Behaviour Equivalence and Compatibility of Business Process Models with Complex Correspondences

Matthias Weidlich[1], Remco Dijkman[2] and Mathias Weske[3]

[1] *Technion – Israel Institute of Technology, Technion City, Haifa 32000, Israel*
[2] *Eindhoven University of Technology*
Den Dolech 2, 5612 AZ Eindhoven, The Netherlands
[3] *Hasso Plattner Institute, University of Potsdam*
Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam, Germany
Email: weidlich@tx.technion.ac.il, r.m.dijkman@tue.nl, weske@hpi.uni-potsdam.de

Once multiple models of a business process are created for different purposes or to capture different variants, verification of behaviour equivalence or compatibility is needed. Equivalence verification ensures that two business process models specify the same behaviour. Since different process models are likely to differ with respect to their assumed level of abstraction and the actions that they take into account, equivalence notions have to cope with correspondences between sets of actions and actions that exist in one process but not in the other. In this paper, we present notions of equivalence and compatibility that can handle these problems. In essence, we present a notion of equivalence that works on correspondences between sets of actions rather than single actions. We then integrate our equivalence notion with work on behaviour inheritance that copes with actions that exist in one process but not in the other, leading to notions of behaviour compatibility. Compatibility notions verify that two models have the same behaviour with respect to the actions that they have in common. As such, our contribution is a collection of behaviour equivalence and compatibility notions that are applicable in more general settings than existing ones.

Keywords: Behaviour Equivalence, Behaviour Compatibility, Model Verification, Behavioural Models

Received 00 January 2009; revised 00 Month 2009

1. INTRODUCTION

Business process management aims at supporting the operations of large organisations using explicit representations of business processes, i.e., process models [1]. Those models define activities, or actions, the essential steps that need to be conducted, along with constraints on their logical order of execution. As every conceptual model, a process model provides an abstraction that is created for a purpose. The purpose of process modelling determines to which extent the operations shall be captured and how they are abstracted to the model. Different modelling purposes, therefore, may lead to different process models that capture the same business operations. Also, process models are closely related if they do not capture different views on a common business process, but slight variations of a business process. In both cases, the process models may differ with respect to the considered set of activities and their logical ordering; a different granularity of activities may be used and activities present in one model may be absent or differently ordered in another model.

Behavioural equivalence or compatibility analysis of process models reveals whether different models specify the same behaviour. Behaviour compatibility is a weak form of behaviour equivalence. To be compatible, two behaviours only have to be equivalent with respect to the activities that they have in common. There are many drivers for equivalence and compatibility analysis. First and foremost, process models are means for communication. As such, the need to have representations of business operations that are equivalent or at least compatible is inherent; a shared understanding of operations cannot be achieved if different process models specify contradictory

behaviour. Further, the validation of a requirements specification with an according workflow implementation, both captured by process models, requires means for compatibility analysis. Since business processes are undergoing changes frequently, process models tend to drift apart. Techniques to realise change management among process models, therefore, also rely on equivalence or compatibility criteria.

As a preliminary step, equivalence and compatibility analysis of process models requires that correspondences between activities are identified, i.e., between activities that are considered to be equivalent. Since the investigated process models serve different modelling purposes or capture different variations of a business process, it is likely that complex correspondences are identified in this step. In contrast to elementary 1:1 correspondences, complex correspondences relate sets of activities to each other. They stem from differences in the assumed abstraction level.

In this paper, we focus on the challenge imposed by such complex correspondences, when doing equivalence or compatibility analysis. The presence of complex correspondences precludes the direct application of behaviour equivalences as defined in the linear time – branching time spectrum [2, 3], because these equivalence notions are defined for elementary (i.e., 1:1) correspondences only. In this paper, we present notions that lift these equivalences to complex correspondences. These notions are independent of the chosen equivalence criterion. For illustration, we utilise two concrete equivalences as examples: concrete trace equivalence and branching bisimulation. In essence, we assume complex correspondences to induce a relation between *regions* of activities between two process models. The approach relies on the aggregated behaviour for these regions and abstracts from their internal behaviour. This allows for assessing whether the start and end of regions in one model coincide with the start and end of the corresponding regions in another model. In our previous work, we followed a similar approach using a partitioning of traces [4]. Despite a similar underlying idea of how to treat complex correspondences, however, the notions presented in this paper take a different approach by leveraging the notions of a start and an end of a region. As such, they allow for taking the branching structure into account, which our previous work did not.

After lifting the notion of behaviour equivalence to complex correspondences, we also integrate existing work on behaviour inheritance [5, 6]. Notions of behaviour inheritance provide a means to cope with activities in one model that are without counterpart in another model. We combine these notions with the aforementioned notions of equivalence to arrive at compatibility notions that are applicable in a more general setting than the existing ones.

The remainder of this paper is structured as follows. Section 2 discusses the background of our work. We illustrate the challenges of equivalence and compatibility analysis with an example and review the essentials of behaviour inheritance. Section 3 defines the formal framework used throughout this paper. In Section 4, we show how the behaviour of a single region of a process model is aggregated. Following this line, Section 5 lifts the idea of behaviour aggregation from single regions to multiple regions. Section 6 shows how the aggregated behaviour of regions is used to judge on equivalence or compatibility of aligned process models. Section 7 shows how the compatibility notions can be applied to an example from practice. A review of related work and concluding remarks complete this paper.

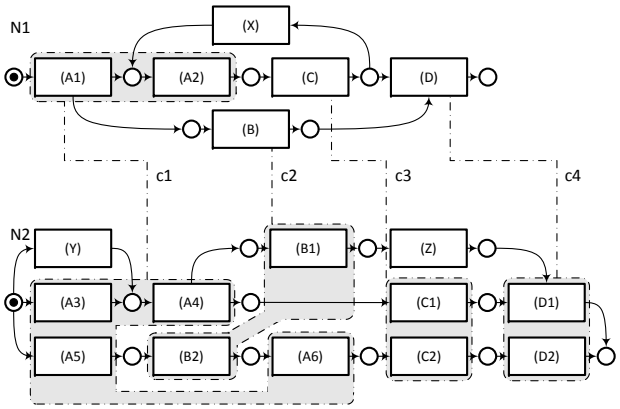

FIGURE 1: Net systems that are aligned by correspondences.

2. BACKGROUND

To illustrate the problem addressed by this paper, this section first introduces an example. Then, we elaborate on related work on behaviour inheritance. It copes with particular aspects of compatibility of process models that are aligned by correspondences.

2.1. An Example

Process models capture the behaviour of a business process by defining a set of activities, or actions, and a logical order between them [1]. Figure 1 depicts two process models as Petri net systems, see [7]. Here, activities are represented by net transitions. The states and state transitions of the net system define the behavioural dependencies for the execution of activities.

The two net systems depicted in Figure 1 are assumed to be closely related in terms of their semantics. They may describe the same business process or very similar variations of a common business process. The semantic closeness of the models is manifested in correspondences between the activities, i.e., the transitions of the Petri net systems. Those correspondences relate activities (or groups thereof) to each other that are considered to be equivalent. We discuss techniques for identifying such correspondences when reviewing related work. Correspondences may be elementary or complex. The

former relate pairs of single activities to each other. Complex correspondences are defined between groups of activities. Here, 1:n correspondences stem from activities in one model that are refined or collapsed in the other model. Further, n:m correspondences represent a relation between sets of activities, for which there are no correspondences between one of their activity subsets. Such a correspondence may stem from a different modularisation of the functionality captured by the activities. In Figure 1, there are four correspondences, all of them are complex. For instance, correspondence $c1$ encodes that the transitions $\{A1, A2\}$ in net system $N1$ *correspond* to the transitions $\{A3, A4, A5, A6\}$ in net system $N2$.

Note that correspondences are defined on the model structure. The behavioural semantics of the correspondences, therefore, are not defined explicitly, but are induced by the correspondences. Consider again correspondence $c1$ of the example. The correspondence defines that the transition sets $\{A1, A2\}$ and $\{A3, A4, A5, A6\}$ jointly refer to the same units of work of the business process. The concrete behavioural relation between both sets, i.e., the behavioural semantics of the correspondence $c1$, thus follows from the behaviour described by the net systems. The behavioural semantics in this case is that an occurrence of transition $A1$ that is followed by one more occurrences of transition $A2$ in net system $N1$ is considered to be equivalent to the potential occurrence of transition $A3$ followed by an occurrence of transition $A4$, *or* the occurrence of both transitions, $A5$ and $A6$, in net system $N2$.

Given a set of correspondences between process models, we are interested in knowing whether the models are compatible in terms of the behavioural semantics of their correspondences. According to Zelewski, behaviour compatibility, or behaviour consistency, of process models refers to a *freedom of contradictions* [8]. One may interpret freedom of contradictions such that process models are compatible if their behaviour is equivalent, modulo activities that have been added, removed, or refined. Earlier, we mentioned various drivers for compatibility analysis. Those suggest, that different notions of behaviour equivalence may be suited to be the basis for compatibility verification. Following van Glabbeek [9], *'two system descriptions are considered equivalent only if the described behaviours share the properties that are essential in the context in which the system will be embedded. It depends on this context and on the interests of a particular user which properties are essential.'*

Behaviour equivalences can be applied directly to judge behaviour compatibility only if the correspondences between two process models induce a bijection over their activities. We argue that this is rarely the case for process models that capture different perspectives on a business process or different variations of a business process. Hence, the application of behaviour equivalences to judge behaviour compatibility has to address the two following questions.

(1) How to cope with activities that are not aligned by any correspondence?

(2) How to cope with complex correspondences?

The first question has been addressed by work on *behaviour inheritance*, of which an overview is presented in the next section. The second question has not been addressed. Although this question is closely related to refinements of behavioural models, we are not aware of any approach that adapts common behaviour equivalence for the setting of complex correspondences. In this paper, we propose a way to lift behaviour equivalences to complex correspondences.

2.2. Behaviour Inheritance

The concept of inheritance is well-known for models that capture static structures. As an example, consider the subclass – superclass relation as defined within the Unified Modeling Language (UML). Here, the notion of inheritance clarifies how the subclass *extends* the superclass by adding class members. Conceptually, behaviour inheritance tries to adapt this concept to behavioural models. As such, it defines how to cope with extensions of behavioural models when assessing behaviour equivalence.

Behaviour inheritance is grounded on two elementary operations, i.e., *hiding* and *blocking* of activities, or actions [5, 6]. Both operations may be applied to all activities that extend one model with respect to another model. Blocking means that the activity is removed before deciding equivalence. All state transitions related to this activity are no longer part of the behavioural models that are compared. Hiding means that an activity cannot be observed. All state transitions related to this activity are invisible when deciding a certain notion of behaviour equivalence. Apparently, the notion of behaviour equivalence is assumed to distinguish visible and invisible transitions. To this end, work on behaviour equivalence mainly relies on branching bisimulation. However, the concepts can directly be ported to other behaviour equivalences.

Hiding and blocking may be combined for different transitions. Following Basten and van der Aalst [5, 6], *protocol inheritance* holds if equivalence is satisfied by blocking activities that relate to model extensions, whereas *projection inheritance* holds if equivalence is satisfied by hiding activities. Behaviour equivalence may be achieved in both cases, by only blocking certain activities and by only hiding activities (called *protocol / projection inheritance*). However, it may be required to combine both operations to satisfy behaviour equivalence. That is, two models are equivalent if one set of activities is blocked and another set of activities is hidden (called *life-cycle inheritance*).

For the use case of deciding compatibility of aligned process models, hiding and blocking may be applied to activities that are not aligned by any correspondence. Consider the example net systems in Figure 1. Transition (X) in system $N1$ is not aligned and may be blocked or hidden when comparing the behaviour of both net systems. In system $N2$, we observe two transitions that are without counterpart in the other system, i.e., transitions (Y) and (Z). Again, the transitions may be blocked or hidden. Note that blocking transition (Z), however, implies that transition $(D1)$ cannot be fired as well.

We consider compatibility to be a symmetric concept. In contrast to behaviour inheritance, therefore, we may decide to block or hide activities in both models that are compared.

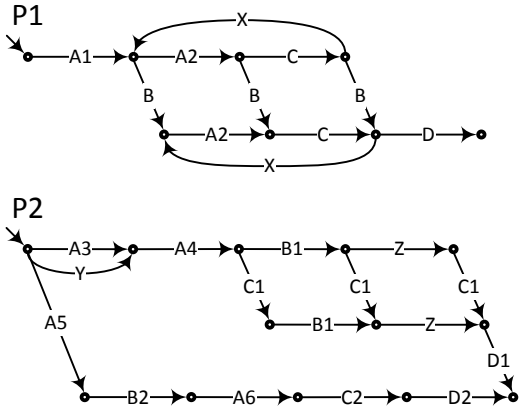

FIGURE 2: Two process graphs, which represent the behaviour of the net systems depicted in Figure 1.

3. FORMAL PRELIMINARIES

This section clarifies notions and notations used in the remainder of this paper. Section 3.1 defines process graphs as the formal grounding of our investigations. Section 3.2 defines abstraction and encapsulation for process graphs. We elaborate on behaviour equivalences in Section 3.3. Section 3.4 introduces the concept of an alignment for process graphs.

