A Classification of BPMN Collaborations Based on Safeness and Soundness Notions

Flavio Corradini, Chiara Muzi, Barbara Re, Francesco Tiezzi

School of Science and Technology, University of Camerino, Italy

Abstract

BPMN standard has a huge uptake in modelling business processes within the same organisation or involving multiple ones. It results that providing a solid foundation to enable BPMN designers to understand their models in a consistent way is becoming more and more important. In our investigation we define and exploit a formal characterisation of the collaborations' semantics, specifically and directly given for BPMN models, to provide their classification. In particular, we refer to collaborations involving processes with arbitrary topology, thus overcoming the well-structuredness limitations. The proposed classification is based on some of the most important correctness properties in the business process domain, namely, safeness and soundness. We prove, with a uniform formal framework, some conjectured and expected results and, most of all, we achieve novel results for BPMN collaborations concerning the relationships between safeness and soundness, and their compositionality, that represent major advances in the state-of-the-art.

Keywords: Business Process Modelling, BPMN Collaboration, Operational Semantics, Safeness, Soundness, Classification.

1. Introduction

Modern organisations recognise the importance of having tools to describe how to behave in order to make sense of their complex reality to better achieve their objective. This is generally reflected in a business process model that is characterised as "*a collection of related and structured activities undertaken by one or more organisations in order to pursue some particular goal. [. . .] Business processes are often interrelated since the execution of a business process often results in the activation of related business processes within the same or other organisa-*

tions" [1]. Up to now, several languages have been proposed to represent business process models. The Object Management Group (OMG) standard Business Process Model and Notation (BPMN) [2] is the most prominent language, since it is widely accepted in both academia and industry. In particular, BPMN collaboration models are used to describe distributed and complex scenarios where multiple participants interact via messages exchange. It results that a BPMN collaboration model gives both a global view on the messages exchange as well as a local view on internal behaviour.

Eventhough widely accepted, BPMN's major drawback is related to the possible misunderstanding of its execution semantics, defined by means of natural text descriptions, sometimes containing misleading information [3]. To overcome this issue, much effort has been devoted to formalise BPMN semantics by means of mapping it to other formal languages. The most relevant attempt is the one to Petri Nets provided by Dijkman et al. [4]. However, models resulting from a mapping inherit constraints given by the target formal language and so far none of them considers BPMN features such as different abstraction levels (i.e., collaboration, process, and sub-process), the asynchronous communication model, and the notion of completion due to different types of end event (i.e., simple, message throwing, and terminate).

Our investigation is based on a formal characterisation of the BPMN semantics specifically given for collaboration models. It is used to formally define a classification of these BPMN collaboration models according to relevant properties of the business process domain. It is worth noticing that our work aims at providing a classification specific for BPMN models. To this aim, our formal semantics is directly defined on BPMN elements. Our intention is to introduce a unique formal framework to allow BPMN designers to achieve a better understanding of their models, and relative properties. This results in a systematic methodological approach to improve the design of BPMN collaborations.

As a distinctive aspect, the proposed semantics supports models with arbitrary topology, including of course also 'well-structured' ones [5]. This is necessary to enable a classification of both structured and unstructured models with respect to specific properties. Our choice to consider model with an arbitrary topology is also motivated by the following reasons. Unstructured models can often be studied via their transformation into their structured version at the cost of increasing the model size [6]. However, this transformation is not always possible [7, 8]. Moreover, looking at the public repository of BPMN models provided by the BPM Academic Initiative (`http://bpmai.org`) we noticed that unstructured models are largely used in practice, especially when the models size is significant (this is wider discussed at the end of the paper).

Regarding the considered properties, our classification relies on a well-known class of properties in the domain of business process management, namely *safeness* [9] and *soundness* [10, 11]. So far, despite the large body of work on this topic, no formal definition of such properties directly given on BPMN is provided. We reconcile in our single framework properties taken into account by different languages, like Petri Nets [12], Workflow Nets [9], and Elementary Nets [13]. Studying different properties in the same framework does not leave any room for ambiguity, and increases the potential for formal reasoning on their relationship. Differently from other formal notations, our framework primitively allows to express important features of the BPMN notation, such as message passing and its impact on soundness.

Hence, the main contribution of the paper is a classification of BPMN collaborations according to relevant properties of the domain. More in detail, we prove that a well-structured collaboration is always safe, but the reverse does not hold. Moreover, well-structuredness implies soundness only at the process level, while there are well-structured collaborations that are not sound. Regarding the relationships between soundness and safeness, we prove that soundness does not imply safeness. Indeed, there are unsafe models that are sound. Similarly, sound models are not necessarily safe.

Moreover, we study safeness and soundness compositionality in the domain of business process modelling, and we show how specific BPMN element, namely terminate event and sub-processes, can move certain BPMN models from one class to another. To illustrate both our formal framework and the considered properties as well as relationships we rely on a running example concerning a travel agency.

The rest of the paper is organised as follows. Sec. 2 provides background notions on BPMN and the considered properties. Sec. 3 introduces a first insight into the obtained results. Sec. 4 introduces the proposed formal framework, Sec. 5 provides the definition of properties, and Sec. 6 makes it clear the relationships between these properties. Sec. 7 presents the study on safeness and soundness compositionality. Finally, Sec. 8 discusses related works, and Sec. 9 concludes the paper.

2. Background

In this section we first provide some basic notions on BPMN elements that can be included in a collaboration diagrams. Then, we present a travel agency scenario, used throughout the paper as a running example.

2.1. Basic Notions on BPMN

Here we do not aim to provide a complete presentation of the standard, but a discussion of the main concepts of BPMN [2] we use in the following. Our choice of the BPMN fragment is driven by practical aspects. Indeed, as shown in [14], even if the BPMN specification is quite wide, only less than 20% of its vocabulary is used regularly in designing business process models.

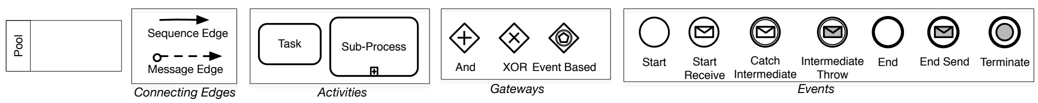


Figure 1: Considered BPMN 2.0 elements.

In the following we report the BPMN elements we consider. They are briefly described below and reported in Fig. 1. **Pools** are used to represent participants or organisations involved in the collaboration, and include details on internal process specifications. **Connecting Edges** are used to connect BPMN elements. *Message Edge* is a dashed connector used to visualise communication flows between organisations, while *Sequence Edge* is a solid connector used to specify the internal flow of the process, thus ordering elements in the same pool. **Activities** are used to represent specific work to be performed within a process. In particular, a *task* is an atomic activity which represents work that can not be interrupted. It can be also used to send and receive messages. A *sub-process* represents work that broken down to a finer level of detail. **Gateways** are used to manage the flow of a process both for parallel activities and choices. Gateways act as either join nodes (merging incoming sequence edges) or split nodes (forking into outgoing sequence edges). Different types of gateways are available. A *XOR gateway* gives the possibility to describe choices. In particular, a XOR-split gateway is used after a decision to fork the flow into branches. When executed, it activates exactly one outgoing edge. A XOR-join gateway acts as a pass-through, meaning that it is activated each time the gateway is reached. An *AND gateway* enables parallel execution flows. An AND-split gateway is used to model the parallel execution of two or more branches, as all outgoing sequence edges are activated simultaneously. An AND-join gateway synchronises the execution of two or more parallel branches, as it waits for all incoming sequence edges to complete before triggering the outgoing flow. An *Event-Based gateway* is used after a decision to fork the flow into branches according to external choice. Its outgoing branches activation depends on taking place of catching events. Basically, such events are in a race condition, where the first event that is triggered wins and disables the other ones. **Events** are used to represent something that can happen. An event can be a *Start Event* representing the point from which a process starts, an *Intermediate Event* representing something that happens during process execution, or an *End Event* representing the process termination. Events are drawn as circles. When an event is source or target of a message edge, it is called *Message Event*. According to the different kinds of message edge connections, we distinguish between: (i) *Start Message Event* is a start event with an incoming message edge; the event element catches a message and starts a process; (ii) *Throw Intermediate Event* is an intermediate event with an outgoing message edge; the event element sends a message; (iii) *Catch Intermediate Event* is an intermediate event with an incoming message edge; the event element receives a message, (iv) *End Message Event* is an end event with an outgoing message edge; the event element sends a message and ends the process. We also refer to a particular type of end event, the *Terminate Event* able to stop and abort the running process.

Finally, a key concept related to the BPMN process execution refers to the notion of *token*. The BPMN standard states that "a token is a theoretical concept that is used as an aid to define the behaviour of a process that is being performed" [2, Sec. 7.1.1]. A token is commonly generated by a start event, traverses the sequence edges of the process and passes through its elements enabling their execution, and it is consumed by an end event when process completes. The distribution of tokens in the process elements is called *marking*, therefore the *process execution* is defined in terms of marking evolution. In the collaboration, the process execution also triggers message flow able to genere messages. We will refer them as message flow token.

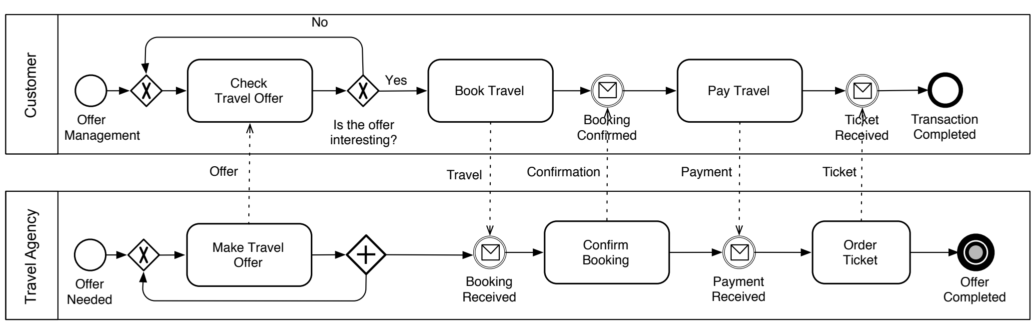


Figure 2: BPMN collaboration model of a travel agency scenario.

2.2. *Travel Agency Collaboration Scenario*

Considering some of the introduced BPMN elements we obtain the travel agency scenario combining in a collaboration model the activities of a Customer and a Travel Agency reported in Fig. 2.

Running Example (1/9). The Travel Agency continuously offers travels to the Customer, until an offer is accepted. If the Customer is interested in one offer, she decides to book the travel and refuses all the other offers that the Travel Agency insistently proposes. As soon as the booking is received by the Travel Agency, it sends back a confirmation message, and asks for the payment of the travel. When this is completed the ticket is sent to the Customer, and the Travel Agency activities end. The processes of the Customer and of the Travel agency are represented inside two *pools*. Considering the Customer pool, from left to right, we have that as soon as the process starts, due to the presence of a *start event*, the Customer checks the travel offer. This is done by executing a *receiving task*. Then, she decides either to book the travel or to wait for other offers, by means of a cycle composed of two XOR *gateways*. After the Customer finds the interesting offer, she books the travel, by sending a message to the Travel Agency by executing a *sending task*, and waits for the booking confirmation. As soon as she receives the booking confirmation, through an *intermediate receiving event*, she pays the travel, receives the ticket from the Agency and her specific works terminate by means of an *end event*. Considering the work of the Travel Agency, as soon as its process starts, it makes travel offers to the Customer, by means of a *sending task*, until an offer is accepted. Thanks to the behaviour of the AND-split combined with the XOR-join in a cycle, it continuously makes offers. At the same time, it proceeds in order to receive a booking via an *intermediate receiving event*. Then, it confirms the booking and sends a notification to the Customer. Finally, after receiving the payment, it orders and sends the ticket, thus completing its activities by means of a *terminate event* which stops and aborts the running process, including the offering of travels.

3. Classification Results

In this section, we informally introduce the considered properties and how our classification advances the state of the art respect to other available classification in the litterature. We also discuss how our framework enables a more precise classification of the BPMN models considering language peculiarities.

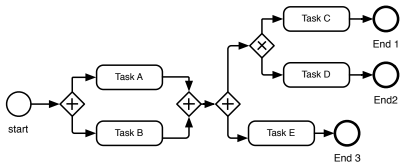


Figure 3: A WS process model.

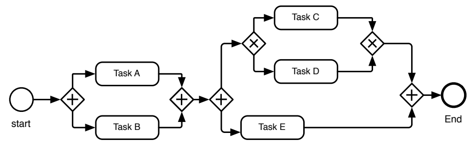


Figure 4: A non WS process model.

3.1. Well-structuredness, Safeness and Soundness for BPMN

Here, we introduce an informal definition of *well-structuredness*, *safeness* and *soundness* for BPMN models, while their formalisation is provided in Section 5. In particular, *well-structuredness* relates to the way the model elements are connected with each other, while *safeness* and *soundness* relate to the way a process model can be executed.

A BPMN proces model is *well-structured*, if for every split gateway there is a corresponding join gateway such that the fragment of the model between the split and the join forms a single-entry-single-exit process fragment (see Def. 4). The notion is inspired by the one defined on WF-Net [5]. As an example, the process in Fig. 3 is the well-structured version of the unstructured process in Fig. 4. It can be extended to process collaborations (see Def. 5) requiring that processes of all involved organisations are well-structured.

A BPMN process model is *safe*[1], if during its execution no more than one token occurs along the same sequence edge (see Def. 7). It is inspired by the Petri Net formalisation where safeness means that the Petri Net does not contain more than one token in all reachable markings [9]. Safeness property naturally extends to process collaborations, considering that the collaboration execution has no more than one token occur on the same sequence edge (see Def. 8).

A BPMN process model is *sound*, if once its execution starts, it is always possible to reach a marking where either (i) each marked end event is marked by at most one token and there are no other tokens around, or (ii) all edges are unmarked (see Def. 10). Also soundness is inspired by the literature that since the mid nineties presents several versions of soundness [9] [10] [11] [15]. The notion can be extended to process collaborations (see Def. 11), reasoning on the whole collaboration execution and requiring also that all sent messages are properly handled (i.e. when sent are properlly received). Finally, at collaboration level we also relax the soundness notion toward the message-relaxed version, inspired by [16],

[1]Notably, the notion of safeness is different from that of safety, since it is a standard term in the BPM literature

allowing pending messages (see Def. 12).

3.2. Advances with respect to already available classifications.

Differently from other classifications reasoning at the process level by means of Workflow Nets [17] [18] and π-calculus [19], our study directly addresses BPMN collaboration models. By relying on a uniform formal framework, we achieved novel results are synthesized in the Euler diagram in Fig. 5, in particular showing that:

(i) all well-structured collaborations are safe, but the reverse does not hold;

(ii) there are well-structured collaborations that are neither sound nor message-relaxed sound;

(iii) there are sound and message-relaxed sound collaborations that are not safe.

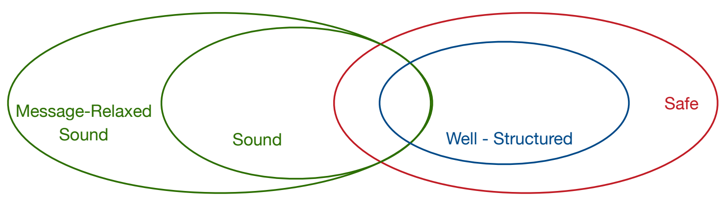


Figure 5: Classification of BPMN collaborations.

Result *(i)* demonstrates that well-structured collaborations represent a subclass of safe collaborations. We also show that such relation is valid at process level, where the classification relaxes the existing results on Workflow Nets, stating that a process model to be safe has to be not only well-structured, but also sound [18].

Result *(ii)* shows that there are well-structured collaborations that are not sound. This is also valid at the process level confirming results provided on Workflow Nets, where well-structuredness implies soundness [20], but relaxing the one obtained in Petri Nets [17], where relaxed soundness and well-structuredness together imply soundness.

Results *(i)* and *(ii)* together confirm limits of well-structuredness as a correctness criterion. Indeed, considering only well-structuredness is too strict, as some safe and sound models that are not well-structured result discarded right from the start.

Result *(iii)* shows that there are sound and message-relaxed sound collaborations that are not safe. This can also be observed at process level resulting in a novel contribution strictly related to the expressiveness of BPMN and its differences with respect to other workflow languages. In fact, Van der Aalst shows that soundness of a Workflow Net is equivalent to liveness and boundedness of the corresponding short-circuited Petri Net [21]. Similarly, in workflow graphs and, equivalently, free-choice Petri Nets, soundness can be characterized in terms of two types of local errors, viz. deadlock and lack of synchronization: a workflow graph is sound if it contains neither a deadlock nor a lack of synchronization [22] [23]. Thus, a sound workflow is always safe. In BPMN instead there are unsafe processes that are sound.

Summing up, result *(i)* together with result *(ii)* and *(iii)* are novel and influenced also the reasoning at process level. This is mainly due to the effects of the behaviour of the terminate event and sub-processes, impacting on the classification of the models, both at the process and collaboration level, as shown in the following.

3.3. Advance in Classifying BPMN Models

Our formalisation considers as first-class citizens BPMN specificities such as: different levels of abstraction (collaboration, process and sub-process levels), asynchronous communication models between pools and the completeness notion able to distinguish the effect of end event from the one of terminate event.

Considering collaboration models, by means of our formalisation we can observe pools that exchange message flow tokens, while in each pool the execution is rendered by the movements of the sequence flow tokens in the process. In this setting, there is a clear difference between the notion of safeness directly defined on BPMN collaborations with respect to that defined on Petri Nets and applied to the Petri Nets resulting from the translation of BPMN collaborations. Safeness of a BPMN collaboration only refers to tokens on the sequence edges of the involved processes, while in its Petri Nets translation it refers to token both on message and sequence edges. Indeed, such distinction is not considered in the available mappings [4] [24], because a message is rendered as a (standard) token in a place. Hence, a safe BPMN collaboration, where the same message is sent more than once (e.g., via a loop), is erroneously considered unsafe by relying on the Petri Nets notion (i.e., 1-boundedness), because enqueued messages are rendered as a place with more than one token. Therefore, the notion of safeness defined for Petri Nets cannot be safely applied as it is to collaboration models. Similarly, regarding the soundness property, we are able to consider different notions of soundness according to the requirements we impose on message queues (e.g., ignoring or not pending messages). Again, due to lack of distinction between message and sequence edges, these fine-grained reasoning are precluded using the current translations from BPMN to Petri Nets.

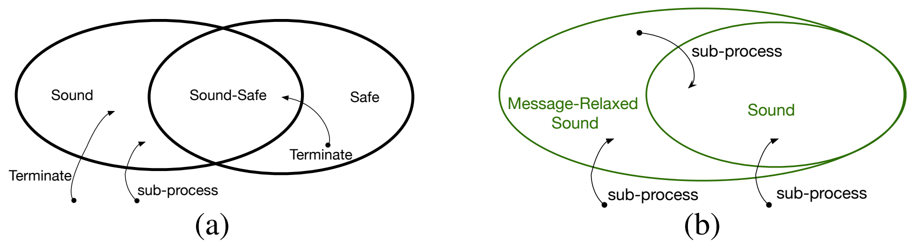


Figure 6: Reasoning at process (a) and collaboration (b) level.

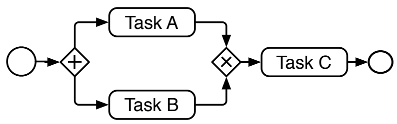


Figure 7: Unsound process.

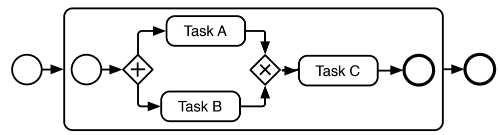


Figure 8: Sound process with an unsound sub-process.

The study of BPMN models via the Petri Nets based frameworks has another limitation concerning the management of the terminate event. Most of the available mappings, such as the ones in [24] and [25], do not consider the terminate event, while in the one provided in [4], terminate events are treated as a special type of error events, which however occur mainly on sub-processes, whose translation assumes safeness. This does not allow reasoning on most models including the terminate event, and more in general on all models including unsafe sub-processes. Nevertheless, given the local nature of Petri Nets transitions, such cancellation patterns are difficult to handle. This is confirmed in [26], stating that modelling a vacuum cleaner, (i.e., a construct to remove all the tokens from a given fragment of a net) is possible but results in a spaghetti-like model.

