Structuring Acyclic Process Models

Artem Polyvyanyy[1], Luciano García-Bañuelos[2], and Marlon Dumas[2]

[1] Hasso Plattner Institute at the University of Potsdam, Germany
Artem.Polyvyanyy@hpi.uni-potsdam.de
[2] Institute of Computer Science, University of Tartu, Estonia
{luciano.garcia,marlon.dumas}@ut.ee

Abstract. This paper addresses the problem of transforming a process model with an arbitrary topology into an equivalent well-structured process model. While this problem has received significant attention, there is still no full characterization of the class of unstructured process models that can be transformed into well-structured ones, nor an automated method to structure any process model that belongs to this class. This paper fills this gap in the context of acyclic process models. The paper defines a necessary and sufficient condition for an unstructured process model to have an equivalent structured model under fully concurrent bisimulation, as well as a complete structuring method.

1 Introduction

In the Business Process Modeling Notation (BPMN) and in similar notations, a process model is composed of nodes (e.g., tasks, events, gateways) connected by a "flow" relation. Although BPMN allows process models to have almost any topology, it is often preferable that process models follow some structure. In this respect, a well-known property of process models is that of *(well-)structuredness* [1], meaning that for every node with multiple outgoing arcs (a *split*) there is a corresponding node with multiple incoming arcs (a *join*), such that the set of nodes between the split and the join form a single-entry-single-exit (SESE) region. For example, Fig.1(a) shows an unstructured process model, while Fig.1(b) shows an equivalent structured model. Note that Fig.1(b) uses short-names for tasks (a, b, c ...), which appear next to each task in Fig.1(a).

This paper studies the problem of automatically transforming process models with arbitrary topology into equivalent well-structured models. The motivation for such a transformation is manifold. Firstly, it has been empirically shown that structured process models are easier to comprehend and less error-prone than unstructured ones [2]. Thus, a transformation from unstructured to structured process model can be used as a refactoring technique to increase process model understandability. Secondly, a number of existing process model analysis techniques only work for structured models. For example, a method for calculating cycle time and capacity requirements of structured process models is outlined in [3], while a method for analyzing time constraints in structured process models is presented in [4]. By transforming unstructured process models to structured ones, we can extend the applicability of these techniques to a larger class of models. Thirdly, a transformation from unstructured to structured process models can be

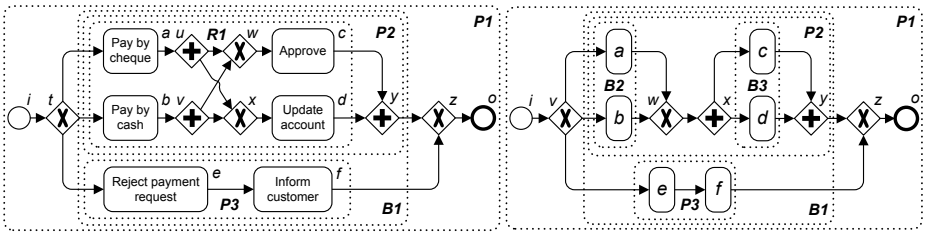


Fig. 1. Unstructured process model and its equivalent structured version

used to implement converters from graph-oriented process modeling languages to structured process modeling languages, e.g., BPMN-to-BPEL.

In the context of flowcharts, without parallel splits and joins, it has been shown that any unstructured flowchart can be transformed into a structured one [5]. If we add parallel splits and joins, this result no longer holds: There exist unstructured process models that do not have equivalent structured ones [1]. Several authors have attempted to classify the sources of unstructuredness in process models [6,7,8] and to define automated methods for structuring process models [9,10,11]. However, these methods are incomplete: There is currently no full characterization of the class of *inherently* unstructured process models, i.e., unstructured process models that have no equivalent structured model. Also, none of the existing structuring methods is complete. In fact, this problem has not been fully solved even for acyclic process models. This paper fills this gap.

To streamline the presentation, we make several assumptions. Firstly, we consider process models composed of nodes (tasks, events, gateways) and control flow relations. In terms of BPMN, this means that we abstract away from other process model elements such as artifacts, annotations, associations, groups, pools, lanes, message flows, sub-process invocations and attributes associated to sub-process invocations, e.g., repetition. Nonetheless, the proposed method is applicable even if these types of elements are present in the input model. Simply, these ancillary elements and attributes need to be moved along with the tasks or events to which they are attached. In the same vein, we do not distinguish between events and tasks since, for the purpose of the transformation, both of these elements are treated equally. Secondly, we consider only sound process models [12]. This restriction is natural since soundness is a widely-accepted correctness criterion for process models. Thirdly, we consider process models in which every node has only one incoming or one outgoing arc. This restriction is merely syntactical because one can trivially split a node with multiple incoming and multiple outgoing arcs into two nodes: one node with a single outgoing arc and the other with a single incoming arc. Fourthly, we consider models with only one start node and one end node. Again, this is not a restriction since every sound model with multiple end nodes can be transformed into an equivalent sound model with a unique end node [12]. The reverse technique can be applied to models with multiple start nodes. Finally, we do not deal with the following BPMN constructs: OR gateways, complex gateways, error events and non-interrupting events. Lifting this latter restriction is left as future work.

The next section presents a taxonomy of (unstructured) process components in process models and reviews related work. Next, Sect.3 introduces the formalism used to represent process models. Sect.4 then introduces the behavioral equivalence used in this paper, viz. fully concurrent bisimulation (FCB), and shows that two acyclic process models are equivalent under this equivalence notion iff they have the same set of ordering relations. This result is used in Sect.5 to characterize the class of acyclic process components that can be structured and to define a structuring algorithm. Finally, Sect.6 concludes the paper.

2 Background and Related Work

This section discusses a complete taxonomy of process components. Next, we analyze previous work with respect to the proposed taxonomy.

2.1 Taxonomy of Process Components

The Refined Process Structure Tree (RPST) [13] is a technique to decompose a process model into a tree of regions. Each node in the RPST maps to a SESE region, herewith called a *process component*. A component in the RPST contains all components at the lower level, and all components at a given level are disjoint.

Each component in the RPST can be classified into one out of four classes [14]: A *trivial* (T) component consists of a single flow arc. A *polygon* (P) represents a sequence of components. A *bond* (B) stands for a set of components that share two common nodes. Any other component is a *rigid* (R). Rigid components explicitly define what makes a process model unstructured.

Fig.1 exemplifies the RPST decomposition in the form of dotted boxes. For instance in Fig.1(a), polygon $P1$ is the root of the RPST and corresponds to the whole process model. Polygon $P1$ contains bond $B1$ that, in turn, contains polygons $P2$ and $P3$. Observe that trivial components and polygons that are composed of two flow arcs are not visualized for simplicity reasons.

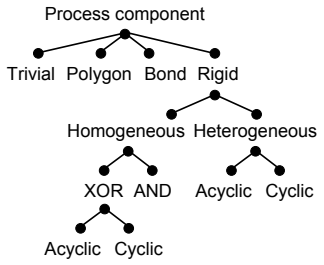


Fig. 2. Taxonomy

Trivials, polygons, and bonds are structured process components. If one could transform each rigid component in the RPST into an equivalent structured component, the entire model could be structured by traversing the RPST bottom-up and replacing each rigid by its equivalent structured component. Accordingly, the rest of the paper focuses on structuring rigid components.

