Well-structuredness, Safeness and Soundness: A Formal Classification of BPMN (Collaborations)

Flavio Corradini, Chiara Muzi, Andrea Morichetta, Barbara Re, Francesco Tiezzi

School of Science and Technology, University of Camerino, Italy

Abstract

The BPMN standard has a huge uptake in modelling business processes within the same organisation or collaborations involving multiple interacting participants. It is widely accepted by the Business Process Management community that a solid formal framework for the notation can help designers to properly understand their BPMN models as well as to state and verify model properties. With this aim in mind, we provide a formal characterisation of BPMN collaborations and some of the most significant 'correctness' properties in the business process domain; namely, well-structuredness, safeness and soundness. We exploit this formalisation to classify BPMN models according to the properties they satisfy and their compositionality, resulting in a systematic study that gives evidence of expected results, closes conjectures and provides novel results. An experimentation to assess the impact of the considered properties on the practice of modelling is carried out on the BPMN models available in a public and populated repository.

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

1. Introduction

Modern organisations recognise the importance of having tools to support and achieve their own objectives. This is properly 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 organisations*" [1].

Several languages have been proposed to model business processes and collaborations. The Object Management Group (OMG) standard Business Process Model and Notation (BPMN) [2] is the most prominent language. In particular, BPMN collaboration models are used to describe distributed and complex scenarios, where multiple participants interact each other via the exchange of messages.

Even though it is widely accepted in both academia and industry, BPMN's major drawback is due to possible misunderstanding of its semantics. It is described in natural language, often ambiguous and sometimes containing misleading information [3]. Much effort has been devoted to formalise BPMN semantics by mapping business processes and collaborations into formal notations (e.g., see [4] for an approach based on Petri Nets). Of course, the resulting models inherit constraints proper of the target language the mapping considers. Consequently, none of them takes into account BPMN features such as: different abstraction levels (i.e., sub-processes, processes and collaborations), asynchronous communication paradigms, notions of completion due to different types of 'end event' (i.e., simple, message throwing and terminate).

In this paper, we provide a formal characterisation of BPMN collaborations and of some structural and behavioural properties. It allows BPMN designers to properly understand their models and their expressiveness, and turns out to be a formal framework supporting the modelling and analysis of BPMN collaborations within their lifecycle.

The formalisation of the BPMN collaborations follows a process description language paradigm, with a formal syntax and an operational semantics describing the step-by-step behaviour of the collaborations. It faithfully extends [5] with a textual notation (instead of a graphical one) and takes into account a larger language (including, e.g., sub-processes).

The formalisation of BPMN model properties takes into account well-known 'correctness' properties in the domain of Business Process Management; namely well-structuredness [6], safeness [7, 8] and soundness [9, 10]. Despite the large body of work on this topic, no formal definition of these properties directly given on BPMN was provided yet, being instead proposed on different notations (as, for instance, Petri Nets [11, 8], Workflow Nets [6, 8, 10] and Elementary Nets [7]). Having a uniform formal framework allowed us to study the relationships between the considered properties and to classify BPMN models according to the properties they satisfy. It turns out that a well-structured collaboration is always safe, but not the vice versa. Well-structuredness implies soundness only at the process level, while this implication does not scale to collaborations (i.e., there are well-structured collaborations that are not sound). Moreover, soundness does not imply safeness and safe models are not necessarily sound. Some of the elements of BPMN collaborations, i.e. sub-processes, message passing and terminate events, have a specific impact on the classification of BPMN collaborations, as their usage can move some models from one class to another.

It is also worth noticing that our framework supports models with arbitrary topology, to enable the management and classification of both well-structured but also unstructured models. Unstructured models can be in some cases studied through their transformation into their structured versions, at the cost of increasing the model size [12]. However, this transformation can either be too large in size, or not possible at all [13, 14].

The relevance of the properties we consider on BPMN collaborations has been empirically studied by looking at their impact on the practice of the real-world modelling. We have analysed the BPMN 2.0 processes and collaborations models available in a well-known, public, well-populated repository provided by the PROSLab, named RePROSitory [15]. The verification of the properties on these models was carried out using the $\mathcal{S}^3$ tool[1] [16], which implements in Java the BPMN operational semantics considered here and uses it for performing properties verification. Notably, as a further contribution of this paper we have extended $\mathcal{S}^3$ in order to include well-structuredness checking. As a result of this empirical study, for instance, it turns out that BPMN models starts to become unstructured when their size grows. Hence, even if well-structuredness is considered as a good modelling practice in the BPMN guidelines, designers tend to deviate by it when modelling complex scenarios.