3.1. Process Graphs

A process graph is a generic behavioural model that allows for defining behaviour by means of states and state transitions, i.e., it is a transition system. Transitions may be labelled with symbols indicating the actions that are performed to realise a state change. For virtually all notations used for modelling business processes – Petri net systems as illustrated earlier or high-level notations such as the Business Model and Notation (BPMN) [10] or Event-Driven Process Chains (EPCs) [11] – execution semantics are defined in terms of states and state transitions. As such, process graphs enable us to discuss behaviour compatibility in the most general case.

We recall basic notions and notations based on [12]. Given an alphabet Σ, we denote all finite sequences over Σ by Σ^* and all infinite sequences over Σ by Σ^∞. We write $\Sigma^\omega = \Sigma^* \cup \Sigma^\infty$. The empty sequence is denoted by ϵ.

Definition 3.1 (Process Graph). *A process graph is a connected, rooted, edge-labelled and directed graph.*

We call the set of nodes that have no outgoing edges the final nodes, such that these represent the completion of a business process. We postulate that $\mathcal{G}$ is the universal set of process graphs and that $\mathcal{A}$ is the universal set of labels, called actions. Further, we assume that there is a dedicated *silent* action, $\tau \notin \mathcal{A}$. For a set of actions $A \subseteq \mathcal{A}$, we write $A_\tau = A \cup \{\tau\}$. Edges are labelled with elements from a set of actions $A_\tau \subseteq \mathcal{A} \cup \{\tau\}$. We also call a node of the process graph a *state* and an edge a *transition* or a *step*. We denote the set of all states of a process graph g as S_g, the set of final states as F_g, the root node as r_g, the set of all actions as A_g and the set of all transitions as $\rightarrow_g$. We omit the subscripts if the respective process graph is clear from the context. Given two states $s_1, s_2 \in S$ of a process graph, we denote a transition from state s_1 to state s_2 that is labelled with action a as $s_1 \xrightarrow{a} s_2$. In this paper, we assume that the root has no incoming edges and that it is the only node in the process graph that does not have any incoming edges. Also, we assume any process graph to show only finitely branching behaviour, i.e., the number of transitions originating from a single state is finite.

We lift transitions to sequences of transitions. We define the relation $\stackrel{\sigma}{\Longrightarrow} \subseteq S \times S$ with $\sigma \in A_\tau^*$ as the smallest relation, such that for all $s_1, s_2, s_3 \in S$ and $a \in A_\tau$ it holds that (1) $s_1 \stackrel{\epsilon}{\Longrightarrow} s_1$, (2) $s_1 \xrightarrow{a} s_2$ with $a \in A_\tau$ implies $s_1 \stackrel{a}{\Longrightarrow} s_2$ with $a \in A_\tau^*$, and (3) $s_1 \stackrel{\sigma}{\Longrightarrow} s_2 \stackrel{\rho}{\Longrightarrow} s_3$ implies $s_1 \stackrel{\sigma}{\Longrightarrow} s_2 \stackrel{\rho}{\Longrightarrow} s_3$ with $\sigma\rho$ as the concatenation of $\sigma \in A_\tau^*$ and $\rho \in A_\tau^\omega$. A state $s_2 \in S$ is reachable from a state $s_1 \in S$, if $s_1 \stackrel{\sigma}{\Longrightarrow} s_2$ for some $\sigma \in A_\tau^*$. The set of states that is reachable from a state $s \in S$ is denoted by s^*. $s \stackrel{\sigma}{\Longrightarrow}$ denotes that there exists some state s', such that $s \stackrel{\sigma}{\Longrightarrow} s'$. Further, if $s_1 \stackrel{\sigma}{\Longrightarrow} s_2$ and $\sigma \in \{\tau\}^*$ then we write $s_1 \Longrightarrow s_2$ to denote state transitions by silent steps.

For this work, we have further assumptions on process graphs. We require process graphs to be free of dead transitions. That is, for each transition $s_1 \xrightarrow{a} s_2$ there exists a transition sequence $r \stackrel{\sigma}{\Longrightarrow} s_1$. In addition, we assume that there are no livelocks, i.e., for each transition $s_1 \xrightarrow{a} s_2$ there exists a final state $f \in F$ and a transition sequence $s_2 \stackrel{\sigma}{\Longrightarrow} f$.

For illustration purposes, consider Figure 2. The process graphs represent the behaviour (aka the state spaces) of the net systems shown in Figure 1. Both graphs are connected, rooted (the root state is highlighted with a small unlabelled arrow), edge-labelled, and directed. The edge labels refer to the transitions of the Petri net systems and indicate the state transitions.

Note that we depict silent steps (neither process graph in the example shows a silent step) by unlabelled arrows between states.

Finally, we define finite and infinite traces of a process graph as sequences of visible actions. A sequence of actions $\sigma = a_1, \ldots, a_n \in A^*$, $n \in \mathbb{N}_0$, is a *finite concrete trace* of a process graph, iff there exist states $s_0, \ldots, s_{2n} \in S$, such that $s_0 = r$ is the root of the process graph and for all $i \in \mathbb{N}$, $1 \leq i \leq n$ it holds $s_{2i-2} \Longrightarrow s_{2i-1} \xrightarrow{a_i} s_{2i}$. A sequence of actions $\sigma = a_1, a_2, \ldots \in A^\infty$, is an *infinite concrete trace* of a process graph, iff there exist states $s_0, s_1, s_2, s_3, s_4 \ldots \in S$, such that $s_0 = r$ is the root of the process graph and for all $s_0 \Longrightarrow s_1 \xrightarrow{a_1} s_3 \Longrightarrow s_4 \xrightarrow{a_2} \ldots$. For a process graph g, we denote the set of all finite concrete traces by $\mathcal{T}_g$ and the set of all infinite concrete by $\mathcal{IT}_g$.

3.2. Abstraction & Encapsulation

In Section 2.2, we discussed that notions of behaviour inheritance either hide or block actions when comparing behaviour. For process graphs, hiding of actions is realised by abstraction, blocking of actions is realised by encapsulation. We recall both notions based on [6].

Definition 3.2 (Abstraction). *Let g be a process graph. For a set of actions $X \subseteq A_g$, the* abstraction operator $\tau_X : \mathcal{G} \mapsto \mathcal{G}$ *rewrites g into the process graph h by renaming all actions in X with the silent action τ, i.e., $S_h = S_g$; $s_1 \xrightarrow{a} s_2 \in \rightarrow_h$, iff $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$ and $a \notin X$; and $s_1 \xrightarrow{\tau} s_2 \in \rightarrow_h$, iff $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$ and $a \in X$.*

Abstraction rewrites a process graph by changing the action labelling. Encapsulation, in turn, removes transitions with certain action labels. Since we require a process graph to be rooted and connected, we also remove the states that may only be reached by using at least one transition with a label that shall be be encapsulated.

Definition 3.3 (Encapsulation). *Let g be a process graph. For a set of actions $X \subseteq A_g$, the* encapsulation operator $\delta_X : \mathcal{G} \mapsto \mathcal{G}$ *rewrites g into the process graph h by removing all transitions with actions in X, i.e., S_h contains all states $s \in S_g$ for which there exists a sequence $\sigma \in A_g^*$ with $r_g \xRightarrow{\sigma} s$ and σ does not contain any action in X; and $s_1 \xrightarrow{a} s_2 \in \rightarrow_h$, iff $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$, $s_1, s_2 \in S_h$, and $a \notin X$.*

3.3. Behaviour Equivalences

To compare the behaviour defined by labelled transition systems, different notions of behaviour equivalence may be applied. Behaviour equivalences have been classified in the seminal work of van Glabbeek [2, 3], yielding the linear time – branching time spectrum for sequential processes. For sequential processes with silent steps (process graphs, respectively), this spectrum is spanned by the notions of *trace equivalence* and *branching bisimulation*. The former is due to Hoare [13] and relates to the equivalence of the observable sequences of actions. As such, it can be seen as the lower bound of the spectrum of behaviour equivalences. Branching bisimulation as defined by van Glabbeek and Weijland [12], in turn, considers not only the observable behaviour, but takes the moment of choice into account. It can be seen as the upper bound of the aforementioned spectrum. Besides these two poles, various equivalences have been proposed and advocated to be applied in the field of business process analysis, see [14]. We focus on the two mentioned notions to exemplify our approach for treating complex correspondences between process models. However, our ideas may also be lifted to other behaviour equivalences.

As mentioned earlier, alignments of process models can be assumed to be partial. Hence, the models that are compared show extensions that may be hidden or blocked according to the notions of behaviour inheritance, cf., Section 2.2. Since hiding of actions is realised by abstraction, we neglect silent steps when comparing traces of process graphs. To highlight that we consider only traces built of concrete actions, we refer to this equivalence as concrete trace equivalence.

Definition 3.4 (Concrete Trace Equivalence). *Two process graphs g and h are* concrete trace equivalent*, denoted by $g \doteq h$, iff $\mathcal{T}(g) = \mathcal{T}(h)$ and $\mathcal{IT}(g) = \mathcal{IT}(h)$.*

Concrete trace equivalence is indeed an equivalence, since reflexivity, transitivity, and symmetry follow directly from the set equivalence. However, this equivalence neglects the branching of a process and considers only the observable sequences of actions. The moment of choice is taken into account by the notion of branching bisimilarity.

Definition 3.5 (Branching Bisimilarity). *Two process graphs g and h are* branching bisimilar*, denoted by $g \sim h$, iff there exists a symmetric relation $\mathcal{R}$ between the nodes of g and h, the branching bisimulation, such that*

1. *the root nodes of the process graphs are related, $r_g \mathcal{R} r_h$;*
2. *if $x \mathcal{R} y$ and $x \xrightarrow{a} x'$, then either $a = \tau$ and $x' \mathcal{R} y$, or there exist states u, v, such that $y \Longrightarrow u \xrightarrow{a} v \Longrightarrow y'$, $x \mathcal{R} u$, $x' \mathcal{R} v$, and $x' \mathcal{R} y'$.*

According to [12], branching bisimilarity is an equivalence relation.

3.4. Alignments of Process Graphs

To capture alignments between process models in the most general setting, we define the concept of a correspondence relation between actions of two process graphs. It relates corresponding actions of both graphs to each other.

Definition 3.6 (Correspondence Relation). *Let g and h be process graphs. A* correspondence relation

$\simeq \subseteq (A_g \setminus A_h) \times (A_h \setminus A_g)$ *associates corresponding actions of both process graphs to each other. Let* $A_1, A_2 \subseteq \mathcal{A}$ *be two sets of actions such that* $A_1 \times A_2 \subseteq \simeq$. *Let* A_1 *and* A_2 *be maximal with respect to set inclusion. Then,* $c = (A_1, A_2)$ *is referred to as a* correspondence *and we also write* $A_1 \simeq A_2$. *With a slight abuse of notation, we also write* $a_1 \simeq a_2$ *iff there exists* $A_1 \simeq A_2$ *and* $a_1 \in A_1$ *and* $a_2 \in A_2$.

Given two process graphs g and h that are aligned by a correspondence relation $\simeq \subseteq (A_g \setminus A_h) \times (A_h \setminus A_g)$, we use a shorthand notation for all actions of the graphs that are part of some correspondence, $A_g^{\simeq} = \{a \in A_g \mid \exists\ b \in A_h : a \simeq b\}$ and $A_h^{\simeq} = \{a \in A_h \mid \exists\ b \in A_g : b \simeq a\}$.

A correspondence between two sets of actions means that the Cartesian product of the sets is subsumed by the correspondence relation. A correspondence $c = (A_1, A_2)$ is called *elementary*, iff $|A_1| = |A_2| = 1$, and *complex* otherwise. Two correspondences $c_1 = (A_1, A_2)$ and $c_2 = (A_1', A_2')$ are *overlapping*, iff $A_1 \cap A_1' \neq \emptyset$ or $A_2 \cap A_2' \neq \emptyset$. A correspondence relation is *overlapping*, iff it induces at least two overlapping correspondences. Otherwise, it is *non-overlapping*.

In the remainder of this paper, we require all correspondence relations to be non-overlapping. For actions that are part of an overlap of two correspondences in one behavioural model, there is no distinct assignment to either correspondence. This may be the case if the functionality encoded in actions is distributed differently among the actions of two process graphs. Then, a 'part' of an action is related to one correspondence, whereas another 'part' of the same action relates to a second correspondences. In this case, however, clarification of the relation of the two correspondences is needed prior to deciding behavioural compatibility. This may be done by splitting up the action that is part of the overlap, so that the derived actions relate only to one of the correspondences and, therefore, unambiguously characterise the relation between them. In practise, most correspondences are non-overlapping. For the alignments investigated in our previous work [22], there are 520 elementary correspondences between 20 pairs of process models. More than 40% of them are part of complex correspondences, whereas less than 7% relate to overlapping correspondences. The correspondence relation discussed for our example in Figure 1 is non-overlapping.