The methods for structuring rigid components differ depending on the types of gateways present in the rigid and whether the rigid contains cycles or not. We classify rigids as follows. A homogeneous rigid contains either only *xor* or only *and* gateways. We call these rigids *(homogeneous) and rigids* and *(homogeneous) xor rigids*, respectively. A heterogeneous rigid contains a mixture of *and*/*xor* gateways. Heterogeneous and homogeneous *xor* rigids are further classified into cyclic, if they contain at least one cycle, or acyclic. Importantly, a safe process model cannot contain homogeneous *and* rigids with cycles. Upon this background, a taxonomy of process components is provided in Fig.2.

2.2 Related Work

The problem of structuring process models is relevant in the context of designing BPMN-to-BPEL transformations. However, BPMN-to-BPEL transformations such as [11] treat rigids as black-boxes that are translated using BPEL links or event handlers, rather than seeking to structure them. In this sense, the present contribution is complementary to this previous work.

A large body of work on flowcharts and GOTO program transformation [5], has addressed the problem of structuring *xor* rigids. In some cases, these transformations introduce additional boolean variables in order to encode part of the control flow, while in other cases they require certain nodes to be duplicated.

In [1], the authors show that not all acyclic *and* rigids can be structured. They do so by providing one counter-example, but do not give a full characterization of the class of models that can be structured nor do they define any automated transformation. Instead, they explore some causes of unstructuredness. In a similar vein, [6] presents a taxonomy of unstructuredness in process models, covering cyclic and acyclic rigids. But the taxonomy is incomplete, i.e., it does not cover all possible cases of models that can be structured. Also, the authors do not define an automated structuring algorithm.

In [7], the authors outline a classification of process components using *region trees*, a predecessor of the RPST. However, the authors do not provide a complete structuring method for acyclic heterogeneous rigids, e.g., the one in Fig.1(a). A similar remark applies to [10]. Meanwhile, [9] proposes a method for restructuring *xor* rigids based on GOTO program transformations, and extends this method to process graphs where such *xor* rigids are nested inside bonds. However, this method cannot deal with *and* rigids nor heterogeneous rigids.

3 Preliminaries

Below we introduce the notations used subsequently to represent process models.

3.1 Petri Nets

Petri nets are a well-known formalism for modeling concurrent systems. Below we present standard definitions of Petri nets and their semantics.

Definition 1 (Petri net). A *Petri net*, or a *net*, is a tuple $N = (P, T, F)$, with P and T as finite disjoint sets of *places* and *transitions*, and $F \subseteq (P \times T) \cup (T \times P)$ as the *flow* relation.

We identify F with its characteristic function on the set $(P \times T) \cup (T \times P)$. We write $X = (P \cup T)$ for all nodes of a net. For a node $x \in X$, $\bullet x = \{y \in X \mid F(y, x) = 1\}$ and $x\bullet = \{y \in X \mid F(x, y) = 1\}$. A node $x \in X$ is an *input* (*output*) node of a node $y \in X$, iff $x \in \bullet y$ ($x \in y\bullet$). For $Y \subseteq X$, $\bullet Y = \bigcup_{y \in Y} \bullet y$ and $Y\bullet = \bigcup_{y \in Y} y\bullet$. We denote by F^+ and F^* irreflexive and, respectively, reflexive transitive closures of F.

Definition 2 (Net semantics). Let $N = (P, T, F)$ be a net.

- $M : P \to \mathbb{N}_0$ is a *marking* of N, where $M(p)$, $p \in P$, returns the number of *tokens* in place p. $[p]$ denotes the marking when place p contains just one token and all other places contain no tokens.

- For any transition $t \in T$ and for any marking M of N, t is *enabled* in M, denoted by $(N, M)[t\rangle$, iff $\forall p \in \bullet t : M(p) \geq 1$.
- If $t \in T$ is enabled in M, then it can *fire*, which leads to a new marking M', denoted by $(N, M)[t\rangle(N, M')$. The new marking M' is defined by $M'(p) = M(p) - F(p,t) + F(t,p)$, for each place $p \in P$.
- A sequence of transitions $\sigma = t_1 \dots t_n$, $n \in \mathbb{N}$, is a *firing sequence*, iff there exist markings $M_0 \dots M_n$, such that for all $1 \leq i \leq n$ holds $(N, M_{i-1})[t_i\rangle(N, M_i)$.
- For any two markings M and M' of N, M' is *reachable* from M in N, denoted by $M' \in [N, M\rangle$, iff there exists a firing sequence σ leading from M to M'.
- A *net system*, or a *system*, is a pair (N, M_0), where N is a net and M_0 is a marking of N. M_0 is called the *initial marking* of N.

Workflow (WF-)nets [15] are a subclass of Petri nets specifically designed to represent business process models. A WF-net is a net with two special places: one to mark the start and the other the end of a workflow execution.

Definition 3 (WF-net, Short-circuit net, WF-system).
A Petri net $N = (P, T, F)$ is a *workflow net*, or a *WF-net*, iff N has a dedicated *source* place $i \in P$, with $\bullet i = \varnothing$, N has a dedicated *sink* place $o \in P$, with $o\bullet = \varnothing$, and the *short-circuit* net $N' = (P, T \cup \{t^\star\}, F \cup \{(o, t^\star), (t^\star, i)\})$, $t^\star \notin T$, of N is strongly connected. A *WF-system* is a pair (N, M_i), where $M_i = [i]$.

Soundness and safeness are basic properties of WF-systems [15]. Soundness states that every execution of a WF-system ends with a token in the sink place, and once a token reaches the sink place, no other tokens remain in the net. Safeness refers to the fact that there is never more than one token in the same place.

In the rest of the paper, we also use three structural subclasses of Petri nets (free-choice net, occurrence net, and causal net), as well as labeled Petri nets to distinguish observable and silent transitions. These are defined below.

Definition 4 (Free-choice net, Occurrence net, Causal net).
A Petri net $N = (P, T, F)$ is a *free-choice* net, iff $\forall p \in P, |p\bullet| > 1 : \bullet(p\bullet) = \{p\}$.
Let $N = (P, T, F)$ be a net such that $\forall x, y \in P \cup T : (x, y) \in F^+ \Rightarrow (y, x) \notin F^+$.
- Net N is an *occurrence net*, iff $\forall p \in P : |\bullet p| \leq 1$.
- Net N is a *causal net*, iff $\forall p \in P : |\bullet p| \leq 1 \wedge |p\bullet| \leq 1$.

Definition 5 (Labeled net). A *labeled* net is a tuple $N = (P, T, F, \mathcal{T}, \lambda)$, where (P, T, F) is a net, $\mathcal{T}$ is a set of *labels*, such that $\tau \in \mathcal{T}$, and $\lambda : T \to \mathcal{T}$ is a function that assigns labels to transitions. If $\lambda(t) \neq \tau$, then t is *observable*; otherwise, t is *silent*. λ is *distinctive* if it is injective on a subset of observable transitions.

3.2 Process Model

As discussed in Sect.1, we consider process models consisting of activities and gateways, as captured in the following definition.