The ability of our formal framework to properly distinguish sequence flow token and message flow tokens, together with our treatment of the terminate event and sub-processes without any of the restrictions mentioned above, has led us to provide a more precise classification of the BPMN models as synthesised by the Euler diagrams in Fig. 6(a) and Fig. 6(b).

In particular, Fig. 6(a) underlines reasoning that can be done at process level on soundness. Here it emerges how the terminate event can affect model soundness, as using it in place of an end event may render sound a model that was unsound. For example, let us consider the process in Fig. 7; it is a simple process that first runs in parallel Task A and Task B, then executes two times Task C. According to the proposed classification the model is unsound. In fact, there is a marking where the end event has two tokens.

Now, let us consider another model, obtained from the one in Fig. 7 by replacing the end event with a terminate event. The resulting model is sound. This is due to the behaviour of the terminate event that, when reached, removes all tokens in the process. We underline that, although the two models are quite similar, in terms of our classification they result to be significantly different.

Also the use of sub-processes can impact on the satisfaction of the soundness property. Fig. 8 shows a simple process model where the unsound process in Fig. 7 is included in the sub-process. Notably, a sub-process is not syntactic sugar that can be removed via a sort of macro expansion. Indeed, according to the BPMN standard, a sub-process completes only when all the internal tokens are consumed, and then just one token is propagated along the including process. Thus, it results that the model in Fig. 8 is sound. Its behaviour would not correspond to that of the process obtained by flattening it, as the resulting model is unsound. Notice, this reasoning is not affected by safeness and in particular, it cannot be extended to collaborations. In fact, as we discuss in Sec. 7, when we compose two sound processes the resulting collaboration could be either sound or unsound.

Interesting situations also arise when focussing on the collaboration level, as highlighted in Fig. 6(b). Worth to notice is the possibility to transform, with a small change, an unsound collaboration into a sound one.

In Fig. 9, Fig. 10 and Fig. 11 we report a simple example showing the impact of sub-processes. Also in this case the models are rather similar, but according to our classification the result is completely different. The collaboration model in Fig. 9 is both unsound and message-relaxed unsound since when ORG A there is a configuration with two tokens on the end event and a pending message. Now let consider another model obtained by Fig. 9 introducing a sub-process the resulting collaboration in Fig. 10 is unsound and message-relaxed sound, since the use of the sub-process mitigates the causes of message-relaxed unsoundness. In fact there will be only the issue of a pending message, since Task C sends two messages and only one will be consumed by Task D. Differently, Fig. 11 shows that enclosing within a sub-process only the part of the model generating multiple tokens we observe a positive effect on the soundness of the model. The collaboration is both sound and message-relaxed sound.

4. Formal Framework

This section presents our BPMN formalisation. Specifically, we first present the syntax and operational semantics we defined for a relevant subset of BPMN elements. The direct semantics proposed in this paper is inspired by [27], but

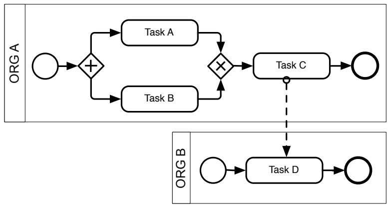


Figure 9: An example of unsound and message-relaxed unsound.

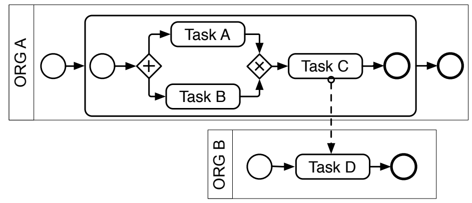


Figure 10: An example of message-relaxed sound and unsound collaboration.

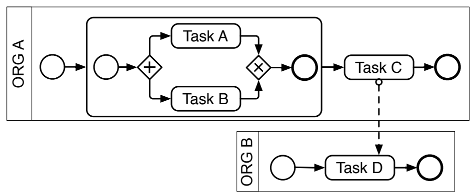


Figure 11: An example of message-relaxed sound and sound collaboration.

its technical definition is significantly different. In particular, configuration states are here defined according to a global perspective, and the formalisation now includes sub-process elements, which were overlooked in the previous semantics definition.

4.1. Syntax of BPMN Collaborations

To enable the formal treatment of collaborations' semantics, we defined a BNF syntax of their model structure (Fig. 12). In the proposed grammar, the non-terminal symbols C and P represent *Collaborations Structure* and *Processes Structure*, respectively. The two syntactic categories directly refer to the corresponding notions in BPMN. The terminal symbols, denoted by the sans serif font, are the typical elements of a BPMN model, i.e. pools, events, tasks, sub-processes and gateways.

$$
\begin{array}{lcl}
C & ::= & \mathsf{pool}(\mathsf{p}, P) \quad | \quad C \parallel C \\
P & ::= & \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \mid \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}) \mid \mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o) \mid \mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_{cmp}) \\
 & | & \mathsf{terminate}(\mathsf{e}_i) \mid \mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_h, \mathsf{e}_{oh})) \\
 & | & \mathsf{andSplit}(\mathsf{e}_i, E_o) \mid \mathsf{xorSplit}(\mathsf{e}_i, E_o) \mid \mathsf{andJoin}(E_i, \mathsf{e}_o) \mid \mathsf{xorJoin}(E_i, \mathsf{e}_o) \\
 & | & \mathsf{task}(\mathsf{e}_i, \mathsf{e}_o) \mid \mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \mid \mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \mid \mathsf{empty}(\mathsf{e}_i, \mathsf{e}_o) \\
 & | & \mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \mid \mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \mid \mathsf{subProc}(\mathsf{e}_i, P, \mathsf{e}_o) \mid P \parallel P
\end{array}
$$

Figure 12: Syntax of BPMN Collaboration Structures.

It is worth noticing that our syntax is too permissive with respect to the BPMN notation, as it allows to write collaborations that cannot be expressed in BPMN. Limiting such expressive power would require to extend the syntax (e.g., by imposing processes having at least one end event), thus complicating the definition of the formal semantics. However, this is not necessary in our work, as we are not proposing an alternative modelling notation, but we are only using a textual representation of BPMN models, which is more manageable for writing operational rules than the graphical notation. Therefore, in our analysis we will only consider terms of the syntax that are derived from BPMN models.

Intuitively, a BPMN collaboration model is rendered in our syntax as a collection of pools and each pool contains a process. More formally, a Collaboration C is a composition, by means of operator $\|$ of pools of the form $\mathsf{pool}(\mathsf{p}, P)$, where: p is the name that uniquely identifies the Pool; P is the Process included in the specific pool, respectively.

In the following, $\mathsf{m} \in \mathbb{M}$ denotes a *message edge*, enabling message exchanges between pairs of participants in the collaboration, while $M \in 2^{\mathbb{M}}$. Moreover, m denotes names uniquely identifying a message edge. We also observe $\mathsf{e} \in \mathbb{E}$ denoting a *sequence edge*, while $E \in 2^{\mathbb{E}}$ a set of edges; we require $|E| > 1$ when it is used in joining and splitting gateways. Similarly, we require that an event-based gateway should contain at least two message events, i.e. $h > 1$ in each $\mathsf{eventBased}$ term. For the convenience of the reader, we refer with e_i to the edge incoming in an element and with e_o to the edge outgoing from an element. In the edge set $\mathbb{E}$ we also include spurious edges denoting the enabled status of start events and the completed status of end events, named *enabling* and *completing* edges, respectively. In particular, we use edge e_{enb}, incoming to a start event, to enable the activation of the process, while e_{cmp} is an edge outgoing from the end events suitable to check the completeness of the process. They are needed to activate sub-processes as well as to check their completion. Moreover, we have that e denotes names uniquely identifying a sequence edge.

The correspondence between the syntax used here to represent a *Process Structure* and the graphical notation of BPMN is as follows.

- $\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o)$ represents a start event that can be activated by means of the enabling edge e_{enb} and that has an outgoing edge e_o.

- $\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp})$ represents an end event with an incoming edge e_i and a completing edge e_{cmp}.

- $\mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o)$ represents a start message event that can be activated by means of the enabling edge e_{enb} as soon as a message m is received and it has outgoing edge e_o.

- $\mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_{cmp})$ represents an end message event with incoming edge e_i, a message m to be sent, and a completing edge e_{cmp}.

- $\mathsf{terminate}(\mathsf{e}_i)$ represents a terminate event with incoming edge e_i.

- $\mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_h, \mathsf{e}_{oh}))$ represents an event based gateway with incoming edge e_i and a list of possible (at least two) message edges, with the related outgoing edges that are enabled by message reception.

- $\mathsf{andSplit}(\mathsf{e}_i, E_o)$ - resp. $\mathsf{xorSplit}(\mathsf{e}_i, E_o)$ - represents an AND - resp. XOR - split gateway with incoming edge e_i and outgoing edges E_o.

- $\mathsf{andJoin}(E_i, \mathsf{e}_o)$ - resp. $\mathsf{xorJoin}(E_i, \mathsf{e}_o)$ - represents an AND - resp. XOR - join gateway with incoming edges E_i and outgoing edge e_o.

- $\mathsf{task}(\mathsf{e}_i, \mathsf{e}_o)$ represents a task with incoming edge e_i and outgoing edge e_o; we can also observe $\mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)$ - resp. $\mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)$ - to consider a task receiving - resp. sending - a message m.

- $\mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)$ (resp. $\mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)$) represents an intermediate receiving - resp. sending - event with an incoming edge e_i and an outgoing edge e_o that are able to receive - resp. sending - a message m.

- $\mathsf{subProc}(\mathsf{e}_i, P, \mathsf{e}_o)$ represents a sub-process element with incoming edge e_i and outgoing edge e_o. When activated, the enclosed sub-process P behaves according to the elements it consists of, including nested sub-process elements (used to describe collaborations with a hierarchical structure).

- $P_1 \parallel P_2$ represents a composition of elements in order to render a process structure in terms of a collection of elements.

Moreover, to simplify the definition of well-structured processes (given later), we include an $empty$ task in our syntax. It permits to connect two gateways with a sequence flow without activities in the middle.

In terms of collaboration the correspondence between the syntax and the graphical notion is as follow.

- $\mathsf{pool}(\mathsf{p}, P)$ represents a pool element with a pool name p. When activated, the enclosed P behaves according to the elements it consists of, including nested process elements.

- $C \parallel C$ represents a composition of elements in order to render a collaboration structure in terms of a collection of elements.

To achieve a compositional definition, each sequence (resp. message) edge of the BPMN model is split in two parts: the part outgoing from the source element and the part incoming into the target element. The two parts are correlated since edge names in the BPMN model are unique. To avoid malformed structure models, we only consider structures in which for each edge labeled by e (resp. m) outgoing from an element, there exists only one corresponding edge labeled by e (resp. m) incoming into another element, and vice versa.

Here in the following we define some auxiliary functions defined on the collaboration and the process structure. Considering BPMN collaborations they may include one or more participants; function $participant(C)$ returns the process structures included in a given collaboration structure. Formally, it is defined as follows.

$$participant(C_1 \parallel C_2) = participant(C_1) \cup participant(C_1)$$
$$participant(\mathsf{pool}(\mathsf{p}, P)) = P$$

Since we also consider process including nested sub-processes to refer to the enabling edges of the start events of the current level we resort to functions $start(P)$.

$$start(P_1 \parallel P_2) = start(P_1) \cup start(P_2)$$
$$start(\mathsf{start}(\mathsf{e}, \mathsf{e}')) = \{\mathsf{e}\} \qquad start(\mathsf{startRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')) = \{\mathsf{e}\}$$
$$start(P) = \emptyset \text{ for any element } P \neq \mathsf{start}(\mathsf{e}, \mathsf{e}') \text{ or } P \neq \mathsf{startRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$$

Notably, we assume that each process/sub-process in the collaboration has only one start event. Function $start(\cdot)$ applied to C will return as many enabling edges as the number of involved participants.

$$start(C_1 \parallel C_2) = start(participant(C_1)) \cup start(participant(C_2))$$
$$start(\mathsf{pool}(\mathsf{p}, P)) = start(P)$$

We similarly define functions $end(P)$ and $end(C)$ on the structure of processes and collaborations in order to refer to end events in the current layer.

$$end(P_1 \parallel P_2) = end(P_1) \cup end(P_2)$$

$$end(\mathsf{endSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}')) = \{\mathsf{e}'\} \qquad end(\mathsf{end}(\mathsf{e}, \mathsf{e}')) = \{\mathsf{e}'\}$$

$$end(P) = \varnothing \text{ for any element } P \neq \mathsf{end}(\mathsf{e}, \mathsf{e}') \text{ or } P \neq \mathsf{endSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$$

Function $end(C)$ on the collaboration structure is defined as follow.

$$end(C_1 \parallel C_2) = end(participant(C_1)) \cup end(participant(C_2))$$

$$end(\mathsf{pool}(\mathsf{p}, P)) = end(P)$$

We also define function $\mathsf{edges}(P)$ to refer the edges in the scope of P.

$$\mathsf{edges}(P_1 \parallel P_2) = \mathsf{edges}(P_1) \cup \mathsf{edges}(P_2)$$

$$\mathsf{edges}(\mathsf{start}(\mathsf{e}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$$

$$\mathsf{edges}(\mathsf{end}(\mathsf{e}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$$

$$\mathsf{edges}(\mathsf{startRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$$

$$\mathsf{edges}(\mathsf{endSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$$

$$\mathsf{edges}(\mathsf{terminate}(\mathsf{e})) = \{\mathsf{e}\}$$

$$\mathsf{edges}(\mathsf{eventBased}(\mathsf{e}, (\mathsf{m}_1, \mathsf{e}'_1), \ldots, (\mathsf{m}_h, \mathsf{e}'_h))) = \{\mathsf{e}, \mathsf{e}'_1, \ldots, \mathsf{e}'_h\}$$

$$\mathsf{edges}(\mathsf{andSplit}(\mathsf{e}, \mathsf{e}'_1, \ldots, \mathsf{e}'_h)) = \{\mathsf{e}, \mathsf{e}'_1, \ldots, \mathsf{e}'_h\}$$

$$\mathsf{edges}(\mathsf{xorSplit}(\mathsf{e}, \mathsf{e}'_1, \ldots, \mathsf{e}'_h)) = \{\mathsf{e}, \mathsf{e}'_1, \ldots, \mathsf{e}'_h\}$$

$$\mathsf{edges}(\mathsf{andJoin}(\mathsf{e}_1, \ldots, \mathsf{e}_h, \mathsf{e}')) = \{\mathsf{e}_1, \ldots, \mathsf{e}_h, \mathsf{e}'\}$$

$$\mathsf{edges}(\mathsf{xorJoin}(\mathsf{e}_1, \ldots, \mathsf{e}_h, \mathsf{e}')) = \{\mathsf{e}_1, \ldots, \mathsf{e}_h, \mathsf{e}'\}$$

$$\mathsf{edges}(\mathsf{task}(\mathsf{e}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$$

$$\mathsf{edges}(\mathsf{taskRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$$

$$\mathsf{edges}(\mathsf{taskSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$$

$$\mathsf{edges}(\mathsf{empty}(\mathsf{e}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$$

$$\mathsf{edges}(\mathsf{interRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$$

$$\mathsf{edges}(\mathsf{interSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$$

$$\mathsf{edges}(\mathsf{subProc}(\mathsf{e}, P, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\} \cup \mathsf{edges}(P)$$

Running Example (2/9). The BPMN model in Fig. 2 is expressed in our syntax as the following collaboration structure (at an unspecified step of execution):

$$\mathsf{pool}(\mathsf{Customer}, P_C) \parallel \mathsf{pool}(\mathsf{TravelAgency}, P_{TA})$$

with P_C expressed as follows (process structure P_{TA} is defined in a similar way) where for simplicity we identify the edges in progressive order e_i (with $i = 0 \ldots 10$):

$$\begin{array}{l}\mathsf{start}(\mathsf{e}_0, \mathsf{e}_1) \\ \parallel \mathsf{xorJoin}(\{\mathsf{e}_1, \mathsf{e}_3\}, \mathsf{e}_2) \parallel \mathsf{taskRcv}(\mathsf{e}_2, \mathsf{Offer}, \mathsf{e}_4) \parallel \mathsf{xorSplit}(\mathsf{e}_4, \{\mathsf{e}_3, \mathsf{e}_5\}) \\ \parallel \mathsf{taskSnd}(\mathsf{e}_5, \mathsf{Travel}, \mathsf{e}_6) \parallel \mathsf{interRcv}(\mathsf{e}_6, \mathsf{Confirmation}, \mathsf{e}_7) \\ \parallel \mathsf{taskSnd}(\mathsf{e}_7, \mathsf{Payment}, \mathsf{e}_8) \parallel \mathsf{interRcv}(\mathsf{e}_8, \mathsf{Ticket}, \mathsf{e}_9) \parallel \mathsf{end}(\mathsf{e}_9, \mathsf{e}_{10})\end{array}$$

Moreover, considering functions we defined on the structure we have: $participant(\mathsf{pool}(\mathsf{Customer}, P_C) \parallel \mathsf{pool}(\mathsf{TravelAgency}, P_{TA})) = \{P_C, P_{TA}\}$, $start(P_C) = \{\mathsf{e}_0\}$, and $end(P_C) = \{\mathsf{e}_{10}\}$. Finally, $\mathsf{edges}(P_C) = \{\mathsf{e}_0, ..., \mathsf{e}_{10}\}$. The others are defined in a similar way. □

Notice, the one-to-one correspondence between the syntax used here to represent a BPMN model and the graphical notation of BPMN, that is exemplified by means of (an excerpt of) our running example in Fig. 2, is also reported in detail in the Appendix A.

4.2. Semantics of BPMN Collaborations

The syntax presented so far permits to describe the mere structure of a collaboration and a process. To describe their semantics we need to enrich it with a notion of execution state, defining the current marking of sequence and message edges. We use *collaboration configuration* and *process configuration* to indicate these stateful descriptions.

Formally, a collaboration configuration has the form $\langle C, \sigma, \delta \rangle$, where: C is a collaboration structure; σ is the part of the execution state at process level, storing for each sequence edge the current number of tokens marking it (notice it refers to the edges included in all the processes of the collaboration), and δ is the part of the execution state at collaboration level, storing for each message edge the current number of message tokens marking it. Moreover, a process configuration has the form $\langle P, \sigma \rangle$, where: P is a process structure; and σ is the execution state at process level. Specifically, a state $\sigma : \mathbb{E} \rightarrow \mathbb{N}$ is a function mapping edges to a number of tokens. The state obtained by updating in the state σ the number of tokens of the edge e to n, written as $\sigma \cdot \{\mathsf{e} \mapsto \mathsf{n}\}$, is defined as follows: $(\sigma \cdot \{\mathsf{e} \mapsto \mathsf{n}\})(\mathsf{e}')$ returns

n if $\mathsf{e}' = \mathsf{e}$, otherwise it returns $\sigma(\mathsf{e}')$. Moreover, a state $\delta : \mathbb{M} \to \mathbb{N}$ is a function mapping message edges to a number of message tokens; so that $\delta(\mathsf{m}) = \mathsf{n}$ means that there are n messages of type m sent by a participant to another that have not been received yet. Update for δ is defined in a way similar to σ's definitions.

Given the notion of configuration, a collaboration is in the *initial state* when each process it includes is in the *initial state*, meaning that the start event of each process must be enabled, i.e. it has a token in its enabling edge, while all other sequence edges (included the enabling edges for the activation of nested sub-processes), and messages edges must be unmarked.

Definition 1 (Initial state of process). *Let $\langle P, \sigma \rangle$ be a process configuration. Predicate $isInit(P, \sigma)$ holds, if $\sigma(start(P)) = 1$, and $\forall\ \mathsf{e} \in \mathsf{edges}(P) \backslash start(P)\ .\ \sigma(\mathsf{e}) = 0$, then process configuration is initial if $isInit(P, \sigma)$ holds.*

Definition 2 (Initial state of collaboration). *Let $\langle C, \sigma, \delta \rangle$ be a collaboration configuration. Predicate $isInit(C, \sigma, \delta)$ holds, if $\forall\ P \in participant(C)$ we have that $isInit(P, \sigma)$, and $\forall\ \mathsf{m} \in \mathbb{M}\ .\ \delta(\mathsf{m}) = 0$, then a collaboration configuration is initial if $isInit(C, \sigma, \delta)$ holds.*

Running Example (3/9). The initial configuration of the collaboration in Fig. 2 is as follows. Given $participant(C) = \{P_C, P_{TA}\}$, we have that $\langle P_C, \sigma \rangle$, $\sigma(\mathsf{e}_0) = 1$ $\sigma(\mathsf{e}_i) = 0\ \ \forall \mathsf{e}_i$ with $i = 1 \ldots 10$, and $\langle P_{TA}, \sigma \rangle$, $\sigma(\mathsf{e}_{11}) = 1$ and $\sigma(\mathsf{e}_j) = 0\ \ \forall \mathsf{e}_j$ with $j = 12..22$. We also have that $\delta(\mathsf{Offer}, \mathsf{Confirmation}, \mathsf{Ticket}, \mathsf{Travel}, \mathsf{Payment}) = 0$. □

The operational semantics is defined by means of a *labelled transition system* (LTS) on collaboration configuration and formalises the execution of message marking evolution according to the process evolution. Its definition relies on an auxiliary transition relation on the behaviour of process.