4. AGGREGATE BEHAVIOUR BASED ON A SINGLE REGION

The main idea behind determining compatibility of complex correspondences is to consider each correspondence as a relation between two *regions*, which occur as a whole, and to determine the compatibility of the *aggregate behaviour* that is determined by those regions. In other words, actions that are part of a complex correspondence are not considered independent of each other. Instead, we assess behaviour compatibility by investigating regions that are formed by sets of actions. In our setting, these sets of actions are induced by complex correspondences. However, the presented approach is generic in the sense that it is grounded on regions, i.e., sets of actions. Hence, we first discuss the aggregation of behaviour based on regions without referring to the notions of an alignment and a correspondence. Once we clarified how the aggregate behaviour of a process graph is defined in terms of the occurrence of regions as a whole, we apply the aggregation to regions induced by correspondences.

As a first step, this section explains how we can determine the aggregate behaviour of a single region of a process graph. We start by introducing the notion of aggregate behaviour, explaining how aggregate behaviour can be determined, and discussing what design decisions are involved. Then, we define the two essential notions for determining aggregate behaviour: starting a region and ending a region.

4.1. Overview

The basis for determining the aggregate behaviour is to consider each region as a single block of activity that takes time. At the start of a process we consider a region inactive. It can be started during the execution of a process and then becomes active. Once it is started, it eventually must end and become inactive again. Therefore, to determine the aggregate behaviour, we need to determine, given a behaviour and the regions within that behaviour, when each region starts and ends. To determine this, two important design decisions must be made:

(1) Do we consider intermediate 'pauses' of regions; i.e.: do we allow regions to end and start again?

(2) When does a region end, if the decision to end a region occurs after other activities (not from the region) have occurred?

Pausing a region is a logical option, in case it is interleaved with activities from another region. For example, consider the process graph in Figure 3 and two regions $\{A1, A2\}$ and $\{B\}$. The region consisting of $A1$ and $A2$ is interleaved with the action B. We can now choose either to pause this region after $A1$ has occurred and start it again when $A2$ occurs; or to leave it active in parallel with the region that contains B. We choose to leave regions active, even if they are interleaved with transitions from other regions. We motivate this from a practical perspective: in most cases a region will be considered as a single unit of work that is started and ended and will remain active, even if it is temporarily interrupted by other work. For example, if an employee checks the eligibility of a client for a loan, the employee may be interrupted by having to enter the client's personal data into an ERP system.

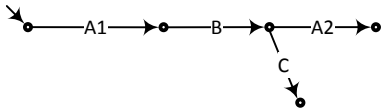

FIGURE 3: Interleaving of regions $\{A1, A2\}$ and $\{B\}$ and uncertainty of the moment of ending region $\{A1, A2\}$.

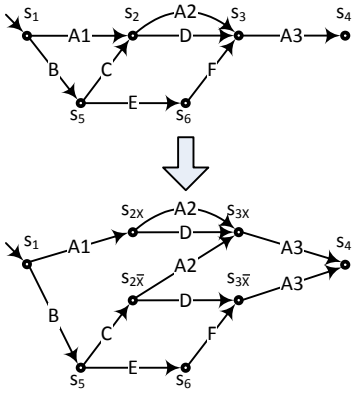

FIGURE 4: Rewrite 'may be started' states for the region $\{A1, A2, A3\}$.

However, conceptually, the employee will not consider the eligibility check as 'ended', because he will have to get back to that later. There is an exception to that in case an activity (such as checking the eligibility for a loan) is performed, after which there is an activity to verify whether the check is performed correctly and, depending on the outcome, the eligibility check must (partly) be performed again. In that case, the employee will consider the eligibility check to be ended after the first check and re-started for the second check. However, without additional information in the process model, it is not possible to distinguish this case. Therefore, we choose to let regions only start and end once in all cases.

It is possible that a region has only ended with certainty, after activities from another region have already occurred. Figure 3 illustrates this. After $A1$ has occurred, it is unclear whether transition $A2$ will still occur or not. Therefore, it is not possible to say with certainty whether or not the region is still active after $A1$. When C occurs, we can say, with hindsight that the region should have been ended after $A1$ occurred. However, then it is too late, because B has already occurred, which did not belong to the region. In deciding how to handle this, we choose to postpone the decision to end a region until the moment at which we are certain that the region has ended. Other possible ways of solving this are by changing the branching time or by 'pausing' the region as explained in the previous paragraph. We can change the branching time by explicitly choosing to end a region or not, at the moment at which this choice is relevant and continue processing depending on this decision. For example, in Figure 3 we can branch along two transition sequences, one that takes us through B and A and one that takes us through B and C. Along the first sequence, the region remains active and along the second sequence the region ended. However, we explicitly aim to preserve branching time in this work and, therefore, do not choose this option. We can also 'pause' the region by ending it after $A1$ and starting it again, should $A2$ occur. However, we explicitly decided not to allow regions to 'pause'. Therefore, we choose to end a region at the moment at which we are certain no more of its activities will occur.

These design choices lead to the following definitions for starting and ending a region.

4.2. Starting a Region

A region starts the first time that an action from that region occurs. We define the notion of starting a region on a process graph. However, for a transition in a process graph, it may not be certain that it represents the first occurrence of an action from a region. This is the case when the transition can be reached both via transition sequences that contain another occurrence of an action from the same region and via sequences that do not. For such a transition, we say that it *may* start the region, because it starts the region if that has not been done earlier in the course of reaching the transition.

DEFINITION 4.1 (Start Region, Started Region). *Let g be a process graph and let $X \subseteq \mathcal{A}$ be a region.*

- *The region X is* started *in $s \in r_g^*$, iff each transition sequence $r_g \stackrel{\sigma}{\Longrightarrow} s$ contains an action $x \in X$ in σ.*
- *The region X is* not started *in $s \in r_g^*$, iff no transition sequence $r_g \stackrel{\sigma}{\Longrightarrow} s$ contains an action $x \in X$ in σ.*
- *The region X* may be started *(or may not be started) in $s \in r_g^*$, iff at least one transition sequence $r_g \stackrel{\sigma}{\Longrightarrow} s$ contains an action $x \in X$ in σ and at least one transition sequence contains no action $x \in X$ in σ.*

Let $s_1 \stackrel{a}{\rightarrow} s_2$ be a transition with $a \in X$ and $s_1, s_2 \in r_g^*$. The transition starts the region, iff the region is not started in s_1 and started in s_2. The transition does not start the region, iff the region is not started in s_2 or started in s_1.

Figure 4 illustrates these situations. In s_{2X}, s_{3X} and s_4 the region $\{A1, A2, A3\}$ was started. In $s_1, s_5, s_6, s_{2\overline{X}}$ and $s_{3\overline{X}}$ the region is not started and in s_2 and s_3 the region may, or may not, be started.

The example in Figure 4 also shows that we cannot always identify the transitions that start the activity of a region. It is unclear whether the transition $s_3 \stackrel{A3}{\longrightarrow} s_4$ starts the region $\{A1, A2, A3\}$ or not. Intuitively, this can be seen, because the region may have been started

before, for example by the transition $s_2 \xrightarrow{A2} s_3$, or may not have been started before, for example, in case s_3 was reached via $s_6 \xrightarrow{F} s_3$. However, if we want to determine equivalence between aggregate processes that are defined in terms of (starting and ending) regions, then we must be able to uniquely identify the transitions that start the activity of a region. Therefore, we introduce a branching time preserving rewrite rule, which rewrites 'may be started' states in such a way that we can always identify whether a transition starts a region or not.

The rule rewrites each state s in which it is not clear whether a region is started or not into a state in which the region is started and a state in which the region is not started. As a convention, we will identify each of the states that were rewritten from a state s as s_X and $s_{\overline{X}}$. Subsequently, each transition that originates from or targets the state s needs to be rewritten. If it starts the region, it must point to a state in which the region is started. If it does not start the region, it needs to leave the region started if it was started and not started if it was not started. Figure 5 shows the different possibilities for rewriting a transition. Each case on the left side of the table, must be rewritten into the case(s) on the right. In the figure, $X?s$ means that in the state s the region X may or may not be started. The figure shows that if the transition comes from s_1, in which it is clear whether X is started or not, and points to s_2, in which it is clear whether X is started or not, then the transition is simply rewritten into an identical transition. The figure also shows that if the transition comes from s_1, in which it is clear whether X is started or not, and points to s_2, in which X may be started, and the transition is itself from the region X, then the rewritten transition points to the state s_{2X}. In that state X is started.

Definition 4.2 (Normalisation for a Region). *Let g be a process graph and let $X \subseteq \mathcal{A}$ be a region. The* normalisation operator $\eta_X : \mathcal{G} \mapsto \mathcal{G}$ *rewrites g into the process graph h by defining S_h and $\rightarrow_h$ as follows.*

S_h is the smallest set that contains:

- *all $s \in S_g$ in which X is started;*
- *all $s \in S_g$ in which X is not started; and*
- *two new states for each $s \in S_g$ in which X may, or may not, be started.*

$\rightarrow_h$ is the smallest set that contains:

- *all $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$ for which X is not 'may be started' in s_1 or s_2;*
- *$s_1 \xrightarrow{a} s_{2X}$, iff $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$, X is not 'may be started' in s_1, X is 'may be started' in s_2 and $a \in X$;*
- *$s_1 \xrightarrow{a} s_{2\overline{X}}$, iff $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$, X is not started in s_1, X is 'may be started' in s_2 and $a \notin X$;*
- *$s_1 \xrightarrow{a} s_{2X}$, iff $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$, X is started in s_1, X is 'may be started' in s_2 and $a \notin X$;*
- *$s_{1X} \xrightarrow{a} s_{2X}$ and $s_{1\overline{X}} \xrightarrow{a} s_{2X}$, iff $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$, X is 'may be started' in s_1, X is 'may be started' in s_2 and $a \in X$;*

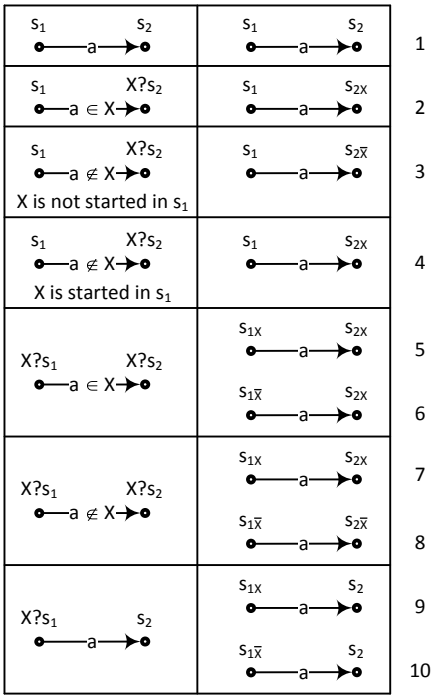

FIGURE 5: Rewrite transitions.

- *$s_{1X} \xrightarrow{a} s_{2X}$ and $s_{1\overline{X}} \xrightarrow{a} s_{2\overline{X}}$, iff $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$, X is 'may be started' in s_1, X is 'may be started' in s_2 and $a \notin X$; and*
- *$s_{1X} \xrightarrow{a} s_2$ and $s_{1\overline{X}} \xrightarrow{a} s_2$, iff $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$, X is 'may be started' in s_1 and X is not 'may be started' in s_2.*

A process graph j is normalised *for X, iff $j = \eta_X(j)$.*

Figure 4 illustrates the rewrite rule. In state s_2, it is unclear whether the region $\{A1, A2, A3\}$ is started or not. Consequently, we rewrite it into two states: s_{2X} in which the region is started and $s_{2\overline{X}}$ in which the region is not started. Similarly, s_3 is rewritten into s_{3X} and $s_{2\overline{X}}$.

What remains to be proven is that the process graph h that was constructed from g using the rewrite rule is branching bisimilar to g and does not contain any states in which region X may, or may not, be started.

Property 1. For a process graph h that is derived by normalising a process graph g for a region $X \subseteq \mathcal{A}$, $h = \eta_X(g)$, it holds that S_h does not contain any states in which X may, or may not, be started.

Proof. In any state that does not have any incoming transitions, X is not started. Therefore, for such states it holds that X is not 'may be started'.

Any state that does have incoming transitions is reachable from r_h via transitions that were generated through the rewrite rules from Definition 4.2. There are 10 possible cases (also see Figure 5). For each of these cases, we prove that X is not 'may be started' in the target states (s_2, s_{2X}, or $s_{2\overline{X}}$) of the transitions. If this holds for the target state, it must also hold for

the source states, because these can, in turn, only be reachable from r_h via the transitions.

A state denoted as s_{2X} is only reachable: via a transition that contains an $a \in X$ (case 2, 5 or 6), or from a state in which X is started and that by Definition 4.1 is only reachable via a transition sequence that contain an $a \in X$ (case 4, 7). Consequently, in a state denoted as s_{2X}, X must be started by Definition 4.1.

A state denoted as $s_{2\overline{X}}$ is only reachable via a transition $a \notin X$ from a state in which X is not started and that by Definition 4.1 is only reachable via a transition sequence that does not contain an $a \in X$ (case 3 and 8). Consequently, in a state denoted as $s_{2\overline{X}}$, X must be not be started by Definition 4.1.

A state s_2, for which it is clear in process g whether the region X is started or not, can only be reached via a state s_1 (case 1) or a 'may be started' state (case 9 and 10). If it is reached via a state s_1 (case 1), in which it is clear was clear in process g whether the region is started or not, we know that the transition takes the criteria into account under which we know whether X is started or not in s_2, because this transition is derived from g without modification. If it is reached via a 'may be started' state (case 9 and 10), the only way in which it can be known in s_2 whether X was started or not, was if $a \in X$, making X started in s_2. This is preserved by the rewrite rule.