Definition 6 (Process model).
A *process model* is a tuple $W = (A, G^+, G^\times, C, \mathcal{A}, \mu)$, where A is a non-empty set of *activities* (or *tasks*), G^+ is a set of *and gateways*, $G^\times$ is a set of *xor gateways* (these sets are disjoint). We write $G = (G^+ \cup G^\times)$ for all gateways and $Z = (A \cup G)$ for all nodes of a model. $C \subseteq Z \times Z$ defines the *control flow*. $\mathcal{A}$ is a non-empty set of *names* and $\mu : A \to \mathcal{A}$ is a function that assigns names to tasks.

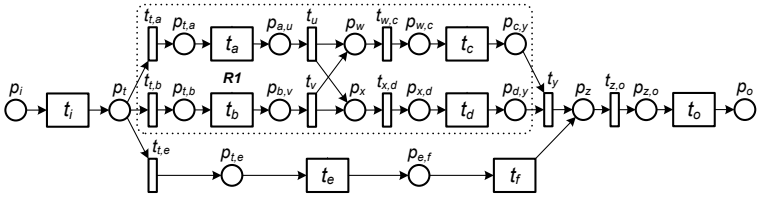


Fig. 3. A WF-net that corresponds to the process model in Fig.1(a)

A task $a \in A$ is a *source*, iff $\bullet a = \varnothing$ and it is a *sink*, iff $a\bullet = \varnothing$, where $\bullet x$, $x \in Z$, stands for a set of immediate predecessors and $x\bullet$ stands for a set of immediate successors of node x. As discussed in Sect.1, we assume that (Z, C) is a graph with a single source, a single sink and is such that every node is on a path from the source to the sink. Each task $a \in A$ has at most one incoming and at most one outgoing arc, i.e., $|\bullet a| \leq 1 \wedge |a\bullet| \leq 1$, while each gateway is either a split or a join: A gateway $g \in G$ is a *split*, iff $|\bullet g| = 1 \wedge |g\bullet| > 1$. A gateway $g \in G$ is a *join*, iff $|\bullet g| > 1 \wedge |g\bullet| = 1$. The execution semantics of process models is defined by a mapping to labeled free-choice Petri nets (cf. Definition 7). For example, Fig.3 shows the WF-net of the process model in Fig.1(a). The figure highlights the subnet that corresponds to rigid component $R1$ in Fig.1(a) (cf. dotted box).

Definition 7 (WF-net of a process model). Let $W = (A, G^{+}, G^{\times}, C, \mathcal{A}, \mu)$ be a process model. Let I and O be sources and sinks of W, respectively. The labeled net $N = (P, T, F, \mathcal{T}, \lambda)$ *corresponding* to W is defined by:

- $P = \{p_x \mid x \in G^{\times}\} \cup \{p_{x,y} \mid (x,y) \in C \wedge y \in A \cup G^{+}\} \cup \{p_x \mid x \in I \cup O\}$.
- $T = \{t_x \mid x \in A \cup G^{+}\} \cup \{t_{x,y} \mid (x,y) \in C \wedge x \in G^{\times}\}$.
- $F = \{(t_x, p_y) \mid (x,y) \in C \wedge x \in A \cup G^{+} \wedge y \in G^{\times}\} \cup \{(t_x, p_{x,y}) \mid (x,y) \in C \wedge x, y \in A \cup G^{+}\} \cup \{(t_{x,y}, p_y) \mid (x,y) \in C \wedge x, y \in G^{\times}\} \cup \{(t_{x,y}, p_{x,y}) \mid (x,y) \in C \wedge x \in G^{\times} \wedge y \in A \cup G^{+}\} \cup \{(p_x, t_{x,y}) \mid (x,y) \in C \wedge x \in G^{\times}\} \cup \{(p_{x,y}, t_y) \mid (x,y) \in C \wedge y \in A \cup G^{+}\} \cup \{(p_x, t_x) \mid x \in I\} \cup \{(t_x, p_x) \mid x \in O\}$.
- $\mathcal{T} = \mathcal{A} \cup \{\tau\}$. $\lambda(t_x) = \mu(x), t_x \in T, x \in A$, otherwise $\lambda(t) = \tau, t \in T$.

Definition 7 states that a task is mapped to a Petri net transition with a single input and a single output arc. An *and* gateway maps to a transition with multiple outgoing arcs (*and*-split) or multiple incoming arcs (*and*-join). An *xor* gateway maps to a place with multiple outgoing arcs (*xor*-split) or multiple incoming arcs (*xor*-join). The places corresponding to *xor*-splits are immediately followed by empty (τ) transitions representing the branching conditions (cf. transitions $t_{x,y}$ introduced in the mapping). Sources and sinks of the process model are mapped to places.

A process model is sound if its corresponding WF-net is sound. In this paper we only consider sound process models. We note that a sound free-choice WF-system is guaranteed to be safe [16] and Definition 7 always produces free-choice WF-nets. Thus the rest of the paper deals with sound and safe process models.

4 Behavioral Equivalence of Process Models

This section motivates fully concurrent bisimulation as the equivalence notion for process models, cf., Sect.4.1, and discusses the procedure of checking equivalence for the class of behavior captured by occurrence nets, cf., Sect.4.2.

4.1 Fully Concurrent Bisimulation

An unstructured model and the corresponding structured model are structurally different, but behaviorally equivalent. There exist many notions of behavioral equivalence for concurrent systems [17]. A common notion of behavioral equivalence for concurrent systems is that of *bisimulation*. Related notions are those of *weak bisimulation* and *branching bisimulation*, which abstract away from silent transitions. These notions have been advocated as being suitable for comparing process models [12]. However, we argue that they are not suitable for our purposes. These three notions adopt an interleaving semantics – i.e., no two tasks are executed exactly at the same time. Thus, a concurrent system and its sequential simulation are considered equivalent. For example, Fig.4 shows the sequential simulation of the net in Fig.3. This net is structured and weakly bisimilar to the net in Fig.3, but it contains no parallel branch. We could take any process model, compute its sequential simulation, structure this sequential net using GOTO program transformations, and transform back the resulting sequential net into a structured process model. This structuring method is complete, but if we start with a process model containing *and* gateways, we obtain a (much larger) structured process model without any parallel branches.

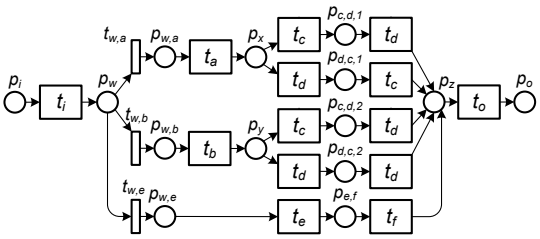


Fig. 4. Sequential simulation of the net in Fig.3

Accordingly, we adopt a notion of equivalence that preserves the level of concurrency of observable transitions, viz. *fully concurrent bisimulation* (FCB) [18]. FCB is defined in terms of concurrent runs of a system, a.k.a. *processes* in the literature (but not to be confused with "business processes" or workflows).

Let $N = (P, T, F)$ be a causal net. A *P-cut* $c \subseteq P$ of N is a maximal set of places unordered w.r.t. F^+. Let $Min(N)$ define the set $\{x \in X \mid \bullet x = \varnothing\}$ and let $Max(N)$ define the set $\{x \in X \mid x\bullet = \varnothing\}$.

