|\text{ea}(L)|\leqslant|L|$ and $\|\text{ea}(L)\|\leqslant\|L\|$ with $\|L\|=\sum_{\sigma\in L}|\sigma|$ [50].

We refer to [50,18] for a detailed discussion and taxonomy on EA techniques and to [41] for an empirical evaluation of EA techniques published through 2024. EA technique [49] aim to improve the accuracy of remaining time predictions that are learned from low-level event logs by applying three EA operators on the log. The three EA operators are designed to mirror the three MA operators sequence, self-loop, and choice. Yet, no theory is developed to prove the correctness of the designed EA operators. Moreover, the proposal is specific to remaining time prediction and does not show how to extend the EA technique with additional operators.

Process tree discovery and formal definitions: The semantics of a process tree $\mathcal{L}(M)$ is the language represented by M [16]. Given an event log L, a process discovery technique pd discovers a process tree M that represents L, i.e., $\text{pd}:\text{B}(\mathcal{A}^*)\rightarrow\mathcal{M}$ with $\mathcal{M}$ the set of all process trees. For example, a process discovery technique, IM [15], leverages the *directly-follows graph* (DFG) $G(L)=(A_L\cup\{\triangleright,\triangleleft\},\mapsto_L)$ with $\triangleright,\triangleleft\notin A_L$[3] and $\mapsto_L$ the *directly-follows* relation to discover M. An event log L and a process tree M can be related based on the notion of *directly-follows completeness* [15], i.e., L and M are directly-follows complete (df-complete), denoted by $L\sim_{df}M$, iff the DFGs $G(L)$ and $G(M)=G(\mathcal{L}(M))$ are equal: $G(L)=G(M)$. Df-completeness captures the behavior in L and M as equivalent to the abstract representation of a DFG. Df-completeness is a condition for the Inductive Miner (IM) to *rediscover* a process tree M from L that is *isomorphic* to M' that was executed for recording the event log L and, as such, is integral to the EA techniques presented in Sect. 4.2 and Sect. 6.

Two process trees M_1,M_2 are isomorphic, formally $M_1\cong M_2$, iff they are syntactically equivalent up to reordering of children for $\wedge$- and $\times$-nodes and the non-first children of $\circlearrowleft$-nodes. A process tree M is *isomorphic rediscoverable* by pd from event log L with $L\subseteq\mathcal{L}(M)$ iff pd discovers a process tree $M'=\text{pd}(L)$ that is isomorphic to M [16]. Isomorphic rediscoverability has been proven for the IM through assuming a restriction $Q(M)$ that must hold for process tree M and a restriction $R(L,M)$ that must hold for L and M. $Q(M)$ requires a process tree without silent activities τ, *duplicate* activities, and joint start and end activities of a $\circlearrowleft$-nodes first child and $R(L,M)$ requires df-completeness [16].

[3] $\triangleright$ is used to denote the start activity $\sigma=\langle v,...\rangle$ as a directly-follows pair $(\triangleright,v)\in\mapsto_L$ and $\triangleleft$ analogously for the end activity of a trace.

3 Synchronization Framework

In this section, we establish the theoretical foundation for synchronizing MA and EA techniques. We first formalize the synchronization problem and requirements, then present our general synchronization approach.

Synchronization problem and requirements. For synchronization to be possible, we must be able to design an EA technique ea_{ma} that transforms the event log such that process discovery yields an abstracted model isomorphic to what we would obtain through direct model abstraction.

Definition 1 (Synchronizability). *Let L be an event log and $M = \text{pd}(L)$ the discovered process tree such that model abstraction* ma *is applicable to M, i.e., $M_a = \text{ma}(M)$. L,* ma*, and* pd *are synchronizable iff there exists event abstraction* ea_{ma} *s.t.* $\text{pd}(\text{ea}_{\text{ma}}(L)) \cong M_a$.

Synchronization approach. We follow a two-step approach for synchronizing MA and EA: first discovering a complex process tree $M = \text{pd}(L)$ followed by applying ma to yield $M_a = \text{ma}(M)$, and abstracting $L_a = \text{ea}_{\text{ma}}(L)$ followed by discovering $M'_a = \text{pd}(L_a)$ should result in isomorphic abstract process trees, i.e., $M_a \cong M'_a$. Therefore, our approach requires two key components:

- MA Technique Adaptation: Not all MA techniques are immediately suitable for synchronization. We need to ensure the MA technique is well-defined and provides sufficient structure for designing a corresponding EA technique.
- Synchronized EA Design: The EA technique must be designed to mirror the abstractions applied by the MA technique while preserving the behavioral relationships needed for correct process discovery.

We demonstrate this approach using the behavioral profile abstraction (BPA) technique [37] for the following reasons. First, synchronizing BPA constitutes a significant challenge because it allows abstracting arbitrary sets of activities to allow abstractions wrt. the data flow (cf. Fig. 1). Hence, the corresponding EA technique must abstract the event log while guaranteeing the correct order of activities in the abstracted event log. Second, BPA aims to reduce model size, which aligns well with the size characteristics of event logs.

We select the Inductive Miner (IM) including fall-throughs for process discovery [15] to balance practical relevance with proof complexity, as isomorphic rediscoverability is already established for IM. Because our synchronization approach requires the novel EA technique ea_{bpa} to result in abstracted event logs L_a for which the IM discovers M_a to maintain the relation between log and model, L_a enables further process intelligence tasks that are grounded in the real-world behavior of L_a and that were not possible before (cf. Sect. 1).

4 BPA-Based Synchronization Method

In this section, we present our concrete synchronization method based on BPA. We first adapt the BPA technique to ensure it meets our synchronization requirements (Sect. 4.1), then design the corresponding synchronized event abstraction technique (Sect. 4.2).

4.1 Adapting BPA for Synchronization

This section introduces the non-order-preserving BPA technique ma_{bpa} and adapts it for synchronization. To illustrate the three steps of ma_{bpa}, we introduce a running example. In addition to the illustrative example in Fig. 1, we present the running example that is more detailed to demonstrate the full behavior of both ma_{bpa} in the following, our synchronized EA technique ea_{bpa} in Sect. 4.2, and our algorithm to generate *minimal* df-complete event logs in Sect. 5.2. Hence, the running example is used to demonstrate the algorithms step-by-step, whereas the illustrative example motivates our approach in Sect. 1 and is revisited in Sect. 7.

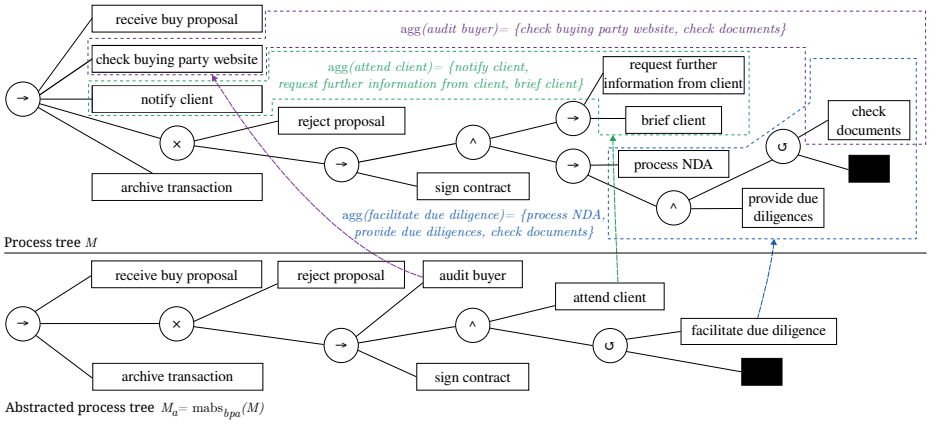

Fig. 2: Running example: Process tree M and abstracted process tree $\text{ma}_{bpa}(M)$ [37] for threshold $w_t = 0.5$.

ref	trace
σ_{ex1}	⟨RBP,CBW,NC,RP,AT⟩
σ_{ex2}	⟨RBP,CBW,NC,RFI,BC,PN,CD,CD,PDD,SC,AT⟩
σ_{ex3}	⟨RBP,CBW,NC,RFI,PN,BC,CD,CD,PDD,SC,AT⟩
σ_{ex4}	⟨RBP,CBW,NC,PN,RFI,BC,CD,CD,PDD,SC,AT⟩

Table 2: Excerpt of event log L_2.

Running example: Figure 2 shows *process tree* M which describes a transaction process by a mergers & acquisitions advisor that sells companies for its clients. The advisor receives buy proposals, checks their websites, and notifies the client about the new proposal. If the proposal is convincing, further information for the buyer is requested from the client and the client is briefed while the advisor processes the non-disclosure agreement (NDA), followed by providing the buyer with confidential documents (due diligence) and checking the documents received from the buyer (multiple times). In the end, an acquisition contract is signed and the proposal is archived. If the proposal is not convincing, it is rejected. Lastly, Table 2 shows four traces σ_{ex1},... of an event log L_2 that is recorded from executing M (activity names are abbreviated by their acronym).

BPA aims to enable unrestricted abstraction of concrete activities into abstract activities. Consequently, we can cluster activities according to their data flow (cf.

Fig. 1 ③) and apply ma_{bpa} to abstract accordingly. At the core of ma_{bpa} lies the *behavioral profile* of a process tree M. It is equivalent to the *footprint* [1] of a process model and constitutes an abstract representation of the behavior allowed by the model similar to the DFG (cf. Sect. 2), but more coarse-grained. Hence, a behavioral profile contains less detailed information on the behavior of the model than the DFG, e.g., a loop $\circlearrowleft(a,b)$ leaves a distinct graph pattern in the DFG, but is indistinguishable from $\wedge(a,b)$ in a behavioral profile. The behavioral profile is defined as follows:

Definition 2 (Behavioral profile, adapted from [37]). *Let M be a process tree, $A = \Sigma(M)$ its activities, and $\mathcal{L}(M)$ its language. Let $\succ \subseteq A \times A$ be the weak-order relation that contains all activity pairs $(x,y) \in \succ$ for which there exists a trace $\sigma = \langle \alpha_1,...,\alpha_n \rangle \in \mathcal{L}(M)$ with $j \in \{1,...,m-1\}$ and $j < k \leqslant m$ such that $\alpha_j = x$ and $\alpha_k = y$. Given the weak-order relation, an activity pair (x,y) is either:*

- *in a strict order relation $\rightsquigarrow_M$: $x \succ y$ and $y \not\succ x$,*
- *in an choice order relation $+_M$: $x \not\succ y$ and $y \not\succ x$,*
- *or in a parallel order relation $\|_M$: $x \succ y$ and $y \succ x$.*

The set of all three relations $\rightsquigarrow_M, +_M$, and $\|_M$ is the behavioral profile p_M of M. The set of all behavioral profiles over activities A is denoted by $\mathcal{BP}_A$.

As aforementioned, we acronym the activity names. For example, we have $\texttt{RBP} \rightsquigarrow_M \texttt{RP}$, $\texttt{RP} +_M \texttt{SC}$, and $\texttt{CD} \|_M \texttt{CD}$ in the behavioral profile p_M of the concrete process tree M in Fig. 2. Given the behavioral profile notion, we can introduce $\text{ma}_{bpa}(M) = M_a$ as three subsequent steps:

S1 The behavioral profile is computed: $\text{pr}(M) = p_M \in \mathcal{BP}_A$.

S2 Given a behavioral profile, the abstract behavioral profile p_{M_a} is derived from p_M by a parametrized function: $\text{dv}_{\text{agg},w_t}(p_M) = p_{M_a}$. The first parameter is a function $\text{agg}: A_a \to 2^A$ with A_a the activities of M_a without the silent activity and $A = \Sigma(M)$. agg specifies which abstract activities correspond to which sets of concrete activities. The second parameter $0 < w_t \leqslant 1$ controls what ordering relation frequencies are selected for p_{M_a} from p_M.

S3 Given an abstract behavioral profile, an abstracted process tree M_a is synthesized whose behavioral profile equals p_{M_a}, i.e., $\text{sy}(p_{M_a}) = M_a$. To uniquely construct a process tree M_a from profile p_{M_a}, the profile is encoded as a graph $G(p_{M_a})$ and the graph's unique *modular decomposition tree* $MDT(G)$ [22] is computed. If each *module* m in $MDT(G)$ is either *linear*, *AND*-, or *XOR*-complete [37], then process tree M_a is constructed by adding a tree node for each module. As a module can be *primitive*, i.e., contains "conflicting" ordering relations, not all profiles p_{M_a} have a corresponding process tree M_a such that this step may fail: $\text{sy}(p_{M_a}) = \perp$.

To illustrate ma_{bpa}, we refer to Fig. 2. The process tree M, the abstracted process tree M_a, and the parameter agg are depicted. First, the behavioral profile p_M of M is computed (S1). Given p_M, the second step $\text{dv}_{\text{agg},w_t}(p_M)$ is computed (S2). The parameter agg is denoted in Fig. 2 by three different colors. For instance, $\text{agg}(\texttt{AB}) = \{\texttt{CBW}, \texttt{CD}\}$. For presentation purposes, the mappings $\text{agg}(y) = \{y\}$ for

Algorithm 1 Derivation of an ordering relation (adapted from [37])

1: **deriveOrderingRelation**$_{\mathrm{agg},w_t}$(**Activity** x,**Activity** y)
2: $w(x \succ_{M_a} y) = |\{\forall (v,u) \in \mathrm{agg}(x) \times \mathrm{agg}(y) : v \rightsquigarrow_M u \vee v \parallel_M u\}|$
3: $w(y \succ_{M_a} x) = |\{\forall (v,u) \in \mathrm{agg}(x) \times \mathrm{agg}(y) : v \rightsquigarrow_M^{-1} u \vee v \parallel_M u\}|$
4: $w(x \not\succ_{M_a} y) = |\{\forall (v,u) \in \mathrm{agg}(x) \times \mathrm{agg}(y) : v \rightsquigarrow_M^{-1} u \vee v +_M u\}|$
5: $w(y \not\succ_{M_a} x) = |\{\forall (v,u) \in \mathrm{agg}(x) \times \mathrm{agg}(y) : v \rightsquigarrow_M u \vee v +_M u\}|$
6: $w_{prod} = |\mathrm{agg}(x)| \cdot |\mathrm{agg}(y)|$
7: $w(x +_{M_a} y) = \min(w(x \not\succ_{M_a} y), w(y \not\succ_{M_a} x)) \cdot \frac{1}{w_{prod}}$
8: $w(x \rightsquigarrow_{M_a} y) = \min(w(x \succ_{M_a} y), w(y \not\succ_{M_a} x)) \cdot \frac{1}{w_{prod}}$
9: $w(x \rightsquigarrow_{M_a}^{-1} y) = \min(w(y \succ_{M_a} x), w(x \not\succ_{M_a} y)) \cdot \frac{1}{w_{prod}}$
10: $w(x \parallel_{M_a} y) = \min(w(x \succ_{M_a} y), w(y \succ_{M_a} x)) \cdot \frac{1}{w_{prod}}$
11: **if** $w(x +_{M_a} y) \geqslant w_t$ **then**
12: **return** $x +_{M_a} y$
13: **else if** $w(x \rightsquigarrow_{M_a} y) \geqslant w_t$ **then**
14: **if** $w(x \rightsquigarrow_{M_a}^{-1} y) > w(x \rightsquigarrow_{M_a} y)$ **then**
15: **return** $x \rightsquigarrow_{M_a}^{-1} y$
16: **else**
17: **return** $x \rightsquigarrow_{M_a} y$
18: **else if** $w(x \rightsquigarrow_{M_a}^{-1} y) \geqslant w_t$ **then**
19: **return** $x \rightsquigarrow_{M_a}^{-1} y$
20: **else if** $w(x \parallel_{M_a} y) \geqslant w_t$ **then**
21: **return** $x \parallel_{M_a} y$
22: **else**
23: **return** $x \parallel_{M_a} y$

$y \in \{\texttt{RBP}, \texttt{RP}, \texttt{SC}, \texttt{AP}\}$ are not visualized in Fig. 2. Note that at this stage, the order of abstract activities in the abstracted process tree M_a is unknown and cannot be derived intuitively, because their mappings of agg may overlap (e.g., AB and FDD) or the order of their concrete activities is in conflict (e.g., NC and BC in agg(AC) vs. CD in agg(FDD) are in strict and interleaving order respectively). ma_{bpa} computes the ordering relations between two abstract activities $x,y \in M_a$ by selecting the most *restrictive* ordering relation among those that occur relatively more frequent than or equally frequent to the threshold w_t. To that end, $\mathrm{dv}_{\mathrm{agg},w_t}$ applies Alg. 1 to each abstract activity pair x and y.

Alg. 1 consists of three blocks: Counting frequencies of weak order relations between the respective concrete activities (line 2-5), deriving relative frequencies for ordering relations from weak order relations (line 6-10), and selecting the most restrictive ordering relation ($+ > \rightsquigarrow^{-1} > \rightsquigarrow > \parallel$) that is equal to or greater than threshold w_t. For example, Alg. 1 applied to AB and AC for $w_t = 0.5$ computes the relative weak order frequency $w(\texttt{AB} \succ_{M_a} \texttt{AC}) = 5/6$, because CBW $\succ$ NC, CBW $\succ$ RFI, CBW $\succ$ BC, CBW $\succ$ NC, CD $\succ$ RFI, and CD $\succ$ BC. Analogously, we have $w(\texttt{AC} \succ_{M_a} \texttt{AB}) = 3/6$, $w(\texttt{AB} \not\succ_{M_a} \texttt{AC}) = 1/6$, and $w(\texttt{AC} \not\succ_{M_a} \texttt{AB}) = 3/6$. The weak order frequencies are transformed to order relation frequencies by taking the minimum of the respective two weak order relations. Thus, $w(\texttt{AB} +_{M_a} \texttt{AC}) = 1/6$, because 1/6 is the minimum of $w(\texttt{AB} \not\succ_{M_a} \texttt{AC})$ and $w(\texttt{AC} \not\succ_{M_a} \texttt{AB})$. Analogously, we have $w(\texttt{AB} \rightsquigarrow_{M_a} \texttt{AC}) = 3/6$, $w(\texttt{AB} \rightsquigarrow_{M_a}^{-1} \texttt{AC}) = 1/6$, and $w(\texttt{AB} \parallel_{M_a} \texttt{AC}) = 3/6$ such that $\texttt{AB} \rightsquigarrow_{M_a} \texttt{AC}$ is the most restrictive ordering relations whose relative frequency is equal to w_t. Overall, the result is $\texttt{AB} \rightsquigarrow_{M_a} \texttt{AC}$.