Consequently, in each state that is reachable via transitions that were generated by the rewrite rules, it is clear whether X is started or not and X is not 'may be started'. □

Using this property, it immediately follows that any process graph that is normalised for a region does not contain any state in which the region may, or may not be started. Further, rewriting preserves branching bisimilarity.

Property 2. For a process graph h that is derived by normalising a process graph g for a region $X \subseteq \mathcal{A}$, $h = \eta_X(g)$, it holds that $h \sim g$.

Proof. h is branching bisimilar to g, because we can construct the branching bisimulation relation $\mathcal{R}$ that meets the two criteria of branching bisimulation (Definition 3.5). We construct this relation as follows. Clearly, g is branching bisimilar to itself (because of the reflexive property of branching bisimilarity). Let $\mathcal{Q}$ be the relation that makes g branching bisimilar to itself. We define $\mathcal{R}$ to be the smallest relation derived from $\mathcal{Q}$ that contains:

- (s_1, s_2) iff $(s_1, s_2) \in \mathcal{Q}$ and X is not 'may be started' in s_1 or s_2;
- (s_{1X}, s_2) and $(s_{1\overline{X}}, s_2)$ iff $(s_1, s_2) \in \mathcal{Q}$ and X is 'may be started' in s_1 and not 'may be started' in s_2;
- (s_1, s_{2X}) and $(s_1, s_{2\overline{X}})$ iff $(s_1, s_2) \in \mathcal{Q}$ and X is not 'may be started' in s_1 and 'may be started' in s_2;
- (s_{1X}, s_{2X}) and $(s_{1\overline{X}}, s_{2\overline{X}})$ iff $(s_1, s_2) \in \mathcal{Q}$ and X is 'may be started' in both s_1 and s_2;

We show that $\mathcal{R}$ satisfies the two criteria.

Criterion 1. For the root state r_g, it holds that $(r_g, r_g) \in \mathcal{Q}$. Also, since the root state has no incoming transitions, X is not started in r_g, such that r_g is also in h (where it is identified as r_h) and is not split up into multiple states, and $(r_g, r_h) \in \mathcal{R}$ by construction of $\mathcal{R}$ above.

Criterion 2. If $x\mathcal{R}y$ and $x \xrightarrow{a} x'$ with $x, x' \in S_g$, then $y \xrightarrow{a} y'$ is derived from $x \xrightarrow{a} x'$ according to Definition 4.2. Also, $x'\mathcal{R}y'$ by construction of $\mathcal{R}$ above. Consequently, there exist states $u, v \in S_h$, such that $y \Longrightarrow u \xrightarrow{a} v \Longrightarrow y'$, $x\mathcal{R}u$, $x'\mathcal{R}v$, and $x'\mathcal{R}y'$, for $y = u$ and $v = y'$.

If $x\mathcal{R}y$ and $x \xrightarrow{a} x'$ with $x, x' \in S_h$, then $y \xrightarrow{a} y'$ is the transition from which $x \xrightarrow{a} x'$ was according to Definition 4.2. Also, $x'\mathcal{R}y'$ by construction of $\mathcal{R}$ above. Consequently, there exist states $u, v \in S_g$, such that $y \Longrightarrow u \xrightarrow{a} v \Longrightarrow y'$, $x\mathcal{R}u$, $x'\mathcal{R}v$, and $x'\mathcal{R}y'$, for $y = u$ and $v = y'$. □

We can now define and prove the property that we need to analyse behaviour at the abstraction level of regions. This property states that a region is started at the first occurrence of a transition from that region.

Property 3. Let h be a process graph that is derived by normalising a process graph g for a region $X \subseteq \mathcal{A}$, $h = \eta_X(g)$. It holds that:

- if a transition $s_1 \xrightarrow{a} s_2$ starts the region X, it must not be preceded by a transition that starts the region on any transition sequence $r_g \xRightarrow{\sigma} s_1$.
- if a transition $s_1 \xrightarrow{a} s_2$ has $a \in X$ it must either start the region or be preceded on each transition sequence $r_g \xRightarrow{\sigma} s_1$ by a transition that starts the region.

Proof. We prove the first part of the property. If a transition $s_1 \xrightarrow{a} s_2$ starts the region X, according to Definition 4.1, the region X must not be started in s_1. Consequently, according to the definition, there must be no transition sequence $r_g \xRightarrow{\sigma} s$ that contains an action $x \in X$ in σ. Therefore $s_1 \xrightarrow{a} s_2$ cannot be preceded by a transition that starts the region, because such a transition must have an action $x \in X$.

We prove the second part of the property by contradiction. Suppose a transition $s_1 \xrightarrow{a} s_2$ has $a \in X$ and neither starts the region nor is preceded on each transition sequence $r_g \xRightarrow{\sigma} s_1$ by a transition that starts the region. Because g was derived by rewriting, it must be the case according to Property 1 that in each state the region must either be started or not started. Consequently, also in s_1 the region X must either be started or not started. According to Definition 4.1 that means that either each transition sequence $r_g \xRightarrow{\sigma} s_1$ contains an action $x \in X$ in σ, or no transition sequence $r_g \xRightarrow{\sigma} s_1$ contains an action $x \in X$ in σ. However, because of the supposition that the transition $s_1 \xrightarrow{a} s_2$

must not be preceded on each transition sequence $r_g \stackrel{\sigma}{\Longrightarrow} s_1$ by a transition that starts the region, it must be the case that no transition sequence $r_g \stackrel{\sigma}{\Longrightarrow} s_1$ contains an action $x \in X$ in σ. This means that the region X must not be started in s_1. Also, because of the supposition that the transition $s_1 \stackrel{a}{\rightarrow} s_2$ does not start the region and Definition 4.1, the region must either already be started in s_1 or not be started in s_2. Since we just concluded that the region must not be started in s_1, it must be the case that the region must not be started in s_2. According to Definition 4.1 that means that no transition sequence $r_g \stackrel{\sigma}{\Longrightarrow} s_2$ contains an action $x \in X$ in σ. This leads to a contradiction, because $s_1 \stackrel{a}{\rightarrow} s_2$ has $a \in X$, such that the sequences $r_g \stackrel{\sigma}{\Longrightarrow} s_2$ that contain this transition contain an action $x \in X$. This proves that, if a transition $s_1 \stackrel{a}{\rightarrow} s_2$ has $a \in X$, it must either start the region or be preceded on each transition sequence $r_g \stackrel{\sigma}{\Longrightarrow} s_1$ by a transition that starts the region. □

4.3. Ending a Region (with Possible Delay)

A region ends the last time that an action from that region occurs. Analogous to the definition of start transitions, we say that a transition may end a region, if that transition may or may not be the last transition of the region to occur. This situation is also explained in Section 4.1 and Figure 3.

Definition 4.3 (Ended Region). *Let g be a process graph and let $X \subseteq \mathcal{A}$ be a region.*

- *The region X has* ended *in $s \in r_g^*$, iff for no $f \in F$ a transition sequence $s \stackrel{\sigma}{\Longrightarrow} f$ contains an action $x \in X$ in σ.*
- *The region X has* not ended *in $s \in r_g^*$, iff for each $f \in F$ each sequence $s \stackrel{\sigma}{\Longrightarrow} f$ contains an action $x \in X$ in σ.*
- *The region X* may have ended *(or may not have ended) in $s \in r_g^*$, iff there exists at least one $f \in F$ for which there exists a sequence $s \stackrel{\sigma}{\Longrightarrow} f$ that contains an action $x \in X$ in σ and there exists at least one $f \in F$ for which there exists a sequence $s \stackrel{\sigma}{\Longrightarrow} f$ that does not contain an action $x \in X$ in σ.*

We would now like to define property that a region ends at the last occurrence of a transition from that region. However, as we explained in Section 4.1, we cannot with certainty say whether a transition ends a region or not. Moreover, unlike for start transitions, we cannot define a branching bisimilarity preserving rewrite rule, such that we can define this property. Therefore, for ending a region, we require a more relaxed property. This property states that a region ends with possible delay at the occurrence of the first transition after which we are sure that no transition from that region can occur anymore.

We define that a region ends, with possible delay, as follows.

Definition 4.4 (End Region with Possible Delay). *Let g be a process graph and let $X \subseteq \mathcal{A}$ be a region. Furthermore, let $s_1 \stackrel{a}{\rightarrow} s_2$ be a transition. The transition ends the region with possible delay, iff the region is not 'ended' or 'may have ended' in s_1 and 'ended' in s_2.*

Now we define and prove the property that a region ends with possible delay at the occurrence of the first transition after which we are sure that no transition from that region can occur anymore.

Property 4. Let h be a process graph that is derived by normalising a process graph g for a region $X \subseteq \mathcal{A}$, $h = \eta_X(g)$. It holds that:

- if a transition $s_1 \stackrel{a}{\rightarrow} s_2$ ends the region X with possible delay, it is not succeeded by a transition $s_1' \stackrel{b}{\rightarrow} s_2'$ that ends the region with possible delay, on a transition sequence $s_2 \stackrel{\sigma}{\Longrightarrow} f$ for any $f \in F_g$.
- a transition $s_1 \stackrel{a}{\rightarrow} s_2$, for which for no $f \in F$ a transition sequence $s_2 \stackrel{\rho}{\Longrightarrow} f$ contains an action $x \in X$ in ρ, must either end the region with possible delay or be preceded on each transition sequence $r_g \stackrel{\sigma}{\Longrightarrow} s_1$ by a transition that ends the region with possible delay.

Proof. We prove the first part of the property. If $s_1 \stackrel{a}{\rightarrow} s_2$ ends the region X with possible delay, the region must have ended in s_2, according to Definition 4.4. Consequently, according to Definition 4.3, for no $f \in F_g$ a transition sequence $s_2 \stackrel{\rho}{\Longrightarrow} f$ contains an action $x \in X$ in ρ. Thus, according to Definition 4.3, X must have ended in all states that succeed $s_1 \stackrel{a}{\rightarrow} s_2$ on a transition sequence $s_2 \stackrel{\rho}{\Longrightarrow} f$ for some $f \in F_g$. A transition on such a transition sequence can not end the region with possible delay, because that requires that the region is not ended. Therefore, $s_1 \stackrel{a}{\rightarrow} s_2$, is not succeeded by a transition $s_1' \stackrel{b}{\rightarrow} s_2'$ that ends the region with possible delay, on a transition sequence $s_2 \stackrel{\sigma}{\Longrightarrow} f$ for any $f \in F_g$.

We prove the part criterion of the property by contradiction. Suppose that a transition $s_1 \stackrel{a}{\rightarrow} s_2$, for which for no $f \in F$ a transition sequence $s \stackrel{\rho}{\Longrightarrow} f$ contains an action $x \in X$ in ρ, neither ends the region with possible delay nor is preceded on each transition sequence $r_g \stackrel{\sigma}{\Longrightarrow} s_1$ by a transition that ends the region with possible delay. Because of the supposition that for no $f \in F$ a transition sequence $s_2 \stackrel{\rho}{\Longrightarrow} f$ contains an action $x \in X$ in ρ and Definition 4.3, region X must have ended in s_2. Because of the supposition that the transition does not end the region and Definition 4.4, either the region is not 'may have ended' and not 'not ended' in s_1, or the region is not 'ended' in s_2. Since we just showed that the region must have ended in s_2, it must be so that the region is not 'may have ended' and not 'not ended' in s_1. By Definition 4.3 this means that not "there exists at least one $f \in F$ for which there exists a sequence $s_1 \stackrel{\rho}{\Longrightarrow} f$ that contains an action

$x \in X$ in ρ and there exists at least one $f \in F$ for which there exists a sequence $s_1 \stackrel{\rho}{\Longrightarrow} f$ that does not contain an action $x \in X$ in ρ" and not "for each $f \in F$ each sequence $s_1 \stackrel{\rho}{\Longrightarrow} f$ contains an action $x \in X$ in ρ". In purely logical notation this reads:
$\neg(\exists f \in F_g : \exists \rho \in A_g^* : s_1 \stackrel{\rho}{\Longrightarrow} f \wedge \exists x \in X : x \text{ in } \rho$
$\wedge$
$\exists f \in F_g : \exists \rho \in A_g^* : s_1 \stackrel{\rho}{\Longrightarrow} f \wedge \neg\exists x \in X : x \text{ in } \rho)$
$\wedge$
$\neg(\forall f \in F_g : \forall \rho \in A_g^* : s_1 \stackrel{\rho}{\Longrightarrow} f \wedge \exists x \in X : x \text{ in } \rho)$
We can rewrite this, using standard logic operations, into:
$\forall f \in F_g : \forall \rho \in A_g^* : s_1 \stackrel{\rho}{\Longrightarrow} f \wedge \neg\exists x \in X : x \text{ in } \rho$
According to Definition 4.3 this means that the region X has ended in s_1. Because of the assumptions that there are no dead tasks and no livelocks (see Section 3.1), each $x \in A_g$ must occur on a transition sequence from r_g to some $f \in F_g$. Consequently, each $x \in X$ must occur as well. With Definition 4.3 this means that in the root state r_g, the region X must either have 'not ended' or 'may have ended'. Consequently, each transition sequence $r_g \stackrel{\sigma}{\Longrightarrow} s_1$ starts with X 'not ended' or 'may have ended' and ends with X 'ended'. Consequently, for some transition $s_1' \stackrel{b}{\rightarrow} s_2'$ in each of these sequences: X is 'not ended' or 'may have ended' in s_1' and 'ended' in s_2'. Thus, according to Definition 4.4, some transition in each of these transition sequences ends the region X with possible delay. This is a contradiction with the supposition that the transition $s_1 \stackrel{a}{\rightarrow} s_2$ is not preceded on each transition sequence $r_g \stackrel{\sigma}{\Longrightarrow} s_1$ by a transition that ends the region with possible delay. This proves that a transition $s_1 \stackrel{a}{\rightarrow} s_2$, for which for no $f \in F$ a transition sequence $s_2 \stackrel{\rho}{\Longrightarrow} f$ contains an action $x \in X$ in ρ, must either end the region with possible delay or be preceded on each transition sequence $r_g \stackrel{\sigma}{\Longrightarrow} s_1$ by a transition that ends the region with possible delay. □

5. AGGREGATE BEHAVIOUR BASED ON MULTIPLE REGIONS

In the previous section, we elaborated on aggregating the behaviour of a process graph based on a single region. Verification of behaviour compatibility between process graphs in the presence of complex correspondences may require to consider multiple regions for each process graph. In this section, we turn the focus on a set of regions. We first lift the normalisation defined for a single region to sets of regions in Section 5.1. Then, Section 5.2 shows how we rewrite a process graph to separate the information of starting and ending for multiple regions. Finally, Section 5.3 discusses properties of this rewriting.