The auxiliary transition relation is a triple $\langle \mathcal{P}, \mathcal{A}, \to \rangle$ where: $\mathcal{P}$, ranged over by $\langle P, \sigma \rangle$, is a set of process configurations; $\mathcal{A}$, ranged over by α, is a set of *labels* (of transitions that process configurations can perform); and $\to \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$ is a *transition relation*. We will write $\langle P, \sigma \rangle \xrightarrow{\alpha} \langle P, \sigma' \rangle$ to indicate that $(\langle P, \sigma \rangle, \alpha, \langle P, \sigma' \rangle) \in \to$ and say that process configuration $\langle P, \sigma \rangle$ performs a transition labelled by α to become process configuration $\langle P, \sigma' \rangle$. Since process execution only affects the current states, and not the process structure, for the sake of readability we omit the structure from the target configuration of the transition.

Thus, a transition $\langle P, \sigma\rangle \xrightarrow{\alpha} \langle P, \sigma'\rangle$ is written as $\langle P, \sigma\rangle \xrightarrow{\alpha} \sigma'$. The labels used by this transition relation are generated by the following production rules.

$$(Actions)\ \alpha \ ::= \ \tau \quad | \quad !\mathsf{m} \quad | \quad ?\mathsf{m} \qquad\qquad (Internal\ Actions)\ \tau \ ::= \ \epsilon \ | \quad kill$$

The meaning of labels is as follows. Label τ denotes an action internal to the process, while $!\mathsf{m}$ and $?\mathsf{m}$ denote sending and receiving actions, respectively. The meaning of internal actions is as follows: ϵ denotes the movement of a token through the process unless the termination action denoted by $kill$.

The transition relation over process configurations formalises the execution of a process; it is defined by the rules in Fig. 13. Before commenting on the rules, we introduce the auxiliary functions they exploit. Specifically, function $inc : \mathbb{S}\times\mathbb{E} \rightarrow \mathbb{S}$ (resp. $dec : \mathbb{S} \times \mathbb{E} \rightarrow \mathbb{S}$), where $\mathbb{S}$ is the set of states, allows updating a state by incrementing (resp. decrementing) by one the number of tokens marking an edge in the state. Formally, they are defined as follows: $inc(\sigma, \mathsf{e}) = \sigma \cdot \{\mathsf{e} \mapsto \sigma(\mathsf{e}) + 1\}$ and $dec(\sigma, \mathsf{e}) = \sigma \cdot \{\mathsf{e} \mapsto \sigma(\mathsf{e}) - 1\}$. These functions extend in a natural ways to sets of edges as follows: $inc(\sigma, \emptyset) = \sigma$ and $inc(\sigma, \{\mathsf{e}\} \cup E)) = inc(inc(\sigma, \mathsf{e}), E)$; the cases for dec are similar. As usual, the update function for δ are defined in a way similar to σ's definitions. We also use the function $zero : \mathbb{S} \times \mathbb{E} \rightarrow \mathbb{S}$ that allows updating a state by setting to zero the number of tokens marking an edge in the state. Formally, it is defined as follows: $zero(\sigma, \mathsf{e}) = \sigma \cdot \{\mathsf{e} \mapsto 0\}$. Also in this case the function extends in a natural ways to sets of edges as follows: $zero(\sigma, \emptyset) = \sigma$ and $zero(\sigma, \{\mathsf{e}\} \cup E)) = zero(zero(\sigma, \mathsf{e}), E)$.

To check the completion of a sub-process we exploit the boolean predicate $completed(P, \sigma)$. It is defined according to the prescriptions of the BPMN standard, which states that "a sub-process instance completes when there are no more tokens in the Sub-Process and none of its Activities is still active" [2, pp. 426, 431]. Thus, the sub-process completion can be formalised as follows.

Definition 3. *Let P be a process included in the sub-process, having the form $\mathsf{end}(\mathsf{e}, \mathsf{e}') \parallel P'$ or $\mathsf{endSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}') \parallel P'$, or $\mathsf{terminate}(\mathsf{e}) \parallel P'$ the predicate $completed(P, \sigma)$ is defined as*

$$\sigma(\mathsf{e}') > 0 \wedge \ \sigma(\mathsf{e}) = 0 \wedge \ isZero(P', \sigma)$$

where $isZero(\cdot)$ is inductively defined on the structure of its first argument as follows:

- $isZero(\mathsf{start}(\mathsf{e}, \mathsf{e}'), \sigma)$ *if* $\sigma(\mathsf{e}) = 0$ *and* $\sigma(\mathsf{e}') = 0$*;*

- $isZero(\mathsf{end}(\mathsf{e}, \mathsf{e}'), \sigma)$ *if* $\sigma(\mathsf{e}) = 0$*;*

- $isZero(\mathsf{startRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}'))$ *if* $\sigma(\mathsf{e}) = 0$ *and* $\sigma(\mathsf{e}') = 0$*;*

- $isZero(\mathsf{endSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}'))$ *if* $\sigma(\mathsf{e}) = 0$*;*

- $isZero(\mathsf{terminate}(\mathsf{e}), \sigma)$ *if* $\sigma(\mathsf{e}) = 0$*;*

- $isZero(\mathsf{eventBased}(\mathsf{e}, (\mathsf{m}_1, \mathsf{e}'_1), \ldots, (\mathsf{m}_k, \mathsf{e}'_h)), \sigma)$ *if* $\sigma(\mathsf{e}) = 0$ *and* $\forall j \in \{1, ..., h\} . \sigma(\mathsf{e}'_j) = 0$*;*

- $isZero(\mathsf{andSplit}(\mathsf{e}, E), \sigma)$ *if* $\sigma(\mathsf{e}) = 0$ *and* $\forall \mathsf{e}' \in E . \sigma(\mathsf{e}') = 0$*;*

- $isZero(\mathsf{xorSplit}(\mathsf{e}, E), \sigma)$ *if* $\sigma(\mathsf{e}) = 0$ *and* $\forall \mathsf{e}' \in E . \sigma(\mathsf{e}') = 0$*;*

- $isZero(\mathsf{andJoin}(E, \mathsf{e}), \sigma)$ *if* $\forall \mathsf{e}' \in E . \sigma(\mathsf{e}') = 0$ *and* $\sigma(\mathsf{e}) = 0$*;*

- $isZero(\mathsf{xorJoin}(E, \mathsf{e}), \sigma)$ *if* $\forall \mathsf{e}' \in E . \sigma(\mathsf{e}') = 0$ *and* $\sigma(\mathsf{e}) = 0$*;*

- $isZero(\mathsf{task}(\mathsf{e}, \mathsf{e}'), \sigma)$ *if* $\sigma(\mathsf{e}) = 0$ *and* $\sigma(\mathsf{e}') = 0$*;*

- $isZero(\mathsf{taskRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}'), \sigma)$ *if* $\sigma(\mathsf{e}) = 0$ *and* $\sigma(\mathsf{e}') = 0$*;*

- $isZero(\mathsf{taskSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}'), \sigma)$ *if* $\sigma(\mathsf{e}) = 0$ *and* $\sigma(\mathsf{e}') = 0$*;*

- $isZero(\mathsf{empty}(\mathsf{e}, \mathsf{e}'), \sigma)$ *if* $\sigma(\mathsf{e}) = 0$ *and* $\sigma(\mathsf{e}') = 0$*;*

- $isZero(\mathsf{interRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}'), \sigma\rangle)$ *if* $\sigma(\mathsf{e}) = 0$ *and* $\sigma(\mathsf{e}') = 0$*;*

- $isZero(\mathsf{interSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}'), \sigma)$ *if* $\sigma(\mathsf{e}) = 0$ *and* $\sigma(\mathsf{e}') = 0$*;*

- $isZero(\mathsf{subProc}(\mathsf{e}, P, \mathsf{e}'), \sigma)$ *if* $\sigma(\mathsf{e}) = \sigma(\mathsf{e}') = 0$ *and* $\forall \mathsf{e}'' \in \mathsf{edges}(P) . \sigma(\mathsf{e}'') = 0$*;*

- $isZero(P_1 \| P_2, \sigma)$ *if* $isZero(P_1, \sigma)$ *and* $isZero(P_2, \sigma)$.

Notably, the completion of a process does not depend on the exchanged messages, and it is defined considering the arbitrary topology of the model, which hence may have one or more end events with possibly more than one token in the completing edges.

Finally, we use the function $marked(\sigma, E)$ to refer to the set of edges in E with at least one token, which is defined as follows:

$$marked(\sigma, \{\mathsf{e}\} \cup E) = \begin{cases} \{\mathsf{e}\} \cup marked(\sigma, E) & \text{if } \sigma(\mathsf{e}) > 0; \\ \quad marked(\sigma, E) & \text{otherwise.} \end{cases}$$

$$marked(\sigma, \emptyset) = \emptyset.$$

We now briefly comment on the operational rules in Fig. 13. Rule $P\text{-}Start$ starts the execution of a process/(sub-)process when it has been activated (i.e., the enabling edge e is marked). The effect of the rule is to increment the number of tokens in the edge outgoing from the start event. Rule $P\text{-}End$ is enabled when there is at least one token in the incoming edge of the end event, which is then moved to the completing edge. Rule $P\text{-}StartRcv$ start the execution of a process when it is in its initial state. The effect of the rule is to increment the number of tokens in the edge outgoing from the start event and remove the token from the enabling edge. A label corresponding to the consumption of a message is observed. Rule $P\text{-}EndSnd$ is enabled when there is at least a token in the incoming edge of the end event, which is then moved to the completing edge. At the same time a label corresponding to the production of a message is observed. Rule $P\text{-}Terminate$ starts when there is at least one token in the incoming edge of the terminate event, which is then removed. Rule $P\text{-}EventG$ is activated when there is a token in the incoming edge and there is a message m_j to be consumed, so that the application of the rule moves the token from the incoming edge to the outgoing edge corresponding to the received message. A label corresponding to the consumption of a message is observed. Rule $P\text{-}AndSplit$ is applied when there is at least one token in the incoming edge of an AND split gateway; as result of its application the rule decrements the number of tokens in the incoming edge and increments that in each outgoing edge. Rule $P\text{-}XorSplit$ is applied when a token is available in the incoming edge of a XOR split gateway, the rule decrements the token in the incoming edge and increment the token in one of the outgoing edges, non-deterministically chosen. Rule $P\text{-}AndJoin$ decrements the tokens in each incoming edge and increments the number of tokens of the outgoing edge, when each incoming edge has at least one token. Rule $P\text{-}XorJoin$ is activated every time there is a token in one of the incoming edges, which is then moved to the outgoing edge. Rule $P\text{-}Task$ deals with simple tasks, acting as a pass through. It is activated only when there is a token in the incoming edge, which is then moved to the outgoing edge. Rule $P\text{-}TaskRcv$ is activated when there is a token in the incoming edge and a label corresponding to the consumption of a message is observed. Similarly, rule $P\text{-}TaskSnd$, instead of consuming, send a message before moving the token to the outgoing edge. A label corresponding to the production of a message is observed. Rule P-*InterRcv* (resp. P-*InterSnd*) follows the same behaviour of rule P-*TaskRcv* (resp. P-*TaskSnd*). Rule P-*Empty* simply propagates tokens, it acts as a pass through. Rules P-*SubProcStart*, P-*SubProcEvolution*, P-*SubProcEnd* and P-*SubProcKill* deal with the behaviour of a sub-process element. The former rule is activated only when there is a token in the incoming edge of the sub-process, which is then moved to the enabling edge of the start event in the sub-process body. Then, the sub-process behaves according to the behaviour of the elements it contains according to the rules P-*SubProcEvolution*. When the sub-process completes the rule P-*SubProcEnd* is activated. It removes all the tokens from the sequence edges of the sub-process body[2], and adds a token to the outgoing edge of the sub-process. Rule P-*SubProcKill* deals with a sub-process element observing a killing action in its behaviour due to an occurrence of Terminate end event. This rule is activated only when there is a token in the incoming edge of termination event by the rule P-*Terminate*. Then, the sub-process stops its internal behaviours and passes the control to the upper layer, indeed the rule removes all the tokens in the sub-process and adds a token to the outgoing edge of the sub-process. Rule P-*Kill* deal with the propagation of killing action in the scope of P and rule P-*Int* deal with interleaving in a standard way for process elements. Notice that we do not need symmetric versions of the last two rules, as we identify processes up to commutativity and associativity of process collection.

[2] Actually, due to the completion definition, only the completing edges of the end events within the sub-process body need to be set to zero.

$$\langle \mathsf{start}(\mathsf{e},\mathsf{e}'),\sigma\rangle \xrightarrow{\epsilon} inc(dec(\sigma,\mathsf{e}),\mathsf{e}') \quad \sigma(\mathsf{e})>0 \qquad (P\text{-}Start)$$

$$\langle \mathsf{end}(\mathsf{e},\mathsf{e}'),\sigma\rangle \xrightarrow{\epsilon} inc(dec(\sigma,\mathsf{e}),\mathsf{e}') \quad \sigma(\mathsf{e})>0 \qquad (P\text{-}End)$$

$$\langle \mathsf{startRcv}(\mathsf{e},\mathsf{m},\mathsf{e}'),\sigma\rangle \xrightarrow{?\mathsf{m}} inc(dec(\sigma,\mathsf{e}),\mathsf{e}') \quad \sigma(\mathsf{e})>0 \qquad (P\text{-}StartRcv)$$

$$\langle \mathsf{endSnd}(\mathsf{e},\mathsf{m},\mathsf{e}'),\sigma\rangle \xrightarrow{!\mathsf{m}} inc(dec(\sigma,\mathsf{e}),\mathsf{e}') \quad \sigma(\mathsf{e})>0 \qquad (P\text{-}EndSnd)$$

$$\langle \mathsf{terminate}(\mathsf{e}),\sigma\rangle \xrightarrow{kill} dec(\sigma,\mathsf{e}) \quad \sigma(\mathsf{e})>0 \qquad (P\text{-}Terminate)$$

$$\langle \mathsf{eventBased}(\mathsf{e},(\mathsf{m}_1,\mathsf{e}'_1),\ldots,(\mathsf{m}_h,\mathsf{e}'_h)),\sigma\rangle \xrightarrow{?\mathsf{m}_j} inc(dec(\sigma,\mathsf{e}),\mathsf{e}'_j) \quad \sigma(\mathsf{e})>0, 1\leqslant j\leqslant h \qquad (P\text{-}EventG)$$

$$\langle \mathsf{andSplit}(\mathsf{e},E),\sigma\rangle \xrightarrow{\epsilon} inc(dec(\sigma,\mathsf{e}),E) \quad \sigma(\mathsf{e})>0 \qquad (P\text{-}AndSplit)$$

$$\langle \mathsf{xorSplit}(\mathsf{e},\{\mathsf{e}'\}\cup E),\sigma\rangle \xrightarrow{\epsilon} inc(dec(\sigma,\mathsf{e}),\mathsf{e}') \quad \sigma(\mathsf{e})>0 \qquad (P\text{-}XorSplit)$$

$$\langle \mathsf{andJoin}(E,\mathsf{e}),\sigma\rangle \xrightarrow{\epsilon} inc(dec(\sigma,E),\mathsf{e}) \quad \forall \mathsf{e}'\in E\,.\,\sigma(\mathsf{e}')>0 \qquad (P\text{-}AndJoin)$$

$$\langle \mathsf{xorJoin}(\{\mathsf{e}\}\cup E,\mathsf{e}'),\sigma\rangle \xrightarrow{\epsilon} inc(dec(\sigma,\mathsf{e}),\mathsf{e}') \quad \sigma(\mathsf{e})>0 \qquad (P\text{-}XorJoin)$$

$$\langle \mathsf{task}(\mathsf{e},\mathsf{e}'),\sigma\rangle \xrightarrow{\epsilon} inc(dec(\sigma,\mathsf{e}),\mathsf{e}') \quad \sigma(\mathsf{e})>0 \qquad (P\text{-}Task)$$

$$\langle \mathsf{taskRcv}(\mathsf{e},\mathsf{m},\mathsf{e}'),\sigma\rangle \xrightarrow{?\mathsf{m}} inc(dec(\sigma,\mathsf{e}),\mathsf{e}') \quad \sigma(\mathsf{e})>0 \qquad (P\text{-}TaskRcv)$$

$$\langle \mathsf{taskSnd}(\mathsf{e},\mathsf{m},\mathsf{e}'),\sigma\rangle \xrightarrow{!\mathsf{m}} inc(dec(\sigma,\mathsf{e}),\mathsf{e}') \quad \sigma(\mathsf{e})>0 \qquad (P\text{-}TaskSnd)$$

$$\langle \mathsf{interRcv}(\mathsf{e},\mathsf{m},\mathsf{e}'),\sigma\rangle \xrightarrow{?\mathsf{m}} inc(dec(\sigma,\mathsf{e}),\mathsf{e}') \quad \sigma(\mathsf{e})>0 \qquad (P\text{-}InterRcv)$$

$$\langle \mathsf{interSnd}(\mathsf{e},\mathsf{m},\mathsf{e}'),\sigma\rangle \xrightarrow{!\mathsf{m}} inc(dec(\sigma,\mathsf{e}),\mathsf{e}') \quad \sigma(\mathsf{e})>0 \qquad (P\text{-}InterSnd)$$

$$\langle \mathsf{empty}(\mathsf{e},\mathsf{e}'),\sigma\rangle \xrightarrow{\epsilon} inc(dec(\sigma,\mathsf{e}),\mathsf{e}') \quad \sigma(\mathsf{e})>0 \qquad (P\text{-}Empty)$$

$$\langle \mathsf{subProc}(\mathsf{e},P,\mathsf{e}'),\sigma\rangle \xrightarrow{\epsilon} inc(dec(\sigma,\mathsf{e}),start(P)) \quad \sigma(\mathsf{e})>0,\ completed(P,\sigma) \qquad (P\text{-}SubProcStart)$$

$$\frac{\langle P,\sigma\rangle \xrightarrow{\alpha} \sigma'}{\langle \mathsf{subProc}(\mathsf{e},P,\mathsf{e}'),\sigma\rangle \xrightarrow{\alpha} \sigma'} \quad (P\text{-}SubProcEvolution)$$

$$\langle \mathsf{subProc}(\mathsf{e},P,\mathsf{e}'),\sigma\rangle \xrightarrow{\epsilon} inc(zero(\sigma,end(P)),\mathsf{e}') \quad completed(P,\sigma) \qquad (P\text{-}SubProcEnd)$$

$$\frac{\langle P,\sigma\rangle \xrightarrow{kill} \sigma'}{\langle \mathsf{subProc}(\mathsf{e},P,\mathsf{e}'),\sigma\rangle \xrightarrow{kill} inc(zero(\sigma',\mathsf{edges}(P)),\mathsf{e}')} \quad (P\text{-}SubProcKill)$$

$$\frac{\langle P_1,\sigma\rangle \xrightarrow{kill} \sigma'}{\langle P_1 \parallel P_2,\sigma\rangle \xrightarrow{kill} zero(\sigma',\mathsf{edges}(P_1\parallel P_2))} \; (P\text{-}Kill) \qquad \frac{\langle P_1,\sigma\rangle \xrightarrow{\alpha} \sigma' \quad \alpha\neq kill}{\langle P_1 \parallel P_2,\sigma\rangle \xrightarrow{\alpha} \sigma'} \; (P\text{-}Int)$$

Figure 13: BPMN Semantics - Process Level.