As the third step (S3), ma_{bpa} attempts synthesizing the abstracted process tree M_a from the abstract behavioral profile p_{M_a} (cf. Algorithm 3.2 in [37]). To construct the different nodes of the process tree M_a, an *order relations graph* $G(p_{M_a}) = (V,E)$ for a given behavioral profile p_{M_a} is constructed. The nodes are the activities $V = A_a$. Edges correspond to the strict order relation and the choice relation without the identity, i.e., $E = \rightsquigarrow_{M_a} \cup +_{M_a} \setminus id_{A_a}$. For example, Fig. 3 (a) depicts the order relations graph $G(p_{M_a})$ for the abstracted behavioral profile p_{M_a} that is derived in step 2 for the running example in Fig. 2.

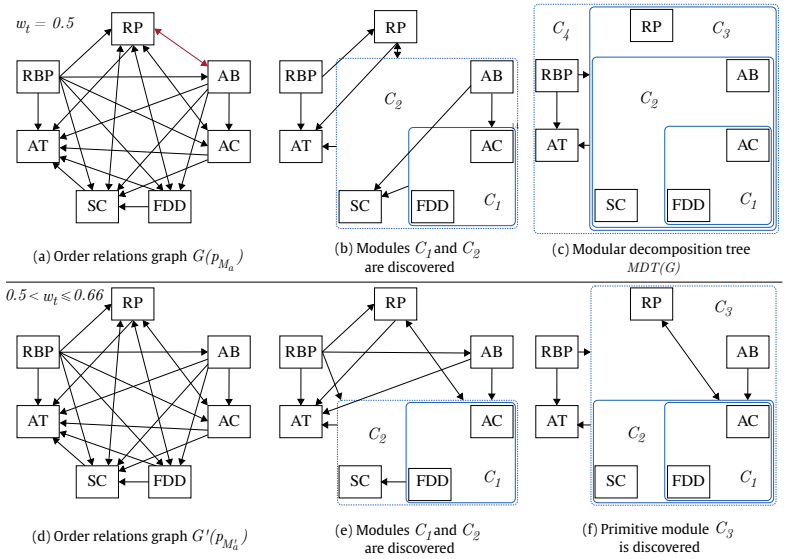

Fig. 3: Modular decomposition of two order relation graphs $G(p_{M_a})$ and $G(p_{M'_a})$ that are derived by ma_{bpa} from the running example M for two different parameters w_t. For $w_t = 0.5$ (a-c), AND-complete module C_1, linear module C_2, XOR-complete module C_3 and linear module C_4 are discovered. For $0.5 < w_t \leqslant 0.66$ (d-e), AND-complete module C_1, linear module C_2, and primitive module C_3 are discovered.

To derive a unique tree structure from $G(p_{M_a})$, the modular decomposition tree $MDT(G)$ [22] is computed. The tree contains a hierarchy of *non-overlapping*[4] modules $C \subseteq V$ that have uniform ordering relations with activities $V \setminus M$, i.e., they "agree" on their ordering relations to other activities. Additionally, modules are classified through the ordering relations between their activities $x \in C$: AND-complete and XOR-complete modules have only activities that are in interleaving order (i.e., they are not connected in $G(p_{M_a})$) and in choice order (i.e., they are completely connected) respectively, while linear modules have only activities that can be linearly

[4] Two modules *overlap* iff they intersect and neither is a subset of the other.

ordered such that their edges do not violate the direction of the linear order. Any other module is primitive.

As depicted in Fig. 3 (a-c), the modular decomposition of $G(p_{M_a})$ discovers four modules that each correspond to a node of M_a (cf. Fig. 2): Linear module C_4 corresponds to the $\rightarrow(...)$ root node in M_a, XOR-complete module C_3 corresponds to the $\times(...)$ node in M_a, linear module C_2 to the second $\rightarrow(...)$ node, and AND-complete module C_1 to the $\wedge(...)$ node. Because the modular decomposition $MDT(G)$ (c) does not contain any primitive module, the abstracted process tree M_a with behavioral profile p_{M_a} can be synthesized. Additionally, step 3 includes a special case for activities $x \in A_a$ that are in parallel order with themselves: $x \,\|\, x \in p_{M_a}$. sy constructs a self-loop node $\circlearrowleft(x,\tau)$, e.g., the self-loop of activity `FDD` in Fig. 2.

In general, parameter agg is assumed to be set (i.e., computed by clustering cf. Sect. 1), as the actual value has no impact on our results. We explicitly add three restrictions (3-5) on ma_{bpa} to guarantee that ma_{bpa} always satisfies its abstraction goal (cf. Fig. 1 ③):

Definition 3 (Behavioral Profile Abstraction (adapted from [37])). *The behavioral profile abstraction* $\mathrm{ma}_{bpa}(M) = M_a$, $M_a = \mathrm{sy}(p_{M_a})$, $p_{M_a} = \mathrm{dv}_{\mathrm{agg},w_t}(p_M)$, $p_M = \mathrm{pr}(M)$ *is applicable to* M *iff*[5]*:*

1. M *has no duplicate activities,*
2. *the modular decomposition tree* $MDT(G)$ *of the abstract behavioral profile's graph* $G(p_{M_a})$ *contains no primitive module,*
3. $(A_{new} := A_a \backslash A \neq \emptyset) \wedge (A_c := A_a \backslash A_{new} \subseteq A) \wedge (A_{new} \cap A_c = \emptyset)$,
4. $(\forall x \in \mathcal{A}_{new}\colon |\mathrm{agg}(x)| > 1) \wedge (|\bigcup_{y \in A_{new}} \mathrm{agg}(y)| > |A_{new}| + 1) \wedge (\forall y \in A_c\colon \mathrm{agg}(y) = \{y\})$,
5. w_t *is restricted to* $0 < w_t \leqslant w_{minmax}$ *with* $w_{minmax} = \min_{x,y \in A_a} w_{max}(x,y)$ *and*

$$w_{max}(x,y) = \max\big(w(x +_{M_a} y), w(x \rightsquigarrow_{M_a} y), w(x \rightsquigarrow^{-1}_{M_a} y), w(x \,\|_{M_a}\, y)\big).$$

Conditions (1-2) are required for the applicability of ma_{bpa} by the original proposal in [37]. Conditions (3-5) are added to guarantee that the resulting process tree M_a is smaller and to prohibit renaming of activities during abstraction.

Conditions (1-2): The original ma_{bpa} [37] additionally required both the process tree M to be of the form $M = \rightarrow(s, M', e)$ for start and end activities s and e and the abstracted process tree M_a to be of the form $M_a = \rightarrow(s_a, M'_a, e_a)$. In M_a, the activities s_a and e_a are either equal to their concrete counterparts s and e or are added in step 3 as artificial start and end activities, if M_a would otherwise not have a start and end activity. Both restrictions are not required for our purpose and, in the case of adding artificial start and end activities, results in a non-synchronizable model abstraction. Moreover, the abstracted process tree M_a may not be smaller than the process tree M, i.e., ma_{bpa} may not be a model abstraction anymore. Consequently, we adapted ma_{bpa} by removing both restrictions.

[5] We denote A_M by A and A_{M_a} by M_a, because the context is clear.

Conditions (3-5): Restrictions (3), (4), and (5) guarantee that the resulting process tree M_a is smaller and prohibit renaming of activities during abstraction, e.g., $|M| = |\text{ma}_{bpa}(M)|$ for any agg that only renames activities in Fig. 2. BPA ma_{bpa} with $A_{new} = \{\texttt{AB},\texttt{AC},\texttt{FCC}\}$, $w_t = w_{minmax} = 0.5$, and agg as depicted in Fig. 2 is applicable to M. To illustrate restriction (5), consider the running example. By setting the parameter w_t to a value between 0.5 and 0.66, the order relations graph misses the choice edge between RP and AB (denoted in red in Fig. 3 (a)), because the default case (line 22) of Alg. 1 is reached for these two activities. Hence, order relations graph $G(p_{M'_a})$ is derived in step 2 as depicted in Fig. 3 (d). Due to the missing edge, a primitive module C_3 is discovered by the modular decomposition (cf. Fig. 3 f). Consequently, no abstracted process tree M_a exists that has the same behavioral profile $p_{M'_a}$. To avoid the default case (line 22) and subsequent discovery of primitive modules, we restrict w_t in restriction (5).

Importantly, these restrictions have not been stated for ma_{bpa} in [37]. Thus, adding (3), (4), and (5) constitutes an adaptation of ma_{bpa}. Our adapted ma_{bpa} is well-defined (cf. Sect. 2). Next, we present our design for ea_{ma}.

4.2 Synchronized Event Abstraction Design

To ensure synchronization for our novel EA technique ea_{bpa}, we design ea_{bpa} to transform L into an abstracted event log L_a such that L_a and M_a are df-complete. We aim for a df-complete L_a, because df-completeness is required for IM's isomorphic rediscoverability (cf. Sect. 2). ea_{bpa} is composed of two steps.

Preliminarily Abstracting The Event Log. First, ea^1_{bpa} constructs a preliminary abstracted event log L_{tmp} by abstracting occurrences of concrete events $e \in A_{\text{ma}}$ (i.e., e is abstracted by ma_{bpa}) into new abstract events x trace by trace. Next, ea^1_{bpa} deletes abstract activities x that are in choice relation to another abstract activity y from traces $\sigma_{abs} \in L_{tmp}$ in which both x and y occur. We illustrate ea^1_{bpa} with the running example (cf. Fig. 2). Let L be an event log such that IM discovers M as depicted. For simplicity, we assume that L is a *minimal* df-complete event log (cf. Sect. 5.2).

To start, we have $L_2 = [\sigma_{ex1}, \sigma_{ex2}, \sigma_{ex3}, \sigma_{ex4}, ...]$ (cf. Tab. 2), $|L_2| = 46$, and $\|L_2\| = 455$[6]. For σ_{ex1}, ea_{mabs} computes $A_{\text{ma}} = \{\texttt{CBW},\texttt{NC}\}$ such that $\sigma_{ex1}[2]$ is the first event abstracted by $\texttt{AB} \in A_{new} = \{\texttt{AB},\texttt{AC},\texttt{FDD}\}$ (line 8). However, the condition in line 9 is not true, because $\texttt{RP} +_{M_a} \texttt{AB} \in p_{M_a}$. Also, for $\sigma_{ex1}[3]$ the abstract activity NC is in choice relation to RP. Thus, $\sigma_{abs,1} = \langle \texttt{RBP},\texttt{RP},\texttt{AT} \rangle$. For σ_{ex2}, ea_{mabs} computes $A_{\text{ma}} = \{\texttt{CBW},\texttt{NC},\texttt{RFI},\texttt{BC},\texttt{PN},\texttt{CD},\texttt{PDD}\}$ such that the condition in line 7 becomes true for any $\sigma'_{ex2}[2]$,...,$\sigma'_{ex2}[8]$. However, only $\sigma'_{ex2}[2]$,$\sigma'_{ex2}[3]$, and $\sigma'_{ex2}[6]$ for AB,AC, and FDD respectively satisfy the condition in line 8. Since no concrete activity $u \in A_{\neg\text{ma}} = \{\texttt{RBP},\texttt{SC},\texttt{AT}\}$ is in choice relation to an abstract activity $x \in A_{new}$, three new abstract events AB,AC, and FDD are added to $\sigma_{abs,2}$ in line 11. Also, abstract activity FDD is in parallel relation to itself, so that FDD is added a second time to σ_{abs2}. Overall, $\sigma_{abs2} = \langle \texttt{RBP},\texttt{AB},\texttt{AC},\texttt{FDD},\texttt{FDD},\texttt{SC},\texttt{AT} \rangle$. Since in every trace the activity CBW always occurs before NC and both always occur before PN (cf. Fig. 2), the next 44 iterations

[6] Minimal df-complete event logs are computed by Alg. 4 in Sect. 5.2

Algorithm 2 First step to synchronize ma_{bpa}: ea^1_{bpa}

Require: : Event log L, process discovery technique pd_{IM}, MA technique ma_{bpa} with corresponding p_{M_a},A_a,A_{new}, and agg (cf. Def. 3)
Ensure: : Preliminary abstracted event log L_{tmp}, abstracted process tree M_a
1: $L_{tmp} \leftarrow \{\}, M \leftarrow \text{pd}_{IM}(L), M_a \leftarrow \text{ma}_{bpa}(M)$,
2: **for all** traces $\sigma \in L$ **do**
3: $\sigma_{abs} \leftarrow \langle\rangle$
4: $A_{\text{ma}} \leftarrow \{e \in \sigma \mid \exists x \in A_{new} : e \in \text{agg}(x)\}$
5: $A_{\neg\text{ma}} \leftarrow \{e \in \sigma\} \backslash A_{\text{ma}}$
6: **for all** $e \in \sigma$ in the order of σ **do**
7: **if** $e \in A_{\text{ma}}$ **then**
8: **for all** $x \in A_{new}$ with $e \in \text{agg}(x)$ first appearing in σ **do**
9: **if** $v \in A_{\neg\text{ma}}$ does not exist s.t. $v +_{M_a} x \in p_{M_a}$ **then**
10: $\sigma_{abs} \leftarrow \sigma_{abs} \cdot \langle x \rangle$
11: **if** $x \parallel_{M_a} x \in p_{M_a}$ **then**
12: $\sigma_{abs} \leftarrow \sigma_{abs} \cdot \langle x' \rangle$
13: **else if** $e \in A_{\neg\text{ma}}$ **then**
14: $\sigma_{abs} \leftarrow \sigma_{abs} \cdot \langle e \rangle$
15: $L_{tmp} \leftarrow L_{tmp} + \{\sigma_{abs}\}$ // standard multiset addition
16: $A_\times \leftarrow \{A \subseteq A_{new} \mid \exists C \in MDT(G(p_{M_a})), \forall x \in A : \text{x belongs to XOR-complete module } C\}$
17: $L_{tmp} \leftarrow \text{deleteChoiceActivities}(L_{tmp}, A_\times)$
18: **return** L_{tmp}, M_a

of the for-loop (line 2) always results in the same abstract trace: $\sigma_{abs2} = ... = \sigma_{abs46}$. and $L_{tmp} = [\sigma_{abs1}, \sigma_{abs2}, ..., \sigma_{abs46}]$. Consequently, $L_{tmp} = [\sigma_{abs1}, \sigma^{45}_{abs2}]$ when the for-loop terminates.

Because no abstract activities $x, y \in A_{new}$ are in choice relation $x +_{M_a} y \in p_{M_a}$ (cf. no XOR-complete module in Fig. 3 (c)), it holds that $A_\times = \varnothing$ in line 17 such that no abstract activities are deleted in line 18. Hence, ea_{bpa} returns L_{tmp} without further changes. In general, $A_\times$ contains sets of activities that are in choice relation to each other, i.e., for any $A \in A_\times$, all abstract activities $x, y \in A$ are in choice relation. Abstract activities that are in choice relation must not both occur in a trace $\sigma_{abs} \in L_{tmp}$. Also, the frequencies of traces in which they occur should be "similar"[7]. While it is important that not always the same abstract activity is deleted from a trace for proving correctness in Sect. 6, similar frequencies ensure that distributions like trace frequencies are maintained as faithfully as possible. The function deleteChoiceActivities ensures that the aforementioned requirements on abstract activities in choice relation are met. For the result L_{tmp} of ea^1_{bpa}, the order of events in L_{tmp} may not adhere to the order of activities in M_a, which is guaranteed through the second step.

Transposing Events To Ensure Correct Orders. The second step ea^2_{bpa} establishes the correct order of events in L_{tmp} with respect to the order of events in the reference minimal df-complete event log L_a of M_a (cf. Alg. 4). The correct order is established through the *Kendall Tau Sequence Distance* $\delta_{kendall}$ [4] that computes the minimal number of transpositions needed to transform one trace σ_1 into the other σ_2. As $\delta_{kendall}$ requires that both traces are permutations of the same multiset of activities, it is undefined otherwise ($\delta_{kendall} = \perp$). Put succinctly, ea^2_{bpa} finds a matching between equivalence classes of traces in the reference L_a (line 3) and

[7] Same frequency means $\forall \sigma_{abs} \in L_{tmp}, A \in A_\times : |A_{\sigma_{abs}} \cap A| \leqslant 1$ and $\forall x, y \in A, A \in A_\times : |\text{freq}_x(L_{tmp}) - \text{freq}_y(L_{tmp})| \leqslant |A|$ with $\text{freq}_x(L) = |\{\sigma_{abs} \in L \mid x \in \sigma_{abs}\}|$

Algorithm 3 Second step to synchronize ma_{bpa}: ea^2_{bpa}

Require: : Preliminary abstracted event log L_{tmp}, Abstracted process tree M_a
Ensure: : Abstracted, df-complete event log L_r
1: $L_a \leftarrow \mathcal{L}_m(M_a)$ // take the minimal df-complete log as reference (cf. Alg. 4)
2: $L_r \leftarrow []$ // initialize empty log for the result
3: $L_a/\sim \leftarrow \{L' \subseteq L_a \mid \forall \sigma_1,\sigma_2 \in L' : \delta_{kendall}(\sigma_1,\sigma_2) \neq \bot\}$
4: $L_{tmp}/\sim \leftarrow \{L' \subseteq L_{tmp} \mid \forall \sigma_1,\sigma_2 \in L' : \delta_{kendall}(\sigma_1,\sigma_2) \neq \bot\}$
5: **for all** $L_a^{class} \in L_a/\sim$ **do**
6: **for all** $L_{tmp}^{class} \in L_{tmp}/\sim$ **do**
7: **if** $\exists \sigma \in L_a^{class}, \sigma_{abs} \in L_{tmp}^{class} : \delta_{kendall}(\sigma,\sigma_{abs}) \neq \bot$ **then**
8: $n_1,...,n_{|L_a^{class}|} \leftarrow \text{evenSplitSizes}(|L_{tmp}^{class}|,|L_a^{class}|)$
9: **for all** $j,\sigma \in \text{enumerate}(L_a^{class})$ **do**
10: $L_{tmp}^{n_j} \leftarrow \text{closestTraces}_{\delta_{kendall}}(n_j, L_{tmp}^{class}, \sigma)$
11: $L_{tmp}^{class} \leftarrow L_{tmp}^{class} - L_{tmp}^{n_j}$
12: $L_{tmp}^{transposed} \leftarrow \text{transposeAll}_{\delta_{kendall}}(L_{tmp}^{n_j}, \sigma)$
13: $L_r \leftarrow L_r + L_{tmp}^{transposed}$
14: **return** L_r

equivalence classes of traces in the input L_{tmp} (line 4) where equivalence is defined modulo transposition (cf. Sect. 6.1). To maintain relative trace multiplicities within an equivalence class of the reference (line 8-9), the traces in the matched equivalence class of the input are evenly split (line 10-11) and transposed (line 12) to exhibit the same order of events as the reference traces (line 13).