5.1. Normalisation based on Multiple Regions

Normalisation in terms of rewriting 'may be started' states has to be done for all regions. We lift the normalisation operator to a set of regions by combining the normalisations for single regions. We assume all regions to be distinct, i.e., for all regions X_1 and X_2 considered for normalisation it holds $X_1 \cap X_2 = \emptyset$. This assumption is necessary to separate the start and end of different regions. Note that a non-overlapping correspondence relation between process graphs, see Section 3.4, induces only distinct regions.

DEFINITION 5.1 (Normalisation for a Set of Regions). *Let $\Omega = \{X_1, \ldots, X_n\}$ be a set of distinct regions, $X_i \subseteq \mathcal{A}$. The* normalisation operator $\eta_\Omega : \mathcal{G} \mapsto \mathcal{G}$ *is defined as the concatenation of normalisations for all regions, i.e., $\eta_\Omega = \eta_{X_1} \circ \ldots \circ \eta_{X_n}$. A process graph g is* normalised *for Ω, iff $g = \eta_\Omega(g)$.*

The normalisation operator for a group of regions represents the joint application of normalisations for single regions. Therefore, it is important to verify that normalisation for one region does not interfere with normalisation for another region in terms of starting and ending regions. Indeed, this is the case.

PROPERTY 5. Let g be a process graph and let $X_1, X_2 \subseteq \mathcal{A}$ be two distinct regions. If process graph h is derived from g by rewriting for region X_1, $h = \eta_{X_1}(g)$, it holds that:

- if X_2 is started / may be started / ended / may have ended in $s \in S_g$ in g and it holds $s \in S_h$, then X_2 is started / may be started / ended / may have ended in s in h.
- if X_2 is started / may be started / ended / may have ended in $s \in S_g$ in g and it holds $s_{X_1}, s_{\overline{X_1}} \in S_h$, then X_2 is started / may be started / ended / may have ended in s_{X_1} and in $s_{\overline{X_1}}$ in h.

Proof. Both criteria follow directly from the fact that normalisation preserves branching bisimilarity (Property 2). □

Rewriting 'may be started' states of different regions is independent of each other. Each normalisation rewrites 'may be started' states for the respective region. Hence, normalisation of a process graph for a set of regions yields a graph that is free of 'may be started' states for all regions.

PROPERTY 6. Let h be a process graph that is derived by normalising a process graph g for a set of distinct regions $\Omega = \{X_1, \ldots, X_n\}$, $X_i \subseteq \mathcal{A}$. It holds that S_h does not contain any states in which any region X_i may, or may not, be started.

Proof. Rewriting a process graph for a single region yields a graph that is free of 'may be started' states for this region by Property 2. According to Property 5, this rewriting step does not introduce any 'may be started' states for another region that have not been part of the process graph before. □

For the process graphs introduced earlier and depicted in Figure 2, we depict the normalised process graphs in Figure 6. For both graphs, we used regions as

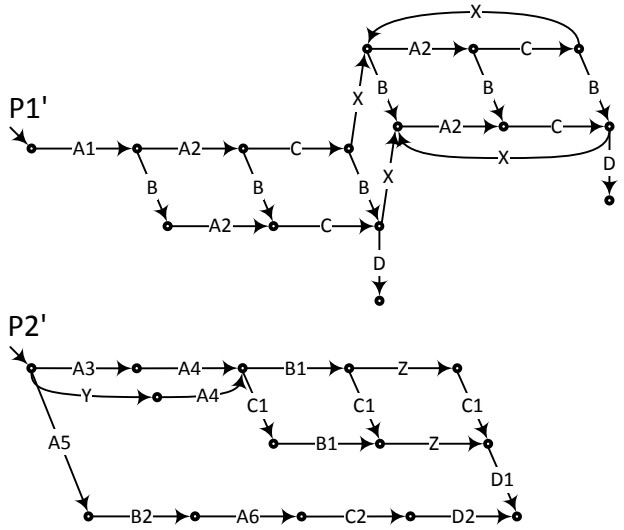

FIGURE 6: Process graphs of Figure 2 normalised for the sets of regions $\{\{A1, A2\}, \{B\}, \{C\}, \{D\}\}$ $(P1')$ and $\{\{A3, A4, A5, A6\}, \{B1, B2\}, \{C1, C2\}, \{D1, D2\}\}$ $(P2')$, respectively.

they are induced by the correspondences. At his stage, we neglected regions formed by actions that are not part of any correspondence. Process graph $P1'$ is obtained by normalising process graph $P1$ for the set of regions $\{\{A1, A2\}, \{B\}, \{C\}, \{D\}\}$. A large part of process graph $P1$ has been duplicated in the course of normalisation. This is caused by the cycles in the process graph $P1$. All states that are part of a cycle are 'may be started' states for region $\{C\}$. For process graph $P2$ in Figure 2, graph $P2'$ is derived by normalisation for the set of regions $\{\{A3, A4, A5, A6\}, \{B1, B2\}, \{C1, C2\}, \{D1, D2\}\}$. Here, states in which region $\{A3, A4, A5, A6\}$ may, or may not, be started have been rewritten.

Finally, we turn the focus on the interplay of starting and ending different regions. We observe that a single transition starts at most one region.

Property 7. Let g be a process graph that is normalised for a set of distinct regions $\Omega = \{X_1, \ldots, X_n\}$, $X_i \subseteq \mathcal{A}$. Any transition $s_1 \xrightarrow{a} s_2$ starts at most one region $X_i \in \Omega$.

Proof. According to Definition 4.1, a region X can only be started by a transition with $a \in X$. Since we assume that for all regions X_1, X_2, it holds that $X_1 \cap X_2 = \emptyset$, a transition can start at most one region. □

Since we adopted a notion of ending a region that may incorporate a delay, a single transition may end more than one region.

Property 8. Let g be a process graph that is normalised for a set of distinct regions $\Omega = \{X_1, \ldots, X_n\}$, $X_i \subseteq \mathcal{A}$. A transition $s_1 \xrightarrow{a} s_2$ can end more than one region $X_i \in \Omega$.

Proof. Consider the process graph $P1'$ in Figure 6 that is normalised for the set of regions $\{\{A1, A2\}, \{B\}, \{C\}, \{D\}\}$. Both transitions labelled with action D end regions $\{A1, A2\}$ and $\{C\}$. □

5.2. Separation of Regions

To assess behaviour compatibility of two process graphs based on regions, we rewrite the graphs so that they capture only the aggregated behaviour in terms of starting and ending regions. Technically, we assign labels that indicate starting and ending of a region to those transitions, whereas all other transitions are renamed to silent steps. Two aspects need to be taken into account when rewriting a process graph to separate regions.

First, a transition starts at most one region (Property 7), but may end multiple regions (Property 8). Since we rely on interleaving semantics, in which parallellism of actions is equal to arbitrary interleaving of actions, the latter shall be equivalent to all interleavings of ending the respective regions. Thus, we rewrite transitions that end multiple regions into a subgraph that encodes all these interleavings. Now, consider the case that a transition ends multiple regions but also starts a region. Assuming distinct regions, such a transition must be labelled with an action of the region that is started. As a consequence, we know that all regions ended (but not started) by the transition are ended with a delay. This observation lets us prioritise ending of regions. Following the discussion presented in Section 4.1, we aim at capturing the earliest state in which a region has definitely ended. Hence, we first end all regions, before a region is started.

Second, it may be the case that a single transition starts and ends a region. For our use case, we abstract from the internal behaviour of a region. Hence, starting and ending a region with one transition shall be equivalent to starting the region and closing it at a later stage. Our rewriting to separate regions splits up a transition that starts and ends a region into two transitions.

Note that both phenomena may occur together. A single transition may end multiple regions among them one region that is also started by the transition. In this case, we first end all regions except the region that is also started. Then, the latter region is started and ended with separate transitions.

We obtain a process graph with separate regions by rewriting. To keep the formalisation concise, we use regions, i.e., sets of actions, to construct actions that relate to these regions. We assign the actions $s(X_i)$ or $e(X_i)$ to all transitions that start or end a region X_i, respectively. The silent action is assigned to all transitions that neither start or end any region. As part of this step, all transitions that end more than one region are replaced by a subgraph capturing all interleavings of ending the regions. All transitions that start and end a region X_i, are expanded into an intermediate state

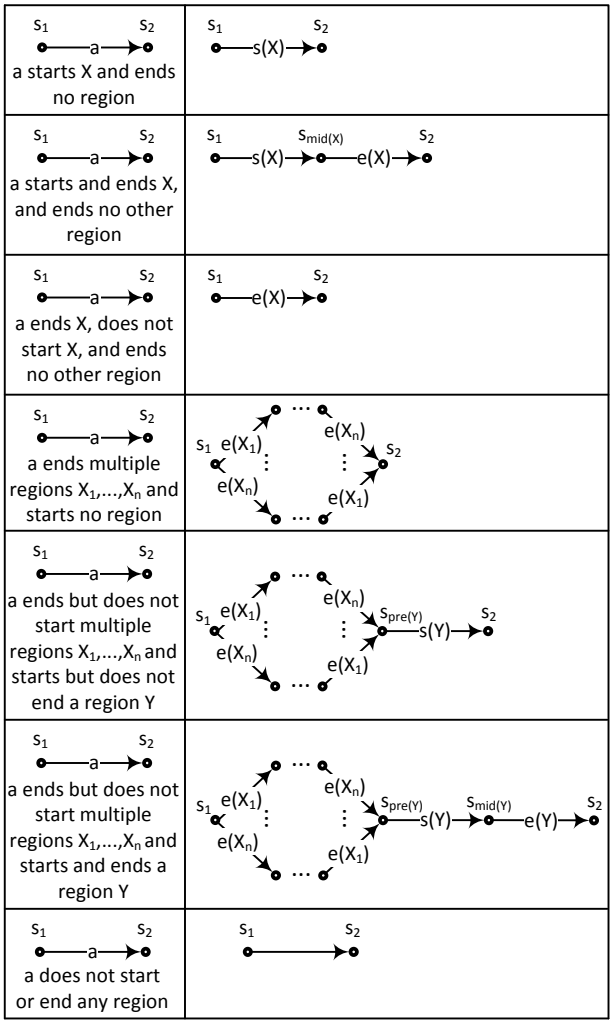

FIGURE 7: Overview of the rewriting to separate regions.

and two transitions labelled with $s(X_i)$ and $e(X_i)$. The different cases in terms of starting and ending regions for a single transition are shown in Figure 7.

The result of region separation is illustrated for an example in Figure 8. Here, we used a short-hand notation to reference the regions, e.g., label A refers to the region $\{A1, A2\}$. Besides relabelling the transitions that start or end any of the three regions $\{A1, A2\}$, $\{B1, B2\}$, and $\{C\}$, two transitions that end more than one region are rewritten to capture all interleavings. Also, we first end regions $\{A1, A2\}$ and $\{B1, B2\}$, before starting the region $\{C\}$ by the respective transition.

Formally, the rewriting is defined as follows. Note that, given a set M, we denote the powerset of M by $\wp(M)$.