Definition 8 (Process). A *process* $\pi = (N_\pi, \rho)$ of a system $S = (N, M_0)$, $N = (P, T, F)$, consists of a causal net $N_\pi = (P_\pi, T_\pi, F_\pi)$ and a function $\rho : X_\pi \to X$:

- $\rho(P_\pi) \subseteq P,\ \rho(T_\pi) \subseteq T$,
- $Min(N_\pi)$ is a P-cut, which corresponds to the initial marking M_0, that is $\forall\ p \in P : M_0(p) = |\rho^{-1}(p) \cap Min(N_\pi)|$, and
- $\forall\ t \in T_\pi\ \forall\ p \in P : (F(p, \rho(t)) = |\rho^{-1}(p) \cap \bullet t|) \wedge (F(\rho(t), p) = |\rho^{-1}(p) \cap t \bullet|)$.

A process π of S is *initial*, iff $T_\pi = \varnothing$.

A process π' is an extension of a process π if it is possible to observe π before one observes π'. Consequently, process π is a prefix of π'.

Definition 9 (Prefix, Process extension).
Let $\pi = (N_\pi, \rho)$, $N_\pi = (P_\pi, T_\pi, F_\pi)$, be a process of $S = (N, M_0)$, $N = (P, T, F)$. Let c be a P-cut of N_π and let $c^\downarrow$ be the set $\{x \in X_\pi \mid \exists\ y \in c : (x, y) \in F^*\}$. A process $\pi^\downarrow_c$ is a *prefix* of π, iff $\pi^\downarrow_c = ((P_\pi \cap c^\downarrow, T_\pi \cap c^\downarrow, F \cap (c^\downarrow \times c^\downarrow)), \rho|_{c^\downarrow})$. A process π' is an *extension* of process π if π is a prefix of π'.

In order to define FCB, we need two auxiliary definitions: λ-abstraction of a process, which is a process footprint that ignores silent transitions, and the order-isomorphism of λ-abstractions.

Definition 10 (Abstraction of a process of a labeled system).
Let $S = (N, M_0)$, $N = (P, T, F, \mathcal{T}, \lambda)$, be a labeled system and let $\pi = (N_\pi, \rho)$, $N_\pi = (P_\pi, T_\pi, F_\pi)$, be a process of S. The λ-abstraction of π, denoted by $\alpha_\lambda(\pi) = (T'_\pi, \prec, \lambda')$, is defined by $T'_\pi = \{t \in T_\pi \mid \lambda(\rho(t)) \neq \tau\}$, $\prec = \{(t_1, t_2) \in T'_\pi \times T'_\pi \mid (t_1, t_2) \in F^+\}$, and $\lambda' : T'_\pi \to \mathcal{T}$, such that $\lambda'(t) = \lambda(\rho(t))$, $t \in T'_\pi$.

Two λ-abstractions are order-isomorphic if there exists a one-to-one correspondence between transitions of both abstractions that also preserves the ordering of the corresponding transitions in the respective abstractions.

Definition 11 (Order-isomorphism of abstractions).
Let $\alpha_{\lambda_1} = (T_1, \prec_1, \lambda_1)$ and $\alpha_{\lambda_2} = (T_2, \prec_2, \lambda_2)$ be two λ-abstractions, both with labels in $\mathcal{T}$. Then α_{λ_1} and α_{λ_2} are *order-isomorphic*, denoted by $\alpha_{\lambda_1} \cong \alpha_{\lambda_2}$, iff there is a bijection $\beta : T_1 \to T_2$ such that $\forall\ t \in T_1 : \lambda_1(t) = \lambda_2(\beta(t))$ and $\forall\ t_1, t_2 \in T_1 : t_1 \prec_1 t_2 \Leftrightarrow \beta(t_1) \prec_2 \beta(t_2)$.

Given the above, fully concurrent bisimulation is defined as follows.

Definition 12 (Fully concurrent bisimulation).
Let $S_1 = (N_1, M_0^1)$ and $S_2 = (N_2, M_0^2)$ be labeled systems, $N_1 = (P_1, T_1, F_1, \mathcal{T}_1, \lambda_1)$ and $N_2 = (P_2, T_2, F_2, \mathcal{T}_2, \lambda_2)$. S_1 and S_2 are *fully concurrent bisimilar*, denoted by $S_1 \approx S_2$, iff there is a set $\mathcal{B} \subseteq \{(\pi_1, \pi_2, \beta)\}$, such that:

(i) π_1 is a process of S_1, π_2 is a process of S_2, and β is a relation between the non-τ transitions of π_1 and π_2.
(ii) If π_0^1 and π_0^2 are the initial processes of S_1 and S_2, respectively, then $(\pi_0^1, \pi_0^2, \varnothing) \in \mathcal{B}$.
(iii) If $(\pi_1, \pi_2, \beta) \in \mathcal{B}$, then β is an order-isomorphism between the λ_1-abstraction of π_1 and the λ_2-abstraction of π_2.
(iv) $\forall (\pi_1, \pi_2, \beta) \in \mathcal{B}$:
  (a) If π'_1 is an extension of π_1, then $\exists (\pi'_1, \pi'_2, \beta') \in \mathcal{B}$ where π'_2 is an extension of π_2 and $\beta \subseteq \beta'$.
  (b) Vice versa.

FCB defines an equivalence relation on labeled systems that is stricter than weak bisimulation and related notions. The nets in Fig.4 and Fig.3 are weakly bisimilar but not FCB-equivalent. Meanwhile, the two models in Fig.1 are FCB-equivalent (with the understanding that two process models are FCB-equivalent if the corresponding Petri nets are FCB-equivalent).

4.2 Behavioral Equivalence and Ordering Relations

The above definition of FCB-equivalence is abstract and hardly of any use when synthesizing structured nets from unstructured ones. Accordingly, we employ a more convenient way of reasoning about FCB-equivalence based on the *ordering relations* of occurrence nets. The idea is that any pair of nodes in an occurrence net can be in a precedence, conflict, or concurrent ordering relation as defined below, and these ordering relations can be used to reason about FCB-equivalence.

Definition 13 (Ordering relations).
Let $N = (P, T, F)$ be an occurrence net and let $x, y \in X$ be two nodes of N.

- x *precedes* y, denoted by $x \rightsquigarrow_N y$, iff $(x, y) \in F^+$.
- x and y are in *conflict*, denoted by $x \#_N y$, iff $\exists\ t_1, t_2 \in T, t_1 \neq t_2 : (\bullet t_1 \cap \bullet t_2 \neq \emptyset) \wedge t_1 \rightsquigarrow_N x \wedge t_2 \rightsquigarrow_N y$.
- x and y are *concurrent*, denoted by $x \|_N y$, iff they are neither in precedence, nor in conflict.

The set $\mathcal{R} = \{\rightsquigarrow_N, \#_N, \|_N\}$ forms the *ordering relations* of N.

Let $N = (P, T, F, \mathcal{T}, \lambda)$ be a labeled occurrence net and let $T' = \{t \in T \mid \lambda(t) \neq \tau\}$. The λ-*ordering relations* of N are formed by the set $\mathcal{R}_\lambda = \{\rightsquigarrow_N \cap\ T' \times T', \#_N \cap\ T' \times T', \|_N \cap\ T' \times T'\}$. We say that two ordering relations are isomorphic if for each pair of observable transitions the ordering relation coincides.