To continue the illustration, $\mathcal{L}_m$ computes $L_a = [\sigma_1,...,\sigma_4]$ with $\sigma_1 = \langle \texttt{RBP}, \texttt{RP}, \texttt{AP} \rangle$, $\sigma_2 = \langle \texttt{RBP}, \texttt{AB}, \texttt{AC}, \texttt{FDD}, \texttt{FDD}, \texttt{SC}, \texttt{AP} \rangle$, $\sigma_3 = \langle \texttt{RBP}, \texttt{AB}, \texttt{FDD}, \texttt{AC}, \texttt{FDD}, \texttt{SC}, \texttt{AP} \rangle$ and $\sigma_4 = \langle \texttt{RBP}, \texttt{AB}, \texttt{FDD}, \texttt{FDD}, \texttt{AC}, \texttt{SC}, \texttt{AP} \rangle$. Hence, $L_{tmp}/\sim = \{[\sigma_{abs1}], [\sigma_{abs2}^{45}]\}$ (line 3) and $L_a/\sim = \{[\sigma_1], [\sigma_2,\sigma_3,\sigma_4]\}$ (line 4) are the two quotient sets modulo transposition. In the first iteration of the for-loop in line 5, the equivalence class $L_a^{class} = [\sigma_1]$ and the equivalence class $L_{tmp}^{class} = [\sigma_{abs1}]$ satisfy the condition in line 7. Therefore, the one trace of L_{tmp}^{class} is evenly split to the one trace of L_a^{class}, i.e., $n_{|L_a^{class}|} = n_1 = 1$. Next, the single closest trace σ_{abs1} of L_{tmp}^{class} is assigned to $L_{tmp}^{n_1}$ (line 10), removed from the equivalence class L_{tmp}^{class} (line 11), transposed according to the zero distance of $\delta_{kendall}$ between σ_{abs1} and σ_1 (i.e., no transposition is applied), and assigned to the result: $L_r = [\sigma_{abs1}]$.

In the second iteration of the for-loop in line 5, the equivalence class $L_a^{class} = [\sigma_2,\sigma_3,\sigma_4]$ is matched with the equivalence class $L_{tmp}^{class} = [\sigma_{abs2}^{45}]$ by satisfying the condition in line 7. The 45 traces in L_{tmp}^{class} are evenly split across the 3 traces of L_a^{class} in line 8: $n_1 = 15, n_2 = 15$, and $n_3 = 15$. Because 45 can be divided without remainder, each split size n_1,... is of equal size and the remainder does not have to be spread across the splits. The enumeration in line 9 of L_a^{class} results in $j \in \{1,2,3\}$. Hence, closestTraces finds the 15 closest traces $\sigma' \in L_{tmp}^{class}$ ($j = 1$, $n_1 = 15$, $\sigma = \sigma_2$) that have the smallest distance $\delta_{kendall}(\sigma',\sigma_2)$ to the current trace σ_2 of L_a^{class} (line 10). Because all traces in L_{tmp}^{class} have the same distance of 0 to σ_2, 15 traces of L_{tmp}^{class} are assigned to $L_{tmp}^{n_1} = [\sigma_{abs}^{15}]$. Subsequently, $L_{tmp}^{n_1}$ is removed from L_{tmp}^{class} (line 11), no transpositions are applied (line 12), and $L_r = [\sigma_1,\sigma_2^{15}]$.

For the second iteration of the for-loop in line 9 ($j = 2$, $n_2 = 15$, $\sigma = \sigma_3$), closestTraces finds the next 15 closest traces $\sigma' \in L_{tmp}^{class}$ that have the smallest distance $\delta_{kendall}(\sigma', \sigma_3) = 1$ to the current trace σ_3 of L_a^{class} (line 10). Because all 30 traces in L_{tmp}^{class} have the same smallest distance of 1 to σ_3, another 15 traces are assigned to $L_{tmp}^{n_2}$ (line 10) and removed from L_{tmp}^{class} (line 11). As the distance greater than zero, the respective transposition to transform each $\sigma' \in L_{tmp}^{n_2}$ into σ_2, i.e., transposing the `AC` with the directly-following `FDD`, are applied in line 12. Hence, $L_{tmp}^{transposed} = [\langle \texttt{RBP},\texttt{AB},\texttt{FDD},\texttt{AC},\texttt{FDD},\texttt{SC},\texttt{AP} \rangle^{15}]$ and $L_r = [\sigma_{abs1}, \sigma_{abs2}^{15}, \langle \texttt{RBP},\texttt{AB},\texttt{FDD},\texttt{AC},\texttt{FDD},\texttt{SC},\texttt{AP} \rangle^{15}]$ in line 13. Analogously, the third iteration applies the two transpositions to transform each σ_{abs2} of the 15 remaining traces in L_{tmp}^{class} into trace σ_4, resulting in $L_r = [\sigma_{abs1}, \sigma_{abs2}^{15}, \langle \texttt{RBP},\texttt{AB},\texttt{FDD},\texttt{AC},\texttt{FDD},\texttt{SC},\texttt{AP} \rangle^{15}, \langle \texttt{RBP},\texttt{AB},\texttt{FDD},\texttt{FDD},\texttt{AC},\texttt{SC},\texttt{AP} \rangle^{15}]$. Obviously, the resulting abstracted event log L_r is df-complete wrt. M_a: $L_a \subseteq L_r$.

Importantly, only the *transpose* edit operation on traces for swapping the order of two directly-following events in a trace is allowed, i.e., any other distance metric on traces cannot be applied here. Since ea_{bpa}^1 substitutes and deletes concrete events of abstract activities, inserts abstract activities for self-loops, and deletes events that are in choice order, it already applies the *substitute*, *delete*, and *insert* edit operations in a controlled manner. Consequently, the perfect matching of equivalence classes between $L_a/\sim$ and $L_{tmp}/\sim$ uniquely exists, because the determinants for the number of equivalence classes in both quotient sets are aligned by ea_{bpa}^1.

First, choice relations are satisfied in the previous step and no loops other than the self-loop can occur and are included in L_{tmp} (cf. Def. 5). Second, L_{tmp} and L_a share exactly the same activities. Third, L_a has fewer traces and is smaller than any concrete event log L from which L_{tmp} was abstracted (cf. Lemma 1), so no equivalence class of L_a can have more traces than the corresponding equivalence class in L_{tmp}. Finally, we point out that transpositions render the timestamp attribute incorrect. To heal the timestamp after transposition, we can either find a timestamp that adheres to the new ordering of events among the concrete events in the "concrete" event attribute (cf. last step) or we must interpolate it and flag it as artificially-created accordingly to avoid confusion during process intelligence.

To sum up, we define $\text{ea}_{bpa} = \text{ea}_{bpa}^2 \circ \text{ea}_{bpa}^1$. In the next section, we prove that ea_{bpa} composed of ea_{bpa}^1 and ea_{bpa}^2 synchronizes ma_{bpa} under restrictions. To that end, we prove the correctness of our design: ea_{bpa} returns df-complete event logs $L_a \subseteq L_r$, i.e., both have the same DFG.

5 Theoretical Foundations

This section establishes the theoretical foundations needed to prove synchronization correctness. We define process tree classes that support isomorphic rediscoverability (Sect. 5.1), propose an algorithm ntl to generate minimal df-complete event logs with size metrics (Sect. 5.2), establish conditions under which our approach produces well-defined results in (Sect. 5.3), and prove that these conditions actually establish well-defined results (Sect. 5.4). For full proofs, we refer to the respective lemma in the appendix Sect. A.

5.1 Process Tree Classes and Rediscoverability

The first step for proving synchronizability is to connect the levels of model abstraction and event logs based on isomorphic rediscoverability. As stated in Sect. 2, process tree M is isomorphic rediscoverable by a process discovery algorithm pd (in this work IM) from event log L iff $M' = pd(L)$ is isomorphic to M. In the following, we will show that any abstracted process tree $M_a = \mathrm{ma}_{bpa}(M)$ is isomorphic rediscoverable under certain model restrictions. These restrictions are summarized by classes of process trees, i.e., C_c and C_a in Def. 4 such that M_a always satisfies the model restrictions of class C_a.

Definition 4 (Class C_c, C_a). *$\oplus(M_1, ..., M_n)$ denotes a node at any position in process tree M. M is in class C_c iff restrictions 1. and 2. are met and in class C_a iff restrictions 1.–3. are met:*

1. *M has no duplicate activities, i.e., $\forall i \neq j : A_{M_i} \cap A_{M_j} = \emptyset$,*
2. *If $\oplus = \circlearrowleft$, then the node is a self-loop, i.e., $\circlearrowleft(v,\tau)$ for some activity $v \in A_M$ (i.e., any other loop $\circlearrowleft(M_1,...M_n)$ is prohibited),*
3. *No τ's outside of the self-loops are allowed: If $\oplus \neq \circlearrowleft$, then $\forall i \leqslant n : M_i \neq \tau$.*

M_a in the running example is in class C_a. These classes differ from standard IM restrictions regarding loop and τ handling, requiring separate rediscoverability proofs. Note that we number the lemmata in this extended version according to our main paper, i.e., lemma 1 and 2 of the main paper are similarly numbered in the extended version.

Lemma 3 (Process trees in C_a are isomorphic rediscoverable). *Let M be a process tree and L be an event log. If M is in class C_a and M and L are df-complete (cf. Sect. 2), i.e., $M \sim_{df} L$, then pd_{IM} discovers a process tree M' from L that is isomorphic to M.*

The proof strategy is to distinguish whether M in C_a contains a self-loop or not. If M does not contain a self-loop, M adheres to the restrictions in [16]. If M contains a self-loop, we extend the base case of the induction in Theorem 14 [16] to also hold for any splitted log $L_v \subseteq [\langle v,v\rangle^m, \langle v,v,v\rangle^n, ...]$ for which the "Strict Tau Loop" fall through discovers the self-loop. Since any M in C_a is isomorphic rediscoverable, what is left to prove is that $M_a = \mathrm{ma}_{bpa}(M)$ is in class C_a.

Lemma 4 (ma_{bpa} abstracted process trees are in C_a). *If ma_{bpa} is applicable to process tree M, then $M_a = \mathrm{ma}_{bpa}(M)$ is in class C_a.*

The main idea of the proof lies in the inability of a behavioral profile p_{M_a} to distinguish whether activities are in a $\wedge$-node or in a $\circlearrowleft$-node. ma_{bpa} handles the inability by always synthesizing a $\wedge$-node for AND-complete modules in the $MDT(G)$. The only τ in M_a can occur due to the additional step that adds a self-loop node to M_a (cf. step 3 in Sect. 4.1). Overall, Lemma 3 and Lemma 4 together imply isomorphic rediscoverability of M_a. Because the isomorphic rediscoverability of M_a is conditioned on df-complete event logs L_a, we generate L_a given M_a using the semantics $\mathcal{L}(M_a)$ in the next section.

5.2 Minimal, Directly-follows Complete Event Logs

Given a process tree M either in C_c or in C_a, there exist countably infinite many df-complete event logs L due to the self-loop node. However, an EA ea_{bpa} must reduce the size of the event log. Therefore, we generate the *minimal*, df-complete (mdf-complete) event log $\mathcal{L}_m(M_a)$ of the countably infinite set of df-complete event logs as a target for our EA technique. Because there are no further $\circlearrowleft$-nodes in M other than self-loop nodes, minimality of $|\mathcal{L}_m(M_a)|$ is equivalent to minimality of $\|\mathcal{L}_m(M_a)\|$. Hence, $\mathcal{L}_m(M_a)$ can be easily computed using the recursive definition of M's language $\mathcal{L}(M)$ for $\oplus$-nodes with $\oplus \in \{\times, \rightarrow, \wedge\}$ and leaves $M = \tau$ or $M = v$. The only difference of $\mathcal{L}_m(M_a)$ compared to $\mathcal{L}(M)$ is the case for the self-loop node $\circlearrowleft(v,\tau)$ that simply assigns the trace $\langle v,v \rangle$.

Algorithm 4 Computing trace number and lengths of $\mathcal{L}_m(M)$: ntl

Require: Process tree M in C_c
Ensure: Number of traces $|\mathcal{L}_m(M)|$ and sequence of trace lengths $\text{lens}(M) = \text{lens}(\mathcal{L}_m(M))$
1: **if** $M = \tau$ **then**
2: **return** $|\mathcal{L}_m(M)| \leftarrow 1$, $\text{lens}(M) \leftarrow \langle 0 \rangle$
3: **else if** $M = \text{v}$ **then**
4: **return** $|\mathcal{L}_m(M)| \leftarrow 1$, $\text{lens}(M) \leftarrow \langle 1 \rangle$
5: **else if** $M = \circlearrowleft(v,\tau)$ for some $v \in A_M$ **then**
6: **return** $|\mathcal{L}_m(M)| \leftarrow 1$, $\text{lens}(M) \leftarrow \langle 2 \rangle$
7: **else if** $M = \times(M_1,...,M_n)$ **then**
8: **return**

$$|\mathcal{L}_m(M)| \leftarrow \sum_{i=1}^{n} |\mathcal{L}_m(M_i)|, \text{lens}(M) \leftarrow \bigodot_{i=1}^{n} \text{lens}(M_i),$$

//where $\odot$ concatenates an ordered collection of sequences
9: **else if** $M = \rightarrow(M_1,...,M_n)$ **then**
10: **return**

$$|\mathcal{L}_m(M)| \leftarrow \prod_{i=1}^{n} |\mathcal{L}_m(M_i)|, \text{lens}(M) \leftarrow \bigodot_{k=1}^{|\mathcal{L}_m(M)|} < \sum_{i=1}^{n} \text{lens}(M_i)[\iota_{k,i}] >$$

//where ι is a bijection $\iota : \{1,...,|\mathcal{L}_m(M)|\} \rightarrow \times_{i=1}^{n} \{1,...,|\text{lens}(M_i)|\}$ and
//$\iota_{k,i} = \pi_i(\iota(k))$ selects the ith element of $\iota(k) = (l_1,...,l_n)$.
11: **else if** $M = \wedge(M_1,...,M_n)$ **then**
12: **return**

$$|\mathcal{L}_m(M)| \leftarrow \sum_{k=1}^{\prod_{i=1}^{n} |\mathcal{L}_m(M_i)|} \binom{m_k}{\bigodot_{i=1}^{n} \langle \text{lens}(M_i)[\iota_{k,i}] \rangle},$$

//where $\binom{m}{\langle l_1,...,l_n \rangle} = \frac{m!}{l_1! * ... l_n!}$ is the multinomial coefficient with $m = \sum_{i=1}^{n} l_i$ and
//ι is a bijection $\iota : \{1,...,\prod_{i=1}^{n} |\mathcal{L}_m(M_i)|\} \rightarrow \times_{i=1}^{n} \{1,...,|\text{lens}(M_i)|\}$

$$\text{lens}(M) \leftarrow \bigodot_{k=1}^{\prod_{i=1}^{n} |\mathcal{L}_m(M_i)|} < \sum_{i=1}^{n} \text{lens}(M_i)[\iota_{k,i}] > ^{\binom{m_k}{\bigodot_{i=1}^{n} \langle \text{lens}(M_i)[\iota_{k,i}] \rangle}}$$

The semantics $\mathcal{L}(M)$ neither provide the number of traces $|\mathcal{L}(M)|$ nor the size $\|\mathcal{L}(M)\|$ generated for arbitrary process trees M in C_c. An existing algorithm $\mathcal{L}'_m$ for computing the number of traces $|\mathcal{L}(M)|$ in [9] is limited compared to our proposed ntl (*n*umber of *t*races and their *l*engths) in Alg. 4 for two reasons. First, ntl′ does not generate the sequence of lengths $\langle |\sigma_1|,...,|\sigma_k| \rangle = \text{lens}(\mathcal{L}_m(M))$ of the $k = |\mathcal{L}_m(M)|$ traces $\sigma_1,...,\sigma_k \in \mathcal{L}_m(M)$ required to compute the log size. Second, $\mathcal{L}'_m$ is limited to $\wedge$-nodes with fixed lengths of traces in the language of its children such that $\times$-nodes

that result in varying lengths cannot be arbitrarily nested with $\wedge$-nodes. Although [9] fix this limitation by transforming the process tree, the resulting process tree duplicates labels such that it violates the restrictions of C_c and C_a, increases the size, and is not isomorphic rediscoverable.

We illustrate $\mathcal{L}_m(M)$ and ntl with M_a of the running example (cf. Fig. 2). Both recur until either a leaf (line 2 and 4) or a self-loop $\circlearrowleft(x,\tau)$ (line 6) is reached. Hence, for each of the six leaves M' with activities RBP, RP, ..., $\mathcal{L}_m(M)$ and ntl reach line 4, e.g., for $M_1 = \texttt{RBP}$, we have $\mathcal{L}_m(M_1) = [\langle\texttt{RBP}\rangle]$, $|\mathcal{L}_m(M_1)| = 1$, and $\text{lens}(M_1) = \langle 1\rangle$. For the self-loop $M_7 = \circlearrowleft(\texttt{FDD},\tau)$, we have $\mathcal{L}_m(M_7) = [\langle\texttt{FDD},\texttt{FDD}\rangle]$, $|\mathcal{L}_m(M_7)| = 1$, and $\text{lens}(M_7) = \langle 2\rangle$. Next, for $M_8 = \wedge(\texttt{AC}, M_7)$, we have $\mathcal{L}_m(M_8) = [\langle\texttt{AC},\texttt{FDD},\texttt{FDD}\rangle,\langle\texttt{FDD},\texttt{AC},\texttt{FDD}\rangle,\langle\texttt{FDD},\texttt{FDD},\texttt{AC}\rangle]$, $|\mathcal{L}_m(M_8)| = \sum_{k=1}^{1}\binom{3}{\langle \text{lens}(M_1)[\pi_1(1,1)],\text{lens}(M_2)[\pi_2(1,1)]\rangle} = \frac{3!}{1!2!} = 3$, and $\text{lens}(M_8) = \langle 3,3,3\rangle$. For $M_9 = \rightarrow(\texttt{AB}, M_8, \texttt{SC})$, we have $\mathcal{L}_m(M_9) = [\langle\texttt{AB},\texttt{AC},\texttt{FDD},\texttt{FDD},\texttt{SC}\rangle,...]$, $|\mathcal{L}_m(M_9)| = 3$, and $\text{lens}(M_9) = \langle 5,5,5\rangle$. Next, for $M_{10} = \times(\texttt{RP}, M_9)$, we have $\mathcal{L}_m(M_{10}) = [\langle\texttt{RP}\rangle,...]$, $|\mathcal{L}_m(M_{10})| = 4$, and $\text{lens}(M_{10}) = \langle 1,5,5,5\rangle$. Finally, $\mathcal{L}_m$ and ntl return $\mathcal{L}_m(M_a) = [\langle\texttt{RBP},\texttt{RP},\texttt{AP}\rangle,...]$, $|\mathcal{L}_m(M_a)| = 4$, and $\text{lens}(M_a) = \langle 3,7,7,7\rangle$ (cf. Sect. 4.2 for the full event log).