DEFINITION 5.2 (Region Separation). *Let g be a process graph that is normalised with respect to a set of regions $\Omega = \{X_1, \ldots, X_n\}$, $X_i \subseteq \mathcal{A}$. The* separation operator $\zeta_\Omega : \mathcal{G} \mapsto \mathcal{G}$ *rewrites g into the process graph h by defining S_h and $\rightarrow_h$ as follows.*

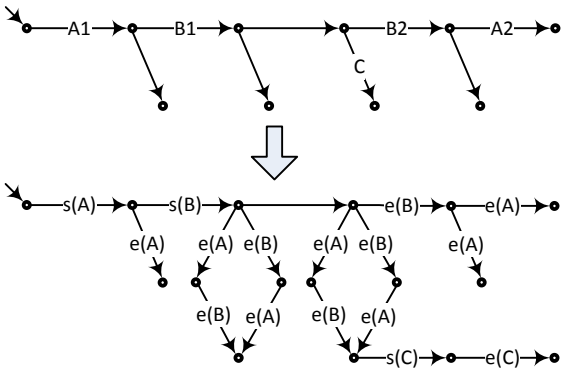

FIGURE 8: Rewriting to separate the regions $\{A1, A2\}$, $\{B1, B2\}$, and $\{C\}$.

S_h *is the smallest set that contains:*

- *all $s \in S_g$;*
- *$2^m - m$ new states s_p for each transition $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$ that ends m regions $X_1, \ldots, X_m \in \Omega$, $1 < m$, but starts none of them, with $p \in \wp(\{X_1, \ldots, X_m\}) \setminus \{\{\}, \{X_1, \ldots, X_m\}\}$ being a set that represents all permutations of regions except for the empty set or the set that contains all regions.*
- *one new state $s_{pre(X_i)}$ for each transition $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$ that ends at least one region and starts the region $X_i \in \Omega$.*
- *one new state $s_{mid(X_i)}$ for each transition $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$ that starts and ends the region $X_i \in \Omega$.*

$\rightarrow_h$ *is the smallest set that contains:*

- *$s_1 \xrightarrow{s(X_i)} s_2$, iff $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$ starts the region $X_i \in \Omega$ and ends no region;*
- *$s_1 \xrightarrow{s(X_i)} s_{mid(X_i)} \xrightarrow{e(X_i)} s_2$, iff $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$ starts and ends a single region $X_i \in \Omega$ and ends no other region;*
- *$s_1 \xrightarrow{e(X_i)} s_2$, iff $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$ ends a single region $X_i \in \Omega$, ends no other region, and starts no region;*
- *$s_1 \xrightarrow{e(X_i)} s_{pre(X_j)}$, iff $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$ ends a single region $X_i \in \Omega$, ends no other region, and starts another region $X_j \in \Omega$;*
- *$s_p \xrightarrow{e(X_i)} s_q$, iff $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$ ends more than one region, among them $X_i \in \Omega$ and either*
  - *$s_p = s_1$ and $X_i \in q$,*
  - *$X_i \notin p$ and $X_i \in q$,*
  - *$s_q = s_2$ and $X_i \notin p$, and $s_1 \xrightarrow{a} s_2$ does not start any region,*
  - *$s_q = s_{pre(X_j)}$ and $X_i \notin p$, and $s_1 \xrightarrow{a} s_2$ starts the region $X_j \in \Omega$.*
- *$s_{pre(X_i)} \xrightarrow{s(X_i)} s_2$, iff $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$ starts the region $X_i \in \Omega$ and ends at least one other region.*
- *$s_{pre(X_i)} \xrightarrow{s(X_i)} s_{mid(X_i)} \xrightarrow{e(X_i)} s_2$, iff $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$ starts and ends the region $X_i \in \Omega$ and ends at least one other region.*
- *$s_1 \xrightarrow{\tau} s_2$, iff $s_1 \xrightarrow{a} s_2 \in \rightarrow_g$ neither starts nor ends any region $X_i \in \Omega$.*

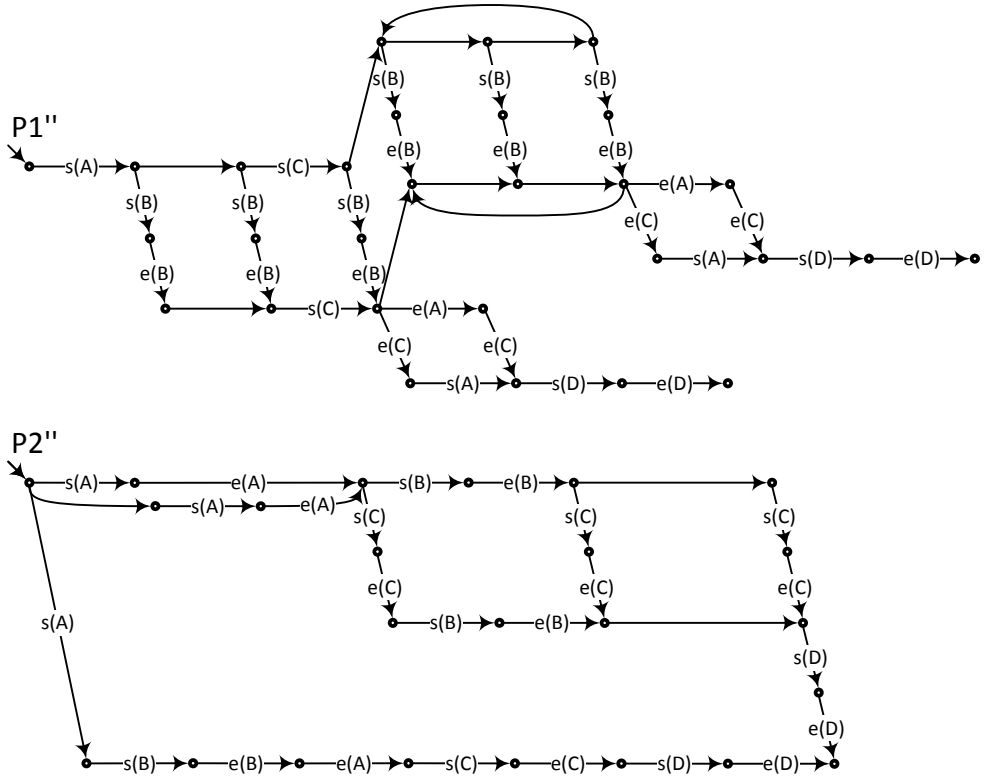

FIGURE 9: Process graphs of Figure 6 for which the following regions have been separated: $\{\{A1, A2\}, \{B\}, \{C\}, \{D\}\}$ for process graph $P1''$ and $\{\{A3, A4, A5\}, \{B1, B2, B3\}, \{C1, C2\}, \{D1, D2\}\}$ for process graph $P2''$.

For our initial example, region separation is illustrated in Figure 9. For the process graphs depicted in Figure 6, we separated all regions that are induced by the correspondences between the two behavioural models as illustrated initially in Figure 1. We used a shorthand notation to indicate the regions, e.g., the label A refers to the region $\{A1, A2\}$ or $\{A3, A4, A5, A6\}$. We observe that several transitions of the graphs in Figure 6 are split up in two transitions to indicate the start and end of the regions, respectively. Also, in graph $P1'$ of Figure 6, two transitions end more than one region. Using the proposed rewriting, we obtain the graph $P1''$ that comprises all interleavings of ending these regions.

5.3. Properties for Separated Regions

The process after region separation should be equivalent to the process before region separation. However, clearly we cannot prove branching bisimilarity between the two processes, because the region separation operator changes the labels of the processes. This is done deliberately, to encode the events of starting a region and ending a region, as they have been explained in the previous section. Therefore, we show a weaker property for the region separation operator. In particular, we show that the order in which regions are started or ended is preserved by the operator.

Property 9. Let g be a process graph that is normalised with respect to a set of regions $\Omega = \{X_1, \ldots, X_n\}$, $X_i \subseteq \mathcal{A}$, and let $h = \zeta_\Omega(g)$ be the region separated process graph for g. Let x and y be actions of starting some region, ending some region or not starting or ending any region. Furthermore, let g contain the transition sequence $s_1 \xrightarrow{a} s_2 \xrightarrow{b} s_3$, where $s_1 \xrightarrow{a} s_2$ also contains a transition labelled with x and $s_2 \xrightarrow{b} s_3$ also contains a transition labelled with y. It holds that $s_1, s_2, s_3 \in S_h$ and there exists a transition sequence $s_1 \overset{\sigma}{\Longrightarrow} s_2 \overset{\rho}{\Longrightarrow} s_3$ in h and all transition sequences $s_1 \overset{\sigma}{\Longrightarrow} s_2$ in h also contain a transition labelled with x and all transition sequences $s_2 \overset{\rho}{\Longrightarrow} s_3$ in h also contain a transition labelled with y.

Proof. The proof is by construction. All $s \in S_g$ are also in S_h and each transition $s \xrightarrow{a} s' \in \rightarrow_g$ that contains a transition labelled with x is rewritten (by the rewrite rules that are illustrated in Figure 8) into transition sequences that connect s and s' and that all comprise a transition labelled with x. □

6. EQUIVALENCE AND COMPATIBILITY

This section shows how we can lift existing notions of behaviour equivalence and compatibility, which work for 1:1 correspondences, to notions of behaviour equivalence and compatibility that work for complex correspondences. Aggregation of behaviour based on regions, as it is introduced in the previous sections, is the basis for these notions.

First, we show how equivalence of behaviours with complex correspondences can be defined in Section 6.1. Then, we follow the work on behaviour inheritance to cope with model extensions. We elaborate on notions of projection compatibility, which incorporate action abstraction, in Section 6.2. Then, we turn the focus to protocol compatibility, which relies on action encapsulation, in Section 6.3. Finally, abstraction and encapsulation are combined to arrive at the notion of lifecycle compatibility in Section 6.4.

6.1. Region Equivalence

Given two process graphs that are aligned by (potentially complex) correspondences, behaviour equivalence can be decided based on the introduced aggregate behaviours of the two process graphs. To this end, we construct a set of regions, such that each correspondence between two process graphs forms a region. Further, all transitions that are not part of any correspondence are considered separate regions. Using the resulting set of regions, we apply region normalisation and separation to both process graphs. Thus, we obtain two process graphs that capture aggregated behaviour, i.e., the behavioural dependencies between the regions induced by the correspondences. For these process graphs, we check behaviour equivalence. In the following definition, we rely on concrete trace equivalence and branching bisimulation, as introduced in Section 3.3. Note that we also use the notations for sets of actions that are part of correspondences as defined in Section 3.4.

Definition 6.1 (Region Equivalence). *Let g and h be two process graphs and $\simeq \subseteq (A_g \setminus A_h) \times (A_h \setminus A_g)$ a correspondence relation.*

- *Let $\Omega = \{X_1, \ldots, X_n\}$ be a set of regions that is induced by correspondences and transitions that are not aligned, i.e., it holds $X \in \Omega$, iff either*
  - *$X = X' \cup X''$ and $X' \simeq X''$, or*
  - *$X = \{x\}$ and $x \in (A_g \cup A_h) \setminus (A_g^{\simeq} \cup A_h^{\simeq})$.*
- *Let g' and h' be the process graphs derived from g and h by region normalisation and separation for Ω, i.e., $g' = \zeta_\Omega(\eta_\Omega(g))$ and $h' = \zeta_\Omega(\eta_\Omega(h))$.*
- *The graphs g and h are* region concrete trace equivalent*, denoted by $g \doteq_\Omega h$, iff $g' \doteq h'$.*
- *The graphs g and h are* region branching bisimilar*, denoted by $g \sim_\Omega h$, iff $g' \sim h'$.*

For our initial example, depicted in Figure 1 and Figure 2, none of the region equivalences is satisfied. This is caused by actions that are not part of any correspondence. For instance, the process graph derived from the graph $P1$ in Figure 2 by region normalisation and separation contains transitions labelled with $s(X)$ and $e(X)$ to indicate starting and ending of the region $\{X\}$. (Recall that for the graphs shown in Figure 6, we neglected actions that are not aligned by any correspondence.) Starting and ending the region $\{X\}$ is apparently not mirrored by the graph derived by region normalisation and separation from $P2$ in Figure 2.

6.2. Projection Compatibility

Following the work on behaviour inheritance, actions may be hidden before behaviour equivalence is decided. Hiding of actions is realised by abstraction, as introduced in Section 3.2. We define projection compatibility in the line of projection inheritance. We use the term compatibility instead of inheritance to highlight that abstraction may be applied to both process graphs under investigation. In contrast to inheritance, compatibility does not assume the relation between both process graphs to be directed in the sense of a subclass – superclass relation.

Projection compatibility holds between two process graphs, if they are equivalent once a set of actions is abstracted. The original definition of projection inheritance, see [6], does not restrict the set of actions that can be abstracted. However, we assume that correspondences are defined explicitly to indicate corresponding actions. Those actions shall not be subject to abstraction, so that we restrict the set of actions that may be abstracted to all actions that are not part of a correspondence. To decide behaviour equivalence in the presence of complex correspondences, we rely on the notions of region equivalence introduced in the previous section. Region equivalence may be based on traces or the branching structure.