Definition 14 (Isomorphism of ordering relations).
Let $N_1 = (P_1, T_1, F_1, \mathcal{T}_1, \lambda_1)$ and $N_2 = (P_2, T_2, F_2, \mathcal{T}_2, \lambda_2)$ be two labeled occurrence nets with distinctive labelings. Let T_1' and T_2' denote non-τ transitions of N_1 and N_2, respectively. Two λ-ordering relations $\mathcal{R}_{\lambda_1}$ of N_1 and $\mathcal{R}_{\lambda_2}$ of N_2 are *isomorphic*, denoted by $\mathcal{R}_{\lambda_1} \cong\ \mathcal{R}_{\lambda_2}$, iff there is a bijection $\gamma : T_1' \to T_2'$, such that:

- $\forall\ t \in T_1' : \lambda_1(t) = \lambda_2(\gamma(t))$, and
- $\forall\ t_1, t_2 \in T_1' : (t_1 \rightsquigarrow_{N_1} t_2 \wedge \gamma(t_1) \rightsquigarrow_{N_2} \gamma(t_2)) \vee (t_2 \rightsquigarrow_{N_1} t_1 \wedge \gamma(t_2) \rightsquigarrow_{N_2} \gamma(t_1)) \vee (t_1 \#_{N_1} t_2 \wedge \gamma(t_1) \#_{N_2} \gamma(t_2)) \vee (t_1 \|_{N_1} t_2 \wedge \gamma(t_1) \|_{N_2} \gamma(t_2))$.

Finally, we show that two occurrence nets with isomorphic ordering relations are FCB-equivalent, and vice-versa. This result is exploited in the next section.

Theorem 1. *Let $S_1 = (N_1, M_i^1)$, $N_1 = (P_1, T_1, F_1, \mathcal{T}_1, \lambda_1)$, and $S_2 = (N_2, M_i^2)$, $N_2 = (P_2, T_2, F_2, \mathcal{T}_2, \lambda_2)$, be two labeled occurrence systems with distinctive labelings and $T_1' \subseteq T_1$, $T_2' \subseteq T_2$ observable transitions, such that there exists bijection $\psi : T_1' \to T_2'$ for which holds $\lambda_1(t) = \lambda_2(\psi(t))$, for all $t \in T_1'$. Let $\mathcal{R}_{\lambda_1}$ and $\mathcal{R}_{\lambda_2}$ be the λ-ordering relations of N_1 and N_2. Then, it holds:*

$$S_1 \approx S_2 \Leftrightarrow \mathcal{R}_{\lambda_1} \cong\ \mathcal{R}_{\lambda_2}.$$

Proof. We prove each direction of the equality separately.

($\Rightarrow$) Let S_1 and S_2 be FCB-equivalent. We want to show that $\mathcal{R}_{\lambda_1} \cong\ \mathcal{R}_{\lambda_2}$.
Let us assume that $S_1 \approx S_2$ holds, but $\mathcal{R}_{\lambda_1} \cong\ \mathcal{R}_{\lambda_2}$ does not hold. Furthermore, let us consider transitions $t_i^1, t_j^1 \in T_1'$ that are in one-to-one correspondence with transitions $t_i^2, t_j^2 \in T_2'$, i.e., $\lambda_1(t_i^1) = \lambda_2(\psi(t_i^2))$ and $\lambda_1(t_j^1) = \lambda_2(\psi(t_j^2))$. All scenarios can be reduced to the following two cases:

Case 1: ($t_i^1 \|_{N_1} t_j^1$ or $t_i^1 \rightsquigarrow_{N_1} t_j^1$, and $t_i^2 \#_{N_2} t_j^2$). If $t_i^1 \|_{N_1} t_j^1$ or $t_i^1 \rightsquigarrow_{N_1} t_j^1$, then there exists process π_1 in S_1 that contains t_i^1 and t_j^1. If $t_i^2 \#_{N_2} t_j^2$, then there exists no process π_2 in S_2 that contains t_i^2 and t_j^2.

Case 2: ($t_i^1 \rightsquigarrow_{N_1} t_j^1$, and $t_j^2 \rightsquigarrow_{N_2} t_i^2$ or $t_i^2 \|_{N_2} t_j^2$). Let π_1 be a process in S_1 that contains t_i^1 and t_j^1, and let π_2 be a process in S_2 that contains t_i^2 and t_j^2. Then, there exists no $\phi \subseteq \psi$, such that ϕ is an order-isomorphism between λ-abstractions of π_1 and π_2.

In both cases we reach the contradiction, i.e., systems $S1$ and $S2$ cannot be FCB-equivalent if the λ-ordering relations are not isomorphic.

($\Leftarrow$) Let $\mathcal{R}_{\lambda_1} \cong \mathcal{R}_{\lambda_2}$. We want to show that S_1 and S_2 are FCB-equivalent. Let us assume that $\mathcal{R}_{\lambda_1} \cong \mathcal{R}_{\lambda_2}$ holds, but $S_1 \approx S_2$ does not hold. Then, for instance, in S_1 there exists process π'_1 that has no corresponding order-isomorphic process in S_2. Suppose that π'_1 has the minimal size among all such processes, i.e., any prefix of π'_1 has a corresponding order-isomorphic process in S_2. Let π'_1 be an extension of π_1 by exactly one observable transition $t^1_j \in T'_1$. Let π_2 be a process in S_2 that is order-isomorphic with π_1. Let $t^2_j \in T'_2$ be in one-to-one correspondence with t^1_j, i.e., $\lambda_1(t^1_j) = \lambda_2(\psi(t^2_j))$. All scenarios can be reduced to the following three cases:

Case 1: There exists process π'_2 that contains t^2_j and is an extension of π_2 by one observable transition. Moreover, there exists $t^1_i \in T'_1$ in π_1, such that $t^1_i \rightsquigarrow_{N_1} t^1_j$. However, it holds $t^2_i \parallel_{N_2} t^2_j$, for $t^2_i \in T'_2$, such that $\lambda_1(t^1_i) = \lambda_2(\psi(t^2_i))$; otherwise there exists an order-isomorphism $\phi \subseteq \psi$ between π'_1 and π'_2.

Case 2: There exists no process π'_2 that contains t^2_j and is an extension of π_2. Moreover, there exists $t^1_i \in T'_1$ in π_1, such that $t^1_i \rightsquigarrow_{N_1} t^1_j$. However, it holds $t^2_i \#_{N_2} t^2_j$, for $t^2_i \in T'_2$, such that $\lambda_1(t^1_i) = \lambda_2(\psi(t^2_i))$.

Case 3: There exists process π'_2 that contains t^2_j and is an extension of π_2, but not by only one observable transition. Then, there exists $t^2_k \in T'_2$ and process π''_2 in S_2, such that $t^2_k \rightsquigarrow_{N_2} t^2_j$, π''_2 is prefix of π'_2, and π_2 is prefix of π''_2. However, $t^1_k \in T'_1$, $\lambda_1(t^1_k) = \lambda_2(\psi(t^2_k))$, is not in π'_1 and, hence, $t^1_k \not\rightsquigarrow_{N_1} t^1_j$.