Since we know the number of traces $|\mathcal{L}_m(M_a)|$ and the lengths $\text{lens}(M_a)$, we can compute the size $\|\mathcal{L}_m(M_a)\|$ by summing the trace lengths to yield $\|\mathcal{L}_m(M_a)\| = 24$. In Lemma 5, we prove the correctness of ntl.

Lemma 5 (Number of traces and size of $\mathcal{L}_m(M)$.log). *If M is in C_c, then $\mathcal{L}_m(M).tr = |\mathcal{L}_m(M).lens|$ and $\mathcal{L}_m(M).\log = \mathcal{L}_m(M)$ with $|\mathcal{L}_m(M)| = \text{ntl}(M).\text{tr}$ traces and size $\|\mathcal{L}_m(M)\| = \sum_{k=1}^{\text{ntl}(M).tr}\text{ntl}(M).\text{lens}[k]$.*

The proof strategy is to sketch the reasoning for the induction step of a structural induction on process tree M in C_c. The reasoning for the trace lengths of a $\rightarrow$-node is that the lengths of concatenated traces from children is the sum of the respective children's trace lengths as indexed by ι. The reasoning for the number of traces of a $\wedge$-node is to characterize interleaving $\prod_{i=1}^{n}\text{ntl}(M_i).\text{tr}$ different trace combinations of varying trace lengths that are indexed by ι as shuffling of n card decks [2]. Shuffling the first two traces and then iteratively shuffling the next trace into each existing interleaving results in as many interleavings as the multinomial coefficient computes. Similar to $\rightarrow$-nodes, trace lengths are computed except that each individual trace length is repeated as often as there are interleavings corresponding to the kth combination of trace lengths.

It follows that we can compute the number of traces and the size of $\mathcal{L}_m(M_a)$ for any abstracted process tree M_a. Given how the number of traces and the size of $\mathcal{L}_m(M)$ are computed, we order $\oplus$-nodes with respect to both their number of traces and sizes.

Lemma 6 (Operator ordering wrt. their mdf-complete log). *Let $M = \oplus(M_1,...,M_n)$ with $\oplus \in \{\wedge,\rightarrow,\times\}$ be three process trees in C_c. If all children M_i of M have at least two traces, i.e., $|\mathcal{L}_m(M_i)| \geqslant 2, i \in \{1,...,n\}$ then: $|L_\times| < |L_\rightarrow| < |L_\wedge|$ and $\|L_\times\| < \|L_\rightarrow\| < \|L_\wedge\|$ with $L_\oplus = \mathcal{L}_m(\oplus(M_1,...,M_n))$. If all children M_i of M have between one and two traces, i.e., $|\mathcal{L}_m(M_i)| \in \{1,2\}, i \in \{1,...,n\}$, then $|L_\times| \leqslant |L_\wedge|$, $|L_\rightarrow| < |L_\wedge|$, and $\|L_\times\| \leqslant \|L_\rightarrow\| < \|L_\wedge\|$.*

Proof. (Sketch) Let $k_1,...,k_n$ be the number of traces in the $M_1,...,M_n$ children, i.e., $k_i = |\mathcal{L}_m(M_i)|$. For $k_i \geqslant 2$, it holds that $\sum_{i=1}^{n} k_i < \prod_{i=1}^{n} k_i < \sum_{k=1}^{\prod_{i=1}^{n} k_i} \binom{m_k}{\odot_{i=1}^{n} \langle \text{lens}(M_i)[\iota_{k,i}] \rangle}$, because multiplication grows "faster" than summation and the sum of $\prod_{i=1}^{n} k_i$ multinomial coefficients "faster" than $\prod_{i=1}^{n} k_i$ itself. Hence, $|L_\times| < |L_\rightarrow| < |L_\wedge|$. Because the children M_i are the same for $L_\times$,$L_\rightarrow$, and $L_\wedge$, their trace lengths $\text{lens}(M_i)$ are the same such that $\|L_\times\| < \|L_\rightarrow\| < \|L_\wedge\|$ follows.
For $k_i \in \{1,2\}$, the sum can grow "faster" than the product: $\sum_{i=1}^{n} k_i > \prod_{i=1}^{n} k_i$, e.g., if all k_i are equal to 1. Still, $\|L_\times\| \leqslant \|L_\rightarrow\|$, because even if the number of traces is larger for $L_\times$, all the events in $e \in L_\times$ at least occur once in $L_\rightarrow$ in fewer traces instead of in their individual traces. The sum of $\prod_{i=1}^{n} k_i$ multinomial coefficients can be equal to the $\sum_{i=1}^{n} k_i$, e.g., if all k_i are equal to 1. Nevertheless, the sum $\sum_{i=1}^{n} k_i$ does not exceed the sum of multinomial coefficients: $\sum_{i=1}^{n} k_i \leqslant \sum_{k=1}^{\prod_{i=1}^{n} k_i} \binom{m_k}{\odot_{i=1}^{n} \langle \text{lens}(M_i)[\iota_{k,i}] \rangle}$. $\prod_{i=1}^{n} k_i$ only "grows", if $k_i = 2$ occurs "often". Yet, $k_i = 2$ occurring "often" results in a large factorial in the multinomial coefficient. Hence, $\prod_{i=1}^{n} k_i < \sum_{k=1}^{\prod_{i=1}^{n} k_i} \binom{m_k}{\odot_{i=1}^{n} \langle \text{lens}(M_i)[\iota_{k,i}] \rangle}$.
To sum up, $|L_\times| \leqslant |L_\wedge|$ and $|L_\rightarrow| < |L_\wedge|$. From $|L_\rightarrow| < |L_\wedge|$ and the observation that the trace lengths in $L_\rightarrow$ and $L_\wedge$ are equal except for their multiplicities in $\text{ntl}(M)$.lens (cf. Algorithm 3 line 10 and line 12 in the main paper), it follows that $\|L_\rightarrow\| < \|L_\wedge\|$. Even if $|L_\times| = |L_\wedge|$, still $\|L_\times\| < \|L_\wedge\|$, because the $\text{lens}(\wedge(M_1,...,M_n)$ in line 12 of Algorithm 3 in our main paper (cf. ntl Algorithm 3) adds the children's trace lengths $\sum_{i=1}^{n} \text{lens}(M_i)[\iota_{k,i}]$ before repeating them as often as there are interleavings.

The $\oplus$-node ordering is important for synchronizability. Any non-order-preserving MA must ensure that abstract activities in M_a are not children of a greater $\oplus$-node than the $\oplus'$-node of M in which the corresponding abstracted activities are children of to avoid that the mdf-complete event log $\mathcal{L}_m(M_a)$ becomes larger than the mdf-complete event log $\mathcal{L}_m(M)$. Hence, if a MA technique fails to consider the $\oplus$-node ordering, there does not exist a EA technique that synchronizes. In the next section, we present the restrictions necessary to ensure that the mdf-complete event log $\mathcal{L}_m(M_a)$ of $M_a = \text{ma}_{bpa}(M)$ is smaller than the mdf-complete event log $\mathcal{L}_m(M)$ of M.

5.3 Event Log Restrictions

For our EA technique to be well-defined, the df-complete event log L_a for the abstracted process model must not be larger than what the concrete event log L can support.

Definition 5 (Restricted event log). *Event log L is restricted iff* pd_{IM} *discovers a process tree* $M = \text{pd}_{IM}(L)$ *such that:*

1. ***Fall throughs:*** pd_{IM} *has only executed the "Strict Tau Loop" to discover a self-loop* $\circlearrowleft(v,\tau)$ *for* $v \in A_L$ *and the "Empty Traces"*[8]*,*
2. ***Cuts:*** pd_{IM} *has only executed choice, parallel, and sequence cuts*[9]*,*

[8] Hence, no nested tau loops $\circlearrowleft(M_1,\tau), M_1 \neq v$ are discovered by "Strict Tau Loop" and no fall throughs "Activity Once Per Trace", "Activity Concurrent", "Tau Loop", and "Flower Model" are executed.

[9] Hence, no loop cut is found.

3. ***Base cases:*** pd_{IM} *has executed any number of base cases, and*
4. ***Model structure:*** *If* $\oplus(M_1,...,M_n)$ *is a node in* M *with at least one child being an activity or a self-loop, i.e.,* $M_i = v$ *or* $M_i = \circlearrowleft(v,\tau)$ *for* $v \in A_L$ *and* $i \in \{1,...,n\}$*, then* $\oplus = \times$ *or* $\oplus = \wedge$*.*

We motivate and illustrate each restriction imposed on a event log L by giving relaxed restrictions and counterexamples.

Fall throughs

"Activity Once Per Trace": The syn (cf. step 3 Section 3.2 in our main paper and Algorithm 3.2 in [37]) of process trees from an abstracted behavioral profile p_{M_a} can be the reason for larger mdf-complete event logs $L_a = \mathcal{L}_m(M_a)$. The "Activity Once Per Trace" can discover process trees M whose mdf-complete event logs significantly exceed the number of traces and size of the original event log L. Therefore, the IM must not execute this fall through during discovery of the abstracted process tree M_a from an abstracted event log L_a.

Counterexample: $L = \{\langle a,b\rangle,\langle e,c,d\rangle,\langle d,e,c\rangle\}$, $\mathrm{pd}_{IM}(L) = M = \times(\rightarrow(a,b),\wedge(e,\wedge(c,d)))$ Obtain $\mathrm{ma}_{bpa}(M) = M_a = \times(x,\wedge(c,d,e))$ for $\mathrm{agg}(x) = \{a,b\}$ and $w_t \leqslant 0.5$ with $L_a = \{\langle x\rangle,\langle c,d,e\rangle,\langle c,e,d\rangle,...,\langle e,d,c\rangle\}$. The number of traces $|L_a| = 7$ and the size is $\|L_a\| = \sum_{k=1}^{7}\langle 1,3,3,3,3,3,3\rangle[k] = 19$ and, thus, both the number of traces and the size of L_a exceed that of L respectively. Hence, no ea exists that can synchronize to ma_{bpa}.

"Activity Concurrent": The following counterexamples exploits the property of ma_{bpa} to change the behavior of the process tree M even for activities that are not abstracted:

Counterexample: $L = \{\langle a,b\rangle,\langle f,c,d,e\rangle,\langle c,d,e,f,f\rangle,\langle c,d,e,f\rangle\}$ and $\mathrm{pd}_{IM}(L) = M = \times(\rightarrow(a,b),\wedge(\times(\circlearrowleft(f,\tau),\tau),\rightarrow(c,d,e)))$. Set $\mathrm{agg}(x) = \{a,b\}$ and $w_t \leqslant 0.5$ to obtain $M_a = \times(x,\wedge(\circlearrowleft(b,\tau),\rightarrow(c,d,g))$ with $L_a = \{\langle x\rangle,\langle b,b,c,d,g\rangle,\langle b,c,b,d,g\rangle,...,\langle g,d,c,b,b\rangle\}$. Both the number of traces and the size of L_a exceeds that of L respectively, as $|L_a| = 11 > 4 = |L|$ and $\|L_a\| = \sum_{k=1}^{11}\langle 1,5,...,5\rangle[k] = 51 > 15 = \|L\|$.

Fall throughs, Cuts (Loop)

"Strict Tau Loop", "Tau Loop", "Flower Model", "Loop Cut": In Lemma 4, we have established that any M_a is in class C_a, i.e., does not contain any $\circlearrowleft$-node other than a self-loop due to the indistinguishability of a $\circlearrowleft$-node and a $\wedge$-node in the BP p_{M_a}. The same property of BPs implies that any $\circlearrowleft$-node $M_{\circlearrowleft}$ in M is replaced by a $\wedge$-node, if all of the activities $v \in \mathcal{A}_{M_{\circlearrowleft}}$ are not abstracted by ma_{bpa} ($A_{M_{\circlearrowleft}} \subseteq A_c$ in Definition 2 in the main paper). Through two counterexamples, we show that if M contains a node $M_{\circlearrowleft}$ that is not a self-loop node $\circlearrowleft(v,\tau)$ for $v \in A_L$, it is neither guaranteed that ma_{bpa} is a MA nor is the mdf-complete event log L_a necessarily smaller than L such that synchronizability becomes impossible. To discover a self-loop both for M and for M_a, either the "Strict Tau Loop" or the "Tau Loop" with a restriction on only discovering a self-loop are suitable. Since "Strict Tau Loop" exactly matches the $\langle v,v\rangle$ directly-follows pattern in an event log, we opt for the "Strict Tau Loop" in Def. 5.

Tau Loop Counterexample: $L = \{\langle a,c,a\rangle,\langle b,a\rangle,\langle d,e\rangle\}$ and $M = \mathrm{pd}_{IM}(L)$ $= \times(\circlearrowleft(\times(b,\rightarrow(a,\times(c,\tau))),\tau),\rightarrow(d,e))$. Apply ma_{bpa} for $\mathrm{agg}(x) = \{d,e\}$ and $w_t \leqslant 0.5$ to obtain the abstracted model $M_a = \times(\wedge(\circlearrowleft(a,\tau),\circlearrowleft(b,\tau),\circlearrowleft(c,\tau)),x)$ with $L_a = \{\langle a,a,c,c,b,b\rangle,...,\langle c,c,b,b,a,a\rangle,\langle x\rangle\}$. Hence, $|L_a| = \binom{6}{\langle 2,2,2\rangle} + 1 = 91 > 3 = |L|$ and $\|L_a\| = \sum_{k=1}^{91}\langle 1,6,...,6\rangle[k] = 541 > 7 = \|L\|$.

Flower Model Counterexample: $L = \{\langle a,b\rangle,\langle a,b,c\rangle,\langle c,a\rangle,\langle d,e\rangle\}$ and $\mathrm{pd}_{IM}(L) = M = \times(\circlearrowleft(\times(a,b,c),\tau),\rightarrow(d,e))$[10]. Apply ma_{bpa} for $\mathrm{agg}(x) = \{d,e\}$ and $w_t \leqslant 0.5$ to obtain the abstracted model $M_a = \times(\wedge(\circlearrowleft(a,\tau),\circlearrowleft(b,\tau),\circlearrowleft(c,\tau)),x)$ with $|M_a| = 12 > 10 = |M|$, i.e., ma_{bpa} is not a MA for process trees M that contain a flower model.
Loop Cut Counterexample: $L = \{\langle a,b,a\rangle,\langle c,d,e\rangle\}$ and $M = \mathrm{pd}_{IM}(L) = \times(\circlearrowleft(a,b),\rightarrow(d,e,f))$. Apply ma_{bpa} for $\mathrm{agg}(x) = \{c,d\}$ and $w_t \leqslant 0.5$ to obtain the abstracted model $M_a = \times(\wedge(a,b),\rightarrow(x,f))$ with $L_a = \{\langle a,b\rangle,\langle b,a\rangle,\langle x,f\rangle\}$. Hence, $|L_a| = 3 > 2 = |L|$ and $\|L_a\| = 6 = \|L_a\|$.

Model structure

The $\mathrm{dv}_{w_t,\mathrm{agg}}$ (cf. Algorithm 3.1 in [37]) has a static order of returning ordering relations for abstracted activities x and y: First, a choice order $x +_{M_a} y$ is returned (line 12), then the inverse strict order $x \rightsquigarrow^{-1}_{M_a} y$ (line 15 and line 19) followed by the strict order $x \rightsquigarrow_{M_a} y$ (line 17) and, finally, the parallel order $x \parallel_{M_a} y$ is returned (line 21), as soon as the threshold w_t is below the respective relative frequency and an "earlier" order relation is not returned already ($\circledcirc$). As long as a node $M' = \rightarrow(M_1,...,M_n)$ in M has only children M_i whose number of traces in the corresponding mdf-complete event log $\mathcal{L}_m(M_i)$ are all at least 2, the static order in $\mathrm{dv}_{w_t,\mathrm{agg}}$ aligns with the order of operators in a process tree as shown in Lemma 6. However, if M' has a child M_i whose number of traces is below 2, returning a choice order $x +_{M_a} y$ can increase the number of traces in L_a, as the order of operators with respect to a sequence node and a choice node depends on the children $M_1,...,M_n$ (cf. Lemma 6). As $\mathrm{dv}_{w_t,\mathrm{agg}}$ does not dynamically depend on the children $M_1,...,M_n$ of a node M' in the process tree M, we must exclude M' that has a child M_i with less than two traces.
Counterexample: $L = \{\langle a,b,c\rangle\}$ and $M = \mathrm{pd}_{IM}(L) = \rightarrow(a,b,c)$. Apply ma_{bpa} for $\mathrm{agg}(x) = \{a,c\}$ and $w_t \leqslant 0.5$ to obtain $M_a = \times(v,b)$ with $L_a = \{\langle v\rangle,\langle b\rangle\}$. Although $\|L_a\| = 2 < 3 = \|L\|$, the number of traces increases $|L_a| = 2 > 1 = |L|$.

For example, the model structure restriction prohibits that L_a has more traces than $\mathcal{L}_m(M)$: $|L_a| > |\mathcal{L}_m(M)|$. Because the process tree $M = \mathrm{pd}_{IM}(L)$ discovered from restricted event log L is in class C_c (cf. Def. 4), we can generate mdf-complete event logs for M[11]. Since ma_{bpa} can change the order of sequentially-ordered activities ($\rightsquigarrow_M$) to choice-ordered activities ($+_{M_a}$) given that $\mathrm{dv}_{\mathrm{agg},w_t}(x,y)$ prioritizes $+$ over $\rightsquigarrow$ (cf. Sect. 4.1), the number of traces in L_a can become larger than the number of traces in L: $|L_a| > |L|$ (compare line 8 and line 10 in Alg. 4). The model structure restriction could be avoided, if we allow an EA ea to split traces of the concrete event log L. Nevertheless, an abstraction should intuitively maintain the semantics of the event log while reducing its complexity (in terms of discovered model size). Splitting traces, however, means that during abstraction new process instances can be created, which is clearly not desired during abstraction. Thus, we can either prohibit the model structure or restrict what activities can be aggregated, i.e., restrict agg, to avoid more traces in L_a.

For example, M in Fig. 2 violates the model structure restriction, but agg as depicted constitutes a case for which $|L_a| < |L|$. Yet, a different function agg′ that

[10] "Strict Tau Loop" is already restricted to only discover a self-loop and "Tau Loop" excluded entirely
[11] We prove that M is in C_c in Lemma 11.

only abstracts the start `RBP` and end activity `AT` would result in $|L_a| > |L|$. We opt to prohibit the model structure, because we aim to synchronize ma_{bpa} with an EA technique and not apply further modifications to ma_{bpa}. Through the restrictions on an event log, we prove that L can always support the mdf-complete event log L_a required for well-definedness of ea_{bpa} in the next section.