Definition 6.2 (Projection Compatibility). *Let g and h be two process graphs and $\simeq \subseteq (A_g \setminus A_h) \times (A_h \setminus A_g)$ a correspondence relation.*

- *The graphs g and h are* trace projection compatible*, iff there are sets of actions $X \subseteq A_g \setminus A_g^{\simeq}$ and $Y \subseteq A_h \setminus A_h^{\simeq}$, such that $\tau_X(g) \doteq_\Omega \tau_Y(h)$.*
- *The graphs g and h are* projection compatible*, iff there are sets of actions $X \subseteq A_g \setminus A_g^{\simeq}$ and $Y \subseteq A_h \setminus A_h^{\simeq}$, such that $\tau_X(g) \sim_\Omega \tau_Y(h)$.*

Projection compatibility for our initial example is determined using the process graphs depicted in Figure 9. Here, all actions that are not aligned by correspondences, see Figure 1, have been abstracted. Note that, again, we interpret Figure 9 such that the labels are shorthand notations for the respective regions, e.g., the label A refers to the region $\{A1, A2, A3, A4, A5, A6\}$ that is formed by actions of either process graphs. Despite their commonalities, the process graphs in Figure 9 do not satisfy (trace) projection compatibility. In process graph $P1''$, region $\{C, C1, C2\}$ may be started after region $\{A1, A2, A3, A4, A5, A6\}$ has been started, but before it has ended. This behaviour is not possible in process graph $P2''$.

6.3. Protocol Compatibility

Besides hiding of actions, blocking of actions may be applied before behaviour equivalence is decided.

Blocking of actions is realised by encapsulation and gives rise to the notion of protocol inheritance. In the same vein, we introduce protocol compatibility that encapsulates a set of actions before a notion of region equivalence is decided.

Definition 6.3 (Protocol Compatibility). *Let g and h be two process graphs and $\simeq \subseteq (A_g \setminus A_h) \times (A_h \setminus A_g)$ a correspondence relation.*

- *The graphs g and h are* trace protocol compatible*, iff there are sets of actions $X \subseteq A_g \setminus A_g^{\simeq}$ and $Y \subseteq A_h \setminus A_h^{\simeq}$, such that $\delta_X(g) \doteq_\Omega \delta_Y(h)$.*
- *The graphs g and h are* protocol compatible*, iff there are sets of actions $X \subseteq A_g \setminus A_g^{\simeq}$ and $Y \subseteq A_h \setminus A_h^{\simeq}$, such that $\delta_X(g) \sim_\Omega \delta_Y(h)$.*

To verify (trace) protocol compatibility for our initial example, we need to encapsulate the actions X, Y, and Z of the process graphs. As mentioned earlier, encapsulation of action Z for the process graph $P2$ in Figure 2 means that once the transition labelled with $A3$ is taken, no transition labelled with an action from the region $\{D, D1, D2\}$ may be taken from some reachable state. Since this region is started and ended for all transition sequences leading to a final state in process graph $P1$ in Figure 2, both process graphs do not satisfy (trace) protocol compatibility.

6.4. Lifecycle Compatibility

It may be necessary to combine hiding and blocking of actions to satisfy behaviour equivalence. In this framework for behaviour inheritance, this case is represented by lifecycle inheritance. Adapting this notion for region equivalence yields lifecycle compatibility.

Definition 6.4 (Lifecycle Compatibility). *Let g and h be two process graphs and $\simeq \subseteq (A_g \setminus A_h) \times (A_h \setminus A_g)$ a correspondence relation.*

- *The graphs g and h are* trace lifecycle compatible*, iff there are sets of actions $V, W \subseteq A_g \setminus A_g^{\simeq}$, $V \cap W = \emptyset$, and $X, Y \subseteq A_h \setminus A_h^{\simeq}$, $X \cap Y = \emptyset$, such that $\tau_V \circ \delta_W(g) \doteq_\Omega \tau_X \circ \delta_Y(h)$.*
- *The graphs g and h are* lifecycle compatible*, iff there are sets of actions $V, W \subseteq A_g \setminus A_g^{\simeq}$, $V \cap W = \emptyset$, and $X, Y \subseteq A_h \setminus A_h^{\simeq}$, $X \cap Y = \emptyset$, such that $\tau_V \circ \delta_W(g) \sim_\Omega \tau_X \circ \delta_Y(h)$.*

For the initial example introduced in Figure 2, Figure 10 shows the process graphs obtained after hiding and blocking of actions has been combined. That is, we encapsulated actions X and Y, and abstracted action Z. All remaining actions are part of correspondences defined between both process graphs. Using the regions induced by the correspondences, we applied region normalisation and separation to both process graphs. Figure 10 illustrates that both process graphs are trace lifecycle compatible. The sets of (in this example only finite) concrete traces coincide for the process graphs $P1^*$ and $P2^*$. In other words, the observable behaviour of both process graphs is the same, once the behaviour for correspondences is aggregated and actions that are not aligned are blocked or hidden.

Figure 10 also illustrates that both process graphs are not lifecycle compatible. Despite the same observable behaviour, both process graphs show differences in their branching structure. In $P1^*$, the transition that starts the region $\{A1, A2, A3, A4, A5, A6\}$ leads to a state that may be left by a transition that closes this region or starts the region $\{B, B1, B2\}$. This branching behaviour is not in line with the process graph $P2^*$.

7. EXTENDED EXAMPLE

This section explains how the compatibility notions that were presented in the previous section, can be applied to an example from practice.

Figure 11a shows two process models that represent business processes from practice. They are simplified versions of processes that we used in previous research [22]. The top shows a reference process for applying for a permit to chop down a tree. The bottom process is an implementation of that process at a municipality. Compatibility between the two processes can be verified to determine the extent to which the implementation conforms to the reference process. The activities in the processes have been annotated with letters to indicate correspondences; activities with the same letter correspond.

Figure 11b shows the respective process graphs. The graphs represent the start and end transitions of each region as well as transitions that do not correspond to the start or end of a region. The latter are silent transitions, but they are labelled with the letter that refers to the activity from which the transition was derived. This convention is used here to be able to refer to the transitions.

Using Figure 11, we can see that the processes are not projection compatible, because hiding activities H ('Check admissible') and I ('Notify not admissible') enables the trace $s(A)$, $e(A)$ in the bottom graph, which is not possible in the top graph. In practical terms the bottom process allows an application to be received and then be discarded, because it is inadmissible, while the top process does not allow this. Using protocol compatibility solves this problem, because it blocks rather than hides activity I ('Notify not admissible'). Blocking this activity makes it impossible to discard claims due to inadmissibility in the bottom process. However, using protocol compatibility introduces problems at other places in the process, because blocking activities E, F or H leads to dead tasks. Blocking transition G leads to the situation in which the top process ends after activity C ('Send approval'), while the bottom process still must perform activity D ('Send bill').

The processes are lifecycle compatible, i.e.: they are

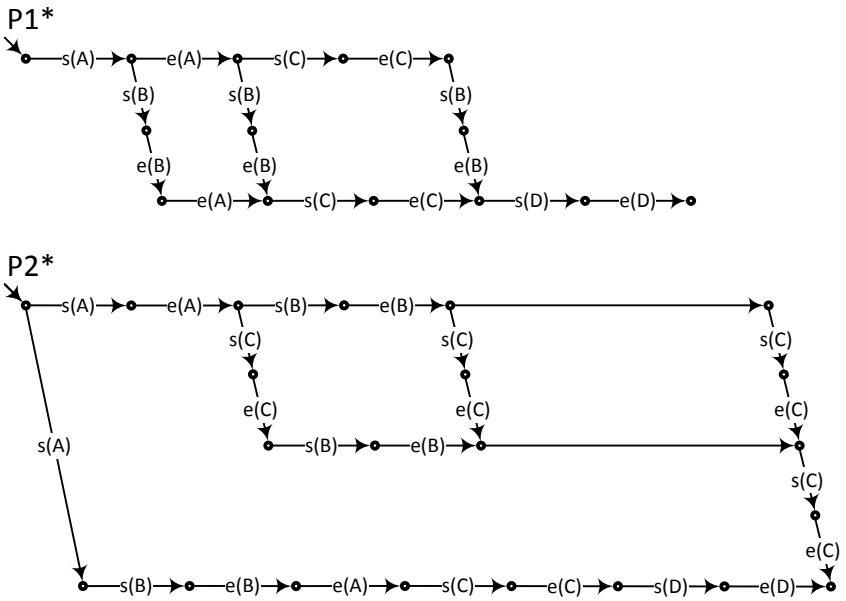

FIGURE 10: Process graphs of Figure 2 for which we encapsulated actions X and Y, abstracted action Z, and applied region normalisation and separation for the regions $\{\{A1, A2, A3, A4, A5, A6\}, \{B, B1, B2\}, \{C, C1, C2\}, \{D, D1, D2\}\}$.

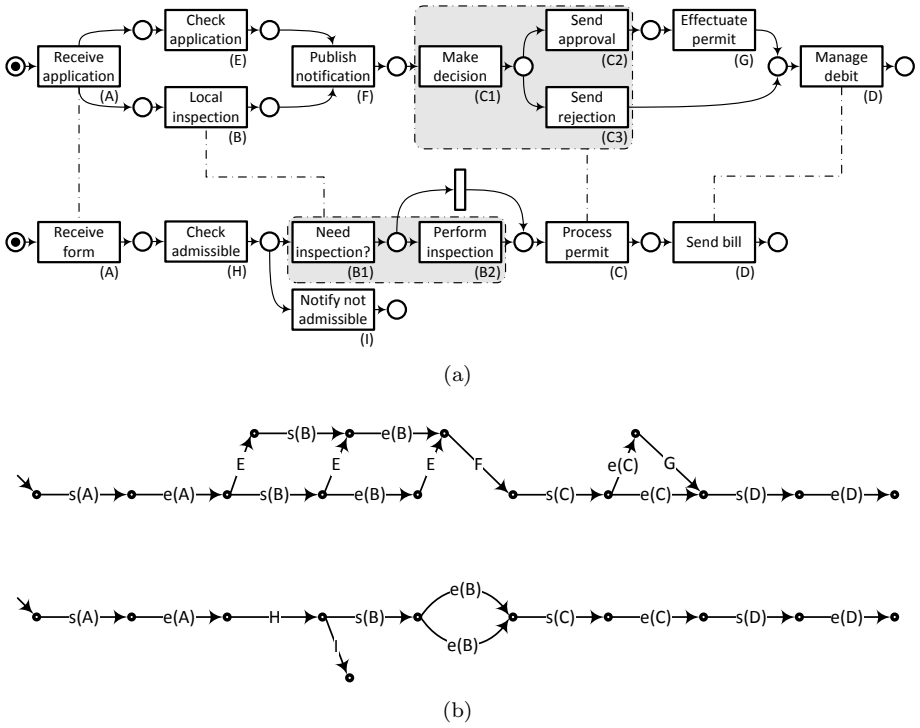

FIGURE 11: (a) net systems for related processes; (b) corresponding process graphs with start and end region transitions.

both trace equivalent and branching bisimilar when blocking activity I and hiding activities E, F, G and H. In practical terms, this means that in order to make the implemented process compatible with the reference process, rejecting an application due to inadmissibility must be made impossible, while the admissibility check must be skipped. In the reference process, checking the application, publishing a notification and effectuating the permit must be skipped. The information that is obtained by this compatibility check, can be used to, for example, configure a software application that has been built to support the reference process and to discuss the suitability of that software for the organization.

8. RELATED WORK

Behaviour equivalence and compatibility relate to different streams of research. In this section, we focus on model matching, similarity of process models, and model refinements and compatibility.

8.1. Model Matching

We argued that verification of behaviour compatibility, first and foremost, requires the identification of corresponding activities of two business process models, or corresponding actions of two process graphs. Such correspondences may implicitly be given once business process models originate from a common source. An example would be models that are derived by customisation of a reference model [15, 16]. In other use cases, however, correspondences need to be defined explicitly. Even though correspondences may be defined manually, compatibility verification in a large scale requires automated support for suggesting correspondences. In general, the detection of correspondences between business process models can be seen as an instantiation of the *matching problem* [17] known from the field of data integration or ontology matching. Hence, techniques proposed to match data schemas or ontologies, see [17, 18, 19] for thorough overviews, may be adapted to detected process model correspondences.

For business process models, most matching approaches rely on a textual techniques that are applied to the labels of activities. Further, the model structure and the behaviour defined by the models may be exploited, for instance in [20, 21]. That is, structural and behavioural similarity measures are used in combination with textual measures to identify correspondences. We review such similarities separately. However, except for a few exceptions, e.g., the ICoP framework [22], matching of business process models focusses on elementary correspondences and largely neglects complex correspondences.

8.2. Process Model Similarity

Besides guiding the identification of correspondences, similarity measures for business process models allow for drawing conclusions on the extent to which two models differ once they do not show behaviour compatibility.

Textual similarity measures are at the very core of most approaches to judge similarity of business process models. In addition, the graph structure may be leveraged using the notions of maximum common subgraph isomorphism and graph edit distance [23]. An overview of graph matching techniques can be found in [24]. The graph edit distance has been used to judge similarity of business process models in [25, 26]. In essence, this measure is based on the fractions of activities and flow arcs between them that need to be inserted or deleted in order to transform one graph into another. Other work follows a similar idea, but relies on high-level and hierarchical change operations instead of elementary graph operations [27, 28].

Depending on the kind of semantics assumed for business process models, similarity measures may be defined for transitions systems such as process graphs used in this work, traces that represent valid sequences of visible actions, or behavioural abstractions. The extent to which two transition systems simulate each other was leveraged in [29, 21]. Those works iteratively evaluate the similarity of states based on the similarity of neighbouring state transitions and states. The overall similarity is obtained once a fixpoint or a predefined number of iterations is reached. Further, the edit distances for sets of traces, or different representations thereof, have been proposed as similarity measures [30, 31]. Yet other work uses behavioural abstractions as the basis of similarity measures. Such measures are grounded on causal footprints that capture sets of causal predecessors and successors for activities [32, 26], or behavioural profiles that capture the order of potential execution for pairs of activities [33].