In all three cases we reach the contradiction, i.e., the λ-ordering relations cannot be isomorphic if systems S_1 and S_2 are not FCB-equivalent. □

5 Synthesis of Structured Process Models

The key idea of the proposed structuring method is to compute the ordering relations of every rigid component, and to synthesize a structured process component from these ordering relations (if such a structured process component exists). A structured process component is one whose RPST contains only trivials, bonds and polygons. Accordingly, what we need is to find such structures in the graph induced by the ordering relations of the component. To this end, we rely on the concept of modular decomposition [19]. Below we discuss how to compute the ordering relations of a process component and then we use the output of this step to synthesize a structured component based on the modular decomposition.

5.1 Computing Ordering Relations

In order to compute the ordering relations of tasks in a process component, we first need to build a corresponding occurrence net, using a procedure known as unfolding [20]. For example, Fig.5 presents the occurrence net for the rigid component $R1$ in Fig.3. The occurrence net may include multiple transitions referring to the same task, e.g., transitions $t_{c,1}$ and $t_{c,2}$ refer to task c. If we used the ordering relations computed from the occurrence net to synthesize a structured process component, the component would contain many duplicate tasks. Fortunately, for any safe net there exists a prefix of its occurrence net, called the *complete prefix unfolding* [20], that is more compact than the occurrence net

but contains all the information about markings contained in the occurrence net. Moreover, this prefix is finite (even for safe nets with cycles). The complete prefix unfolding is obtained by truncating the occurrence net in points where the information about reachable markings starts to be redundant.

Definition 15 (Complete Prefix Unfolding, Cutoff transition).
Let $N = (P, T, F)$ be an occurrence net.

- A *local configuration* $\lceil t \rceil$ of a transition t in an occurrence net is the set of transitions that precede t, i.e., $\lceil t \rceil = \{t' \in T \mid (t', t) \in F^*\}$.
- The *final marking* of a local configuration $Mark(\lceil t \rceil)$ is the set of places that are marked after all the transitions in $\lceil t \rceil$ fire.
- An *adequate order* $\lhd$ is a strict well-founded partial order on local configurations, so that $\lceil t \rceil \subset \lceil t' \rceil$ implies $\lceil t \rceil \lhd \lceil t' \rceil$[3].
- A transition t of an occurrence net is a *cutoff transition* if there exists a *corresponding transition* t', such that $Mark(\lceil t \rceil) = Mark(\lceil t' \rceil)$ and $\lceil t' \rceil \lhd \lceil t \rceil$.
- A *complete prefix unfolding* is the greatest backward closed subnet of an occurrence net containing no transitions after cutoff transitions.

The dotted lines in Fig.5 indicate which parts of the occurrence net are truncated in the complete prefix unfolding. In the unfolding, transition t_v is a cutoff transition.

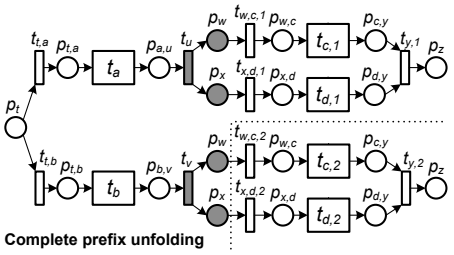


Fig. 5. Occurrence net and complete prefix unfolding of the running example

Alg.1 (adapted from [21]) computes the ordering relations based on a complete prefix unfolding. This algorithm has a low polynomial time to the size of the net. However, the overall complexity of computing ordering relations is dominated by the exponential worst-case complexity of computing the prefix unfolding, which is an NP-complete problem. Observe that in the case of *and* rigids this step is not required as the corresponding WF-net is always an occurrence net. Besides, we do not compute the prefix unfolding over the whole net, but only on individual rigid components of the net. Tests we have conducted with sample process models show that the prefix unfolding computation takes sub-second times[4]. This finding is in line with other work that have empirically shown that prefix unfolding computation is efficient in practice [20].

Alg.1 comprises two phases. First, it computes the ordering relations of transitions on the unfolding according to Definition 13. Then, it updates the relations of transitions in the local configuration of every cutoff transition to overcome the effects of truncation. The update must be performed in reverse topological order. For instance, the algorithm will assert $(t_b \rightsquigarrow t_{c,1})$ in addition to $(t_a \rightsquigarrow t_{c,1})$, i.e., task c may be preceded by either a or b. Note that we impose an additional requirement

[3] Several definitions of *adequate order* exist; we use the one defined in [20], because it has been shown to generate compact unfoldings.

[4] Using the Mole tool for prefix unfolding `http://www.fmi.uni-stuttgart.de/szs/tools/mole/` which implements the algorithm in [20].

Algorithm 1: Compute Ordering Relations of a Complete Prefix Unfolding

Input: A WF-net $W = (P_w, T_w, F_w)$, its Complete Prefix Unfolding $U = (P, T, F)$, and the mapping function $l_w : T \to T_w$
Output: Matrix $ORel$ containing the ordering relations of transitions in U
foreach $t_i, t_j \in T$ **do** Assert $(t_i \parallel_U t_j)$ in $ORel$
foreach $t_i \in T$ following a preorder traversal of U **do**
  foreach $t_j \in T$ such that $t_j \in \bullet(\bullet t_i)$ **do**
    Assert $(t_j \rightsquigarrow_U t_i)$ in $ORel$
    foreach $t_k \in T$ such that $(t_k \rightsquigarrow_U t_j) \in ORel$ **do**
      Assert $(t_k \rightsquigarrow_U t_i)$ in $ORel$
    foreach $t_k \in T$ such that $(t_k \#_U t_j) \in ORel$ **do**
      Assert $(t_k \#_U t_i)$, $(t_i \#_U t_k)$ in $ORel$
  foreach $t_j \in T$ such that $t_i \neq t_j \wedge \bullet t_i \cap \bullet t_j \neq \varnothing$ **do**
    Assert $(t_k \#_U t_i)$ in $ORel$
    foreach $t_k \in T$ such that $(t_j \rightsquigarrow_U t_k) \in ORel$ **do**
      Assert $(t_k \#_U t_i)$, $(t_i \#_U t_k)$ in $ORel$
`// Iterate over cutoff transitions in reverse topological order`
foreach $t_i, t_j, t_k \in T$ such that t_i is a cutoff in U, t_j is not a cutoff in U, $Mark(\lceil t_i \rceil) = Mark(\lceil t_j \rceil)$, $l_w(t_i)\bullet = l_w(t_j)\bullet$, and $l_w(t_i) \bullet \cap \bullet\, l_w(t_k) \neq \varnothing$ **do**
  foreach $t_m, t_n \in T$ such that $(t_k \rightsquigarrow_U t_m) \in ORel \vee t_k = t_m$ and $t_n \in \lceil t_i \rceil$ **do**
    Assert $(t_n \rightsquigarrow_U t_m)$ in $ORel$
return $ORel$

i.e., postsets of a cutoff transition and its corresponding transition must map to the same set of places in the WF-net, for all cutoff transitions. If a complete prefix unfolding does not meet this requirement, it must be expanded.

5.2 From Ordering Relations to Process Models

This section presents the algorithm for synthesizing a well-structured process model that is fully concurrent bisimilar with a given (unstructured) model. Also, we identify the cases when an equivalent well-structured model does not exist.

According to Theorem 1, two process models are fully concurrent bisimilar, iff they demonstrate same ordering relations. Given an (unstructured) process model, the algorithm proceeds by computing its ordering relations, as discussed in Sect. 5.1. Afterwards, the algorithm attempts to synthesize a well-structured model with the same ordering relations.