5.4 Well-definedness

In this section, we prove the necessary condition for synchronization: Event log L can never become smaller than L_a. If the opposite would hold, we could not define an EA technique, because the EA technique would need to either insert new traces and or events or both into L for df-completeness.

To prove the necessary condition, we must consider under what parametrization agg and w_t the $L_a = \mathcal{L}_m(M_a)$ with $M_a = \text{ma}_{bpa}(M)$ becomes maximal. Hence, we prove for what parameters $\mathcal{L}_m(M_a)$ becomes maximal.

Lemma 7 (Maximal $\mathcal{L}_m(M_a)$). *If L is restricted and ma_{bpa} applicable to $M = \text{pd}_{IM}(L)$, $w_t = w_{minmax}$, $\text{agg}(x) = \{v,u\}$, and $\text{agg}(z) = \{z\}$ with $z \in A \backslash \{v,u\}$, i.e., ma_{bpa} aggregates exactly two activities into x, then $M_a = \text{ma}_{bpa}(M)$ has the most traces and maximal size of all $\mathcal{L}_m(M'_a)$ generated for other M'_a with $A'_c \cap \{v,u\} = \varnothing$ and $|\bigcup_{y \in A'_{new}} \text{agg}'(y)| > |\bigcup_{y \in A_{new}} \text{agg}(y)|$ (cf. Def. 3).*

The main idea of the proof is to compare the behavioral profile p_{M_a} with all behavioral profiles $p_{M'_a}$, as the respective process trees M_a and M'_a are similarly synthesized given the BP. Because $p_{M'_a}$ can only contain less concrete activities $q \in A_c$ than p_{M_a} and both v and u must also be abstracted in $p_{M'_a}$, the only cause for more traces or events in $\mathcal{L}_m(M'_a)$ can be due to different order relations in $p_{M'_a}$. Yet, the restricted event log L aligns the priorization of $\text{dv}_{\text{agg},w_t}$ to return order relations from $+$ to $\|$ (cf. Sect. 4.1) with the operator ordering in Lemma 6. Hence, $p_{M'_a}$ can only contain order relations that imply less traces and events in $\mathcal{L}_m(M'_a)$.

Given the restrictions on event log L, we show that L_a for the parameters that maximize it, is always smaller than L.

Lemma 1 ($\mathcal{L}_m(M_a)$ is smaller). *If L is restricted and ma_{bpa} is applicable to $M = \text{pd}_{IM}(L)$, then $L_a = \mathcal{L}_m(M_a)$ has fewer traces and is smaller than L: $|L_a| < |L|$ and $\|L_a\| < \|L\|$.*

The proof strategy is to apply induction on the size of $M = \text{pd}(L)$ and compare the mdf-complete event log $\mathcal{L}_m(M)$ with the maximal mdf-complete event log $\mathcal{L}_m(M_a)$ for M_a abstracted through parameters $\text{agg}(x) = \{v,u\}$ and $w_t = w_{minmax}$. Only abstracting two concrete activities and setting $w_t = w_{minmax}$ maximizes $\mathcal{L}_m(M_a)$ (cf. Lemma 7). Hence, we take the smallest representative on the larger side L and the largest representative on the smaller side L_a. For the induction step we apply a case distinction on the operator in $M = \oplus(...)$ with $\oplus \in \{\times, \rightarrow, \wedge\}$. In all cases, if v and u both occur in the same child of M, the statement follows by the induction hypothesis (IH). We exploit the semantical indifference of M's children order for $\times$- and $\wedge$-nodes and the symmetry of the $+$ and $\|$ order relations to decompose $M_a = \times(...)$ into

children that are not abstracted and the two children that are abstracted. Through the (IH) and the language-preserving reduction of process trees into their *normal form* [15], we prove the statement for decomposed $\times$- and $\wedge$-nodes. In case of a $\rightarrow$-node, we derive two different tree structures for M_a that are shown to have fewer traces and events through a combinatorial proof given the respective equations in Alg. 4.

On the grounds of Lemma 1, we establish the correctness of the synchronized EA technique in Sect. 6.1. Hence, we prove the main synchronization result by correctness of the EA technique in Sect. 6.2.

6 Synchronization Proof

We prove the main synchronization result in Sect. 6.2 by showing that our EA technique produces df-complete logs that enable correct model rediscovery in Sect. 6.1. For full proofs, we refer to the respective lemma in the appendix Sect. A.

6.1 Correctness of the EA Technique

In Sect. 5.4, we have established that the mdf-complete event log $L_a = \mathcal{L}_m(M_a)$ has strictly less traces and is strictly smaller than any event log L. Thus, a well-defined EA technique exists. As ea_{bpa} consists of two steps ea^1_{bpa} and ea^2_{bpa}, we show that these two steps are correctly specified, i.e., that ea_{bpa} is a well-defined EA technique that returns df-complete event logs $L_r = \text{ea}_{bpa}(L)$ for the abstracted process tree $M_a = \text{ma}_{bpa}(\text{pd}_{IM}(L))$: $L_a \subseteq L_r$. To that end, we define the transposition equivalence, because ea^2_{bpa} is specified on the grounds of this equivalence relation.

Definition 6 (Transposition equivalence). *Let $\sigma_1, \sigma_2 \in \mathcal{A}^*$ be two traces and $\delta_{kendall}$ [4] be the minimum number of transpositions required to be applied to σ_1 yielding σ_1' such that $\sigma_1' = \sigma_2$, if both σ_1 and σ_2 are permutations of the multiset of activities $\text{bag}(\sigma_1) = \text{bag}(\sigma_2)$ with $\text{bag}(\sigma) = [\sigma[i] \mid i \in \{1,...,|\sigma|\}]$, and undefined $\delta_{kendall} = \perp$ otherwise. The transposition relation $\sim \in \mathcal{A}^* \times \mathcal{A}^*$ is defined by*

$$\sigma_1 \sim \sigma_2 \text{ iff } \delta_{kendall}(\sigma_1, \sigma_2) \neq \perp.$$

Trivially, $\sim$ is an equivalence relation. Because ea^2_{bpa} searches for a matching between equivalence classes of $L_{tmp} = \text{ea}^1_{bpa}(L)$'s quotient set modulo $\sim$ and equivalence classes of L_a's quotient set modulo $\sim$, we formalize what the matching is.

Definition 7 (Matching). *Let L_1, L_2 be two event logs. Given the transposition equivalence $\sim$, a matching between the two quotient sets $L_1/\sim$ and $L_2/\sim$ is a bijective function $\text{matching}: L_1/\sim \rightarrow L_2/\sim$ such that:*

- *for every equivalence class $L_1^{class} \in L_1/\sim$ and corresponding equivalence class $\text{matching}(L_1^{class}) = L_2^{class}$ it holds: $\exists \sigma_1 \in L_1^{class}, \sigma_2 \in L_2^{class}$ with $\sigma_1 \sim \sigma_2$, i.e., the respective traces in matched equivalence classes are not distinguishable modulo $\sim$, i.e., not distinguishable modulo transposition.*

It does not matter whether we define the equivalence of traces in matched pairs of equivalence classes by existence or by universal quantification, as from existence and equivalence the universal quantification directly follows. We use the existence in Def. 7 to emphasize that we only need to check equivalence once for two traces to decide equivalence. Observe that the condition for pairs of equivalence classes in the matching is checked in the if-condition of ea^2_{bpa} (line 7 Alg. 3). In the following, we prove that a matching between the two quotient sets $L_{tmp}/\sim$ and $L_a/\sim$ exists and that the even split of traces from equivalence classes in the former can always be assigned to traces from equivalence classes in the latter quotient set.

Lemma 8 (Matching of quotient sets). *If L is restricted and ma_{bpa} applicable to $M = \mathrm{pd}_{IM}(L)$, there exists a matching between the quotient set $L_{tmp}/\sim$ of event log $L_{tmp} = \mathrm{ea}^1_{bpa}(L, \mathrm{pd}_{IM}, \mathrm{ma}_{bpa})$ and the quotient set $L_a/\sim$ of $L_a = \mathcal{L}_m(M_a)$ for $M_a = \mathrm{ma}_{bpa}(\mathrm{pd}_{IM}(L))$, i.e., $\mathrm{matching}: L_1/\sim \rightarrow L_2/\sim$ exists. Also, for every matched pair of equivalence classes $L^{class}_{tmp} \in L_{tmp}/\sim$ and $L^{class}_a = \mathrm{matching}(L^{class}_{tmp})$ it holds that L^{class}_{tmp} has equal to or more traces than L^{class}_a: $|L^{class}_a| \leqslant |L^{class}_{tmp}|$ (ii).*

The proof strategy is to establish (i) the existence of a matching between the quotient set $L_{tmp}/\sim$ of L_{tmp} and the quotient set $L_a/\sim$ of the reference event log L_a modulo transposition, and (ii) that for any matched pair of equivalence classes L^{class}_a and L^{class}_{tmp} from $L_a/\sim$ and $L_{tmp}/\sim$ respectively it holds: $|L^{class}_a| \leqslant |L^{class}_{tmp}|$. We prove (i) by contradiction through the restrictions on an event log (cf. Def. 5), structural properties on abstracted process trees M_a (e.g., no loops other than the self-loop), and by code inspection of ea^1_{bpa}. Hence, the search for a matching in line 5-7 of Alg. 3 is correctly specified. We prove (ii) by (i), the minimality of L_a, and Lemma 1.

It follows that the transposition with even splits (line 8-13) returns L_r such that $L_a \subseteq L_r$:

Lemma 2 (ea_{bpa} returns df-complete logs). *If L is restricted and ma_{bpa} applicable to $M = \mathrm{pd}_{IM}(L)$, the event log $L_r = \mathrm{ea}_{bpa}(L, \mathrm{pd}_{IM}, \mathrm{ma}_{bpa})$ is a df-complete event log for $M_a = \mathrm{ma}_{bpa}(\mathrm{pd}_{IM}(L))$.*

Proof. From Lemma 8 and the applied transpositions in line 12 to satisfy every directly-follows relation in L_a by matched sets of traces from their respective equivalence class (line 10 and line 7), it follows that for every trace $\sigma \in L_a$ there exists a trace $\sigma_{abs} \in L_r$ such that $\sigma = \sigma_{abs}$.

Hence, we can prove the synchronizability as required by our approach.

6.2 Main Synchronization Theorem

Given the correctness of the EA technique ea_{bpa} and the isomorphic rediscoverability of abstracted process trees M_a, we prove the main synchronization result.

Theorem 1 (IM and ma_{bpa} are synchronizable).
If L is restricted and ma_{bpa} applicable to $M = \mathrm{pd}_{IM}(L)$, then $\mathrm{pd}_{IM}(L_a)$ discovered from $L_a = \mathrm{ea}_{bpa}(L, \mathrm{pd}_{IM}, \mathrm{ma}_{bpa})$ is isomorphic to $\mathrm{ma}_{bpa}(\mathrm{pd}(L))$.

Proof. From Lemma 2, it follows that event log L_a is a df-complete event log for $M_a = \mathrm{ma}_{bpa}(\mathrm{pd}_{IM}(L))$. Since M_a is isomorphic rediscoverable (cf. Lemma 4), it follows that: $\mathrm{pd}_{IM}(L_a) \cong M_a$.

Therefore, we can abstract discovered process model M through non-order-preserving ma_{bpa} and synchronously abstract L through ea_{bpa} such that both M_a and L_a are available for further process intelligence tasks. In the next section, we revisit the illustrative example in Fig. 1 to show what impact the main synchronization theorem has.

7 Demonstration on the Illustrative Example

To put the theoretical results of synchronization into perspective, we demonstrate their impact on the illustrative scenario that we have introduced in Sect. 1 and Fig. 1. To that end, we show how the event log L (cf. Fig. 1 ①) that contains traces of the bank's trading process for two different products (derivative and fixed income) is abstracted by ea_{bpa} in Sect. 7.1. Recall that the bank already applied the BPA model abstraction ma_{bpa} on the discovered process model M, as it expects the result M_a to have a more suitable granularity for analyzing the common trading process than M. Subsequently, we delineate how the abstracted event log $L_a = \mathrm{ea}_{bpa}(L)$ and M_a help the bank in analyzing the common trading process through process intelligence in Sect. 7.2.

7.1 Abstracting the Event Log

To begin with, BPA ma_{bpa} with $A_{new} = \{\mathtt{RQ}, \mathtt{OT}, \mathtt{N}, \mathtt{CT}\}$, $w_t = w_{minmax} = 5/9$, and agg as depicted in Fig. 1 ③ is applicable to M. We illustrate ea^1_{bpa} (cf. Alg. 2) with the illustrative example. Let L be an event log such that IM discovers M as depicted. For simplicity, we assume that L is a minimal df-complete event log. Due to irrelevance of further event attributes for proofs (cf. Sect. 5-Sect. 6), we only represent activities in traces, but discuss how further attributes can be added. For example $\sigma_1 = \langle \mathtt{OLS}, \mathtt{TLS}, \mathtt{TOS}, \mathtt{C}, \mathtt{T}, \mathtt{T}, \mathtt{C}, \mathtt{TCS} \rangle$ as denoted by "Tr." in Fig. 1 ① is a trace. To start, we have $L = [\sigma_1, \sigma_2, ..., \sigma_9]$, $|L| = 9$, and $\|L\| = 6*8+5+2*2 = 57$. Because in a minimal df-complete event log each activity in a self-loop is repeated once, L contains the six interleavings of $\langle \mathtt{T}, \mathtt{T} \rangle$ and $\langle \mathtt{C}, \mathtt{C} \rangle$ in $\sigma_1, \sigma_3, ..., \sigma_7$ plus σ_2 and plus the two traces $\sigma_8 = \langle \mathtt{GO}, \mathtt{RO} \rangle$ and $\sigma_9 = \langle \mathtt{OLS}, \mathtt{ODS} \rangle$. Given $A_{new} = A_{M_a}$ (cf. Fig. 1 ⑤ and Def. 3), ea^1_{bpa} always computes $A_{\mathrm{ma}} = \{\sigma[1], ..., \sigma[|\sigma|]\}$, i.e., all events are abstracted (line 4). Thus, $\sigma_1[1]$ is the first event abstracted by $\mathtt{RQ} \in A_{new}$ (line 8) and the condition in line 9 is always true, because there exists no concrete event v.

Next, for the two events OLS and TLS, the single abstract event OT is added due to the first appearance condition in line 8. The abstract event N for the four events C,T,T,C is repeated, because $\mathtt{N} \parallel_{M_a} \mathtt{N}$ holds and triggers adding a self-loop (cf. Sect. 4.1) that can be rediscovered by repeating N once. Altogether, we have $\sigma_{abs1} = \langle \mathtt{RQ}, \mathtt{OT}, \mathtt{N}, \mathtt{N}, \mathtt{CT} \rangle \in L_a$ (cf. Fig. 1 ④) and analogously for the remaining eight traces resulting in $L_{tmp} = [\sigma^7_{abs1}, \sigma^2_{abs8}]$.

Before we proceed with line 16, we point out how further attributes can be added. In line 10, new abstract events x are created. Here, any attribute of the current event e or any aggregation function over the set of x's concrete events $A_{\text{ma}}(x) = \{e \in \sigma \mid e \in \text{agg}(x)\}$ can be copied or computed. For instance, we have applied a *last attribute aggregation* with *even time split* for repeated x on every attribute in Fig. 1 ④: Because $A_{\text{ma}}(\texttt{N}) = \{\texttt{T},\texttt{C}\}$, the four corresponding events e's attributes in σ_1 are taken together, evenly split[12] by respecting time, and the respective last attribute is assigned to `N`. The result shows that the event with ID "a3" Fig. 1 ④ has timestamp t_5 and the "Terms" value of the event with ID "5" in Fig. 1 ①. Yet, proposing and evaluating different attribute aggregations goes beyond the scope of this paper. Still, we advocate adding the "concrete" event attribute to every abstract event for storing the respective $A_{\text{ma}}(x)$ and, thus, maintaining flexibility.

Because $\texttt{DQ} +_{M_a} y \in p_{M_a}$ (cf. the initial XOR-gateway in Fig. 1 ⑤) for any other activity $y \in C := A_{M_a} \backslash \{\texttt{RQ}\}$, it holds that $A_\times = \{C\}$ in line 17, i.e., module C is the only XOR-complete module in $MDT(G)$. Nevertheless, the corresponding concrete events `RO` and `DCS` never occur together with any concrete event $e \in A_{\text{ma}}(y)$ of other abstract activities $y \in C$ in any trace $\sigma \in L$, because they were in choice relation already. To illustrate, for σ_2, we have $y = \texttt{OT}, A_{\text{ma}}(y) = \{\texttt{OW}\}$ and `OW` never occurs with `RO` or `ODS` (cf. Fig. 1 ③). Hence, no events are deleted in line 17 and ea^1_{bpa} returns $L_{tmp} = [\sigma^7_{abs1}, \sigma^2_{abs8}]$ that contains 9 traces.

To continue the illustration, $\mathcal{L}_m$ computes $L_a = [\sigma_{abs1}, \sigma_{abs8}]$ with $\sigma_{abs8} = \langle \texttt{RQ},\texttt{DQ} \rangle$ (line 1) such that $L_a / \sim = \{[\sigma_{abs1}], [\sigma_{abs8}]\}$ is the quotient set of L_a by $\delta_{kendall}$ (line 3). Likewise, $L_{tmp} / \sim = \{[\sigma^7_{abs1}], [\sigma^2_{abs8}]\}$ is the quotient set of L_{tmp} (line 4). Next, we iterate over equivalence classes of both quotient sets (line 5-6). First, $L^{class}_a = [\sigma_{abs1}]$ and $L^{class}_{tmp} = [\sigma^7_{abs1}]$ share traces that are permutations of the same multiset of activities, i.e., the condition in line 7 is true. Because $|L^{class}_{tmp}| / |L^{class}_a| = 7/1 = 7$ is an integer quotient, no remainder is left to be evenly spread across the $n_1, \ldots, n_{|L^{class}_a|}$ sizes of splits, i.e., $n_1 = 7$ (line 8). Given the split sizes, the jth trace $\sigma \in L^{class}_a$ is the reference for the n_j closest traces in L^{class}_{tmp} in terms of number of transpositions ($\delta_{kendall}$) (line 10).

Due to $n_1 = 7$ and the single trace in L^{class}_a, the closestTraces operator assigns all seven traces in L^{class}_{tmp} with a distance of 0 transpositions to $L^{n_1}_{tmp}$. Consequently, L^{class}_{tmp} is empty afterwards (line 11), no transpositions must be applied to traces $L^{transposed}_{tmp} = L^{n_1}_{tmp}$ (line 12), and the seven traces in $L^{n_1}_{tmp}$ are added to the result L_r (line 13): $L_r = [\sigma^7_{abs1}]$. In Fig. 1 ④, the first two traces of the σ_{abs1} trace variant are depicted. Analogously, the second for-loop iteration (line 5) yields $L_r = [\sigma^7_{abs1}, \sigma^2_{abs8}]$, i.e., no transpositions were necessary overall.