Now, the labelled transition relation on collaboration configurations formalises the execution of message marking evolution according to process evolution. In the case of collaborations, this is a triple $\langle \mathcal{C}, \mathcal{A}, \rightarrow \rangle$ where: $\mathcal{C}$, ranged over by $\langle C, \sigma, \delta \rangle$, is a set of collaboration configurations; $\mathcal{A}$, ranged over by α, is a set of *labels* (of transitions that collaboration configurations can perform as well as the process configuration); and $\rightarrow \subseteq \mathcal{C} \times \mathcal{A} \times \mathcal{C}$ is a *transition relation*. We will write $\langle C, \sigma, \delta \rangle \xrightarrow{\alpha} \langle C, \sigma', \delta' \rangle$ to indicate that $(\langle C, \sigma, \delta \rangle, \alpha, \langle C, \sigma', \delta' \rangle) \in \rightarrow$ and say that collaboration configuration $\langle C, \sigma, \delta \rangle$ performs transition labelled by α to become collaboration configuration $\langle C, \sigma', \delta' \rangle$. Since collaboration execution only affects the current states, and not the collaboration structure, for the sake of readability we omit the structure from the target configuration of the transition. Thus, a transition $\langle C, \sigma, \delta \rangle \xrightarrow{\alpha} \langle C, \sigma', \delta' \rangle$ is written as $\langle C, \sigma, \delta \rangle \xrightarrow{\alpha} \langle \sigma', \delta' \rangle$. We recall α are the following: label τ denotes an action internal to the process, while $!\mathsf{m}$ and $?\mathsf{m}$ denote sending and receiving actions, respectively. The rules related to the collaboration are defined in Fig. 14

$$\frac{\langle P, \sigma \rangle \xrightarrow{\tau} \sigma'}{\langle \mathsf{pool}(\mathsf{p}, P), \sigma, \delta \rangle \xrightarrow{\tau} \langle \sigma', \delta \rangle} \quad (C\text{-}Internal)$$

$$\frac{\langle P, \sigma \rangle \xrightarrow{?\mathsf{m}} \sigma' \quad \delta(\mathsf{m}) > 0}{\langle \mathsf{pool}(\mathsf{p}, P), \sigma, \delta \rangle \xrightarrow{?\mathsf{m}} \langle \sigma', dec(\delta, \mathsf{m}) \rangle} \quad (C\text{-}Receive)$$

$$\frac{\langle P, \sigma \rangle \xrightarrow{!\mathsf{m}} \sigma'}{\langle \mathsf{pool}(\mathsf{p}, P), \sigma, \delta \rangle \xrightarrow{!\mathsf{m}} \langle \sigma', inc(\delta, \mathsf{m}) \rangle} \quad (C\text{-}Deliver)$$

$$\frac{\langle C_1, \sigma, \delta \rangle \xrightarrow{\alpha} \langle \sigma', \delta' \rangle}{\langle C_1 \parallel C_2, \sigma, \delta \rangle \xrightarrow{\alpha} \langle \sigma', \delta' \rangle} \quad (C\text{-}Int)$$

Figure 14: BPMN Semantics - Collaboration Level.

The first three rules allow a single pool, representing organisation p, to evolve according to the evolution of its enclosed process P. In particular, if P performs an internal action, rule *C-Internal*, or a receiving/delivery action, rule *C-Receive*/*C-Deliver*, the pool performs the corresponding action at collaboration layer. Notably, rule *C-Receive* can be applied only if there is at least one message available (premise $\delta(\mathsf{m}) > 0$); of course, one token is consumed by this transition. Recall indeed that at process label $?\mathsf{m}$ just indicates the willingness of a process to consume a received message, regardless the actual presence of messages. Moreover, when a process performs a sending action, represented by a transition labelled by $!\mathsf{m}$, such message is delivered to the receiving organization by applying rule *C-Deliver*. The resulting transition has the effect of increasing the number of tokens in the message edge m. Rule *C-Int* permits to interleave the execution of actions performed by pools of the same collaboration, so that if a part of a larger collaboration evolves, the whole collaboration evolves accordingly. Notice that we do not need symmetric versions of rule *C-Int*, as we identify collaborations up to commutativity and associativity of pools collection.

5. Properties of BPMN Collaborations

In this section we provide a rigorous characterisation, with respect to the BPMN formalisation presented so far, of the key properties studied in this work: well-structuredness, safeness and soundness. We characterise these properties both at process and collaboration levels.

5.1. Well-Structured BPMN Collaborations

The standard BPMN allows process models to have almost any topology. However, it is often desirable that models abide some structural rules. In this respect, a well-known property of a process model is that of *well-structuredness*. In this paper we have been inspired by the definition of well-structuredness given by Kiepuszewski et al. [5]. Such a definition was given on workflow models and it is not expressive enough for BPMN, so we extend it to well-structured collaborations including all the elements defined in our semantics (i.e. not only based element included in workflow models but also event based gateway and sub-processes).

Before providing a formal characterisation of well-structured BPMN processes and collaborations, we need to introduce some auxiliary definitions. In particular, we inductively define functions $in(P)$ and $out(P)$, which determine the incoming and outgoing sequence edges of a process element P as follows:

$$
\begin{array}{ll}
in(\mathsf{start}(\mathsf{e},\mathsf{e}')) = \varnothing & out(\mathsf{start}(\mathsf{e},\mathsf{e}')) = \{\mathsf{e}'\} \\
in(\mathsf{end}(\mathsf{e},\mathsf{e}')) = \{\mathsf{e}\} & out(\mathsf{end}(\mathsf{e},\mathsf{e}')) = \varnothing \\
in(\mathsf{startRcv}(\mathsf{e},\mathsf{m},\mathsf{e}')) = \varnothing & out(\mathsf{startRcv}(\mathsf{e},\mathsf{m},\mathsf{e}')) = \{\mathsf{e}'\} \\
in(\mathsf{endSnd}(\mathsf{e},\mathsf{m},\mathsf{e}')) = \{\mathsf{e}\} & out(\mathsf{endSnd}(\mathsf{e},\mathsf{m},\mathsf{e}')) = \varnothing \\
in(\mathsf{terminate}(\mathsf{e})) = \{\mathsf{e}\} & out(\mathsf{terminate}(\mathsf{e})) = \varnothing \\
in(\mathsf{andSplit}(\mathsf{e},E)) = \{\mathsf{e}\} & out(\mathsf{andSplit}(\mathsf{e},E)) = E \\
in(\mathsf{xorSplit}(\mathsf{e},E)) = \{\mathsf{e}\} & out(\mathsf{xorSplit}(\mathsf{e},E)) = E \\
in(\mathsf{andJoin}(E,\mathsf{e}')) = E & out(\mathsf{andJoin}(E,\mathsf{e}')) = \{\mathsf{e}'\} \\
in(\mathsf{xorJoin}(E,\mathsf{e}')) = E & out(\mathsf{xorJoin}(E,\mathsf{e}')) = \{\mathsf{e}'\} \\
in(\mathsf{eventBased}(\mathsf{e},(\mathsf{m}_1,\mathsf{e}'_1),\ldots,(\mathsf{m}_h,\mathsf{e}'_h))) & out(\mathsf{eventBased}(\mathsf{e},(\mathsf{m}_1,\mathsf{e}'_1),\ldots,(\mathsf{m}_h,\mathsf{e}'_h))) \\
\qquad\qquad = \{\mathsf{e}\} & \qquad\qquad = \{\mathsf{e}'_j\} \quad with \quad 1 < j < h \\
in(\mathsf{task}(\mathsf{e},\mathsf{e})) = \{\mathsf{e}\} & out(\mathsf{task}(\mathsf{e},\mathsf{e}')) = \{\mathsf{e}'\} \\
in(\mathsf{taskRcv}(\mathsf{e},\mathsf{m},\mathsf{e}')) = \{\mathsf{e}\} & out(\mathsf{taskRcv}(\mathsf{e},\mathsf{m},\mathsf{e}')) = \{\mathsf{e}'\} \\
in(\mathsf{taskSnd}(\mathsf{e},\mathsf{m},\mathsf{e})) = \{\mathsf{e}\} & out(\mathsf{taskSnd}(\mathsf{e},\mathsf{m},\mathsf{e}')) = \{\mathsf{e}'\} \\
in(\mathsf{empty}(\mathsf{e},\mathsf{e}')) = \{\mathsf{e}\} & out(\mathsf{empty}(\mathsf{e},\mathsf{e}')) = \{\mathsf{e}'\} \\
in(\mathsf{interRcv}(\mathsf{e},\mathsf{m},\mathsf{e}')) = \{\mathsf{e}\} & out(\mathsf{interRcv}(\mathsf{e},\mathsf{m},\mathsf{e}')) = \{\mathsf{e}'\} \\
in(\mathsf{interSnd}(\mathsf{e},\mathsf{m},\mathsf{e}')) = \{\mathsf{e}\} & out(\mathsf{interSnd}(\mathsf{e},\mathsf{m},\mathsf{e}')) = \{\mathsf{e}'\} \\
in(\mathsf{subProc}(\mathsf{e},P_1,\mathsf{e}')) = \{\mathsf{e}\} & out(\mathsf{subProc}(\mathsf{e},P_1,\mathsf{e}')) = \{\mathsf{e}'\} \\
in(P_1 \parallel P_2) = (in(P_1) \cup in(P_2)) & out(P_1 \parallel P_2) = (out(P_1) \cup out(P_2)) \\
\qquad\qquad \setminus (out(P_1) \cup out(P_2)) & \qquad\qquad \setminus (in(P_1) \cup in(P_2))
\end{array}
$$

Moreover, to simplify the definition of well-structuredness for processes, we also provide the definition of well-structured core by means of the boolean predicate $isWSCore(\cdot)$.

Definition 4 (Well-structured processes). *A process P is* well-structured *(WS) if P has one of the following forms:*

$$\mathsf{start}(\mathsf{e},\mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'',\mathsf{e}''') \tag{1}$$
$$\mathsf{start}(\mathsf{e},\mathsf{e}') \parallel P' \parallel \mathsf{terminate}(\mathsf{e}'') \tag{2}$$
$$\mathsf{start}(\mathsf{e},\mathsf{e}') \parallel P' \parallel \mathsf{endSnd}(\mathsf{e}'',\mathsf{m},\mathsf{e}''') \tag{3}$$
$$\mathsf{startRcv}(\mathsf{e},\mathsf{m},\mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'',\mathsf{e}''') \tag{4}$$
$$\mathsf{startRcv}(\mathsf{e},\mathsf{m},\mathsf{e}') \parallel P' \parallel \mathsf{terminate}(\mathsf{e}'') \tag{5}$$
$$\mathsf{startRcv}(\mathsf{e},\mathsf{m},\mathsf{e}') \parallel P' \parallel \mathsf{endSnd}(\mathsf{e}'',\mathsf{m},\mathsf{e}''') \tag{6}$$

where $in(P') = \{\mathsf{e}'\}$, $out(P') = \{\mathsf{e}''\}$, and $isWSCore(P')$.
$isWSCore(\cdot)$ is inductively defined on the structure of its first argument as follows:

1. $isWSCore(\mathsf{task}(\mathsf{e},\mathsf{e}'))$;
2. $isWSCore(\mathsf{taskRcv}(\mathsf{e},\mathsf{m},\mathsf{e}'))$;
3. $isWSCore(\mathsf{taskSnd}(\mathsf{e},\mathsf{m},\mathsf{e}'))$;
4. $isWSCore(\mathsf{empty}(\mathsf{e},\mathsf{e}'))$;
5. $isWSCore(\mathsf{interRcv}(\mathsf{e},\mathsf{m},\mathsf{e}'))$;
6. $isWSCore(\mathsf{interSnd}(\mathsf{e},\mathsf{m},\mathsf{e}'))$;

$$\frac{\forall j \in [1..n] \;\; isWSCore(P_j), \;\; in(P_j) \subseteq E, \;\; out(P_j) \subseteq E'}{\begin{array}{l}7.\;\; isWSCore(\mathsf{andSplit}(\mathsf{e},E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{andJoin}(E',\mathsf{e}''))\\ 8.\;\; isWSCore(\mathsf{xorSplit}(\mathsf{e},E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E',\mathsf{e}''))\end{array}}$$

$$9.\;\; \frac{\forall j \in [1..n] \;\; isWSCore(P_j), \;\; in(P_j) = \mathsf{e}'_j, \;\; out(P_j) \subseteq E}{isWSCore(\mathsf{eventBased}(\mathsf{e},\{(\mathsf{m}_j,\mathsf{e}'_j) | j \in [1..n]\}) \parallel P_1 \parallel \ldots \mid P_n \parallel \mathsf{xorJoin}(E,\mathsf{e}''))}$$

$$10.\;\; \frac{\begin{array}{c} isWSCore(P_1), isWSCore(P_2),\\ in(P_1) = \{\mathsf{e}'\}, out(P_1) = \{\mathsf{e}^{\mathsf{iv}}\},\\ in(P_2) = \{\mathsf{e}^{\mathsf{vi}}\}, out(P_2) = \{\mathsf{e}''\}\end{array}}{isWSCore(\mathsf{xorJoin}(\{\mathsf{e}'',\mathsf{e}'''\},\mathsf{e}') \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}},\{\mathsf{e}^{\mathsf{v}},\mathsf{e}^{\mathsf{vi}}\}))}$$

$$\frac{isWSCore(P_1'), in(P_1') = \{\mathsf{e}''\}, out(P_1') = \{\mathsf{e}'''\}}{\begin{array}{l} 11(a).\ isWSCore(\mathsf{subProc}(\mathsf{e}, \mathsf{start}(\mathsf{e}', \mathsf{e}'') \parallel P_1' \parallel \mathsf{end}(\mathsf{e}''', \mathsf{e}^{\mathsf{iv}}), \mathsf{e}^{\mathsf{v}})) \\ 11(b).\ isWSCore(\mathsf{subProc}(\mathsf{e}, \mathsf{start}(\mathsf{e}', \mathsf{e}'') \parallel P_1' \parallel \mathsf{terminate}(\mathsf{e}'''), \mathsf{e}^{\mathsf{iv}})) \\ 11(c).\ isWSCore(\mathsf{subProc}(\mathsf{e}, \mathsf{start}(\mathsf{e}', \mathsf{e}'') \parallel P_1' \parallel \mathsf{endSnd}(\mathsf{e}''', \mathsf{m}, \mathsf{e}^{\mathsf{iv}}), \mathsf{e}^{\mathsf{v}})) \\ 11(d).\ isWSCore(\mathsf{subProc}(\mathsf{e}, \mathsf{startRcv}(\mathsf{e}', \mathsf{m}, \mathsf{e}'') \parallel P_1' \parallel \mathsf{end}(\mathsf{e}''', \mathsf{e}^{\mathsf{iv}}), \mathsf{e}^{\mathsf{v}})) \\ 11(e).\ isWSCore(\mathsf{subProc}(\mathsf{e}, \mathsf{startRcv}(\mathsf{e}', \mathsf{m}, \mathsf{e}'') \parallel P_1' \parallel \mathsf{terminate}(\mathsf{e}'''), \mathsf{e}^{\mathsf{iv}})) \\ 11(f).\ isWSCore(\mathsf{subProc}(\mathsf{e}, \mathsf{startRcv}(\mathsf{e}', \mathsf{m}, \mathsf{e}'') \parallel P_1' \parallel \mathsf{endSnd}(\mathsf{e}''', \mathsf{m}, \mathsf{e}^{\mathsf{iv}}), \mathsf{e}^{\mathsf{v}})) \end{array}}$$

$$12.\quad \frac{isWSCore(P_1), isWSCore(P_2), out(P_1) = in(P_2)}{isWSCore(P_1 \parallel P_2)}$$

According to the definition 4, well-structured processes are given in the forms (1-6), that is as a (core) process included between any possible combination of different types of the start and end events included in the semantics. We allow a start event or a start message event and one simple end event or terminate event or end messege event. The (core) process between the start and end events can be composed by any element up to the well-structured core definition. Any single task or intermediate event is a well-structured core (cases 1-6); a composite process starting with an AND (resp. XOR, resp. Event-based) split and closing with an AND (resp. XOR, resp. XOR) join is well-structured core if each edge of the split is connected to a given edge of the join by means of a well-structured core processes (cases 7-9); a loop of sequence edges ($\mathsf{e}_1 \rightarrow \mathsf{e}_4 \rightarrow \mathsf{e}_6 \rightarrow \mathsf{e}_2$) formed by means of a XOR split and a XOR join is well-structured core if the body of the loop consists of well-structured core processes (case 10). Notably, only loops formed by XOR gateways are well-structured. For a better understanding, cases 7 - 10 are graphically depicted in Fig. 15. A subprocess is well structure core if it includes a well-structured core process (case 11). A process element collection is well-structured core if its processes are well-structured and sequentially composed (case 12).

Well-structuredness can be also extended to collaborations, by requiring each process involved in a collaboration to be well-structured.

Definition 5 (Well-structured collaborations). *Let C be a collaboration, $isWS(C)$ is inductively defined as follows:*

- $isWS(\mathsf{pool}(\mathsf{p}, P))$ *if P is well-structured;*
- $isWS(C_1 \parallel C_2)$ *if $isWS(C_1)$ and $isWS(C_2)$.*

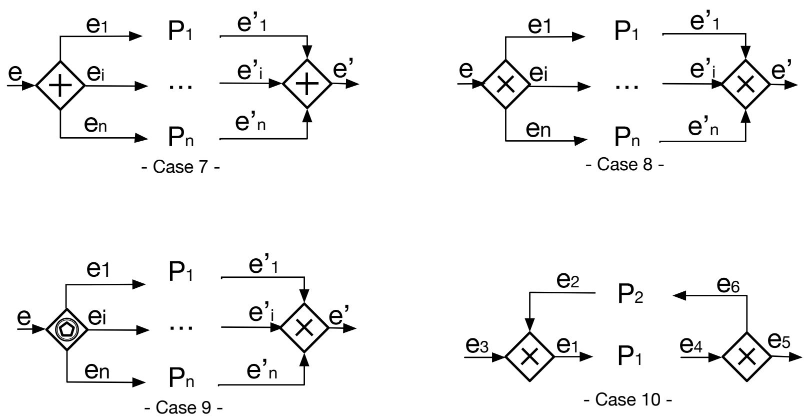


Figure 15: Well-structured nodes (cases 7-10).

Running Example (4/9). Considering the proposed running example and according to the above definitions, process P_C is well-structured, while process P_{TA} is not well-structured, due to the presence of the unstructured loop formed by the XOR join and an AND split. Thus, the overall collaboration is not well-structured. □

5.2. Safe BPMN Collaborations

A relevant property in business process domain is *safeness*, i.e the occurrence of no more than one token along the same sequence edge during the process execution.