Let $N = (P, T, F, \mathcal{T}, \lambda)$ be a labeled occurrence net. The *ordering relations graph* of N is a triple $\mathcal{G}_\lambda = (V, E, \mathcal{L})$, where V is the set of non-τ transitions of N, $\mathcal{L} = \{\epsilon, \rightsquigarrow, \#, \parallel\}$ is a set of labels, and $E : V \times V \to \mathcal{L}$ is an edge labeling function, such that $E(x, y) = \oplus$, $x, y \in V$ and $\oplus \in \mathcal{L} \setminus \epsilon$, if $x \oplus_N y$, otherwise $E(x, y) = \epsilon$. Self-relations are ignored, i.e., $E(x, x) = \epsilon$, $x \in V$. Observe that $\mathcal{G}_\lambda$ is an alternative representation of λ-ordering relations $\mathcal{R}_\lambda$ of N.

Fig.6(a) shows the ordering relations graph of a complete prefix unfolding that is given in Fig.5. As the conflict and concurrency relations are symmetric, the corresponding edges are visualized as two-sided arrows; solid and dotted for the conflict $(a \# b)$ and concurrency $(c \parallel d)$ relation, respectively. Regular arrows reflect the precedence relation, which is transitive and asymmetric. Edges that have ϵ labels are not visualized. Because of the precedence relation, most of the ordering relations graphs are asymmetric.

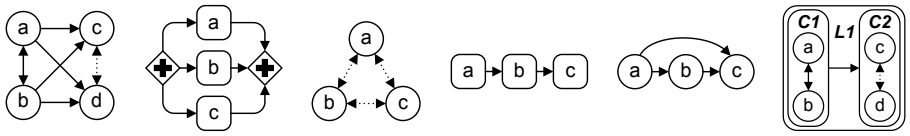


Fig. 6. (a) An ordering relations graph, (b) an *and* bond component, (c) an *and* complete module, (d) a polygon component, (e) a linear module, (f) the MDT of (a)

The RPST of a well-structured model is composed of trivial, polygon, and bond (either *and* or *xor*) components. Contrary to a rigid component that can have an arbitrary topology, the structure of each component of a well-structured model is well-defined and has a precise structural characterization in terms of the corresponding ordering relations graph. The ordering relations graph of a bond is a complete graph, or a clique. All edges in the graph have the same label: # for *xor* bonds and ∥ for *and* bonds. This topology is consistent with the intuition behind: all nodes in a *xor* bond are in conflict, i.e., only one is executed; all nodes in an *and* bond are concurrently executed. Fig.6(b) shows an *and* bond with three parallel branches, whereas Fig.6(c) shows the corresponding clique of concurrent relations. In the cases of a trivial and polygon component, the ordering relations graph is a direct acyclic graph representing the transitive closure, or the total order, of the precedence relation. All edges of the graph are labeled $\rightsquigarrow$. Fig.6(d) shows a polygon composed of three activities, whereas Fig.6(e) presents the corresponding transitive closure over the precedence relation.

Let $\mathcal{G} = (V, E, \mathcal{L})$ be an ordering relations graph. A *module* $M \subseteq V$ of $\mathcal{G}$ is a non-empty subset of transitions that have a uniform relations with transitions $V \setminus M$, i.e., $\forall\ x, y \in M\ \forall\ z \in V \setminus M : E(x, z) = E(y, z) \wedge E(z, x) = E(z, y)$. Note that singleton sets of V are referred to as *trivial* modules.

Definition 16 (Complete, Linear, Primitive).
Let M be a non-singleton module of $\mathcal{G}$.

- M is *complete* (C), iff $\exists\ l \in \{\#, \|\}\ \forall\ x, y \in M, x \neq y : E(x, y) = l$, i.e., the subgraph induced by M is a complete graph, or a clique. If $l = \#$, then M is *xor* complete, otherwise M is *and* complete.
- M is *linear* (L), iff there exists a linear order $(x_1, ..., x_{|T'|})$ of elements of T', such that $E(x_i, x_j) = \rightsquigarrow$, if $i < j$, and $E(x_i, x_j) = \epsilon$ otherwise.
- If M is neither complete, nor linear, then M is *primitive* (P).

The following proposition summarizes relations between components of a process model and modules of an ordering relations graph.

Proposition 1. Let C_1 be a process component and let M_1 be the corresponding ordering relations graph. Let M_2 be an ordering relations graph and let C_2 be the corresponding process component.

1. If C_1 is trivial or polygon, then M_1 is linear.
2. If M_2 is linear, then there exists C_2 that is trivial or polygon.
3. If C_1 is *and* (*xor*) bond, then M_1 is *and* (*xor*) complete.
4. If M_2 is *and* (*xor*) complete, then there exists C_2 that is *and* (*xor*) bond.

Two modules M_1 and M_2 of $\mathcal{G}$ *overlap*, iff they intersect and neither is a subsetof

the other, i.e., $M_1 \setminus M_2$, $M_1 \cap M_2$, and $M_2 \setminus M_1$ are all non-empty. M_1 is *strong*, iff there exists no module M_2 of $\mathcal{G}$, such that M_1 and M_2 overlap. The *modular decomposition* substitutes each strong module of a graph by a new vertex and proceeds recursively. The result is a rooted, unique tree called the *Modular Decomposition Tree*, which can be computed in linear time [19].

Definition 17 (Modular Decomposition Tree). Let $\mathcal{G} = (V, E, \mathcal{L})$ be an ordering relations graph. The *Modular Decomposition Tree* (MDT) of $\mathcal{G}$, denoted by $MDT(\mathcal{G})$, is a containment hierarchy of all strong modules of $\mathcal{G}$.

Fig.6(f) shows the MDT of the ordering relations graph that is proposed in Fig.6(a). Each module is enclosed in a box with rounded corners. Note that module names hint at their class. For instance, module $C1$ is a complete module, and is composed of two nodes a and b that are in conflict relation, $a \# b$. Therefore, $C1$ is a *xor* complete module. Similarly, $C2$ is an *and* complete module. By treating both modules as singletons, the modular decomposition identifies that they are in total order and, hence, form a linear module $L1$.

We are now ready to present the main result of this section.

Theorem 2. *Let $\mathcal{G}$ be an ordering relations graph. The Modular Decomposition Tree of $\mathcal{G}$ has no primitive module, iff there exists a well-structured process model W such that $\mathcal{G}$ is the ordering relations graph of W.*

Proof. Let $\mathcal{G} = (V, E, \mathcal{L})$ be an ordering relations graph.

($\Rightarrow$) Assume that the MDT of $\mathcal{G}$ has no primitive module. We show now by structural induction on the MDT of $\mathcal{G}$ that there exists a well-structured process model W with ordering relations $\mathcal{G}$. The MDT of $\mathcal{G}$ contains singleton, linear, *and* complete, and *xor* complete modules.

Base: If the MDT of $\mathcal{G}$ consists of a single module M, then M is singleton and W is a process model composed of a single task $m \in M$.

Step: Let M be a module of the MDT of $\mathcal{G}$ such that each child module of M has a corresponding well-structured process model. If M is linear, then W can be a trivial or polygon component composed from children of M, cf., 2 in Prop. 1. If M is complete, then W can be a bond component, either *and* or *xor*, composed from children of M, cf., 4 in Prop. 1. In both cases, M has a corresponding well-structured process model.