To sum up, the abstracted event log L_r maintains the multiplicities of the concrete event log L and the respective distributions of values in the additional event attributes. In particular, the abstracted event log L_r is equivalently related to M_a as the concrete L was related to M. In the next section, we demonstrate the impact of having both L_a and M_a for subsequent process intelligence.

[12] Ties can be decided towards the first abstract event.

7.2 Impact on Process Intelligence

Coming back to the bank scenario in Fig. 1, correct synchronized abstraction of both M and L into M_a (cf. Fig. 1 ⑤) and L_a (cf. Fig. 1 ④) allows the bank to analyze the common trading process of both products simultaneously with, e.g., process enhancement (cf. Fig. 1 ⑧). To that end, L_a allows to enhance M_a with the frequency information that seven trades negotiated the terms in particular wrt. the "spread" as recorded in basis points (bp). Here, the bank is most interested in the question: "Does the client and requested product affect the final terms after renegotiation?". To answer the question, the bank must define a classification problem that relates the `OT`'s event attribute value for attributes "Client" and "Product" to the last event attribute value of the second `N` event in each trace that has the `CT` event as the end event. Because the synchronized EA technique ea_{bpa} enables to automatically compute the abstracted L_a that contains the seven trades corresponding to the abstracted process model M_a, the bank can define the classification problem given L_a and M_a.

Having only the abstracted process model M_a, the bank is neither able to enhance the abstracted process model with further perspectives for, e.g., presenting different aspects of the process to different stakeholders in the bank, nor is it able to define the classification problem, as it would lack the abstracted event log L_a. Moreover, it can start predicting and simulating the common trading process by learning both a simulation and prediction model through L_a and M_a. To sum up, the bank is able to flexibly apply a similar set of process intelligence tasks on an abstracted process model and abstracted event log as it was used to on the concrete event log and process model.

8 Conclusion

We propose a novel synchronization approach that closes the gap between MA and EA techniques. Consequently, we can apply MA technique ma on discovered process models M and compute the corresponding abstracted event log L_a through the synchronized EA technique ea_{ma}. The proposed approach is the first to formalize the impact of EA techniques on the discovered process model and the first to enable process intelligence tasks grounded in the real-world behavior contained in L_a. So far, our approach is limited to the IM and its relatively simple event log conceptualization. Furthermore, our approach focuses on size as an indicator of complexity and only briefly discusses event attribute abstractions that go beyond the control-flow, but are driven by the data flow. In future work, we will investigate extended conceptualization of event logs. Second, we will extend the set of process discovery techniques and integrate optimization frameworks for MA that capture further complexity metrics. Third, we will parametrize synchronized EA techniques to allow for even more faithful abstraction of event logs.

Appendix

The appendix includes Lemma 1 and Lemma 2 from the main paper with full proofs. Moreover, the appendix includes the additional Lemma 3, Lemma 4, Lemma 5, Lemma 7, and Lemma 8 from this extended version with full proofs. In addition, we report a simple helper lemma in Lemma 11.

A Lemmata With Full Proofs

Lemma 3 (Process trees in C_a are isomorphic rediscoverable). *Let M be a process tree and L be an event log. If M is in class C_a and M and L are df-complete, i.e., $M \sim_{df} L$, then pd_{IM} discovers a process tree M' from L that is isomorphic to M.*

Proof. M meets all model restrictions for the class of language-rediscoverable process trees in [16] except for the self-loop $\circlearrowleft(v,\tau)$ with $v \in A$ that can occur as a (sub-)tree in M. The self-loop violates both the restriction on disjoint start and end activities for the first branch of the loop (cf. restriction 2 in [16]) and the restriction that no τ's are allowed in M (cf. restriction 3 in [16]). Hence, we distinguish whether M contains a self-loop. For both cases, we build on the proof of Theorem 14 [16], because Theorem 14 is proven through showing isomorphic rediscoverability by induction on the size of M.
Case no self-loop: Because M meets all model restrictions for the class of language-rediscoverable process trees, the conditions of Theorem 14 in [16] are met and M isomorphic rediscoverable.
Case self-loop: Because the self-loop does not contain nested subtrees, it suffices to extend Theorem 14's induction on the size of M by an additional base case: $\circlearrowleft(v,\tau)$ with $v \in A$. The additional base case holds, because the "Strict Tau Loop" fall through rediscovers the self-loop from any splitted log $L_v \subseteq [\langle v,v\rangle^m, \langle v,v,v\rangle^n, ...]$ that occurs during recursion of IM.

Lemma 4 (ma_{bpa} abstracted process trees are in C_a). *If ma_{bpa} is applicable to process tree M, then $M_a = \mathrm{ma}_{bpa}(M)$ is in class C_a.*

Proof. The abstracted process tree M_a does not contain duplicate activities, because M does not contain duplicate activities (condition 1 in Def. 4) and new activities $x \in A_{new}$ (cf. condition 4) are only added once to the process tree M_a during synthesis (cf. Algorithm 3.2 in [37]).
A $\circlearrowleft$-node $M_\circlearrowleft = \circlearrowleft(M_1,...,M_n)$ and a $\wedge$-node $M_\wedge = \wedge(M'_1,...,M'_m)$ in a process tree M are indistinguishable through the BP p_M (i.e., $x_i \parallel y_j \in p_M$ for all $x_i, y_j \in A_{M_\circlearrowleft}$ where x_i is an activity in child M_i of $M_\circlearrowleft$ and y_j is an activity from another child M_j, $i \neq j$ of $M_\circlearrowleft$ and similarly $x'_i \parallel y'_j \in p_M$ for two activities from different children of $M_\wedge$), because the order relations in the BP p_M are defined through the eventually-follows relation. For ma_{bpa}, there exists a choice in the synthesis step of ma_{bpa}: Either construct a loop node for an AND-complete module in the modular decomposition tree or construct a parallel node, since there is no information in the abstracted BP

p_{M_a} that allows to differentiate these two nodes. ma_{bpa} chooses the parallel node and always constructs a parallel node for AND-complete modules with the exception of singleton modules $x \| x \in p_{M_a}$ in which case it constructs the self-loop $\circlearrowleft(x,\tau)$ (cf. line 7-8 and 15-16 in Algorithm 3.2 in [37]).
Modules of the modular decomposition tree $MDT(G)$ of the graph $G(p_{M_a})$ can by definition [37] only contain activities.

Lemma 5 (Number of traces and size of $\text{ntl}(M)$**.log).** *If M is in C_c, then $|\mathcal{L}_m(M)|$ is computed by* ntl, $|\mathcal{L}_m(M)| = |\text{lens}(M)|$, *and the size is* $\|\mathcal{L}_m(M)\| = \sum_{k=1}^{|\mathcal{L}_m(M)|} \text{lens}(M)[k]$.

Proof. We sketch the structural induction proof for the number of traces and size by case distinction on the process tree's node operator.
$M = \times(M_1,...,M_n)$: The union of logs $L = \bigcup_{i=1}^{n} \mathcal{L}_m(M_i)$ constructs a log L whose number of traces equals the sum of trace numbers and whose trace lengths equal the concatenation of all children's M_i sequence of trace lengths $\text{lens}(M_i)$. The length of concatenating n sequences of length $\text{lens}(M_i)$ equals the sum of n lengths $|\text{lens}(M)| = |\mathcal{L}_m(M)|$.
$M = \rightarrow(M_1,...,M_n)$: The set of all sequential concatenations $L = \{\sigma_1 \cdot \sigma_2 \cdot ... \cdot \sigma_n \,|\, \forall i \in \{1...n\} : \sigma_i \in \mathcal{L}_m(M_i)\}$ constructs a log L whose number of traces equal the number of ordered pairs in the cartesian product $\times_{i=1}^{n} \{1,...,$
$|\mathcal{L}_m(M_i)|\}$. Hence, $|\mathcal{L}_m(M)| = \prod_{i=1}^{n} |\mathcal{L}_m(M_i)|$. The bijection ι enumerates the ordered pairs for summation.
$M = \wedge(M_1,...,M_n)$: Interleaving of multiple event logs $\mathcal{L}_m(M_i)$ through a $\wedge$-node requires interleaving of $\prod_{i=1}^{n} |\mathcal{L}_m(M_i)|$ different ordered pairs of traces we refer to as combinations. As each of these trace combinations can have traces of varying lengths, the function ntl enumerates the respective combinations of traces lengths through the bijection ι and sums the respective number of interleaving the trace combinations. Each combination of traces $(\sigma_1,...\sigma_n) \in \mathcal{L}_m(M_1) \times ... \times \mathcal{L}_m(M_n)$ to be interleaved is enumerated by index k in the domain of ι: $k \in \text{dom}(\iota)$. Hence, the kth enumerated trace combination has its corresponding sequence of trace lengths $\bigodot_{i=1}^{n} \langle \text{lens}(M_i)[\iota_{k,i}] \rangle$ (line 12 in Algorithm 3 of the main paper). The number of interleavings of two traces σ_1 and σ_2 is $\binom{m}{\langle l_1, l_2 \rangle} = \frac{(l_1+l_2)!}{l_1! l_2!}$ with $|\sigma_1| = l_1$ and $|\sigma_2| = l_2$, because interleaving two sequences without changing their respective order is equivalent to shuffling two card decks without changing the card order of the two decks [2]. As the number of riffle shuffle permutations equals $\frac{(p+q)!}{p!q!}$ for p the number of cards in the first deck and q the number of cards in the second deck [2], the number of traces $\sigma \in \sigma_1 \diamond \sigma_2$ equals $\frac{(k_1+k_2)!}{k_1! k_2!}$ with $|\sigma_1| = k_1$ and $|\sigma_2| = k_2$. Generalizing the number of two trace interleavings to the number of n traces interleaved yields $\binom{m_1}{\langle l_1, l_2 \rangle} * \binom{m_2}{\langle l_1+l_2, l_3 \rangle} * ... * \binom{m_n}{\langle \sum_{i=1}^{n-1} l_i, l_n \rangle} = \frac{(l_1+l_2)!}{l_1! l_2!} * \frac{(l_1+l_2+l_3)!}{l_3!(l_1+l_2)!} * \frac{(l_1+l_2+l_3+l_4)!}{l_4!(l_1+l_2+l_3)!} * ... * \frac{(\sum_{i=1}^{n} l_i)!}{l_n!(\sum_{i=1}^{n-1} l_i)!} = \frac{(\sum_{j=1}^{n} l_j)!}{\prod_{j=1}^{n} l_j!} = \binom{m}{\langle l_1,...,l_n \rangle}$, because after two traces have been interleaved, we can take the already interleaved trace as a new trace for the next trace to be interleaved with. Hence, the number of n traces interleaved equals the multinomial coefficient for the n different trace lengths $l_1,...,l_n$. Consequently, for each kth combination of traces in the ntl function at line 12, the multinomial coefficient is computed for the corresponding sequence of trace lengths

$\langle l_{k,1},...,l_{k,n}\rangle = \bigodot_{i=1}^{n}\langle \text{lens}(M_i)[\iota_{k,i}]\rangle$. Each of these interleaved traces are summed together to yield the number of traces.
Since there are $\prod_{j=1}^{n}|\mathcal{L}_m(M_j)|$ different trace combinations to be interleaved, there can only be $\prod_{j=1}^{n}|\mathcal{L}_m(M_j)|$ different lengths of traces as a result of interleaving. The interleaved trace length for the kth trace combination equals $m_k = \sum_{i=1}^{n}\text{lens}(M_i)[\iota_{k,i}] = \sum_{i=1}^{n} l_{k,i}$, which has to be repeated in $\text{lens}(M)$ for the number of interleavings: $\langle m_k \rangle^{\left(\bigodot_{i=1}^{n}\langle \text{lens}(M_i)[\iota_{k,i}]\rangle\right)^{m_k}}$.

Lemma 7 (Maximal $\mathcal{L}_m(M_a)$). *If L is restricted and ma_{bpa} applicable to $M = \text{pd}_{IM}(L)$, $w_t = w_{minmax}$, $\text{agg}(x) = \{v,u\}$, and $\text{agg}(z) = \{z\}$ with $z \in A\backslash\{v,u\}$, i.e., ma_{bpa} aggregates exactly two activities into x, then $M_a = \text{ma}_{bpa}(M)$ has the most traces and maximal size of all $\mathcal{L}_m(M'_a)$ generated for other M'_a with $A'_c \cap \{v,u\} = \varnothing$ and $|\bigcup_{y\in A'_{new}}\text{agg}'(y)| > |\bigcup_{y\in A_{new}}\text{agg}(y)|$ (cf. Definition 2 in our main paper).*

Proof. Let M_a be the abstracted process model for $w_t = w_{minmax}$,$\text{agg}(x) = \{v,u\}$ and M'_a be the abstracted process model for $w_t \leqslant w_{minmax}$ and an aggregate function that abstracts more than two concrete activities, i.e., $|A'_a \cap A_M| < |A_M| - 2$. From condition (4) Definition 2 in our main paper, it follows that $|A_{M'_a}| < |A_{M_a}|$, because either are at least three concrete activities abstracted into a single abstract activity in M'_a or m abstract activities aggregate at least $m+1$ concrete activities. Thus, M'_a cannot have more traces and events in $\mathcal{L}_m(M'_a)$ than $\mathcal{L}_m(M_a)$ as a result of more activities. Hence, M'_a must have a "different" tree structure in terms of node operators and their children.
The abstract process tree is constructed given the modular decomposition (MDT) of the ordering relations graph $G(p_{M'_a})$ and $G(p_{M_a})$ such that structural tree differences must be caused by differences in the the behavioral profiles $p_{M'_a}$ and p_{M_a}. The two behavioral profiles are different

1. with respect to their sizes $|p_{M'_a}| < |p_{M_a}|$,
2. with respect to order relations that involve activities $q \in A_c$, i.e., activities q are not abstracted in M_a but are abstracted in M'_a,
3. with respect to order relations of abstract activities $y' \in A'_{new}$ to $q' \in A'_a$ (cf. condition 3 Definition 2 in our main paper) and $y \in A_{new}$ to $q \in A_a$.

The first two differences imply more traces and events in $\mathcal{L}_m(M_a)$. Hence, a larger number of traces and events in $\mathcal{L}_m(M'_a)$ can only be the result of different order relations between abstract activities y' to q' and y to q. From Lemma 6, more traces and events can only occur, if order relations of abstract activities y' to a q' are less restrictive than from y to q.
By definition, L is restricted and $w_t = w_{minmax}$. Hence, M is in C_c and meets the requirement on the model structure that excludes children of less than two traces for any node $\rightarrow(...)$ in M. From Lemma 6, it follows that the order of returning order relations in , i.e., first $+_{M_a}$, then $\rightsquigarrow^{-1}_{M_a}$, then $\rightsquigarrow_{M_a}$, then $\|_{M_a}$ (cf. ⓞ), aligns with the order of process tree operators for a node $M_{\oplus}$ whose children remain the same, but whose root node changes. By code inspection of Algorithm 3.1 in [37] (that corresponds to $\text{dv}_{\text{agg},w_t}$ in our main paper), an order relation $x \diamond_{M_a} q$ is only

returned, if $v \diamond_M q$ for $v \in \text{agg}(x)$ and $q \in A_M$ occurs either relatively more often in the behavioral profile p_M or, in case of equal relative frequencies, the more restrictive relation according to the order ⓞ is selected.
In any case, $x \diamond_{M_a} q$ is only returned, if the respective order relation is either already the most prevalent in M, i.e., the most prevalent in the behavioral profile p_M of M, or, in case of conflicts between order relation frequencies, it is the more "restrictive" according to ⓞ. Hence, Algorithm 3.1 in [37] chooses an order relation between two activities that corresponds to fewer traces and fewer events in $\mathcal{L}_m(M'_a)$ over an order relation between two activities that corresponds to more traces and more events. Therefore, order relations between abstract activities y' and q' cannot result in more traces and events in $\mathcal{L}_m(M'_a)$. Also, setting $w_t < w_{minmax}$ can only further decrease the number of traces and events in $\mathcal{L}_m(M'_a)$.
Altogether, all three differences between $p_{M'_a}$ and p_{M_a} imply that $\mathcal{L}_m(M'_a) \leqslant \mathcal{L}_m(M_a)$.

Lemma 1 ($\mathcal{L}_m(M_a)$ is smaller). *If L is restricted and ma_{bpa} applicable to $M = \text{pd}_{IM}(L)$, then $L_a = \mathcal{L}_m(M_a)$ has fewer traces and is smaller than L: $|L_a| < |L|$ and $\|L_a\| < \|L\|$.*

Proof. From Lemma 11 it follows that we can generate mdf-complete event logs $\mathcal{L}_m(M)$ for M. From Lemma 5, it follows that $\mathcal{L}_m(M)$ is the smallest event log for which IM discovers the same process tree M, i.e., it is the smallest representative of all restricted event logs: $|\mathcal{L}_m(M)| \leqslant |L'|$ and $\|\mathcal{L}_m(M)\| \leqslant \|(L')\|$ for every $L' \in [L]_M = \{L' \subseteq \mathcal{E}^* \mid \text{pd}_{IM}(L') \cong M \wedge L' \text{is restricted}\}$. Additionally, from Lemma 7 it follows that it suffices to only consider ma_{bpa} with $w_t = w_{minmax}$ that abstracts two concrete activities: $\text{agg}(x) = \{v,u\}$ and $\text{agg}(z) = \{z\}$ for $v,u \in A_M$ and $z \in A_M \backslash \{v,u\}$. Thus, L_a is generated for $M_a = \text{ma}_{bpa}(M)$ in which two concrete activities are abstracted with $w_t = w_{minmax}$. Consequently, the following induction on the size $|M|$ proves $|L_a| < |\mathcal{L}_m(M)|$ and $\|L_a\| < \|\mathcal{L}_m(M)\|$.
Base Cases:

- $M = \wedge(a,b)$: $\text{agg}(x) = \{a,b\}$ and $w_{minmax} = 0.5$ results in $M_a = \circlearrowleft(x,\tau)$ with $L_a = \{\langle x,x\rangle\}$. Hence, $|L_a| = 1 < |\mathcal{L}_m(M)| = 2$ and $\|L_a\| = 2 < 4 = \|\mathcal{L}_m(M)\|$.
- $M = \times(a,b)$: $\text{agg}(x) = \{a,b\}$ and $w_{minmax} = 1$ results in $M_a = x$ with $L_a = \{\langle x\rangle\}$. Hence, $|L_a| = 1 < |\mathcal{L}_m(M)| = 2$ and $\|L_a\| = 1 < 2 = \|\mathcal{L}_m(M)\|$.
- $M = \wedge(a, \circlearrowleft(b,\tau))$: $\text{agg}(x) = \{a,b\}$ and $w_{minmax} = 0.75$ results in $M_a = x$ with $L_a = \{\langle x\rangle\}$. Hence, $|L_a| = 1 < |\mathcal{L}_m(M)| = 3$ and $\|L_a\| = 1 < 9 = \|\mathcal{L}_m(M)\|$.
- $M = \times(a, \circlearrowleft(b,\tau))$: $\text{agg}(x) = \{a,b\}$ and $w_{minmax} = 0.75$ results in $M_a = \circlearrowleft(x,\tau)$ with $L_a = \{\langle x,x\rangle\}$. Hence, $|L_a| = 1 < |\mathcal{L}_m(M)| = 2$ and $\|L_a\| = 2 < 3 = \|\mathcal{L}_m(M)\|$.