Before providing a formal characterisation of safeness BPMN processes and collaborations, we need to introduce the auxiliary function $maxMarking(\cdot)$ that, given a configuration $\langle P, \sigma\rangle$, determines the maximum number of tokens marking the sequence edges of elements in P according to the state σ (this function relies on the standard function $max(\cdot)$ returning the maximum in a list of natural numbers).

$$
\begin{aligned}
&maxMarking(\mathsf{start}(\mathsf{e},\mathsf{e}'),\sigma) = \sigma(\mathsf{e}')\\
&maxMarking(\mathsf{end}(\mathsf{e},\mathsf{e}'),\sigma) = \sigma(\mathsf{e})\\
&maxMarking(\mathsf{startRcv}(\mathsf{e},\mathsf{m},\mathsf{e}'),\sigma) = \sigma(\mathsf{e}')\\
&maxMarking(\mathsf{endSnd}(\mathsf{e},\mathsf{m},\mathsf{e}'),\sigma) = \sigma(\mathsf{e})\\
&maxMarking(\mathsf{terminate}(\mathsf{e}),\sigma) = \sigma(\mathsf{e})\\
&maxMarking(\mathsf{andSplit}(\mathsf{e},E),\sigma) = max(\sigma(\mathsf{e}),\sigma(\mathsf{e}').\forall \mathsf{e}' \in E)\\
&maxMarking(\mathsf{xorSplit}(\mathsf{e},E),\sigma) = max(\sigma(\mathsf{e}),\sigma(\mathsf{e}').\forall \mathsf{e}' \in E)\\
&maxMarking(\mathsf{andJoin}(E,\mathsf{e}),\sigma) = max(\sigma(\mathsf{e}').\forall \mathsf{e}' \in E,\sigma(\mathsf{e}))\\
&maxMarking(\mathsf{xorJoin}(E,\mathsf{e}),\sigma) = max(\sigma(\mathsf{e}').\forall \mathsf{e}' \in E,\sigma(\mathsf{e}))\\
&maxMarking(\mathsf{task}(\mathsf{e},\mathsf{e}'),\sigma) = max(\sigma(\mathsf{e}),\sigma(\mathsf{e}'))\\
&maxMarking(\mathsf{taskRcv}(\mathsf{e},\mathsf{m},\mathsf{e}'),\sigma) = max(\sigma(\mathsf{e}),\sigma(\mathsf{e}'))\\
&maxMarking(\mathsf{taskSnd}(\mathsf{e},\mathsf{m},\mathsf{e}'),\sigma) = max(\sigma(\mathsf{e}),\sigma(\mathsf{e}'))\\
&maxMarking(\mathsf{empty}(\mathsf{e},\mathsf{e}'),\sigma) = max(\sigma(\mathsf{e}),\sigma(\mathsf{e}'))\\
&maxMarking(\mathsf{interRcv}(\mathsf{e},\mathsf{m},\mathsf{e}'),\sigma) = max(\sigma(\mathsf{e}),\sigma(\mathsf{e}'))\\
&maxMarking(\mathsf{interSnd}(\mathsf{e},\mathsf{m},\mathsf{e}'),\sigma) = max(\sigma(\mathsf{e}),\sigma(\mathsf{e}'))\\
&maxMarking(\mathsf{subProc}(\mathsf{e},P,\mathsf{e}'),\sigma) = max(\sigma(\mathsf{e}),maxMarking(P,\sigma),\sigma(\mathsf{e}'))\\
&maxMarking(P_1 \parallel P_2,\sigma) = max(maxMarking(P_1,\sigma),maxMarking(P_2,\sigma))
\end{aligned}
$$

$maxMarking(\cdot)$ can be also simply extended to collaborations $\langle C,\sigma\rangle$, to determine the maximum number of tokens marking the sequence edges of elements in all the processes P included in the collaboration.

$$
\begin{aligned}
&maxMarking(\mathsf{pool}(\mathsf{p},P),\sigma) = maxMarking(P,\sigma)\\
&maxMarking(C_1 \parallel C_2,\sigma) =\\
&\qquad max(maxMarking(participant(C_1),\sigma), maxMarking(participant(C_2),\sigma))
\end{aligned}
$$

We also need the following definition determining the safeness of a process in a given state.

Definition 6 (Current state safe process). *A process configuration $\langle P,\sigma\rangle$ is current state safe (*cs-safe*) if and only if $maxMarking(P,\sigma) \leqslant 1$.*

We can finally conclude with the definition of safe processes and collaborations which requires that cs-safeness is preserved along the computations. Now, a process (collaboration) is defined to be safe if it is preserved that the maximum marking does not exceed one along the process (collaboration) execution. We use $\rightarrow^*$ to denote the reflexive and transitive closure of $\rightarrow$.

Definition 7 (Safe processes). *A process P is* safe *if and only if, given σ such that $isInit(P, \sigma)$, for all σ' such that $\langle P, \sigma\rangle \to^* \sigma'$ we have that $\langle P, \sigma'\rangle$ is cs-safe.*

Definition 8 (Safe collaborations). *A collaboration C is* safe *if and only if, given σ and δ such that $isInit(C, \sigma, \delta)$, for all for all σ' and δ' such that $\langle C, \sigma, \delta\rangle \to^* \langle \sigma', \delta'\rangle$ we have that $maxMarking(C, \sigma') \leqslant 1$.*

Running Example (5/9). Let us consider again our running example depicted in Fig. 2. Process P_C is safe since there is not any process fragment capable of producing more than one token. Process P_{TA} instead is not safe. In fact, if task Make Travel Offer is executed more than once, we would have that the AND split gateway will produce more than one token in the sequence flow connected to the Booking Received event. Thus, also the resulting collaboration is not safe. □

5.3. Sound BPMN Collaborations

Also soundness is considered as a relevant property. We defined it both at the process and collaboration level. In a process it ensures that, if once its execution starts with a token in the start event, it is always possible to reach one of these scenarios. The first one *(i)* where all marked end events are marked exactly by a single token and all sequence edges are unmarked. While the second *(ii)* when no token are observed in the configuration (i.e. in the case of a token reaches a terminate event). The definition extends to collaboration by considering the combined execution of the included processes and taking into account that all the messages are handled during the execution (i.e. no pending message tokens are observed).

Definition 9 (Current state sound process). *A process configuration $\langle P, \sigma\rangle$ is current state sound (*cs-sound*) if and only if one of the following hold:*

(i) $\forall\, \mathsf{e} \in marked(\sigma, end(P))\ .\ \sigma(\mathsf{e}) = 1$ *and* $isZero(P, \sigma)$*;*

(ii) $\forall\, \mathsf{e} \in \mathsf{edges}(P)\ .\ \sigma(\mathsf{e}) = 0$.

Definition 10 (Sound process). *A process P is* sound *if and only if, given σ such that $isInit(P, \sigma)$, for all σ' such that $\langle P, \sigma\rangle \to^* \sigma'$ we have that there exists σ'' such that $\langle P, \sigma'\rangle \to^* \sigma''$, and $\langle P, \sigma''\rangle$ is cs-sound.*

Definition 11 (Sound collaboration). *A collaboration C is* sound *if and only if, given σ and δ such that $isInit(C, \sigma, \delta)$, for all σ' and δ' such that $\langle C, \sigma, \delta\rangle \rightarrow^* \langle \sigma', \delta'\rangle$ we have that there exist σ'' and δ'' such that $\langle C, \sigma', \delta'\rangle \rightarrow^* \langle \sigma'', \delta''\rangle$, $\forall$ P in C we have that $\langle P, \sigma''\rangle$ is cs-sound, and $\forall\, \mathsf{m} \in \mathbb{M} \,.\, \delta''(\mathsf{m}) = 0$.*

Thanks to the expressibility of our formalisation to distinguish sequence tokens from message tokens we relax the soundness property by defining message-relaxed soundness. It extends the usual soundness notion by considering sound also those collaborations in which asynchronously sent messages are not handled by the receiver.

Definition 12 (Message-relaxed sound collaboration). *A collaboration C is* Message-relaxed sound *if and only if, given σ and δ such that $isInit(C, \sigma, \delta)$, for all σ' and δ' such that $\langle C, \sigma, \delta\rangle \rightarrow^* \langle \sigma', \delta'\rangle$ we have that there exist σ'' and δ'' such that $\langle C, \sigma', \delta'\rangle \rightarrow^* \langle \sigma'', \delta''\rangle$, and $\forall$ P in C we have that $\langle P, \sigma''\rangle$ is cs-sound.*

Running Example (6/9). Let us consider again our running example. It is easily to see that process P_C is sound, since it is always possible to reach the end event and when reached there is no token marking the sequence flows. Also process P_{TA} is sound, since when a token reaches the terminate event, all the other tokens are removed from the edges by means of the killing effect. However, the resulting collaboration is not sound. In fact, when a travel offer is accepted by the customer, we would have that the AND-Split gateway will produce two tokens, one of which re-activates the task Make Travel Offer. Thus, even if the process completes, the message lists are not empty. However, the collaboration satisfied the message-relaxed soundness property. □

6. Relationships among Properties

In this section we study the relationships among the considered properties both at the process and collaboration level. In particular we investigate the relationship between (i) well-structuredness and safeness, (ii) well-structuredness and soundness, and (iii) safeness and soundness. The proofs of these results are reported in the Appendix B.

6.1. Well-structuredness vs. Safeness in BPMN

Considering well-structuredness and safeness we demonstrate that all well-structured models are safe (Theorem 1), and that the vice versa does not hold. To this aim, first we show that a process in the initial state is cs-safe (Lemma 1). Then, we show that cs-safeness is preserved by the evolution of well-structured core process elements (Lemma 2) and processes (Lemma 3). These latter two lemmas rely on the notion of *reachable* processes/core elements of processes (that is process elements different from start, end, and terminate events). In fact, the syntax in Fig. 12 is too liberal, as it allows terms that cannot be obtained (by means of transitions) from a process in its initial state. This last notion, in its own turn, needs the definition of initial state for a core process element.

Definition 13 (Initial state of core elements in P). *Let P be a process, then $isInitEl(P, \sigma)$ is inductively defined on the structure of process P as follows:*

$isInitEl(\mathsf{task}(\mathsf{e}, \mathsf{e}'), \sigma)$ *if* $\sigma(\mathsf{e}) = 1$ *and* $\sigma(\mathsf{e}') = 0$
$isInitEl(\mathsf{taskRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}'), \sigma)$ *if* $\sigma(\mathsf{e}) = 1$ *and* $\sigma(\mathsf{e}') = 0$
$isInitEl(\mathsf{taskSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}'), \sigma)$ *if* $\sigma(\mathsf{e}) = 1$ *and* $\sigma(\mathsf{e}') = 0$
$isInitEl(\mathsf{empty}(\mathsf{e}, \mathsf{e}'), \sigma)$ *if* $\sigma(\mathsf{e}) = 1$ *and* $\sigma(\mathsf{e}') = 0$
$isInitEl(\mathsf{interRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}'), \sigma)$ *if* $\sigma(\mathsf{e}) = 1$ *and* $\sigma(\mathsf{e}') = 0$
$isInitEl(\mathsf{interSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}'), \sigma)$ *if* $\sigma(\mathsf{e}) = 1$ *and* $\sigma(\mathsf{e}') = 0$
$isInitEl(\mathsf{andSplit}(\mathsf{e}, E), \sigma)$ *if* $\sigma(\mathsf{e}) = 1$ *and* $\forall \mathsf{e}' \in E \,.\, \sigma(\mathsf{e}') = 0$
$isInitEl(\mathsf{xorSplit}(\mathsf{e}, E), \sigma)$ *if* $\sigma(\mathsf{e}) = 1$ *and* $\forall \mathsf{e}' \in E \,.\, \sigma(\mathsf{e}') = 0$
$isInitEl(\mathsf{andJoin}(E, \mathsf{e}), \sigma)$ *if* $\forall \mathsf{e}' \in E \,.\, \sigma(\mathsf{e}') = 1$ *and* $\sigma(\mathsf{e}) = 0$
$isInitEl(\mathsf{xorJoin}(E, \mathsf{e}), \sigma)$ *if* $\exists \mathsf{e}' \in E \,.\, \sigma(\mathsf{e}') = 1$ *and* $\sigma(\mathsf{e}) = 0$
$isInitEl(\mathsf{eventBased}(\mathsf{e}, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_k, \mathsf{e}_{ok})), \sigma)$ *if* $\sigma(\mathsf{e}) = 1$
and $\forall \mathsf{e}' \in \{\mathsf{e}_{o1}, \ldots, \mathsf{e}_{ok}\} \,.\, \sigma(\mathsf{e}') = 0$
$isInitEl(\mathsf{subProc}(\mathsf{e}, P, \mathsf{e}'))$ *if* $\sigma(\mathsf{e}) = 1$, $\sigma(\mathsf{e}') = 0$
and $\forall \mathsf{e}'' \in \mathsf{edges}(P) \,.\, \sigma(\mathsf{e}'') = 0$
$isInitEl(P_1 \| P_2, \sigma)$ *if* $\forall \mathsf{e} \in in(P_1 \parallel P_2) \,:\, isInitEl(\mathsf{getEl}(\mathsf{e}, P_1 \parallel P_2))$
and $\forall \mathsf{e} \in (\mathsf{edges}(P_1 \parallel P_2) \backslash in(P_1 \parallel P_2)) \,:\, \sigma(\mathsf{e}) = 0$

where $\mathsf{getEl}(\mathsf{e}, P)$ *returns the element in P with incoming edge* e*:*

- $\mathsf{getEl}(\mathsf{e}, \mathsf{task}(\mathsf{e}', \mathsf{e}'')) = \begin{cases} \mathsf{task}(\mathsf{e}', \mathsf{e}'') & \text{if } \mathsf{e} = \mathsf{e}' \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{taskRcv}(\mathsf{e}', \mathsf{m}, \mathsf{e}'')) = \begin{cases} \mathsf{taskRcv}(\mathsf{e}', \mathsf{m}, \mathsf{e}'') & \text{if } \mathsf{e} = \mathsf{e}' \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{taskSnd}(\mathsf{e}', \mathsf{m}, \mathsf{e}'')) = \begin{cases} \mathsf{taskSnd}(\mathsf{e}', \mathsf{m}, \mathsf{e}'') & \text{if } \mathsf{e} = \mathsf{e}' \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{empty}(\mathsf{e}', \mathsf{e}'')) = \begin{cases} \mathsf{empty}(\mathsf{e}', \mathsf{e}'') & \text{if } \mathsf{e} = \mathsf{e}' \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{interRcv}(\mathsf{e}', \mathsf{m}, \mathsf{e}'')) = \begin{cases} \mathsf{interRcv}(\mathsf{e}', \mathsf{m}, \mathsf{e}'') & \text{if } \mathsf{e} = \mathsf{e}' \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{interSnd}(\mathsf{e}', \mathsf{m}, \mathsf{e}'')) = \begin{cases} \mathsf{interSnd}(\mathsf{e}', \mathsf{m}, \mathsf{e}'') & \text{if } \mathsf{e} = \mathsf{e}' \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{andSplit}(\mathsf{e}', E)) = \begin{cases} \mathsf{andSplit}(\mathsf{e}', E) & \text{if } \mathsf{e} = \mathsf{e}' \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{andJoin}(E, \mathsf{e}')) = \begin{cases} \mathsf{andJoin}(E, \mathsf{e}') & \text{if } \mathsf{e} \in E \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{xorSplit}(\mathsf{e}', E)) = \begin{cases} \mathsf{xorSplit}(\mathsf{e}', E) & \text{if } \mathsf{e} = \mathsf{e}' \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{xorJoin}(E, \mathsf{e}')) = \begin{cases} \mathsf{xorJoin}(E, \mathsf{e}') & \text{if } \mathsf{e} \in E \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{eventBased}(\mathsf{e}', (\mathsf{m}_1, \mathsf{e}''_1), \ldots, (\mathsf{m}_k, \mathsf{e}''_k))) = \begin{cases} \mathsf{eventBased}(\mathsf{e}', (\mathsf{m}_1, \mathsf{e}''_1), \ldots, (\mathsf{m}_k, \mathsf{e}''_k)) & \text{if } \mathsf{e} = \mathsf{e}' \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{subProc}(\mathsf{e}', P, \mathsf{e}'')) = \begin{cases} \mathsf{subProc}(\mathsf{e}', P, \mathsf{e}'') & \text{if } \mathsf{e} = \mathsf{e}' \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, P_1 \parallel P_2) = \mathsf{getEl}(\mathsf{e}, P_1), \mathsf{getEl}(\mathsf{e}, P_2)$

Definition 14 (Reachable processes). *A process configuration* $\langle P, \sigma \rangle$ *is* reachable *if there exists a configuration* $\langle P, \sigma' \rangle$ *such that* $isInit(P, \sigma')$ *and* $\langle P, \sigma' \rangle \rightarrow^{*} \sigma$.

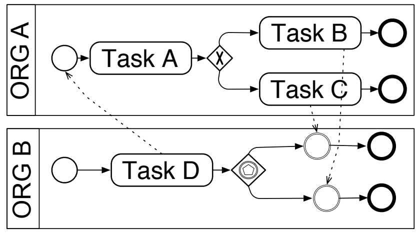


Figure 16: A safe BPMN collaboration not well-structured

Definition 15 (Reachable core process element). *A process configuration $\langle P, \sigma\rangle$ is* core reachable *if there exists a configuration $\langle P, \sigma'\rangle$ such that $isInitEl(P, \sigma')$ and $\langle P, \sigma'\rangle \rightarrow^{*}\sigma$.*

Lemma 1. *Let P be a process, if $isInit(P, \sigma)$ then $\langle P, \sigma\rangle$ is cs-safe.*

Proof (sketch). Trivially, from definition of $isInit(P, \sigma)$. □

Lemma 2. *Let $isWSCore(P)$, and let $\langle P, \sigma\rangle$ be a core reachable and cs-safe process configuration, if $\langle P, \sigma\rangle \xrightarrow{\alpha} \sigma'$ then $\langle P, \sigma'\rangle$ is cs-safe.*

Proof (sketch). We proceed by induction on the structure of well-structured core process elements. □

Lemma 3. *Let P be WS, and let $\langle P, \sigma\rangle$ be a process configuration reachable and cs-safe, if $\langle P, \sigma\rangle \xrightarrow{\alpha} \sigma'$ then $\langle P, \sigma'\rangle$ is cs-safe.*

Proof (sketch). We proceed by case analysis on the structure of P, which is a WS process (see Definition 4). □

Theorem 1. *Let P be a process, if P is well-structured then P is safe.*

Proof (sketch). We show that if $\langle P, \sigma\rangle \rightarrow^{*}\sigma'$ then $\langle P, \sigma'\rangle$ is cs-safe, by induction on the length n of the sequence of transitions from $\langle P, \sigma\rangle$ to $\langle P, \sigma'\rangle$. □

The reverse implication of Theorem 1 is not true. In fact there are safe processes that are not well-structured. The collaboration diagram represented in Fig. 16 is an example. The involved processes are trivially safe, since there are not fragments capable of generating multiple tokens; however they are not well-structured.

We now extend the previous results to collaborations.

Theorem 2. *Let C be a collaboration, if C is well-structured then C is safe.*

Proof (sketch). We proceed by contradiction. □

6.2. Well-structuredness vs. Soundness in BPMN

Considering the relationship between well-structuredness and soundness. We prove that a well-structured process is always sound (Theorem 3), but there are sound processes that are not well-structured. To this aim, first we show that a reachable well-structured core process element can always complete its execution (Lemma 4). This latter Lemma 4 is based on the auxiliary definition of the final state of core elements in a process, given for all elements with the exception of start and end events.