Therefore, there exists a well-structured process model W composed from children of module V of $\mathcal{G}$ that has ordering relations graph $\mathcal{G}$.

($\Leftarrow$) Let W be a well-structured process model with ordering relations graph $\mathcal{G}$. We want to show that the MDT of $\mathcal{G}$ has no primitive module. Because W is well-structured, the RPST of W has no rigid component. The corresponding ordering relations graph of a non-rigid component, i.e., trivial, polygon, or bond component, is either complete or linear, cf., 1 and 3 in Prop. 1. If W is composed of a single task, then $\mathcal{G}$ consists of one singleton trivial module. □

Finally, we detail the approach for structuring acyclic rigid components in Alg.2.

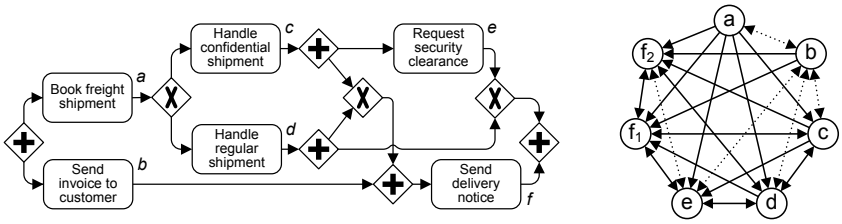
Based on all previous results, Alg.2 synthesizes, whenever possible, the RPST of an FCB-equivalent well-structured component for a given acyclic rigid component. No variables are introduced. Task duplication depends on the "quality" of the prefix unfolding. The complexity of the algorithm is determined by the exponential

Algorithm 2: Restructure an Acyclic Rigid Process Component

Input: An acyclic rigid process component
Output: The RPST of a well-structured process component
Compute the complete prefix unfolding of the input process component
Compute the ordering relations of the unfolding (Alg. 1)
Restrict the ordering relations to the set of non-τ transitions
Compute the MDT of the graph formed with the restricted ordering relations
Construct the RPST by traversing each module M of the MDT (in postorder)
- If M is trivial singleton, then generate a task
- If M is *and* complete, then generate an *and* bond component
- If M is *xor* complete, then generate a *xor* bond component
- If M is linear, then generate a trivial or polygon component
- If M is primitive, then FAIL `// component cannot be restructured`

return the RPST

complexity of the unfolding (see earlier discussion). All other steps are polynomial. In the case of an *and* rigid, the unfolding is not needed because the WF-net of an *and* rigid is already an occurrence net. The algorithm fails if the input process component is inherently unstructured, such as the process component in Fig.7(a). In this particular case, the ordering relations graph forms a single primitive module, cf., Fig.7(b). Note that the unfolding step duplicates task f.



(a) (b)

Fig. 7. (a) A rigid process component and (b) its ordering relations graph

6 Conclusion

We conclude that a sound and safe acyclic process model is inherently unstructured if its RPST has a rigid component for which the modular decomposition of its ordering relations contains a primitive. In all other cases, Algorithm 2 applied to each rigid in the RPST constructs an equivalent structured model. We have thus provided a characterization of the class of structured acyclic process models under FCB equivalence, and a complete structuring method. This method is implemented in a tool, namely `bpstruct`, that structures BPMN models exported from Oryx[5]. The tool is available at `https://code.google.com/p/bpstruct/`.

This method can also be used to structure models with SESE cycles, even if these cycles contain unstructured components. In this case, the unstructured components and the cycles are in different nodes of the RPST. However, the

[5] http://oryx-project.org/

proposed method cannot deal with models with arbitrary cycles. Also, the results do not apply to models with OR-joins, complex gateways, exception handlers and non-interrupting events. Future work will aim at lifting these restrictions.

Acknowledgments. This research is partly funded by the ERDF via the Estonian Center of Excellence in Computer Science.

References

1. Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.: On Structured Workflow Modelling. In: CAiSE. Volume 1789 of LNCS. (2000) 431–445
2. Laue, R., Mendling, J.: The Impact of Structuredness on Error Probability of Process Models. In: UNISCON. Volume 5 of LNBIP. (2008) 585–590
3. Laguna, M., Marklund, J.: Business Process Modeling, Simulation, and Design. Prentice Hall (2005)
4. Combi, C., Posenato, R.: Controllability in Temporal Conceptual Workflow Schemata. In: BPM. Volume 5701 of LNCS. (2009) 64–79
5. Oulsnam, G.: Unravelling unstructured programs. Comput. J. **25** (1982) 379–387
6. Liu, R., Kumar, A.: An Analysis and Taxonomy of Unstructured Workflows. In: BPM. Volume 3649 of LNCS. (2005) 268–284
7. Hauser, R., Friess, M., Küster, J.M., Vanhatalo, J.: An Incremental Approach to the Analysis and Transformation of Workflows Using Region Trees. IEEE Transactions on Systems, Man, and Cybernetics, Part C **38** (2008) 347–359
8. Polyvyanyy, A., García-Bañuelos, L., Weske, M.: Unveiling Hidden Unstructured Regions in Process Models. In: OTM. Volume 5870 of LNCS. (2009) 340–356
9. Hauser, R., Koehler, J.: Compiling Process Graphs into Executable Code. In: GPCE. Volume 3286 of LNCS. (2004) 317–336
10. Koehler, J., Hauser, R.: Untangling Unstructured Cyclic Flows - A Solution Based on Continuations. In: OTM. Volume 3290 of LNCS. (2004) 121–138
11. Ouyang, C., Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M., Mendling, J.: From business process models to process-oriented software systems. ACM Trans. Softw. Eng. Methodol. **19** (2009)
12. Kiepuszewski, B., ter Hofstede, A.H.M., van der Aalst, W.M.P.: Fundamentals of Control Flow in Workflows. Acta Inf. **39** (2003) 143–209
13. Vanhatalo, J., Völzer, H., Koehler, J.: The Refined Process Structure Tree. Data & Knowledge Engineering **68** (2009) 793–818
14. Polyvyanyy, A., Vanhatalo, J., Völzer, H.: Simplified computation and generalization of the refined process structure tree. Technical Report RZ 3745, IBM (2009)
15. van der Aalst, W.M.P.: Verification of Workflow Nets. In: Application and Theory of Petri Nets. Volume 1248 of LNCS. (1997) 407–426
16. van der Aalst, W.M.P.: Workflow verification: Finding control-flow errors using petri-net-based techniques. In: BPM. Volume 1806 of LNCS. (2000) 161–183
17. van Glabbeek, R.J.: The Linear Time-Branching Time Spectrum (Extended Abstract). In: CONCUR. Volume 458 of LNCS. (1990) 278–297
18. Best, E., Devillers, R.R., Kiehn, A., Pomello, L.: Concurrent bisimulations in petri nets. Acta Inf. **28** (1991) 231–264
19. McConnell, R.M., de Montgolfier, F.: Linear-time modular decomposition of directed graphs. Discrete Applied Mathematics **145** (2005) 198–209
20. Esparza, J., Römer, S., Vogler, W.: An Improvement of McMillan's Unfolding Algorithm. FMSD **20** (2002) 285–310
21. Kondratyev, A., Kishinevsky, M., Taubin, A., Ten, S.: Analysis of Petri Nets by Ordering Relations in Reduced Unfoldings. FMSD **12** (1998) 5–38