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, while Sec. 6 makes it clear the relationships between these properties. Sec. 7 presents the study on safeness and soundness compositionality. Sec. 8 presents the $\mathcal{S}^3$ tool and provides a clearer idea of the impact of well-structuredness, safeness, and soundness on the real-world modelling practice. Finally, Sec. 9 discusses related works, and Sec. 10 concludes the paper.

[1] `http://pros.unicam.it/s3/`

2. Basic Notions on BPMN Collaborations

In this section we introduce the considered elements of BPMN collaborations [2]. We provide here a detailed explanation of the elements, jointly with their correspondent textual representation that will be part of the process description language we take into account. Still, we present only the intended meaning of the elements, because the formal semantics will be given at Sec. 4.

This section is also the occasion to present our running example, taken from a travel agency scenario.

2.1. Pools

Pools (see Table 1) are used to represent participants or organisations involved in a collaboration. They are drawn as rectangles and include a unique name p for the Pool and a process specification P. The corresponding textual description is $\mathsf{pool}(\mathsf{p}, P)$, meaning that, when activated, p behaves according to the process specification P.

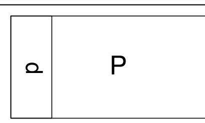
Pool - Graphical Representation	**Pool - Textual Notation**
	$\mathsf{pool}(\mathsf{p}, P)$

Table 1: Graphical and textual description of Pools.

2.2. Activities

Activities (see Table 2) are used to represent specific works to perform within a process. A *task* is an atomic activity, which cannot be interrupted during its execution. Tasks can also send and receive messages. A *sub-process*, instead, represents a work that brokes down into a finer level of detail. Activities are drawn as rectangles with rounded corners. The corresponding textual notation is as follows.

- $\mathsf{task}(\mathsf{e}, \mathsf{e}')$ denotes the task with incoming edge e and outgoing edge e',
- $\mathsf{taskRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$, denotes the task receiving a message m,
- $\mathsf{taskSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$, denotes the task sending a message m,
- $\mathsf{subProc}(\mathsf{e}, P, \mathsf{e}')$ denotes the sub-process activity with incoming edge e and outgoing edge e'. When activated, the (sub-)process P behaves according to its specification (it can include nested sub-process activities, of course).

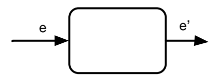
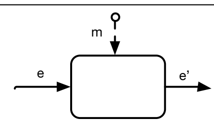
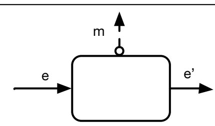
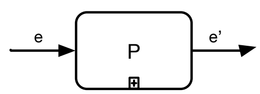
Activities - Graphical Representation	Activities - Textual Notation
	$\mathsf{task}(\mathsf{e},\mathsf{e}')$
	$\mathsf{taskRcv}(\mathsf{e},\mathsf{m},\mathsf{e}')$
	$\mathsf{taskSnd}(\mathsf{e},\mathsf{m},\mathsf{e}')$
	$\mathsf{subProc}(\mathsf{e},P,\mathsf{e}')$

Table 2: Graphical and textual description of Activities.

2.3. Gateways

Gateways (see Table 3) 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.

An *AND gateway* enables parallel execution flows. In particular, 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. The corresponding textual notation is as follows.

- $\mathsf{andSplit}(\mathsf{e},\{\mathsf{e}'_1,\ldots,\mathsf{e}'_n\})$ denotes an AND split gateway with incoming edge e and outgoing edges $\mathsf{e}'_1,\ldots,\mathsf{e}'_n$.

- $\mathsf{andJoin}(\{\mathsf{e}_1,\ldots,\mathsf{e}_n\},\mathsf{e}')$ denotes an AND join gateway with incoming edges $\mathsf{e}_1,\ldots,\mathsf{e}_n$ and outgoing edge e'.

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. The corresponding textual notation is as follows.