Definition 16 (Final state of core elements in P). *Let P be a process, then $isCompleteEl(P, \sigma)$ is inductively defined on the structure of process P as follows:*

$isCompleteEl(\mathsf{task}(\mathsf{e}, \mathsf{e}'), \sigma)$ *if* $\sigma(\mathsf{e}) = 0$ *and* $\sigma(\mathsf{e}') = 1$

$isCompleteEl(\mathsf{taskRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}'), \sigma)$ *if* $\sigma(\mathsf{e}) = 0$ *and* $\sigma(\mathsf{e}') = 1$

$isCompleteEl(\mathsf{taskSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}'), \sigma)$ *if* $\sigma(\mathsf{e}) = 0$ *and* $\sigma(\mathsf{e}') = 1$

$isCompleteEl(\mathsf{empty}(\mathsf{e}, \mathsf{e}'), \sigma)$ *if* $\sigma(\mathsf{e}) = 0$ *and* $\sigma(\mathsf{e}') = 1$

$isCompleteEl(\mathsf{interRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}'), \sigma)$ *if* $\sigma(\mathsf{e}) = 0$ *and* $\sigma(\mathsf{e}') = 1$

$isCompleteEl(\mathsf{interSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}'), \sigma)$ *if* $\sigma(\mathsf{e}) = 0$ *and* $\sigma(\mathsf{e}') = 1$

$isCompleteEl(\mathsf{andSplit}(\mathsf{e}, E), \sigma)$ *if* $\sigma(\mathsf{e}) = 0$ *and* $\forall \mathsf{e}' \in E \,.\, \sigma(\mathsf{e}') = 1$

$isCompleteEl(\mathsf{xorSplit}(\mathsf{e}, E), \sigma)$ *if* $\sigma(\mathsf{e}) = 0$ *and* $\exists \mathsf{e}' \in E \,.\, \sigma(\mathsf{e}') = 1$
and $\forall \mathsf{e}'' \in E \backslash \mathsf{e}' \,.\, \sigma(\mathsf{e}'') = 0$

$isCompleteEl(\mathsf{andJoin}(E, \mathsf{e}), \sigma)$ *if* $\forall \mathsf{e}' \in E \,.\, \sigma(\mathsf{e}') = 0$ *and* $\sigma(\mathsf{e}) = 1$

$isCompleteEl(\mathsf{xorJoin}(E, \mathsf{e}), \sigma)$ *if* $\forall \mathsf{e}' \in E \,.\, \sigma(\mathsf{e}') = 0$ *and* $\sigma(\mathsf{e}) = 1$

$isCompleteEl(\mathsf{eventBased}(\mathsf{e}, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_k, \mathsf{e}_{ok})), \sigma)$ *if* $\sigma(\mathsf{e}) = 0$
and $\exists \mathsf{e}' \in \{\mathsf{e}_{o1}, \ldots, \mathsf{e}_{ok}\} \,.\, \sigma(\mathsf{e}') = 1$
and $\forall \mathsf{e}'' \in \{\mathsf{e}_{o1}, \ldots, \mathsf{e}_{ok}\} \backslash \mathsf{e}' \,.\, \sigma(\mathsf{e}'') = 0$

$isCompleteEl(\mathsf{subProc}(\mathsf{e}, P, \mathsf{e}'))$ *if* $\sigma(\mathsf{e}) = 0$, $\sigma(\mathsf{e}') = 1$
and $\forall \mathsf{e}'' \in \mathsf{edges}(P) \,.\, \sigma(\mathsf{e}'') = 0$

$isCompleteEl(P_1 \| P_2, \sigma)$ *if* $\forall \mathsf{e} \in out(P_1 \parallel P_2) \;:\; isCompleteEl(\mathsf{getEl}(\mathsf{e}, P_1 \parallel P_2))$
and $\forall \mathsf{e} \in (\mathsf{edges}(P_1 \parallel P_2) \backslash out(P_1 \parallel P_2)) \;:\; \sigma(\mathsf{e}) = 0$

where $\mathsf{getEl}(\mathsf{e}, P)$ *returns the element in* P *with outgoing edge* e*:*

- $\mathsf{getEl}(\mathsf{e}, \mathsf{task}(\mathsf{e}', \mathsf{e}'')) = \begin{cases} \mathsf{task}(\mathsf{e}', \mathsf{e}'') & \text{if } \mathsf{e} = \mathsf{e}'' \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{taskRcv}(\mathsf{e}', \mathsf{m}, \mathsf{e}'')) = \begin{cases} \mathsf{taskRcv}(\mathsf{e}', \mathsf{m}, \mathsf{e}'') & \text{if } \mathsf{e} = \mathsf{e}'' \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{taskSnd}(\mathsf{e}', \mathsf{m}, \mathsf{e}'')) = \begin{cases} \mathsf{taskSnd}(\mathsf{e}', \mathsf{m}, \mathsf{e}'') & \text{if } \mathsf{e} = \mathsf{e}'' \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{empty}(\mathsf{e}', \mathsf{e}'')) = \begin{cases} \mathsf{empty}(\mathsf{e}', \mathsf{e}'') & \text{if } \mathsf{e} = \mathsf{e}'' \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{interRcv}(\mathsf{e}', \mathsf{m}, \mathsf{e}'')) = \begin{cases} \mathsf{interRcv}(\mathsf{e}', \mathsf{m}, \mathsf{e}'') & \text{if } \mathsf{e} = \mathsf{e}'' \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{interSnd}(\mathsf{e}', \mathsf{m}, \mathsf{e}'')) = \begin{cases} \mathsf{interSnd}(\mathsf{e}', \mathsf{m}, \mathsf{e}'') & \text{if } \mathsf{e} = \mathsf{e}'' \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{andSplit}(\mathsf{e}', E)) = \begin{cases} \mathsf{andSplit}(\mathsf{e}', E) & \text{if } \mathsf{e} \in E \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{andJoin}(E, \mathsf{e}')) = \begin{cases} \mathsf{andJoin}(E, \mathsf{e}') & \text{if } \mathsf{e} = \mathsf{e}' \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{xorSplit}(\mathsf{e}', E)) = \begin{cases} \mathsf{xorSplit}(\mathsf{e}', E) & \text{if } \mathsf{e} \in E \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{xorJoin}(E, \mathsf{e}')) = \begin{cases} \mathsf{xorJoin}(E, \mathsf{e}') & \text{if } \mathsf{e} = \mathsf{e}' \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{eventBased}(\mathsf{e}', (\mathsf{m}_1, \mathsf{e}''_1), \ldots, (\mathsf{m}_k, \mathsf{e}''_k))) =$
  $\begin{cases} \mathsf{eventBased}(\mathsf{e}', (\mathsf{m}_1, \mathsf{e}''_1), \ldots, (\mathsf{m}_k, \mathsf{e}''_k)) & \text{if } \mathsf{e} \in \{\mathsf{e}''_1, \ldots, \mathsf{e}''_k\} \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, \mathsf{subProc}(\mathsf{e}', P, \mathsf{e}'')) = \begin{cases} \mathsf{subProc}(\mathsf{e}', P, \mathsf{e}'') & \text{if } \mathsf{e} = \mathsf{e}' \\ \epsilon & \text{otherwise} \end{cases}$

- $\mathsf{getEl}(\mathsf{e}, P_1 \parallel P_2) = \mathsf{getEl}(\mathsf{e}, P_1), \mathsf{getEl}(\mathsf{e}, P_2)$

Lemma 4. *Let* $isWSCore(P)$ *and let* $\langle P, \sigma \rangle$ *be core reachable, then there exists* σ' *such that* $\langle P, \sigma \rangle \rightarrow^* \sigma'$ *and* $isCompleteEl(P, \sigma')$.

Proof (sketch). We proceed by induction on the structure of well-structured core process. □

Theorem 3. *Let* P *be a WS process, then* P *is sound.*

Proof (sketch). We proceed by case analysis. □

The reverse implication of Theorem 3 is not true. In fact there are sound processes that are not well-structured; see for example the process represented in Fig. 17. This process is surely unstructured, and it is also trivially sound, since it is always possible to reach an end event without leaving tokens marking the sequence flows.

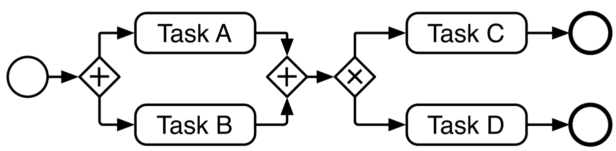


Figure 17: An example of sound process not Well-Structured.

However, Theorem 3 does not extend to the collaboration level. In fact, when we put well-structured processes together in a collaboration, this could be either sound or unsound. This is also valid for message-relaxed soundness.

Theorem 4. *Let* C *be a collaboration,* C *is WS does not imply* C *is sound.*

Proof (sketch). We proceed by contradiction. □

Theorem 5. *Let* C *be a collaboration,* C *is WS does not imply* C *is message-relaxed sound.*

Proof (sketch). We proceed by contradiction. □

6.3. Safeness vs. Soundness in BPMN

Considering the relationship between safeness and soundness. We demonstrate that there are unsafe models that are sound. This is a peculiarity of BPMN, faithfully implemented in our semantics, thank to its capability to support the terminate end event and (unsafe) sub-processes. Let us first reason at process level considering some examples.

Theorem 6. *Let P be a process, P is unsafe does not imply P is unsound.*

Proof (sketch). We proceed by contradiction. □

Let us consider now the collaboration level. We have that unsafe collaborations could either sound or unsound, as proved by the following Theorem.

Theorem 7. *Let C be a collaboration, C is unsafe does not imply C is unsound.*

Proof (sketch). We proceed by contradiction. □

Running Example (7/9). Considering the collaboration in our running example, Customer is both safe and sound, while the process of the Travel Agency is unsafe but sound, since the terminate event permits a to reach a marking where all edges are umarked. The collaboration is not safe, and it is also unsound but message-relaxed sound, since there could be messages in the message lists.

7. Compositionality of Safeness and Soundness

In this section we study safeness and soundness compositionality, i.e. how the behaviour of processes affects that of the entire resulting collaboration. In particular, we show the interrelationship between the studied properties at collaboration and at process level. At process level we also consider the compositionality of sub-processes, investigating how sub-processes behaviour impacts on the safeness and soundness of process including them.

7.1. On Compositionality of Safeness

We show here that safeness is compositional, that is the composition of safe processes always results in a safe collaboration.

Theorem 8. *Let C be a collaboration, if all processes in C are safe then C is safe.*

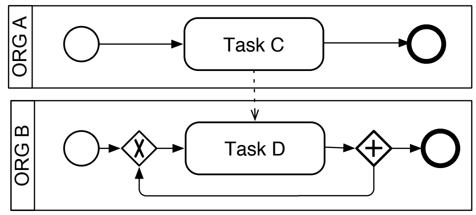


Figure 18: Safe collaboration with safe and unsafe processes.

Proof (sketch). We proceed by contradiction (see Appendix B). □

We also show that the unsafeness of a collaboration cannot be in general determined by information about the unsafeness of the processes that compose it. Indeed, putting together an unsafe process with a safe or unsafe one, the obtained collaboration could be either safe or unsafe. Let us consider now some cases.

Running Example (8/9). In our example, the collaboration is composed by a safe process and an unsafe one. In fact, focussing on the process of the Travel Agency, we can immediately see that it is not safe: the loop given by a XOR join and an AND split produces multiple tokens on one of the outgoing edges of the AND split. Now, if we consider this process together with the safe process of Customer, the resulting collaboration is not safe. Indeed, the XOR split gateway, which checks if the offer is interesting, forwards only one token on one of the two paths. As soon as a received offer is considered interesting, the Customer process proceeds and completes. Thus, due to the lack of safeness, the travel agency will continue to make offers to the customer, but no more offer messages arriving from the Travel Agency will be considered by the customer. □

Example 1. *Another example refers to the case in which a collaboration composed by a safe process and an unsafe one results in a safe collaboration, as shown in Fig. 18. If we focus only on the process in* ORG B *we can immediately notice that it is not safe: again the loop given by a XOR join and an AND split produces multiple tokens on the same edge. However, if we consider this process together with the safe process of* ORG A*, the resulting collaboration is safe. In fact, task D receives only one message, producing a token that is successively split by the AND gateway. No more message arrives from the send task, so, although there is a token is blocked, we have no problem of safeness.* □

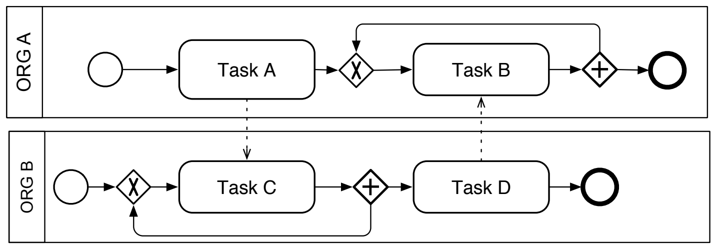


Figure 19: Safe collaboration with unsafe processes.

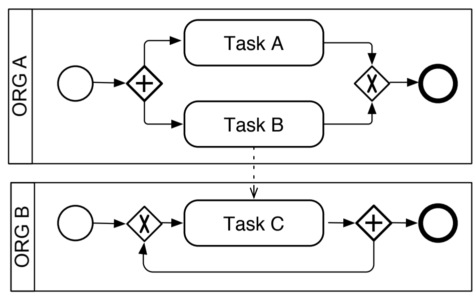


Figure 20: Unsafe collaboration with unsafe processes.

Example 2. *In Fig. 19 we have two unsafe processes, since each of them contains a loop capable of generating an unbounded number of tokens. However, if we consider the collaboration obtained by the combination of these processes, it turns out to be safe. Indeed, as in the previous example, tasks C and B are executed only once, as they receive only one message. Thus, the two loops are blocked and cannot effectively generate multiple tokens.* □

Example 3. *Also the collaboration in Fig. 20 is composed by two unsafe processes: process in* ORG A *contains an AND split followed by a XOR join that produces two tokens on the outgoing edge of the XOR gateway; process in* ORG B *contains the same loop as in the previous examples. In this case the collaboration composed by these two processes is unsafe. Indeed, the XOR join in* ORG A *will effectively produce two tokens since the sending of task B is not blocking.* □

Let us now to consider processes including sub-processes. We show that the composition of unsafe sub-processes always results in un-safe processes, but the vice versa does not hold. There are also un-safe processes including safe sub-process when the unsafeness does not depend from the behaviour of the sub-process.

Theorem 9. *Let P be a process including a sub-process* $\mathsf{subProc}(\mathsf{e}_i, P_1, \mathsf{e}_o)$*, if P_1 is unsafe then P is unsafe.*

Proof (sketch). We proceed by contradiction (see Appendix B). □

7.2. On Compositionality of Soundness

As well as for the safeness property, we show now that it is not feasible to detect the soundness of a collaboration by relying only on information about soundness of processes that compose it. However, the unsoundness of processes implies the unsoundness of the resulting collaboration.

Theorem 10. *Let C be a collaboration, if some processes in C are unsound then C is unsound.*

Proof (sketch). We proceed by contradiction (see Appendix B). □

On the other hand, when we put together sound processes, the obtained collaboration could be either sound or unsound, since we have also to consider messages. It can happen that either a process waits for a message that will never be received or it receive more than the number of messages it is able to process. Let us consider some examples.

Running Example (9/9). In our running example, the collaboration is composed by two sound processes. In fact, the Customer process is well-structured, thus sound. Focussing on the process of the Travel Agency, it is also sound since when it completes the terminate end event aborts all the running activities and removes all the tokens still present (more details will follow in Section 3). However, the resulting collaboration is not sound, since the message lists could not be empty. □

Example 4. *In Fig. 21 we have a collaboration resulting from the composition of two sound processes. If we focus only on the processes in* ORG A *and* ORG B *we can immediately note that they are sound. However, the resulting collaboration is not sound. In fact, for instance, if Task A is executed, Task C in* ORG B *will never receive the message and the AND-Join gateway cannot be activated, thus the process of* ORG B *cannot complete its execution.* □

Example 5. *Also the collaboration in Fig. 22 is trivially composed by two sound processes. However, in this case also the resulting collaboration is sound. In fact, Task E will always receive the message by Task B and the processes of* ORG A *and* ORG B *can correctly complete.* □

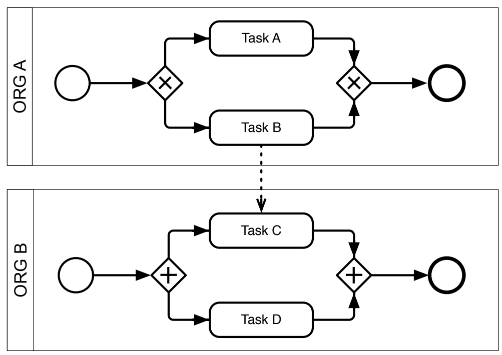


Figure 21: An example of unsound collaboration with sound processes.

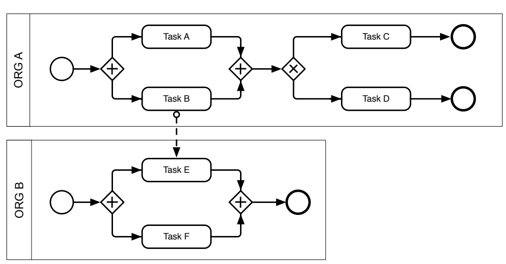


Figure 22: Sound collaboration with sound processes.

Let's now to consider soundness in a multi-layer structure. We show that the composition of unsound sub-processes does not results in un-sound processes. There are also sound processes including unsound sub-process. In fact, when we put unsound sub-process together in a process, this could be either sound or unsound.

Theorem 11. *Let P be a process including a sub-process* $\mathsf{subProc}(\mathsf{e}_i, P_1, \mathsf{e}_o)$*, if P_1 is unsound does not imply P is unsound.*

Proof (sketch). We proceed by contradiction (see Appendix B). □

8. Related Work

In this paper we provide a formal characterisation of well-structuredness BPMN models. To do that we have been inspired by the definition of well-structuredness given in [5]. Other attempts are also available in the literature. Van der Aalst et al. [28] state that a workflow net is well-structured if the split/join constructions are properly nested. El-Saber and Boronat [29] propose a formal definition of well-structured processes, in terms of a rewriting logic, but they do not extend this definition at collaboration level.

We then consider safeness, showing that this is a significant correctness property. Dijkman et al. [4] discuss about safeness in Petri Nets resulting from the translation of BPMN. In such work, safeness of BPMN terms means that no activity will ever be enabled or running more than once concurrently. This definition is given using natural language, while in our work we give a precise characterisation of safeness for both BPMN processes and collaborations. Other approaches introducing mapping from BPMN to formal languages, such as YAWL [30] and COWS [31], do not consider safeness, even if it is recognised as an important characteristic [32].

Moreover, soundness is considered as one of the most important correctness criteria.

There is a jungle of other different notions of soundness in the literature, referring to different process languages and even for the same process language, e.g. for EPC a soundness definition is given by Mendling in [33], and for Workflow Nets by van der Aalst [10] provides two equivalent soundness definitions. However, these definitions cannot be used directly for BPMN because of its peculiarities. In fact, although the BPMN process flow resembles to some extent the behaviour of Petri Nets, it is not the same. BPMN 2.0 provides a comprehensive set of elements that go far beyond the definition of mere place/transition flows and enable modelling at a higher level of abstraction.

Other studies try to characterize inter-organizational soundness are available. A first attempt was done using a framework based on Petri Nets [11]. The authors investigate IO-soundness presenting an analysis technique to verify the correctness of an inter-organizational workflow. However, the study is restricted to structured models. Soundness regarding collaborative processes is also given in [34] in the field of the Global Interaction Nets, in order to detect errors in technology-independent collaborative business processes models. However, differently from our work, this approach does not apply to BPMN, which is the modelling notation aimed by our study. Concerning message-relaxed soundness, we have been motivated by Puhlmann and Weske [16], who define interaction soundness, which in turn is based on lazy soundness [19]. The use of a mapping into π-calculus, rather than of a direct semantics, bases the reasoning on constrains given by the target language. In particular, the authors refer to a synchronous communication model not compliant with the BPMN standard. Our framework instead natively implements the BPMN communication model via an asynchronous approach. Moreover, the interaction soundness assumes structural soundness as a necessary condition that we relax.

Therefore, our investigation of properties at collaboration level provides novel insights with respect to the state-of-the-art of BPMN formal studies.

9. Concluding Remarks

Our study formally defines some important correctness properties, namely well-structuredness, safeness, and soundness, both at the process and collaboration level. We demonstrate the relationships between the studied properties, with the aim of classifying BPMN collaboration diagrams according to the properties they satisfy. Rather than converting the BPMN model to a Petri or Workflow Net and studying relevant properties on the model resulting from the mapping we directly report such properties on BPMN considering its complexity. In doing this the approach is based an uniform formal framework and it is not limited to models of a specific topology.

Specifically, we show that well-structured collaborations represent a subclass of safe ones. In fact, there is a class of collaborations that are safe, even if with an unstructured topology. We also show there are well-structured collaborations that are neither sound nor message-aware sound. These models are typically discarded by the modelling approaches in the literature, as they are over suspected of carrying bugs. However, we have shown that some of these models, hence they can play a significant role in practice. Finally, we demonstrate there are sound and message-aware sound collaborations that are not safe. Resulting classification provide a novel contribution by extending the reasoning from process to BPMN collaborations. Moreover, being close to the BPMN standard give use to catch the language peculiarities as the asynchronous communication models, and the completeness notion distinguishing the effect of end event and the terminate event.

Relevance into Practice. To get a clearer idea of the impact of well-structuredness, safeness, and soundness on the real-world modelling practice, we have analyzed the BPMN 2.0 collaboration models available in a well-known, public, well-populated repository provided by the BPM Academic Initiative (`http://bpmai.org`). From the raw dataset, to avoid uncompleted models and low quality ones, we have selected only those with 100% of connectedness (i.e., all model elements are connected). This results on 2.740 models suitable for our investigation. To better understand the trend in Table 1, the models are grouped in terms of number of contained elements. From the technical point of view, well-structuredness has been checked using the PromniCAT platform[3], while safeness and soundness have been checked using the S^3 tool[4].

We have found that 86% of models in the repository are well-structured. Anyway, more interesting is the trend of the number of well-structured models with respect to their size. It shows that in practice BPMN models starts to become unstructured when their size grows. This means that structuredness should be regarded as a general guideline but one can deviate from it if necessary, especially in modelling complex scenarios. The balancing between the two classes motivates, on the one hand, our design choice of considering in our formalisation BPMN models with an arbitrary topology and, on the other hand, the necessity of studying well-structuredness and the related properties.