Induction hypothesis (IH): For any process tree M' of smaller size than M that is discovered from restricted event log L' such that $M'_a = \text{ma}_{bpa}(M')$ is applicable to M' for $w'_t = w'_{minmax}$, the mdf-complete log L'_a has fewer traces and is smaller than $\mathcal{L}_m(M)'$: $|L'_a| < |\mathcal{L}_m(M)'|$ and $\|L'_a\| < \|\mathcal{L}_m(M)'\|$.
Induction step: Let $M = \oplus(M_1,\ldots,M_n)$. The loop operator cannot occur as the root node, as it can only occur in a self-loop $\circlearrowleft(v,\tau)$ with $v \in A_L$ in a process tree M discovered from restricted event log L (cf. Lemma 11), i.e., it is either covered in base cases or part of M as a subtree in one of the M_i's. Thus, apply case distinction on the operator node $\oplus \in \{\times, \rightarrow, \wedge\}$:

– Case $\oplus = \times$:

There exist two cases of how $\text{agg}(x) = \{v,u\}$,$\text{agg}(z) = \{z\}$ for $z \in A_M \backslash \{v,u\}$ can be defined on M:

- Case $v,u \in M_i$,$i \in \{1,...,n\}$: Hence, both concrete activities occur in the same child M_i of M. By (IH), the inequalities hold.
- Case $v \in M_i$,$u \in M_j$,$i,j \in \{1,...,n\}$,$i \neq j$: Hence, the two concrete activities v and u occur in two different children M_i and M_j of M. Because the choice order relation $+$ is symmetric and the children $M_1,\dots,M_n$ of M can be reordered without changing the language $\mathcal{L}_m(M)$, the abstracted process tree can be decomposed: $M'_a = \times(\text{ma}_{bpa}(\times(M_i,M_j)),M_{r_1},\dots,M_{r_m})$ with $m = n-2$ and $r_1,...,r_m \in \{1,...,n\}\backslash\{i,j\}$. M'_a may not always be in normal form as the process tree $\text{ma}_{bpa}(\times(M_i,M_j))$ may have a choice operator as a root node. Nevertheless, reducing M'_a to a normal form with the reduction rules in Definition 5.1 [15] yields the abstracted process tree M_a and the reduction rules preserve the language (ⓡ), i.e., $\mathcal{L}_m(M'_a) = \mathcal{L}_m(M_a)$ such that the number of traces and sizes are equal. Hence, $M_a = \text{ma}_{bpa}(M)$ is decomposable as specified by M'_a such that the abstracted process tree only differs to M in the abstraction of M_i and M_j. Because $|\times(M_i,M_j)| < |M|$, the inequalities follow from (IH).

– Case $\oplus = \rightarrow$:

There exist two cases of how $\text{agg}(x) = \{v,u\}$,$\text{agg}(z) = \{z\}$ for $z \in A_M \backslash \{v,u\}$ can be defined on M:

- Case $v,u \in M_i$,$i \in \{1,...,n\}$: Analogous to case $\oplus = \times$.
- Case $v \in M_i$,$u \in M_j$,$i,j \in \{1,...,n\}$,$i \neq j$: Hence, the two concrete activities v and u occur in two different children M_i and M_j of M. Without loss of generality, we assume $i < j$. If $i > 1$ or $j < n$, abstraction of M can be decomposed into $M_a = \rightarrow(M_1,...,M_{i-1},\text{ma}_{bpa}(M_{i \leqslant r \leqslant j}),M_{j+1},...,M_n)$ with $M_{i \leqslant r \leqslant j} = \rightarrow(M_i,M_{i+1},...,M_j)$, because the behavioral profile p_{M_a} differs to the behavioral profile p_M only for ordering relations of activities $y \in A_{M_{i \leqslant r \leqslant j}}$. Since ⓡ and $|M_{i \leqslant r \leqslant j}| < |M|$, the inequalities hold by (IH).
  If $i = 1$ and $j = n$, then $M_{i \leqslant r \leqslant j} = M$, i.e., ma_{bpa} abstracts the whole process tree M. For any $q \in \bigcup_{r \in \{2,...,n-1\}} A_{M_r}$, it holds that $v \rightsquigarrow_M q$ and $u \rightsquigarrow_M^{-1} q$. Thus, all relative frequencies of ordering relations (line 7-10 in Algorithm 3.1 in [37]) are equal to 0.5 such that $+_{M_a}$ is always returned: $w_{max}(x,q) = w(x +_{M_a} q) = 0.5$. On the one hand, for any $q_1 \in A_{M_1} \backslash \{v\}$, it holds $u \rightsquigarrow_M^{-1} q_1$. Thus, if $v +_M q_1$ or $v \rightsquigarrow_M q_1$, then $w_{max}(x,q_1) = w(x +_{M_a} q_1) = 0.5$. If $v \rightsquigarrow_M^{-1} q_1$ or $v \parallel_M q_1$, then $w_{max}(x,q_1) = w(x \rightsquigarrow_{M_a}^{-1} q_1) \geqslant 0.5$. On the other hand, for any $q_n \in A_{M_n} \backslash \{u\}$, it holds $v \rightsquigarrow_M q_n$. Thus, if $u +_M q_n$ or $u \rightsquigarrow_M^{-1} q_n$, then $w_{max}(x,q_n) = w(x +_{M_a} q_n) = 0.5$. If $u \rightsquigarrow_M q_n$ or $u \parallel_M q_n$, then $w_{max}(x,q_n) = w(x \rightsquigarrow_{M_a} q_n) \geqslant 0.5$. Additionally, $w_{max}(x,x) = w(x +_{M_a} x) = 0.5$, i.e., x is not added as a self-loop $\circlearrowleft(x,\tau)$ to M_a during the synthesis step of ma_{bpa}.
  Altogether, the abstract activity x is in choice relation $v +_{M_a} z$ to any activity q of $M_2,...,M_{n-1}$, x is either in choice $x +_{M_a} q_1$ or inverse strict order relation

$x \rightsquigarrow^{-1}_{M_a} q_1$ to any activity q_1 of M_1, and x is either in choice $x +_{M_a} q_n$ or strict order relation $x \rightsquigarrow_{M_a} q_n$ to any activity q_n of M_n. If x is in choice relation to all activities q_1 and q_n in M_1 and M_n respectively, then

$$M_a = \times(x, \rightarrow(M'_1, M_2, ..., M'_n)) \qquad \text{(I)}$$

Although $M_2, \ldots, M_{n-1}$ may, in fact, change through ma_{bpa}, if they contain an *optional* node $\times(..., \tau, ...)$, we do not consider optional nodes for simplicity. Optional nodes are always abstracted by removing the τ (if the node has more than two children) or by "moving up" the other activity $q \in A_{\times(..., \tau, ...)}$ as a direct child to the parent node. In both cases, the number of traces and events of the mdf-complete log $\mathcal{L}_m(M_r)$ for $r \in \{2, ..., n-1\}$ decreases, so the L_a we consider in the following by ignoring optional nodes is at least as large as the L'_a that would result from also considering abstraction of optional nodes.

Let M_v and M_u be the nodes of M_1 and M_n in which v and u occur as children respectively. If $M_v = \oplus(v, M_{v,2})$ or $M_u = \oplus(u, M_{u,2})$, then the other node $M_{v,2}$ and $M_{u,2}$ respectively "move up" as children to the parent node of M_v and M_u respectively. If M_v or M_u has more than two children, child v and u is eliminated from M_v and M_u respectively. For all four cases of how M_1 and M_n are changed into M'_1 and M'_n through changing M_v and M_u respectively, it holds $|\mathcal{L}_m(M'_1)| < |\mathcal{L}_m(M_1)|$ and $\|\mathcal{L}_m(M'_1)\| < \|\mathcal{L}_m(M_1)\|$ as well as similarly for M_n and M'_n. Thus, for (I), it follows that:

$$|L_a| = 1 + |\mathcal{L}_m(M'_1)| * |\mathcal{L}_m(M'_n)| * \prod_{r \in \{2, ..., n-1\}} |\mathcal{L}_m(M_r)| < |\mathcal{L}_m(M)|$$

because from line 9 of Algorithm 3 in our main paper, we can compute the number of traces of a sequence as the product of its children's number of traces and two children with less traces always outweigh the new trace $\langle v \rangle \in L_a$.

The size of $\mathcal{L}_m(M)$ is:

$$\begin{aligned} \|\mathcal{L}_m(M)\| &= \sum_{j=1}^{|\mathcal{L}_m(M)|} \text{lens}(M)[j] \\ &= \sum_{j=1}^{|\mathcal{L}_m(M)|} \Big(\bigodot_{k=1}^{|\mathcal{L}_m(M)|} < \sum_{i=1}^{n} \text{lens}(M_i)[\iota_{k,i}] > \Big)[j] \end{aligned}$$

Break the trace lengths sequence up and rewrite:

$$= \sum_{i=1}^{n} \sum_{k=1}^{|\mathcal{L}_m(M)|} \text{lens}(M_i)(\iota_{k,i})$$

Holding i constant for ι to separate each M_i:

$$
\begin{aligned}
&= \sum_{i=1}^{n} \Big(\sum_{k=1}^{|\mathcal{L}_m(M_i)|} \text{lens}(M_i)[k] \Big) * \prod_{j \in \{1,\ldots,n\} \setminus \{i\}} |\mathcal{L}_m(M_j)| \Big) \\
&= \sum_{i=1}^{n} \big(\|\mathcal{L}_m(M_i)\| * \prod_{j \in \{1,\ldots,n\} \setminus \{i\}} |\mathcal{L}_m(M_j)| \big) \\
&< \big(\sum_{i=1}^{n} \|\mathcal{L}_m(M_i)\| \big) * |\mathcal{L}_m(M)| = |\mathcal{L}_m(M)| \sum_{i=1}^{n} \|\mathcal{L}_m(M_i)\| \qquad \text{(Eq. 2)}
\end{aligned}
$$

For the size of L_a it holds:

$$
\begin{aligned}
\|L_a\| &< 1 + |L_a| * \big(\|\mathcal{L}_m(M_1')\| + \|\mathcal{L}_m(M_n')\| \\
&\qquad + \sum_{r \in \{2,\ldots,n-1\}}^{n} \|\mathcal{L}_m(M_r)\| \big) \\
&< 1 + |L_a| \sum_{r \in \{1,\ldots,n\}}^{n} \|\mathcal{L}_m(M_r)\|) < \|\mathcal{L}_m(M)\|
\end{aligned}
$$

because due to (Eq. 2) we can bound the size of L_a by the number of traces times the sum of each children's log size and since $|L_a| < |\mathcal{L}_m(M)|$ proves the inequality even for the upper bound of unchanged M_i and M_j children in M_a.

If v is also in inverse strict order relation to some activities q_1 in M_1, then M_1 is split into two process trees $M_{1,\rightsquigarrow^{-1}}$ that contains the activities q_1 in inverse strict order relation to v and $M_{1,\times}$ that contains the activities q_1 in choice relation to v. Analogously, if u is also in strict order relation to some activities q_n in M_n, then M_n is split into two process trees $M_{n,\rightsquigarrow}$ that contains the activities q_n in strict order relation to u and $M_{n,\times}$ that contains the activities q_n in choice relation to u. Taken together:

$$
M_a = \rightarrow (M_{1,\rightsquigarrow^{-1}}, \times (x, \rightarrow (M_{1,\times}, M_2, \ldots, M_{n-1}, M_{n,\times}), M_{n,\rightsquigarrow}) \qquad \text{(II)}
$$

The mdf-complete log L_a is largest both in terms of number of traces and size, if M_1 and M_n only contain either parallel $\wedge(\ldots)$ or sequence $\rightarrow(\ldots)$ nodes (cf. Lemma 6). It follows that all activities q_1 in M_1 are in inverse strict order relation to v, i.e., $v \rightsquigarrow^{-1} q_1$, and all activities q_n in M_n are in strict order relation to v, i.e., $v \rightsquigarrow q_n$. Hence, the M_a with the largest L_a for (II) is:

$$
M_a = \rightarrow (M_i', \times (x, \rightarrow (M_2, \ldots, M_{j-1}), M_j') \qquad \text{(III)}
$$

For (III), it follows that:

$$\begin{aligned}|L_a| &= |\mathcal{L}_m(M_1')| * |\mathcal{L}_m(M_n')| * (1 + \prod_{r \in \{2,\ldots,n-1\}} |\mathcal{L}_m(M_r)|) \\ &= |\mathcal{L}_m(M_1')| * |\mathcal{L}_m(M_n')| && \text{(New)} \\ &\quad + |\mathcal{L}_m(M_1')| * |\mathcal{L}_m(M_n')| * \prod_{r \in \{2,\ldots,n-1\}} |\mathcal{L}_m(M_r)|) && \text{(Abs.)} \\ &< |\mathcal{L}_m(M)|\end{aligned}$$

because M_1' and M_n' both have parallel root nodes[13] and the factorials in the multinomial coefficients of $|\mathcal{L}_m(M_1')|$ and $|\mathcal{L}_m(M_n')|$ decrease "faster" through multiplication in (Abs.) than $|L_a|$ gains traces through adding $|\mathcal{L}_m(M_1')| * |\mathcal{L}_m(M_n')|$ in (New). For the size of the log L_a it holds:

$$\begin{aligned}\|L_a\| &< |L_a| * (1 + \|\mathcal{L}_m(M_1')\| + \|\mathcal{L}_m(M_n')\| + \sum_{r \in \{2,\ldots,n-1\}}^{n} \|\mathcal{L}_m(M_r)\|) \\ &< \|\mathcal{L}_m(M)\|\end{aligned}$$

because we can bound the size of L_a by the number of traces times the sum of each children's log size (Eq. 2) and two times a smaller log $\mathcal{L}_m(M_1')$ and $\mathcal{L}_m(M_2')$ outweighs the additional event.

– Case $\oplus = \wedge$: Analogous to case $\oplus = \times$.

Lemma 8 (Matching of quotient sets). *If L is restricted and ma_{bpa} applicable to $M = \mathrm{pd}_{IM}(L)$, there exists a matching between the quotient set $L_{tmp}/\sim$ of event log $L_{tmp} = \mathrm{ea}^1_{bpa}(L, \mathrm{pd}_{IM}, \mathrm{ma}_{bpa})$ and the quotient set $L_a/\sim$ of $L_a = \mathcal{L}_m(M_a)$ for $M_a = \mathrm{ma}_{bpa}(\mathrm{pd}_{IM}(L))$, i.e., $\mathrm{matching}: L_1/\sim \to L_2/\sim$ exists. Also, for every matched pair of equivalence classes $L_{tmp}^{class} \in L_{tmp}/\sim$ and $L_a^{class} = \mathrm{matching}(L_{tmp}^{class})$ it holds that L_{tmp}^{class} has equal to or more traces than L_a^{class}: $|L_a^{class}| \leqslant |L_{tmp}^{class}|$ (ii).*

Proof. We prove the statement in four steps (I-IV). First, we prove that L_{tmp} and L_a share the same activities $A_{L_{tmp}} = A_{L_a}$. Second, we prove that the quotient sets have the same size: $|L_{tmp}/\sim| = |L_a/\sim|$. Third, we prove that for every equivalence class $L_{tmp}^{class} \in L_{tmp}/\sim$, there exists exactly one equivalence class $L_a^{class} \in L_a/\sim$ such that their traces are indistinguishable modulo transposition: $\forall \sigma_a \in L_{tmp}^{class} \sigma \in L_a^{class}$: $\delta_{kendall}(\sigma_a, \sigma) \neq \perp$. From the three steps I-III, the existence of a matching follows. Fourth, we prove the inequality (ii).
I. By code inspection of ea^1_{bpa} it follows that $A_{L_{tmp}} = A_{L_a}$, because the parameter agg is equally applied to L for abstraction of concrete events into their abstract

[13] pd_{IM} discovers process trees in *normal form* [15] such that for $M = \rightarrow(M_1,\ldots,M_n)$ no children of M can have the sequence as a root node.

counterparts as it is applied to M for abstraction of concrete activities into their abstract counterparts.

II. Second, we establish that there are as many equivalence classes in $L_{tmp}/\sim$ as there are in $L_a/\sim$: $|L_{tmp}/\sim| = |L_a/\sim|$. Towards contradiction, we assume $|L_{tmp}/\sim| \neq |L_a/\sim|$. Given the model structure of M_a (cf. Lemma 4 and Def. 4), the $\times$-node is the only process tree operator that affects the number of equivalence classes in L_a, because $\circlearrowleft$-nodes other than the self-loop as a leaf do not occur and traces in both the language of $\rightarrow$-node and $\wedge$-node are indistinguishable modulo transposition. Likewise, the restricted event log L in Def. 5 is similarly constrained. By code inspection of ea^1_{bpa} it follows that the resulting L_{tmp} satisfies the requirements of Def. 5. Hence, L_{tmp} is also a restricted event log. Let $M_{tmp} = \text{pd}_{IM}(L_{tmp})$ be the process tree that the IM discovers from L_{tmp}. From $|L_{tmp}/\sim| \neq |L_a/\sim|$, from the respective restrictions on both L_{tmp} and M_a, and from Alg. 4, it follows that M_{tmp} and M_a must have a different number of $\times$-nodes. Hence, it follows that $|L_{tmp}| \neq |L_a|$, i.e., the two event logs have different numbers of traces (cf. Lemma 5 and line 8 Alg. 4). Considering that $|L_a| < \mathcal{L}_m(M)$ (cf. Lemma 1) and that ea^1_{bpa} does not add or delete traces from L, it can only be that $|L_{tmp}| > |L_a|$. Consequently, the number of $\times$-nodes in M_{tmp} must be larger than in M_a.