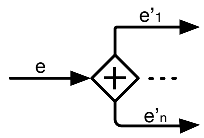
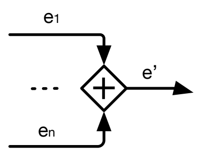
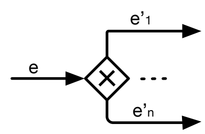
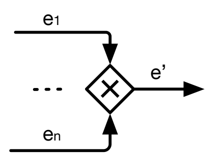
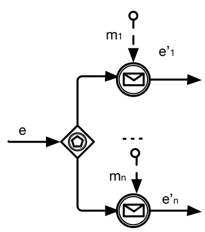
Gateways - Graphical Representation	Gateways - Textual Notation
	$\mathsf{andSplit}(\mathsf{e}, \{\mathsf{e}'_1, \ldots, \mathsf{e}'_n\})$
	$\mathsf{andJoin}(\{\mathsf{e}_1, \ldots, \mathsf{e}_n\}, \mathsf{e}')$
	$\mathsf{xorSplit}(\mathsf{e}, \{\mathsf{e}'_1, \ldots, \mathsf{e}'_n\})$
	$\mathsf{xorJoin}(\{\mathsf{e}_1, \ldots, \mathsf{e}_n\}, \mathsf{e}')$
	$\mathsf{eventBased}(\mathsf{e}, (\mathsf{m}_1, \mathsf{e}'_1), \ldots, , (\mathsf{m}_n, \mathsf{e}'_n))$

Table 3: Graphical and textual description of Gateways.

- $\mathsf{xorSplit}(\mathsf{e}, \{\mathsf{e}'_1, \ldots, \mathsf{e}'_n\})$ denotes a XOR split gateway with incoming edge e and outgoing edges $\mathsf{e}'_1, \ldots, \mathsf{e}'_n$.
- $\mathsf{xorJoin}(\{\mathsf{e}_1, \ldots, \mathsf{e}_n\}, \mathsf{e}')$ denotes a XOR join gateway with incoming edges $\mathsf{e}_1, \ldots, \mathsf{e}_n$ and outgoing edge e'.

An *Event-Based gateway* is used after a decision to fork the flow into branches according to external choices. 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.

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

2.4. Events

Events (see Table 4) are used to represent something that can happen. An event can be a *Start Event* representing the point from which a process starts. A *Start Message Event* is a start event with an incoming message edge; the event element catches a message and starts a process. An event can be an *Intermediate Event* if it happens during a process execution. If it receives a message it is called *Intermediate Receiving Events*, while if it sends a message it is called *Intermediate Sending Events*. An *End Event* represents process termination. There are other different forms for termination. An *End Message Event* is an end event with an outgoing message edge; it sends a message before ending the process. The *Terminate End Event*, instead, stops and aborts the running process.

Events are drawn as circles and the corresponding textual notation is as follows.

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

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

- $\mathsf{interRcv(e, m, e')}$ represents an intermediate receiving event with an incoming edge e and an outgoing edge $\mathsf{e'}$ that are able to receive a message m.

- $\mathsf{interSnd(e, m, e')}$ represents an intermediate sending event with an incoming edge e and an outgoing edge $\mathsf{e'}$ that are able to send a message m.

- $\mathsf{end(e, e')}$ represents an end event with an incoming edge e and a completing edge $\mathsf{e'}$.

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

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

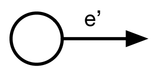
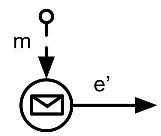
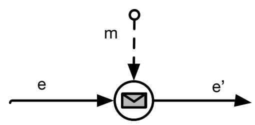
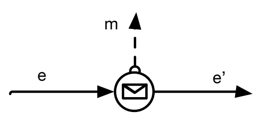
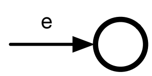
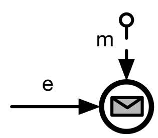
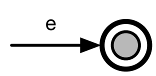
Events - Graphical Representation	Events - Textual Notation
	$\mathsf{start}(\mathsf{e}, \mathsf{e}')$
	$\mathsf{startRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$
	$\mathsf{interRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$
	$\mathsf{interSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$
	$\mathsf{end}(\mathsf{e}, \mathsf{e}')$
	$\mathsf{endSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$
	$\mathsf{terminate}(\mathsf{e})$

Table 4: Graphical and textual description of Events.

2.5. Tokens

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 generate messages. We will refer them as message flow token.