Concerning safeness, it results that 2.689 models are safe. The classes that surely cannot be neglected in our study, as they are suitable to model realistic scenarios, are those with size 20-29, 30-39 and 40-49 including 156 models, of which only 3 are unsafe. This makes evident that modelling safe models is part of the practice, and that imposing well-structuredness is sometimes too restrictive, since there is a huge class of models that are safe even if with an unstructured

[3]`https://github.com/tobiashoppe/promnicat`
[4]`http://pros.unicam.it/s3/`

Size	Dataset	WS	Non-WS	Safe	MR-Sound	Sound
0 - 9	1668	1551(93%)	117(7%)	1647	1077	1133
10 - 19	910	692(76%)	218 (24%)	883	462	487
20 - 29	137	95(69%)	42(31%)	134	51	57
30 - 39	13	4 (27%)	9 (73%)	13	4	4
40 - 49	9	1(14%)	8 (86%)	9	3	3
50 - 59	1	0 (0%)	1 (100%)	1	0	0
60 - 69	0	0	0	0	0	0
70 - 79	2	0 (0%)	2 (100%)	2	0	0
0 - 79	2740	2342 (86%)	398 (14%)	2689	1597	1684

Table 1: Classification of the models in the BPM Academic Initiative repository.

topology.

Concerning soundness, it results that there are 1.684 sound models. It results that modelling in a sound way is a common practice, recognizing soundness as one of the most important correctness criteria. Moreover, the data show that there are well-structured models that are not sound this confirm the limitation of well-structuredness. Concerning message-relaxed soundness, it results that the number of models satisfying this property is 87 more than the sound ones. This highlights the relevance of a set of models, up to now, not considered.

Future Work. We plan to continue our programme to effectively reason on the whole set of of BPMN elements included in a collaborations. In particular, we would like to check if the obtained results are still valid in an extended framework.

References

[1] Lindsay, A., Downs, D., Lunn, K.: Business processes—attempts to find a definition. Information and Software Technology **45**(15) (2003) 1015–1019

[2] OMG: Business Process Model and Notation (BPMN V 2.0) (2011)

[3] Suchenia, A., Potempa, T., Ligeza, A., Jobczyk, K., Kluza, K.: Selected Approaches Towards Taxonomy of Business Process Anomalies. In: Advances in Business ICT: New Ideas from Ongoing Research. Volume 658 of SCI. Springer (2017) 65–85

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

[5] Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.J.: On structured workflow modelling. In: Seminal Contributions to Information Systems Engineering, 25 Years of CAiSE. Volume 9539 of LNCS. Springer (2000) 431–445

[6] Dumas, M., La Rosa, M., Mendling, J., Mäesalu, R., Reijers, H.A., Semenenko, N.: Understanding business process models: the costs and benefits of structuredness. In: CAISE. Volume 7328 of LNCS. Springer (2012) 31–46

[7] Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring acyclic process models. Information Systems **37**(6) (2012) 518–538

[8] Polyvyanyy, A., Garcia-Banuelos, L., Fahland, D., Weske, M.: Maximal Structuring of Acyclic Process Models. The Computer Journal **57**(1) (2014) 12–35

[9] van der Aalst, W.M.: Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. In: Business Process Management, Models, Techniques, and Empirical Studies. Volume 1806 of LNCS. Springer (2000) 161–183

[10] van der Aalst, W., van Hee, K., ter Hofstede, A., Sidorova, N., Verbeek, H., Voorhoeve, M., Wynn, M.: Soundness of workflow nets: classification, decidability, and analysis. FAC **23**(3) (2011) 333–363

[11] van der Aalst, W.M.: Process-oriented architectures for electronic commerce and interorganizational workflow. Information Systems **24**(8) (December 1999) 639–671

[12] Murata, T.: Petri nets: Properties, analysis and applications. IEEE Proceedings **77**(4) (1989) 541–580

[13] Rozenberg, G., Engelfriet, J.: Elementary net systems. In: Lectures on Petri Nets I: Basic Models. Springer (1998) 12–121

[14] Muehlen, M., Recker, J.: How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. In: CAISE. Volume 5074 of LNCS. Springer (2008) 465–479

[15] El-Saber, N.: CMMI-CM compliance checking of formal BPMN models using Maude. PhD thesis, University of Leicester - Department of Computer Science (2015)

[16] Puhlmann, F., Weske, M.: Interaction soundness for service orchestrations. In: Service-Oriented Computing. Volume 4294 of Lecture Notes in Computer Science., Springer (2006) 302–313

[17] Dehnert, J., Zimmermann, A.: On the suitability of correctness criteria for business process models. In: BPM. Volume 3649 of LNCS. Springer (2005) 386–391

[18] van der Aalst, W.M.: Structural characterizations of sound workflow nets. Computing Science Reports **96**(23) (1996) 18–22

[19] Puhlmann, F., Weske, M.: Investigations on soundness regarding lazy activities. In: Business Process Management. Volume 4102 of Lecture Notes in Computer Science., Springer (2006) 145–160

[20] van Hee, K., Oanea, O., Serebrenik, A., Sidorova, N., Voorhoeve, M.: History-based joins: Semantics, soundness and implementation. In: International Conference on Business Process Management, Springer (2006) 225–240

[21] Van der Aalst, W.M.: Verification of workflow nets. In: International Conference on Application and Theory of Petri Nets, Springer (1997) 407–426

[22] Favre, C., Völzer, H.: Symbolic execution of acyclic workflow graphs. Business Process Management (2010) 260–275

[23] Prinz, T.M.: Fast soundness verification of workflow graphs. In: ZEUS. Volume 1029 of LNCS. Springer (2013) 31–38

[24] Kunze, M., Weske, M.: Behavioural Models - From Modelling Finite Automata to Analysing Business Processes. Springer (2016)

[25] Kheldoun, A., Barkaoui, K., Ioualalen, M.: Formal verification of complex business processes based on high-level Petri nets. Information Sciences **385-386** (April 2017) 39–54

[26] Ter Hofstede, A.: Workflow patterns: On the expressive power of (petri-net-based) workflow languages. PhD thesis, University of Aarhus (2002)

[27] Corradini, F., Fornari, F., Polini, A., Re, B., Tiezzi, F.: A formal approach to modeling and verification of business process collaborations. Science of Computer Programming **166** (2018) 35–70

[28] Van Der Aalst, W.M.: The Application of Petri Nets to Workflow Management. Journal of Circuits, Systems and Computers **08**(01) (1998) 21–66

[29] El-Saber, N., Boronat, A.: BPMN Formalization and Verification Using Maude. In: Workshop on Behaviour Modelling-Foundations and Applications, ACM (2014) 1–12

[30] Decker, G., Dijkman, R., Dumas, M., García-Bañuelos, L.: Transforming BPMN diagrams into YAWL nets. In: BPM. Volume 5240 of LNCS. Springer (2008) 386–389

[31] Prandi, D., Quaglia, P., Zannone, N.: Formal Analysis of BPMN Via a Translation into COWS. In: Coordination Models and Languages. Volume 5052 of LNCS. Springer (2008) 249–263

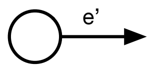
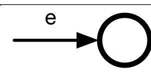
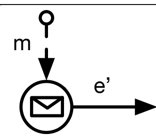
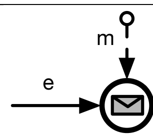
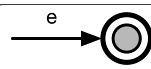
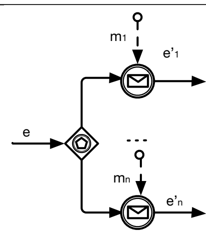
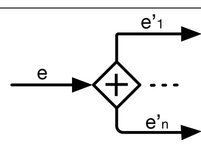
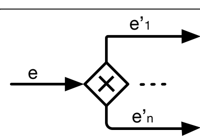
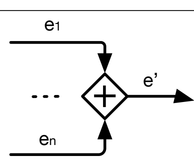
[32] Cheng, A., Esparza, J., Palsberg, J.: Complexity results for 1-safe nets. In: Foundations of Software Technology and Theoretical Computer Science. Volume 761 of LNCS. Springer (1993) 326–337

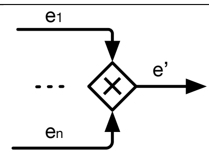
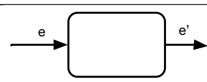
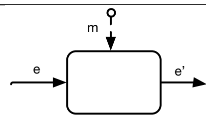
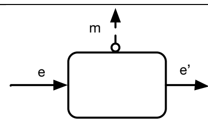
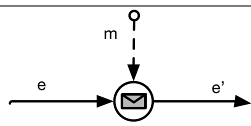
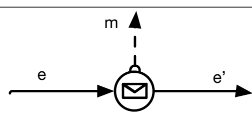
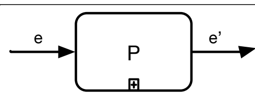
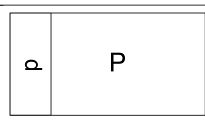
[33] Mendling, J.: Detection and prediction of errors in EPC business process models. PhD thesis, Wirtschaftsuniversität Wien Vienna (2007)

[34] Roa, J., Chiotti, O., Villarreal, P.: A verification method for collaborative business processes. In: International Conference on Business Process Management. Volume 99 of LNBIP. Springer (2011) 293–305

Appendix A. Correspondence

Here we reported the complete correspondence between the BPMN graphical notation and our syntax. For the sake of presentation, join and split gateways include only three incoming/outgoing branching respectively.

Graphical Representation	Textual Notation
	$\mathsf{start}(\mathsf{e}, \mathsf{e}')$
	$\mathsf{end}(\mathsf{e}, \mathsf{e}')$
	$\mathsf{startRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$
	$\mathsf{endSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$
	$\mathsf{terminate}(\mathsf{e})$
	$\mathsf{eventBased}(\mathsf{e}, (\mathsf{m}_1, \mathsf{e}'_1), \ldots, , (\mathsf{m}_n, \mathsf{e}'_n))$
	$\mathsf{andSplit}(\mathsf{e}, \{\mathsf{e}'_1, \ldots, \mathsf{e}'_n\})$
	$\mathsf{xorSplit}(\mathsf{e}, \{\mathsf{e}'_1, \ldots, \mathsf{e}'_n\})$
	$\mathsf{andJoin}(\{\mathsf{e}_1, \ldots, \mathsf{e}_n\}, \mathsf{e}')$

Graphical Representation	Textual Notation
	$\mathsf{xorJoin}(\{\mathsf{e}_1, \ldots, \mathsf{e}_n\}, \mathsf{e}')$
	$\mathsf{task}(\mathsf{e}, \mathsf{e}')$
	$\mathsf{taskRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$
	$\mathsf{taskSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$
— —	$\mathsf{empty}(\mathsf{e}, \mathsf{e}')$
	$\mathsf{interRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$
	$\mathsf{interSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$
	$\mathsf{subProc}(\mathsf{e}, P, \mathsf{e}')$
	$\mathsf{pool}(\mathsf{p}, P)$

Appendix B. Proofs

In this appendix we report the proofs of the results presented in the paper.

Lemma 1. *Let P be a process, if $isInit(P, \sigma)$ then $\langle P, \sigma \rangle$ is cs-safe.*

Proof. Trivially, from definition of $isInit(P, \sigma)$. By definition of $isInit(P, \sigma)$, we have that $\sigma(\mathsf{e}) = 1$ where $\mathsf{e} \in start(P)$ and $\forall\ \mathsf{e}' \in \mathbb{E} \backslash start(P)\ .\ \sigma(\mathsf{e}') = 0$, i.e. only the start event has a marking and all the other edges are unmarked. Hence, we have that $maxMarking(P, \sigma) \leqslant 1$, which allows us to conclude. □

Lemma 2. *Let $isWSCore(P)$, and let $\langle P, \sigma \rangle$ be a core reachable and cs-safe process configuration, if $\langle P, \sigma \rangle \xrightarrow{\alpha} \sigma'$ then $\langle P, \sigma' \rangle$ is cs-safe.*

Proof. We proceed by induction on the structure of WSCore process elements.
Base cases: we show here only few interesting cases among the multiple base cases; since by hypothesis $isWSCore(P)$, it can only be either a task or an intermediate event. Let us consider the simple task, since all the other cases are similar. □

- $P = \mathsf{task}(\mathsf{e}, \mathsf{e}')$. By hypothesis $\langle P, \sigma \rangle$ is cs-safe, then $maxMarking(P, \sigma) = \max(\sigma(\mathsf{e}), \sigma(\mathsf{e}')) \leqslant 1$. The only rule that can be applied to infer the transition $\langle P, \sigma \rangle \xrightarrow{\alpha} \sigma'$ is $P\text{-}Task$. In order to apply the rule there must be $0 < \sigma(\mathsf{e})$; hence $0 < \sigma(\mathsf{e}) \leqslant 1$, i.e. $\sigma(\mathsf{e}) = 1$. We can exploit the fact that $\langle P, \sigma \rangle$ be is a core reachable configuration to prove that $\sigma(\mathsf{e}') = 0$. The application of the rule produces $\sigma' = \langle inc(dec(\sigma, \mathsf{e}), \mathsf{e}') \rangle$, i.e. $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$. Thus, $maxMarking(P, \sigma') = \sigma'(\mathsf{e}')$. Since $\sigma'(\mathsf{e}') = 1$ we have that $maxMarking(P, \sigma') \leqslant 1$, which allows us to conclude.

Inductive cases: we consider the following cases, the other are deal with similarly.

- Let us consider $\langle \mathsf{andSplit}(\mathsf{e}, E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{andJoin}(E', \mathsf{e}'), \sigma \rangle$. There are the following possibilities:
  - $\langle \mathsf{andSplit}(\mathsf{e}, E), \sigma \rangle$ evolves by means of rule $P\text{-}AndSplit$. We can exploit the fact that this is a core reachable well-structured configuration to prove that $\sigma(\mathsf{e}) = 1$ and $\forall \mathsf{e}'' \in E\ .\sigma(\mathsf{e}'') = 0$. Thus, $\langle \mathsf{andSplit}(\mathsf{e}, E), \sigma \rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = inc(dec(\sigma, \mathsf{e}), E)$. Hence, $maxMarking(\mathsf{andSplit}(\mathsf{e}, E), \sigma') = 1$. By hypothesis $\langle \mathsf{andSplit}(\mathsf{e}, E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{andJoin}(E', \mathsf{e}'), \sigma \rangle$ is cs-safe, i.e. if $\forall \mathsf{e}'' \in E\ .\sigma'(\mathsf{e}'') = 1$, that is there is a token on the outgoing edges of the AND-Split in the state $\langle \mathsf{andSplit}(\mathsf{e}, E), \sigma' \rangle$, then all the other edges are unmarked. This means that cs-safeness is not affected. Therefore, the overall term is cs-safe.

- Node $P_1 \parallel \ldots \parallel P_n$ evolves without affecting the split and join gateways. In this case we can easily conclude by inductive hypothesis.
- Node $P_1 \parallel \ldots \parallel P_n$ evolves and affects the split and/or join gateways. In this case we can reason like in the first case, by relying on inductive hypothesis.
- $\langle \mathsf{andJoin}(E', \mathsf{e}'), \sigma \rangle$ evolves by means of rule *P-AndJoin*. We can exploit the fact that this is a core reachable well-structured configuration to prove that $\forall \mathsf{e}'' \in E' . \sigma(\mathsf{e}'') = 1$ and $\sigma(\mathsf{e}') = 0$. Thus $\langle \mathsf{andJoin}(E', \mathsf{e}'), \sigma \rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = inc(dec(\sigma, E'), \mathsf{e}')$. Hence, $maxMarking(\mathsf{andJoin}(E', \mathsf{e}'), \sigma') = 1$. By hypothesis $\langle \mathsf{andSplit}(E, \mathsf{e}) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{andJoin}(E', \mathsf{e}'), \sigma \rangle$ is cs-safe, i.e. if there is a token on the outgoing edge of the AND-Join in the state $\langle \mathsf{andJoin}(E', \mathsf{e}'), \sigma' \rangle$ all the other edges do not have tokens. This means that cs-safeness is not affected. Therefore, the overall term is cs-safe.

- Let us consider $\mathsf{xorJoin}(\{\mathsf{e}_2, \mathsf{e}_3\}, \mathsf{e}_1) \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}_4, \{\mathsf{e}_5, \mathsf{e}_6\})$ with $in(P_1) = \{\mathsf{e}_1\}, out(P_1) = \{\mathsf{e}_4\}, in(P_2) = \{\mathsf{e}_6\}, out(P_2) = \{\mathsf{e}_2\}$
  - $\langle \mathsf{xorJoin}(\{\mathsf{e}_2, \mathsf{e}_3\}, \mathsf{e}_1), \sigma \rangle$ evolves by means of rule *P-XorJoin*. We can exploit the fact that this is a core reachable well-structured configuration to prove that the term is marked $\sigma(\mathsf{e}_1) = 0$ and either $\sigma(\mathsf{e}_2) = 1$ or $\sigma(\mathsf{e}_3) = 1$; let us assume the marking is $\sigma(\mathsf{e}_3) = 1$ (since the other case is similar). Thus $\langle \mathsf{xorJoin}(\{\mathsf{e}_2, \mathsf{e}_3\}, \mathsf{e}_1), \sigma \rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = inc(dec(\sigma, \mathsf{e}_2), \mathsf{e}_1)$. Hence, $maxMarking(\mathsf{xorJoin}(\{\mathsf{e}_2, \mathsf{e}_3\}, \mathsf{e}_1), \sigma' \rangle) = 1$. By hypothesis $\langle \mathsf{xorJoin}(\{\mathsf{e}_2, \mathsf{e}_3\}, \mathsf{e}_1), \sigma \rangle$ is cs-safe, i.e. if there is a token on e_1 in the state $\langle \mathsf{xorJoin}(\{\mathsf{e}_2, \mathsf{e}_3\}, \mathsf{e}_1), \sigma' \rangle$ all the other edges do not have token. This means that cs-safeness is not affected. Therefore, the overall term is cs-safe.
  - Node $P_1 \parallel P_2$ evolves without affecting the split and join gateways. In this case we can easily conclude by inductive hypothesis.
  - Node $P_1 \parallel P_2$ evolves and affects the split xor join and xor split gateways. In this case we can reason like in the first case, by relying on inductive hypothesis.
  - $\langle \mathsf{xorSplit}(\mathsf{e}_4, \{\mathsf{e}_5, \mathsf{e}_6\}), \sigma \rangle$ evolves by means of rule *P-XorSplit*. We can exploit the fact that this is a core reachable well-structured configuration to prove that the term is marked as $\sigma(\mathsf{e}_4) = 1$. Hence, it evolves in a cs-safe term; in fact let us assume that it evolves in this way $\langle \mathsf{xorSplit}(\mathsf{e}_4, \{\mathsf{e}_5, \mathsf{e}_6\}), \sigma \rangle \xrightarrow{\mathsf{e}_i} \sigma'$ with $\sigma' = inc(dec(\sigma, \mathsf{e}_4), \mathsf{e}_5)$. Hence, $maxMarking(\mathsf{xorSplit}(\mathsf{e}_4, \{\mathsf{e}_5, \mathsf{e}_6\}), \sigma') = 1$. By hypothesis $\langle \mathsf{xorJoin}(\{\mathsf{e}_2, \mathsf{e}_3\}, \mathsf{e}_1) \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}_4, \{\mathsf{e}_5, \mathsf{e}_6\}), \sigma \rangle$ is cs-safe, i.e. if there is a token on e_5 in the state $\langle \mathsf{xorSplit}(\mathsf{e}_4, \{\mathsf{e}_5, \mathsf{e}_6\}), \sigma' \rangle$ all the other

edges do not have token. This means that cs-safeness is not affected. Therefore, the overall term is cs-safe.

- Let us consider $\langle P, \sigma \rangle = \langle P_1 \parallel P_2, \sigma \rangle$. The relevant case for cs-safeness is when P evolves by applying $P\text{-}Int_1$. We have that $\langle P_1 \parallel P_2, \sigma \rangle \xrightarrow{\alpha} \sigma'$ with $\langle P_1, \sigma \rangle \xrightarrow{\alpha} \sigma'$. By definition of maxMarking function we have that $maxMarking(P, \sigma)$ = $\max(maxMarking(P_1, \sigma), maxMarking(P_2, \sigma))$. By inductive hypothesis we have that $maxMarking(P_1, \sigma) = maxMarking(P_1, \sigma') \leqslant 1$ which is cs-safe. Since P_2 is well structured and cs-safe, then also $\langle P_2, \sigma' \rangle$ is cs-safe, which permits us to conclude.