From the last two statements, it follows that there exist two traces $\sigma_1, \sigma_2 \in L_{tmp}$ and two events in these traces $\exists x \in \sigma_1, y \in \sigma_2$ such that $x +_{M_{tmp}} y$ holds in the behavioral profile $p_{M_{tmp}}$ of M_{tmp}. Let x and y be activities that are children of one of the additional $\times$-nodes in M_{tmp}, i.e., $x \in A_{M_1}$ and $y \in A_{M_2}$ for two children M_1 and M_2 of the additional $\times$-node. From **1.**, it follows that $x, y \in A_{M_a}$. Additionally, $x \triangle_{M_a} y$ with $\triangle \in \{\rightsquigarrow, \rightsquigarrow^{-1}, \|\}$ in the behavioral profile p_{M_a} of M_a, as otherwise x and y would not be in children of the additional $\times$-node in M_{tmp}. There are two alternative reasons for $x +_{M_{tmp}} y$. First, $x +_{M_{tmp}} y$ holds, because ea^1_{bpa} removed one of the two activities through the deleteChoiceActivities operator in line 16 Alg. 2. If deleteChoiceActivities does not remove the two activities, they must have been in choice relation already. In both cases, however, the activities x and y are in choice relation in the behavioral profile p_{M_a} of M_a, i.e., a contradiction. Second, $x +_{M_{tmp}} y$ holds, because either x or y is a concrete activity v such that one of the two was not added to an abstracted trace σ_{abs} (cf. line 9 Alg. 2). Again, this can only happen, if the two activities are in choice relation in p_{M_a}.

III. Third, we establish that for every equivalence class $L^{class}_{tmp} \in L_{tmp}/\sim$, there exists exactly one equivalence class $L^{class}_a \in L_a/\sim$ such that: $\forall \sigma_{abs} \in L^{class}_{tmp} \sigma_a \in L^{class}_a : \delta_{kendall}(\sigma_{abs}, \sigma_a) \neq \perp$. Towards contradiction, we assume that there exists an equivalence class $L^{class}_{tmp} \in L_{tmp}/\sim$ for which no equivalence class $L^{class}_a \in L_a/\sim$ exists that can be matched. Let L^{class}_{tmp} be the equivalence class for which no matching equivalence class $L^{class}_a \in L_a/\sim$ exists. For every trace $\sigma_{abs} \in L^{class}_{tmp}$ and for every $\sigma_a \in L^{class}_a$ it follows that $\delta_{kendall} = \perp$. Either the traces have all a different length $|\sigma_{abs}| \neq |\sigma_a|$ or have different activities $\text{bag}(\sigma_{abs}) \neq \text{bag}(\sigma_a)$ (cf. Def. 6).

(Different lengths) By code inspection of ea^1_{bpa} it follows that a trace σ_{abs} can only change its length relative to its concrete σ by (1) deletion (line 8), (2) deletion (line 9), (3) insertion (line 12), or (4) deletion (line 17). The first deletion corresponds to abstraction of concrete events $\text{agg}(x) \subseteq A_\sigma$ of activities occurring in σ into their

abstract events x. The second deletion corresponds to choice relations (cf. II). The third deletion corresponds to self-loops. The fourth deletion corresponds to choice relations (cf. II). As the same abstraction operation was applied to corresponding concrete activities in M to yield abstract activities x in M_a, both choice relations are similar for L_a and L_{tmp} (cf. II), and self-loops in M_a corresponds to $\langle ...,z,...,z,... \rangle \in L_a$ (cf. Alg. 4), the corresponding trace length of σ_{abs} must occur for some trace σ'_a in some equivalence class $L_a^{class} \in L_a/\sim$.
(Different multiset of activities) From (I), the different multisets of activities $\text{bag}(\sigma_{abs}) \neq \text{bag}(\sigma_a)$ cannot be due to activities that are only in one of the two event logs L_{tmp} and L_a. Thus, either is an activity $x \in A_{L_{tmp}}$ only in one of the two multisets, i.e., $x \in \text{bag}(\sigma_{abs})$ and $x \notin \text{bag}(\sigma_a)$ without loss of generality, or the multiplicity of an activity x is unequal in the two multisets, i.e, $\text{bag}(\sigma_{abs})(x) \neq \text{bag}(\sigma_a)(x)$. If an activity x is only in one of the two multisets, the only reason can be a corresponding choice relation $x +_{M_a} y$ that prevented the x activity to occur in trace σ_{abs} due to y occurring in σ_{abs}. However, from (II), the choice relation occurs also in M_a, and by correctness of Alg. 4 (cf. Lemma 5), also in L_a. Hence, the trace σ_{abs} without the x activity must have a matching trace $\sigma' \in L_a^{class}$ with the same multiset of activities. If the multiplicity of an activity x is unequal in the two multisets of activities, the only reason can, again, be a corresponding choice relation, as the loop is restricted to a self-loop both in L and in M_a. Thus, there must be a matching trace $\sigma' \in L_a^{class}$ with the same multiset of activities.
Overall, it follows that for every equivalence class $L_{tmp}^{class} \in L_{tmp}/\sim$, there exists exactly one equivalence class $L_a^{class} \in L_a/\sim$ such that: $\forall \sigma_{abs} \in L_{tmp}^{class} \sigma_a \in L_a^{class} : \delta_{kendall}(\sigma_{abs},\sigma_a) \neq \perp$.
IV. Lastly, we prove (ii). From Lemma 1, it follows that $|L_a| < |L|$. Hence, $|L_a| < |L_{tmp}|$, i.e., the event log L_a has fewer traces than the preliminary abstracted event log L_{tmp}, because ea_{bpa}^1 does not delete traces from L. Consequently, the statement follows from (II), (III), $|L_a| < |L_{tmp}|$, and the minimality of L_a (cf. Lemma 5).

Lemma 2 (ea_{bpa} returns mdf-complete logs). *If L is restricted and ma_{bpa} applicable to $M = \text{pd}_{IM}(L)$, the event log $L'_a = \text{ea}_{bpa}(L,\text{pd}_{IM},\text{ma}_{bpa})$ is a mdf-complete event log for $M_a = \text{ma}_{bpa}(\text{pd}_{IM}(L))$.*

Proof. From line 1 of Algorithm 2 in our main paper it follows that L_a is a mdf-complete event log for M_a (cf. Lemma 5). Hence, we must show $L'_a = L_a$.
The first step ea_{bpa}^1 (Algorithm 1 in the main paper) abstracts the events $e \in \sigma$ whose concrete activities $e \in A_{\text{ma}}$ are abstracted by ma_{bpa} into their respective new abstract activities $x \in A_{new} = A_{M_a} \backslash A_M$ (cf. condition 4 Definition 2 in the main paper). Hence, encountering an event $e \in \sigma$ (line 7) with $e \in A_{\text{ma}}$ must trigger the construction of a new, abstract event x (line 10). A new abstract activity $x \in A_{new}$ abstracts two or more concrete activities $\text{agg}(x)$. Thus, all events $e' \in \sigma$ that have an activity to be abstracted by x, i.e., $e' \in \text{agg}(x)$ must be abstracted into a single abstract event x in trace σ. The condition in line 8 ensures that the first occurrence of an event e to be abstracted into x is the only e that triggers the construction of x. Because $\text{agg}(x) \cap \text{agg}(y) \neq \varnothing$ with $x,y \in A_{new}$ is allowed, line 8 captures all new abstract activities $x,y,...$ that abstract a concrete activity $v \in A_{\text{ma}}$ equal to the event e. Since

concrete activities $u \in A_{\neg \mathrm{ma}}$ must not be changed in σ, a choice relation between u and an abstract activity x, i.e., $u +_{M_a} x \in p_{M_a}$, prohibits adding the corresponding abstract event x to the preliminary abstracted L_{tmp} (line 9). Since abstract activities can be in parallel relation to themselves $x \parallel_{M_a} x \in p_{M_a}$, the corresponding abstract event x is added twice to inject the pattern for the self-loop (cf. Section 5.2 in the main paper). Lastly, events e that are not abstracted, i.e., $e \in A_{\neg \mathrm{ma}}$, are added to the abstracted trace σ_{abs}.
Lastly, ea^1_{bpa} computes a set of abstract activity sets $A_\times$ (line 16) such that for each two activities $x, y \in A, A \in A_\times$ it holds that $x +_{M_a} y \in p_{M_a}$. If a trace $\sigma_{abs} \in L_{tmp}$ contains at least two activities x, y that are both in the same A of $A_\times$, we must eliminate all but one of the activities x, y. The function eliminateChoiceActivities ensures elimination of the respective abstract activities per trace σ_{abs} as specified in line 17.
The second step ea^2_{bpa} (Algorithm 2 in the main paper) identifies with $L_=$ all traces $\sigma \in L_a$ that are already contained in L_{tmp} (line 2) and adds them to the abstracted event log L'_a (line 3). While there are traces $\sigma \in L_{open}$, L'_a does not contain all required traces $\sigma \in L_a$. As the only difference between a trace $\sigma \in L_a$ and a trace $\sigma_{abs} \in L_{tmp} \setminus L'_a$ is the order of events, the minimal number of transpositions required to transform trace σ_{abs} into trace σ is computed by the *Kendall Tau Sequence Distance* [4] denoted by $\delta_{kendall}$. As the distance $\delta_{kendall}$ does not only compute the distance metric, but also the required transpositions, we apply the transpositions on σ_{abs} as specified in line 7. From Lemma 1, it follows that $|L_a| < |L_{tmp}|$ such that the while loop in line 5 always terminates. After termination of the while loop, it follows that $L'_a = L_a$.
Overall, it follows that $L'_a = L_a$. Since $|L_a| < |L_{tmp}|$ and $\|L_a\| < \|L\|$ (cf. Lemma 1), the EA ea is well-defined.

Lemma 11 (Restricted discovery). *If L is restricted, $M = \mathrm{pd}_{IM}(L)$ is a process tree in C_c.*

Proof. IM does not discover duplicate activities (cf. Definition 5.7 property $C_B.2$ and Lemma 6.2 in [15]), i.e., M meets requirement 1 of C_c. Because IM does not discover a loop operator through a loop cut, does not execute fall through "Flower Model" as well as "Tau Loop", and executes fall through "Strict Tau Loop" only to discover a self-loop $\circlearrowleft(x, \tau)$ for $x \in A_L$, M meets requirement 2 of C_c.

References

1. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE TKDE **16**(9), 1128–1142 (2004)
2. Aldous, D., Diaconis, P.: Shuffling Cards and Stopping Times. The American Mathematical Monthly **93**(5), 333–348 (1986)
3. Angelastro, S., Ferilli, S.: Process Model Modularization by Subprocess Discovery. In: 2020 International Joint Conference on Neural Networks (IJCNN). pp. 1–8. IEEE (2020)
4. Cicirello, V.A.: Kendall tau sequence distance: Extending kendall tau from ranks to sequences. INIS **7**(23) (4 2020)
5. De San Pedro, J., Carmona, J., Cortadella, J.: Log-Based Simplification of Process Models. In: BPM. pp. 457–474. Springer (2015)

6. Fahland, D., van der Aalst, W.M.P.: Simplifying Mined Process Models: An Approach Based on Unfoldings. In: BPM. pp. 362–378. Springer (2011)
7. Fahland, D., van der Aalst, W.M.P.: Simplifying discovered process models in a controlled manner. Information Systems **38**(4), 585–605 (2013)
8. Ferilli, S.: WoMan: Logic-Based Workflow Learning and Management. IEEE Transactions on Systems, Man, and Cybernetics: Systems **44**(6), 744–756 (2014)
9. Janssenswillen, G., Depaire, B., Jouck, T.: Calculating the Number of Unique Paths in a Block-Structured Process Model. In: ATAED. pp. 138–152. CEUR (2016)
10. Kammerer, K., Kolb, J., Reichert, M.: PQL - A Descriptive Language for Querying, Abstracting and Changing Process Models. In: BPMDS, vol. 214, pp. 135–150. Springer (2015)
11. Kolb, J., Kammerer, K., Reichert, M.: Updatable Process Views for User-Centered Adaption of Large Process Models. In: Service-Oriented Computing. pp. 484–498. Springer (2012)
12. Kolb, J., Reichert, M.: Data flow abstractions and adaptations through updatable process views. pp. 1447–1453. SAC '13, ACM (2013)
13. Kolb, J., Reichert, M.: A flexible approach for abstracting and personalizing large business process models. ACM SIGAPP Applied Computing Review **13**(1), 6–18 (2013)
14. Köpke, J., Eder, J., Künstner, M.: Projections of Abstract Interorganizational Business Processes. In: Database and Expert Systems Applications. pp. 472–479. Springer (2014)
15. Leemans, S.J.J.: Robust Process Mining with Guarantees: Process Discovery, Conformance Checking and Enhancement, LNBIP, vol. 440 (2022)
16. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering Block-Structured Process Models from Event Logs - A Constructive Approach. Tech. rep., Eindhoven University of Technology (2013), no. BPM-13-06
17. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering Block-Structured Process Models from Event Logs - A Constructive Approach. In: Colom, J.M., Desel, J. (eds.) Application and Theory of Petri Nets and Concurrency. pp. 311–329. LNCS, Springer (2013)
18. Lim, J., Song, M.: A framework for understanding event abstraction problem solving: Current states of event abstraction studies. DKE **154**, 102352 (2024)
19. López-Pintado, O., Murashko, S., Dumas, M.: Discovery and Simulation of Data-Aware Business Processes. In: 2024 6th International Conference on Process Mining (ICPM). pp. 105–112 (Oct 2024).
20. Mafazi, S., Grossmann, G., Mayer, W., Schrefl, M., Stumptner, M.: Consistent Abstraction of Business Processes Based on Constraints. J. Data Semant. **4**(1), 59–78 (2015)
21. Mafazi, S., Mayer, W., Grossmann, G., Stumptner, M.: A Knowledge-based Approach to the Configuration of Business Process Model Abstractions. In: KiBP@ KR. pp. 60–74 (2012)
22. McConnell, R.M., de Montgolfier, F.: Linear-time modular decomposition of directed graphs. Discrete Applied Mathematics **145**(2), 198–209 (Jan 2005)
23. Meyer, A., Weske, M.: Data Support in Process Model Abstraction. In: Conceptual Modeling. pp. 292–306. Springer (2012)
24. Milner, R.: The space and motion of communicating agents. Cambridge University Press (2009)
25. Ordoni, E., Mülle, J., Böhm, K.: Reduction of data-value-aware process models: A relevance-based approach. Information Systems **114**, 102157 (2023)
26. Polyvyanyy, A., Smirnov, S., Weske, M.: On application of structural decomposition for process model abstraction. In: BPSC, pp. 110–122 (2009)

27. Pérez-Castillo, R., Fernández-Ropero, M., Guzmán, I.G.R.d., Piattini, M.: MARBLE. A business process archeology tool. In: 2011 27th IEEE International Conference on Software Maintenance (ICSM). pp. 578–581 (Sep 2011)
28. Pérez-Castillo, R., Fernández-Ropero, M., Piattini, M.: Business process model refactoring applying IBUPROFEN. An industrial evaluation. Journal of Systems and Software **147**, 86–103 (2019)
29. Reichert, M.: Visualizing Large Business Process Models: Challenges, Techniques, Applications. In: BPM Workshops, vol. 132, pp. 725–736. Springer (2013)
30. Reichert, M., Kolb, J., Bobrik, R., Bauer, T.: Enabling personalized visualization of large business processes through parameterizable views. pp. 1653–1660. SAC '12, ACM (2012)
31. Senderovich, A., Shleyfman, A., Weidlich, M., Gal et al., A.: P^3-Folder: Optimal Model Simplification for Improving Accuracy in Process Performance Prediction. In: BPM. pp. 418–436. Springer (2016)
32. Senderovich, A., Shleyfman, A., Weidlich, M., Gal, A., Mandelbaum, A.: To aggregate or to eliminate? Optimal model simplification for improved process performance prediction. Information Systems **78**, 96–111 (Nov 2018)
33. Smirnov, S., Dijkman, R., Mendling, J., Weske, M.: Meronymy-Based Aggregation of Activities in Business Process Models. In: ER. pp. 1–14. Springer (2010)
34. Smirnov, S., Reijers, H.A., Weske, M.: From fine-grained to abstract process models: A semantic approach. Information Systems **37**(8), 784–797 (2012)
35. Smirnov, S., Reijers, H.A., Weske, M., Nugteren, T.: Business process model abstraction: a definition, catalog, and survey. Distrib. Parallel Databases **30**(1), 63–99 (2012)
36. Smirnov, S., Weidlich, M., Mendling, J.: Business Process Model Abstraction Based on Behavioral Profiles. In: Service-Oriented Computing. pp. 1–16. Springer (2010)
37. Smirnov, S., Weidlich, M., Mendling, J.: Business process model abstraction based on synthesis from well-structured behavioral profiles. Int. J. Coop. Inf. Syst. **21**(01), 55–83 (2012)
38. Tsagkani, C., Tsalgatidou, A.: Abstracting BPMN models. In: Proceedings of the 19th Panhellenic Conference on Informatics. pp. 243–244. PCI '15, Association for Computing Machinery (2015)
39. Tsagkani, C., Tsalgatidou, A.: Process model abstraction for rapid comprehension of complex business processes. Information Systems **103**, 101818 (Jan 2022)
40. Tsalgatidou, A., Tsagkani, C.: Rule-based Business Process Abstraction Framework:. In: BMSD. pp. 173–178. SCITEPRESS (2016)
41. Van Houdt, G., de Leoni, M., Martin, N., Depaire, B.: An empirical evaluation of unsupervised event log abstraction techniques in process mining. Inf. Syst. **121**, 102320 (2024)
42. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. Data & Knowledge Engineering **68**(9), 793–818 (Sep 2009)
43. Völker, M., Weske, M.: Ontology-Based Abstraction of Bot Models in Robotic Process Automation. In: ER. pp. 239–256 (2023)
44. Wang, N., Sun, S., Liu, Y., Zhang, S.: Business Process Model Abstraction Based on Fuzzy Clustering Analysis. International Journal of Cooperative Information Systems **28**(03), 1950007 (2019)
45. Wang, N., Sun, S., OuYang, D.: Business Process Modeling Abstraction Based on Semi-Supervised Clustering Analysis. Business & Information Systems Engineering **60**(6), 525–542 (2018)
46. Weber, B., Reichert, M., Mendling, J., Reijers, H.A.: Refactoring large process model repositories. Computers in Industry **62**(5), 467–486 (2011)

47. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process Discovery Using Integer Linear Programming. In: PETRI NETS. pp. 368–387 (2008)
48. Xue, G., Zhang, K., Yang, J., Yao, S.: Plain abstraction of business process model. In: ICCIT. pp. 338–341 (2011)
49. Ye, J., Zhang, S., Lin, Y.: Log Optimization Simplification Method for Predicting Remaining Time (2025), arXiv:2503.07683 [cs]
50. van Zelst, S.J., Mannhardt, F., de Leoni, M., Koschmider, A.: Event abstraction in process mining: literature review and taxonomy. Gran. Comp. **6**(3), 719–736 (2021)