8.3. Model Refinements & Compatibility

Complex correspondences between behavioural models are closely related to refinements between them. A refinement refers to the specification of a certain process model activity, or an action of a process graph, in more detail. Hence, a refinement relation between two models induces complex correspondences of a particular type, i.e., 1:n correspondences that are equally directed (all compound activities are part of one model, all refined activities are part of the other model) and non-overlapping. Refinements, therefore, are commonly assumed to be hierarchical. Following this line, a variety of refinements that are behaviour preserving have been presented. An overview of Petri net refinements can be found in [34]. Here, a place or transition is typically substituted by a subnet, see [35, 36]. Refinement operators for process algebra have been introduced in [37, 38].

Refinement operators guide the adaptation of behavioural models and aim at preserving selected properties, such as deadlock freedom or weak bisimulation. As such, models that are aligned by correspondences may be considered to be compatible, if the correspondences can be traced back to refinements. However, such an approach is hard to operationalise once corresponding sets of activities, or regions of actions, are not isolated. For instance, it is unclear how the correspondences between the example net systems in Figure 1 can be traced back to Petri net refinements.

There is little work on notions of behaviour compatibility that is applicable for arbitrary complex correspondences between behavioural models. Recently, we advocated the application of behavioural profiles as an behavioural abstraction to decide compatibility of

process models [39, 40]. This approach assesses whether behavioural relations defined over pairs of activities in one model coincide with the relations defined for corresponding activities in another model. Even though this approach is applicable for correspondences of any type, it considers only an abstraction of trace semantics.

9. CONCLUSIONS

Given a relation between corresponding activities of two business process models, we argued that two challenges arise when verifying equivalence or compatibility: (1) one model can contain activities that do not correspond to any activity in the other model; and (2) complex correspondences can exist between the models, where each complex correspondence represents a relation between a set of activities in one model and a set of activities in the other model. The first challenge can be approached using notions of behaviour inheritances, whereas the second challenge has not been addressed in previous work. In this paper, we focussed on this challenge and introduced a framework that leverages regions of activities to decide equivalence and compatibility in the presence of complex correspondences. We require the regions in one model to coincide with the regions formed by corresponding activities in the other model, in terms of when these regions start and end. We illustrated the approach using two equivalences, concrete trace equivalence and branching bisimulation. In addition, we showed how the approach integrates with the existing work on behaviour inheritance to develop notions of compatibility.

The question of how behaviour equivalences may be applied in the presence of complex correspondence between behavioural models, first and foremost, is a generic question that is independent of any use case. However, we made explicit that any solution to this problem involves several design decisions. Depending on how these design choices are answered, the process graphs are preprocessed such that they are comparable using existing notions of equivalence. Apparently this allows for conclusions only if the applied preprocessing is considered to be valid. We perceive this to be an inherent problem of behaviour compatibility of complex correspondences that cannot be avoided. In this paper, we answered the respective design decisions for behavioural models that capture business processes. For instance, we argued that pausing regions does not seem to be appropriate for this use case. Based on such application specific reasoning, we implemented a preprocessing step that yields an aggregation of the behaviour of multiple regions. For a different setting, e.g., for behavioural models describing electronic circuits or software applications, the aforementioned design decisions may be answered differently. By making the decision explicit, we have outlined how our compatibility notions may be adapted for a different purpose. The discussion of the impact of differently taken design choices, however, has been beyond the scope of this paper.

Besides, the assumption of non-overlapping correspondences has to be seen as a limitation of our work. Even though we argued that the majority of complex correspondences observed in practice are non-overlapping, the model collection described in [22] also shows some overlapping correspondences. Those cannot be addressed by our approach. Further work is needed to investigate the semantics of these correspondences and the actions that are part of the overlap. In particular, splitting up the respective actions to clearly separate correspondences may turn out to be a resolution strategy that, in the end, enables the application of the technique presented in this work.

We discussed that there is a large body of work on refinement operations that preserve behavioural properties. Similarly, transformation rules for behavioural models that preserve the notions of behaviour inheritance have been presented in [6]. We aim at investigating similar rules that go beyond existing rules in terms of expressiveness, but preserve the presented notions of behaviour compatibility in future work. As outlined in [6], such rules shall meet two criteria, computational efficiency for verifying whether the rule satisfies the requirements and usability from a practical point of view. Besides transformation rules, feedback on deviations in case of violated compatibility is a direction for future work.

REFERENCES

[1] Weske, M. (2007) *Business Process Management: Concepts, Languages, Architectures*. Springer.

[2] van Glabbeek, R. J. (2001) Handbook of Process Algebra. Elsevier.

[3] van Glabbeek, R. J. (1993) The linear time - branching time spectrum ii. In Best, E. (ed.), *CONCUR*, Hildesheim, Germany, August 23-26, Lecture Notes in Computer Science, **715**, pp. 66–81. Springer.

[4] Weidlich, M., Dijkman, R. M., and Weske, M. (2010) Deciding behaviour compatibility of complex correspondences between process models. In Hull, R., Mendling, J., and Tai, S. (eds.), *BPM*, Hoboken, NJ, USA, September 13-16, Lecture Notes in Computer Science, **6336**, pp. 78–94. Springer.

[5] van der Aalst, W. M. P. and Basten, T. (1997) Life-cycle inheritance: A petri-net-based approach. In Azéma, P. and Balbo, G. (eds.), *ICATPN*, Toulouse, France, June 23-27, Lecture Notes in Computer Science, **1248**, pp. 62–81. Springer.

[6] Basten, T. and Aalst, W. (2001) Inheritance of behavior. *JLAP*, **47**, 47–145.

[7] Reisig, W. (1985) *Petri Nets: An Introduction*, Monographs in Theoretical Computer Science. An EATCS Series, **4**. Springer.

[8] Zelewski, S. (1995) Petrinetzbasierte Modellierung komplexer Produktionssysteme - Eine Untersuchung des Beitrags von Petrinetzen zur Prozeßkoordinierung in komplexen Produktionssystemen, insbesondere

Flexiblen Fertigungssystemen, Band 9: Beurteilung des Petrinetz-Konzepts. Technical report. University of Leipzig. (in German).

[9] van Glabbeek, R. J. (1990) Comparative Concurrency Semantics and Refinement of Actions. PhD thesis Free University Amsterdam. Introduction available at `http://theory.stanford.edu/~rvg/thesis.html`. Second edition available as *CWI tract* 109, CWI, Amsterdam 1996.

[10] (2011) Business Process Model and Notation (BPMN) Version 2.0. Technical Report. Object Management Group (OMG).

[11] Keller, G., Nüttgens, M., and Scheer, A.-W. (1992) Semantische Prozeßmodellierung auf der Grundlage 'Ereignisgesteuerter Prozeßketten (EPK)'. Technical Report 89. Veröffentlichungen des Instituts für Wirtschaftsinformatik, Saarbrücken.

[12] van Glabbeek, R. J. and Weijland, W. P. (1996) Branching time and abstraction in bisimulation semantics. *J. ACM*, **43**, 555–600.

[13] Hoare, C. A. R. (1980) Communicating sequential processes. In McKeag, R. M. and MacNaghten, A. M. (eds.), *On the construction of programs*. Cambridge University Press.

[14] Hidders, J., Dumas, M., Aalst, W., Hofstede, A., and Verelst, J. (2005) When are two workflows the same? In Atkinson, M. D. and Dehne, F. K. H. A. (eds.), *CATS*, Newcastle, NSW, Australia, January-February 2005, CRPIT, **41**, pp. 3–11. Australian Computer Society.

[15] Rosemann, M. and van der Aalst, W. M. P. (2007) A configurable reference modelling language. *Inf. Syst.*, **32**, 1–23.

[16] Gottschalk, F., van der Aalst, W. M. P., Jansen-Vullers, M. H., and Rosa, M. L. (2008) Configurable workflow models. *Int. J. Cooperative Inf. Syst.*, **17**, 177–221.

[17] Euzenat, J. and Shvaiko, P. (2007) *Ontology Matching*. Springer-Verlag.

[18] Rahm, E. and Bernstein, P. A. (2001) A survey of approaches to automatic schema matching. *VLDB Journal*, **10**, 334–350.

[19] Choi, N., Song, I.-Y., and Han, H. (2006) A survey on ontology mapping. *SIGMOD Record*, **35**, 34–41.

[20] Dijkman, R., Dumas, M., García-Bañuelos, L., and Käärik, R. (2009) Aligning business process models. In: *EDOC*, Auckland, New Zealand, September 1-4, pp.45–53, IEEE Computer Society.

[21] Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S. M., and Zave, P. (2007) Matching and merging of statecharts specifications. *ICSE*, Minneapolis, MN, USA, May 20-26, pp. 54–64. IEEE Computer Society.

[22] Weidlich, M., Dijkman, R. M., and Mendling, J. (2010) The ICoP framework: Identification of correspondences between process models. In Pernici, B. (ed.), *CAiSE*, Hammamet, Tunisia, June 7-9, Lecture Notes in Computer Science, **6051**, pp. 483–498. Springer.

[23] Dumas, M., García-Bañuelos, L., and Dijkman, R. M. (2009) Similarity search of business process models. *IEEE Data Eng. Bull.*, **32**, 23–28.

[24] Bunke, H. (2000) Graph matching: Theoretical foundations, algorithms, and applications. *Proceedings of the International Conference on Vision Interface*, Montreal, Quebec, Canada, May, pp. 82–88.

[25] Dijkman, R. M., Dumas, M., and García-Bañuelos, L. (2009) Graph matching algorithms for business process model similarity search. In Dayal, U., Eder, J., Koehler, J., and Reijers, H. A. (eds.), *BPM*, Ulm, Germany, September 8-10, Lecture Notes in Computer Science, **5701**, pp. 48–63. Springer.

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

[27] Li, C., Reichert, M., and Wombacher, A. (2008) On measuring process model similarity based on high-level change operations. In Li, Q., Spaccapietra, S., Yu, E. S. K., and Olivé, A. (eds.), *ER*, Barcelona, Spain, October 20-24, Lecture Notes in Computer Science, **5231**, pp. 248–264. Springer.

[28] Küster, J., Gerth, C., Förster, A., and Engels, G. (2008) Detecting and resolving process model differences in the absence of a change log. *BPM*, Milan, Italy, September 2-4, LNCS, **5240**, pp. 244–260.

[29] Sokolsky, O., Kannan, S., and Lee, I. (2006) Simulation-based graph similarity. *TACAS*, Vienna, Austria, March 25 - April 2, pp. 426–440.

[30] Wombacher, A. and Rozie, M. (2006) Evaluation of workflow similarity measures in service discovery. In Schoop, M., Huemer, C., Rebstock, M., and Bichler, M. (eds.), *Service Oriented Electronic Commerce*, LNI, **80**, pp. 51–71. GI.

[31] Wombacher, A. and Li, C. (2010) Alternative approaches for workflow similarity. *IEEE SCC*, Miami, Florida, USA, July 5-10, pp. 337–345. IEEE Computer Society.

[32] Dongen, B., Dijkman, R. M., and Mendling, J. (2008) Measuring similarity between business process models. In Bellahsene, Z. and Léonard, M. (eds.), *CAiSE*, Montpellier, France, June 16-20, LNCS, **5074**, pp. 450–464. Springer.

[33] Kunze, M., Weidlich, M., and Weske, M. (2011) Behavioral similarity - a proper metric. In Rinderle-Ma, S., Toumani, F., and Wolf, K. (eds.), *BPM*, Clermont-Ferrand, France, August 30 - September 2, Lecture Notes in Computer Science, **6896**, pp. 166–181. Springer.

[34] Brauer, W., Gold, R., and Vogler, W. (1989) A survey of behaviour and equivalence preserving refinements of Petri nets. In Rozenberg, G. (ed.), *Applications and Theory of Petri Nets*, Bonn, Germany, June, Lecture Notes in Computer Science, **483**, pp. 1–46. Springer.

[35] Peterson, J. L. (1977) Petri nets. *ACM Comput. Surv.*, **9**, 223–252.

[36] Murata, T. (1989) Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, **77**, 541–580.

[37] Aceto, L. and Hennessy, M. (1994) Adding action refinement to a finite process algebra. *Inf. Comput.*, **115**, 179–247.

[38] Wong, P. Y. H. and Gibbons, J. (2007) A process-algebraic approach to workflow specification and refinement. In Lumpe, M. and Vanderperren, W. (eds.), *Software Composition*, Lecture Notes in Computer Science, **4829**, pp. 51–65. Springer.

[39] Weidlich, M., Mendling, J., and Weske, M. (2011) Efficient consistency measurement based on behavioral profiles of process models. *IEEE Trans. Software Eng.*, **37**, 410–429.

[40] Weidlich, M., Polyvyanyy, A., Mendling, J., and Weske, M. (2011) Causal behavioural profiles – efficient computation, applications, and evaluation. *Fundamenta Informaticae (FI)*, **-**, –. To appear.