$\square$

Lemma 3. *Let P be WS, and let $\langle P, \sigma \rangle$ be a process configuration reachable and cs-safe, if $\langle P, \sigma \rangle \xrightarrow{\alpha} \sigma'$ then $\langle P, \sigma' \rangle$ is cs-safe.*

***Proof*.** According to Definition 4, P can have 6 different forms. We proceed by case analysis on the parallel component of $\langle P, \sigma \rangle$ that causes the transition $\langle P, \sigma \rangle \xrightarrow{\alpha} \sigma'$.

We show now the case P= $\mathsf{start}(\mathsf{e}, \mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'', \mathsf{e}''')$.

- $\mathsf{start}(\mathsf{e}, \mathsf{e}')$ evolves by means of the rule *P-$Start$*. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$, hence, by cs-safeness, $\sigma(\mathsf{e}) = 1$. We can exploit the fact that this is a reachable well-structured configuration to prove that $\sigma(\mathsf{e}') = 0$. The rule produces the following transition $\langle \mathsf{start}(\mathsf{e}, \mathsf{e}'), \sigma \rangle \xrightarrow{\epsilon} \sigma'_1$ with $\sigma'_1 = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$ where $\sigma'_1(\mathsf{e}) = 0$ and $\sigma'_1(\mathsf{e}') = 1$. Now, $\langle P, \sigma'_1 \rangle = \langle \mathsf{start}(\mathsf{e}, \mathsf{e}') \parallel P' \mid\mid \mathsf{end}(\mathsf{e}'', \mathsf{e}'''), \sigma'_1 \rangle$ can evolve only through the application of $P\text{-}Int_1$ producing $\langle P, \sigma' \rangle$ with $\sigma'(in(P')) = 1$.

  By hypothesis $\langle P, \sigma \rangle$ is cs-safe, thus $\sigma(\mathsf{e}'') \leqslant 1$, $\sigma(\mathsf{e}''') \leqslant 1$ and $\max(\sigma(\mathsf{edges}(P'))) \leqslant 1$. Now $maxMarking(P', \sigma) \leqslant 1$ and $maxMarking(P', \sigma') \leqslant 1$. Therefore $maxMarking(P, \sigma') = max(0, 1, \sigma'(in(P')), \sigma'(out(P')), \sigma'(\mathsf{e}''), \sigma'(\mathsf{e}''')) \leqslant 1$, then $\langle P, \sigma' \rangle$ is cs-safe.

- $\mathsf{end}(\mathsf{e}'', \mathsf{e}''')$ evolves by means of the rule *P-End*. We can exploit the fact that this is a reachable well-structured configuration to prove that the term is marked as $\sigma(\mathsf{e}'') = 1$ and $\sigma(\mathsf{e}''') = 0$. The rule produces the following transition $\langle \mathsf{end}(\mathsf{e}'', \mathsf{e}'''), \sigma \rangle \xrightarrow{\epsilon} \langle inc(dec(\sigma, \mathsf{e}''), \mathsf{e}''') \rangle$. Now, $\langle P, \sigma \rangle$ can only evolve by applying $P\text{-}Int_1$ producing $\langle P, \sigma' \rangle$.

  By hypothesis $\langle P, \sigma \rangle$ is cs-safe, then $\sigma(\mathsf{e}'') \leqslant 1$, $\sigma(\mathsf{e}''') \leqslant 1$ and P' is cs-safe. Reasoning as previously we can conclude that $\langle P, \sigma' \rangle$ is cs-safe.

- P' moves, that is $\langle P', \sigma\rangle \xrightarrow{\alpha} \sigma'$. By Lemma 2 $\langle P', \sigma'\rangle$ is safe, thus $maxMarking(P', \sigma') \leqslant 1$. By hypothesis, P is cs-safe therefore $maxMarking(\mathsf{start}(\mathsf{e}, \mathsf{e}'), \sigma') \leqslant 1$, $maxMarking(\mathsf{end}(\mathsf{e}'', \mathsf{e}'''), \sigma') \leqslant 1$. We can conclude that $\langle P, \sigma'\rangle$ is safe.

Now we consider the case P= $\mathsf{start}(\mathsf{e}, \mathsf{e}') \parallel P' \parallel \mathsf{terminate}(\mathsf{e}'')$.

- The start event evolves: like the previous case.
- The end terminate event evolves: the only transition we can apply is *P*-$Terminate$. By applying the rule we have$\langle \mathsf{terminate}(\mathsf{e}''), \sigma\rangle \xrightarrow{kill} dec(\sigma, \mathsf{e}'')$ with $\sigma(\mathsf{e}'') > 0$. Now, $\langle P, \sigma\rangle$ can only evolve by applying $P\text{-}Kill_1$ producing $\langle P, \sigma'\rangle$ where σ' is completed unmarked; therefore it is cs-safe.
- P' moves: similar to the previous case.

□

Theorem 1. *Let P be a process, if P is well-structured then P is safe.*

Proof. We have to show that if $\langle P, \sigma\rangle \rightarrow^* \sigma'$ then $\langle P, \sigma'\rangle$ is cs-safe. We proceed by induction on the length n of the sequence of transitions from $\langle P, \sigma\rangle$ to $\langle P, \sigma'\rangle$.
Base Case ($n = 0$): In this case $\sigma = \sigma'$, then $isInit(P, \sigma')$ is satisfied. By Lemma 1 we conclude $\langle P, \sigma'\rangle$ is cs-safe.
Inductive Case: In this case $\langle P, \sigma\rangle \rightarrow^* \langle P, \sigma''\rangle \xrightarrow{\alpha} \langle P, \sigma'\rangle$ for some process $\langle P, \sigma''\rangle$. By induction, $\langle P, \sigma''\rangle$ is cs-safe. By applying Lemma 3 to $\langle P, \sigma''\rangle \xrightarrow{\alpha} \langle P, \sigma'\rangle$, we conclude $\langle P, \sigma'\rangle$ is cs-safe. □

Theorem 2. *Let C be a collaboration, if C is well-structured then C is safe.*

Proof. By contradiction, let us assume C is well-structured and C is unsafe. By Definition 8, there exists a collaboration configuration $\langle C, \sigma', \delta'\rangle$ such that $\langle C, \sigma, \delta\rangle \rightarrow^* \langle C, \sigma', \delta'\rangle$ and $maxMarking(C, \sigma') > 1$ and $\langle P, \sigma'\rangle$ not cs-safe. Thus, there exists P in C such that $\langle P, \sigma\rangle \rightarrow^* \langle P, \sigma'\rangle$. From hypothesis $isInit(C, \delta)$, we have $isInit(P, \sigma)$. From hypothesis C is well-structured, we have that P is WS. Therefore, by Theorem 1, P is safe. By Definition 7, $\langle P, \sigma'\rangle$ is cs-safe, which is a contradiction. □

Lemma 4. *Let $isWSCore(P)$ and let $\langle P, \sigma\rangle$ be core reachable, then there exists σ' such that $\langle P, \sigma\rangle \rightarrow^* \sigma'$ and $isCompleteEl(P, \sigma')$.*

Proof. We proceed by induction on the structure of $isWSCore(P)$. Base cases: by definition of $isWSCore()$, P can only be either a task or an intermediate event; we show here only the case in which it is a non communicating task, the other are dealt with similarly.

- $P = \mathsf{task}(\mathsf{e}, \mathsf{e}')$. The only rule we can apply is P-$Task$. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$. Since $isWSCore(P)$, $\langle P, \sigma\rangle$ is safe, hence $\sigma(\mathsf{e}) = 1$. Since the process configuration is core reachable we have $\sigma(\mathsf{e}') = 0$. The application of the rule produces $\langle \mathsf{task}(\mathsf{e}, \mathsf{e}'), \sigma\rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$. Thus, we have $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$, which permits us to conclude.

Inductive cases: we consider one case, the other are dealt with similarly.

- Let us consider $P = \langle \mathsf{andSplit}(\mathsf{e}, E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{andJoin}(E', \mathsf{e}'), \sigma\rangle$. There are the following possibilities:
  - $\langle \mathsf{andSplit}(\mathsf{e}, E), \sigma\rangle$ evolves by means of rule P-$AndSplit$. We can exploit the fact that this is a core reachable well-structured configuration to prove that $\sigma(\mathsf{e}) = 1$ and $\forall \mathsf{e}'' \in E\ .\sigma(\mathsf{e}'') = 0$. Thus, $\langle \mathsf{andSplit}(\mathsf{e}, E), \sigma\rangle \xrightarrow{\epsilon} \langle inc(dec(\sigma, \mathsf{e}), E)\rangle$. Now, P can evolve only through the application of P-Int_1 producing $\langle P, \sigma'_1\rangle$ with $\sigma'_1(in(P_1)) = \ldots = \sigma''(in(P_n)) = 1$. By inductive hypothesis there exists a state σ'_1 such that $isCompleteEl(P_1 \mid \ldots \mid P_n, \sigma'_1)$. Now, P can only evolve by applying rule P-Int_1, producing $\langle P, \sigma'_2\rangle$ with $\sigma'_2(\mathsf{edges}(E')) = 1$. Now, $\langle \mathsf{andJoin}(E', \mathsf{e}'), \sigma'_2\rangle$ can evolve by means of rule P-$AndJoin$. The application of the rule produces $\langle \mathsf{andJoin}(E', \mathsf{e}'), \sigma_2\rangle \xrightarrow{\epsilon} \sigma'_3$ with $\sigma'_3 = inc(dec(\sigma'_2, E'), \mathsf{e}')$, i.e. $\sigma'_3(\mathsf{e}') = 1$ and $\forall \mathsf{e}''' \in E'\ .\sigma'_3(\mathsf{e}''') = 0$. This permits us to conclude.
  - $P_1 \parallel \ldots \parallel P_n$ evolves without affecting the split and join gateways. In this case we can easily conclude by inductive hypothesis.
  - $P_1 \parallel \ldots \parallel P_n$ evolves and affects the split and/or join gateways. In this case we can reason like in the first case

□

Theorem 3. *Let $\langle P, \sigma\rangle$ be a WS process configuration, then $\langle P, \sigma\rangle$ is sound.*

Proof. According to Definition 4, P can have 6 different forms. We consider now the case P= $\mathsf{start}(\mathsf{e}, \mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'', \mathsf{e}''')$.

Let us assume that $isInit(P, \sigma)$. Thus we have that $\sigma(start(P)) = 1$, and $\forall\ \mathsf{e}^{\mathsf{iv}} \in \mathsf{edges}(P) \backslash start(P)\ .\ \sigma(\mathsf{e}^{\mathsf{iv}}) = 0$. Therefore the only parallel component of P that can infer a transition is the start event. In this case we can apply only the rule P-$Start$. The rule produces the following transition, $\langle \mathsf{start}(\mathsf{e},\mathsf{e}'),\sigma\rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = inc(dec(\sigma,\mathsf{e}),\mathsf{e}')$ where $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$. Now $\langle P,\sigma'\rangle$ can evolve through the application of rule $P\text{-}Int_1$ producing $\langle P,\sigma'_1\rangle$, with $\sigma'_1(in(P')) = 1$. Now P' moves. By hypothesis $isWSCore(P')$, thus by Lemma 4 there exists a process configuration $\langle P',\sigma'_2\rangle$ such that $\langle P',\sigma'_1\rangle \rightarrow^* \sigma'_2$ and $isCompleteEl(P',\sigma'_2)$. The process can now evolve thorough rule $P\text{-}Int_1$. By hypothesis the process is WS, thus, after the application of the rule we obtain $\langle \mathsf{start}(\mathsf{e},\mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'',\mathsf{e}'''),\sigma'_3\rangle$, where $\sigma'_3(\mathsf{e}'') = 1$ and $\forall \mathsf{e}^{\mathsf{v}} \in \mathsf{edges}(P') \,.\, \sigma'_3(\mathsf{e}^{\mathsf{v}}) = 0$. We can now apply rule $P\text{-}End$ that decrements the token in e_i and produces a token in e_{cmp}, which permits us to conclude. □

Theorem 4. *Let C be a collaboration, C is WS does not imply C is sound.*

Proof. Let C be a WS collaboration, and let us suppose that C is sound. Then, it is sufficient to show a counter example, i.e. a WS collaboration that is not sound. Let us consider, for instance, the collaboration in Fig. B.23. By Definition, the collaboration is WS. The soundness of the collaboration instead depends on the evaluation of the condition of the XOR-Split gateway in ORG A. If a token is produced on the upper flow and Task A is executed then Task C in ORG B will never receive the message and the AND-Join gateway can not be activated, thus the process of ORG B can not reach a marking where the end event has a token. □

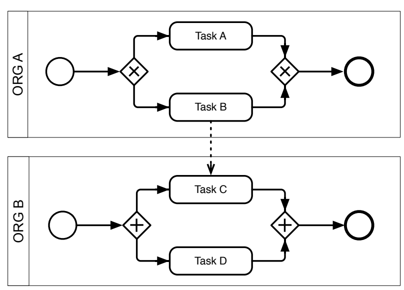


Figure B.23: An example of unsound collaboration with sound WS processes.

Theorem 5. *Let C be a collaboration, C is WS does not imply C is message-relaxed sound.*

Proof. Let C be a WS collaboration, and let us suppose that C is message-relaxed sound. Then, it is sufficient to show a counter example, i.e. a WS collaboration that is not message-relaxed sound. We can consider again the collaboration in Fig. B.23. By reasoning as previously, the message-relaxed soundness of the collaboration depends on the evaluation of the condition of the XOR-Split gateway in ORG A. This permits us to conclude. □

Theorem 6. *Let P be a collaboration, P is unsafe does not imply P is unsound.*

***Proof*.** Let P be a unsafe collaboration, and let us suppose that P is unsound. Then, it is sufficient to show a counter example, i.e. a unsafe collaboration that is sound. We can consider the process in Fig. B.24. It is unsafe since the AND split gateway creates two tokens that are then merged by the XOR join gateway producing two tokens on the outgoing edge of the XOR join. However, after Task C is executed and one token enables the terminate end event, the $kill$ label is produced and the second token in the sequence flow is removed (rule *P-Terminate*), rendering the process sound. □

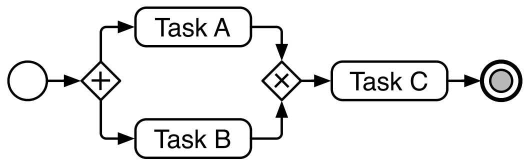


Figure B.24: An example of unsafe but sound process.

Theorem 7. *Let C be a collaboration, C is unsafe does not imply C is unsound.*

***Proof*.** Let C be a unsafe collaboration, and let us suppose that C is unsound. Then, it is sufficient to show a counter example, i.e. a unsafe collaboration that is sound. We can consider the collaboration in Fig. B.25. Process in ORG A and ORG B are trivially unsafe, since the AND split gateway generates two tokens that are then merged by the XOR join gateway producing two tokens on the outgoing edge of the XOR join. By definition of safeness collaboration the considered collaboration is unsafe. Concerning soundness, processes of ORG B and ORG A are sound. In fact, in each process, after one token enables the terminate end event, the kill label is produced and the second token in the sequence flow is removed (rule P-Terminate), resulting in a marking where all edges are unmarked. Thus, the resulting collaboration is sound. □

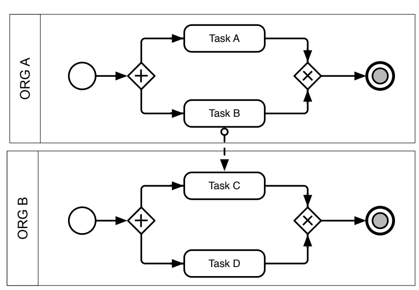


Figure B.25: An example of unsafe but sound collaboration.

Theorem 8. *Let C be a collaboration, if all processes in C are safe then C is safe.*

Proof. By contradiction let C be unsafe, i.e. there exists a collaboration $\langle C, \sigma', \delta' \rangle$ such that $\langle C, \sigma, \delta \rangle \rightarrow^* \langle \sigma', \delta' \rangle$ with $\mathsf{pool}(\mathsf{p}, P)$ in C and $\langle P, \sigma' \rangle$ not cs-safe. By hypothesis all processes of C are safe, hence it is safe the process, say P, of organisation p. As $\langle C, \sigma', \delta' \rangle$ results from the evolution of $\langle C, \sigma, \delta \rangle$, the process $\langle P, \sigma' \rangle$ must derive from $\langle P, \sigma \rangle$ as well, that is $\langle P, \sigma \rangle \rightarrow^* \sigma'$. By safeness of P, we have that $\langle P, \sigma' \rangle$ is cs-safe, which is a contradiction. □

Theorem 9. *Let P be a process including a sub-process $\mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}')$, if P_1 is unsafe then P is unsafe.*

Proof. Let us suppose $P = \mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}') \parallel P_2$ By contradiction let P be safe, i.e. given σ such that $isInit(P, \sigma)$, for all σ' such that $\langle P, \sigma \rangle \rightarrow^* \sigma'$ we have that $\langle P, \sigma' \rangle$ is cs-safe. By hypothesis P_1 is unsafe, i.e. given σ'_1 such that $isInit(P_1, \sigma'_1)$, there exists σ'_2 such that $\langle P_1, \sigma'_1 \rangle \rightarrow^* \sigma'_2$ and $\langle P_1, \sigma'_2 \rangle$ not cs-safe. Thus, we have $maxMarking(P_1, \sigma'_2) \geqslant 1$. By definition of function $maxMarking()$, we have that $maxMarking(P, \sigma'_2) = \max(maxMarking(\mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}')), maxMarking(P_2)) = maxMarking(P_1, \sigma'_2) \geqslant 1$. Thus, P is not cs-safe, which is a contradiction. □

Theorem 10. *Let C be a collaboration, if some processes in C are unsound then C is unsound.*

Proof. Let P_1 and P_2 be two processes such that P_1 is unsound, and let C be the collaboration obtained putting together P_1 and P_2. By contradiction let C be sound, i.e., given σ and δ such that $isInit(C, \sigma, \delta)$, for all σ' and δ' such that $\langle C, \sigma, \delta \rangle \rightarrow^* \langle \sigma', \delta' \rangle$ we have that there exist σ'' and δ'' such that $\langle C, \sigma', \delta' \rangle \rightarrow^* \langle \sigma'', \delta'' \rangle$, and $\forall\ P$ in C we have that $\langle P, \sigma'' \rangle$ is cs-sound and $\forall\, \mathsf{m} \in \mathbb{M}\ .\ \delta''(\mathsf{m}) = 0$. Since P_1 is unsound, we have that, given σ'_1, such that $isInit(P_1, \sigma'_1)$, for all σ'_2 such that $\langle P_1, \sigma \rangle \rightarrow^* \sigma'_2$ we have that does not exist σ'_3 such that $\langle P_1, \sigma'_2 \rangle \rightarrow^* \sigma'_3$, and $\langle P_1, \sigma'_3 \rangle$ is cs-sound. Choosing $\langle C, \sigma', \delta' \rangle$ such that $\mathsf{pool}(\mathsf{p}, P_1)$ in C', by unsoundness of P_1 we have that there exists a process in C' that is not cs-sound, which is a contradiction. □

Theorem 11. *Let P be a process including a sub-process $\mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}')$, if P_1 is unsound does not imply P is unsound.*

Proof. Let P_1 be a unsound, and let us suppose that P is unsound. Then, it is sufficient to show a counter example, i.e. an sound process including an unsound sub-process. We can consider process in Fig. B.26. The process is unsound since when there is a token in the end event of ORG A there is still a pending sequence token to be consumed. If we include the part of the model generating multiple tokens in the scope of a sub-process, as it is shown in Fig. B.27, that is when the process includes a sub-process, the process is sound. In fact, when there is a token in the end event of ORG A no other pending sequence tokens need to be processed. □

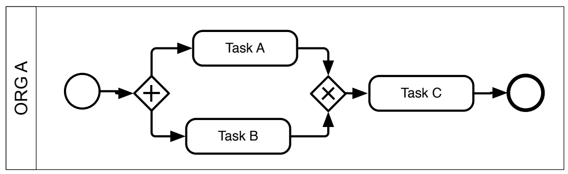


Figure B.26: An example of unsound process.

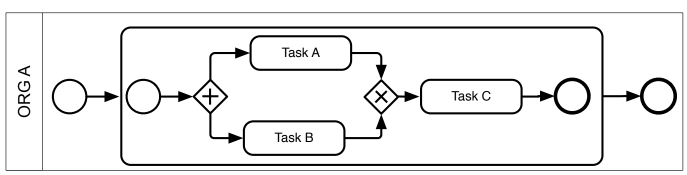


Figure B.27: An example of sound process with unsound sub-process.