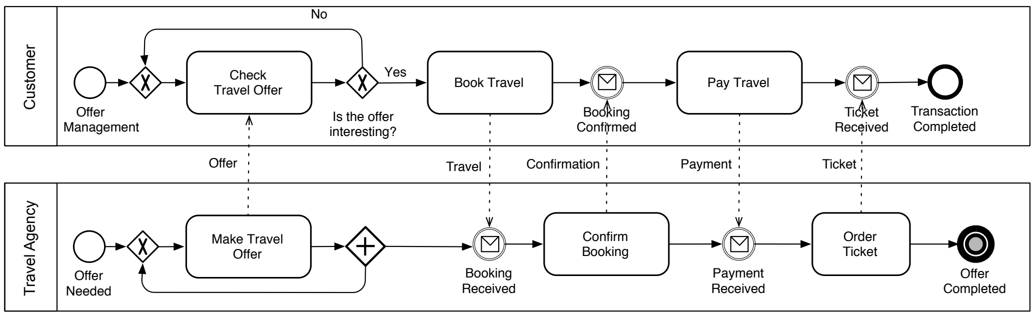


Figure 1: BPMN collaboration model of a travel agency scenario.

2.6. Travel Agency Collaboration Scenario

We introduce here a BPMN collaboration model of a travel agency scenario to be used throughout the paper as a running example.

In this scenario, a Travel Agency continuously offers travels to a Customer, until an offer is accepted. If the Customer is interested in one offer, he/she decides to book the travel and refuses all the others already offered. 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.

Running Example (1/9). We design the travel agency scenario as a collaboration model composed by two *pools*; namely, the Travel Agency and the Customer, as reported in Fig. 1. Let us concentrate on the Customer pool. As soon as the process starts, due to the presence of a *start event*, the Customer checks for the travel offer. This is done by executing a *receiving task*. Then, he/she decides either to book the travel or to wait for other offers, by cycling on two XOR *gateways*. After the Customer finds the interesting offer, he/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 the Customer receives the booking confirmation, through an *intermediate receiving event*, he/she pays the travel, receives the ticket from the Agency and the process terminates by means of an *end event*. Symmetrically, 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 an AND-split combined with a 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.

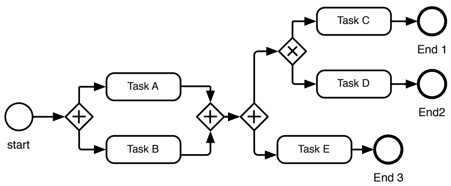


Figure 2: A non WS process model.

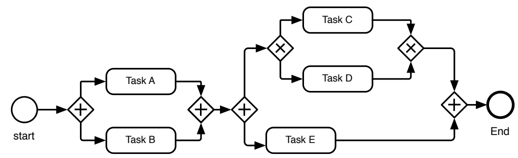


Figure 3: A WS process model.

3. Classification Results

In this section, we informally introduce the properties we consider and provide an explanation of our main results. We also discuss how our framework enables a more precise classification of the BPMN models with respect to others in the literature.

3.1. Well-structuredness, Safeness and Soundness for BPMN

We take into account three well-known classes of 'correctness' properties in the domain of business process management; namely well-structuredness [6], safeness [7, 8] and soundness [9, 10]. Their formal definitions will be given in Section 5. Intuitively, *well-structuredness* relates to the way the model elements are connected with each other, while *safeness* and *soundness* have to do with the process behaviour, i.e. to the way processes can be executed.

A BPMN process model is *well-structured* (WS) 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-Nets [6]. As an example, the process in Fig. 3 is the well-structured version of the unstructured process in Fig. 2. The notion of well-structuredness is extended from process to collaborations (see Def. 5), requiring well-structuredness to all the processes involved in the organisation.

A BPMN process model is *safe*[2], if during its execution no more than one token occurs along the same sequence edge (see Def. 7). This definition is inspired by the Petri Net formalism, where safeness means that a Petri Net does not have more than one token at each place in all reachable markings [8]. Safeness of processes scales to process collaborations, saying that no more than one token occurs on the same sequence edge during a collaboration execution (see Def. 8).

[2]Notably, the notion of safeness is different from that of safety, and is a specific and standard concept in the BPMN literature.

A BPMN process model is *sound* whenever, during its execution, it is always possible to reach a marking where either (i) each marked end event is marked by at most one token and there is no other token around, or (ii) all edges are unmarked (see Def. 10). Soundness is also inspired by the literature that presents several versions on different modelling languages [8] [10] [9] [17]. It is extended to process collaborations (see Def. 11), involving the whole collaboration execution and requiring that all sent messages are properly received. We also consider a variant of this property at collaboration level, which is a message-relaxed version, inspired by [18], that allows pending messages (see Def. 12).

3.2. Advances with respect to already available classifications.

Differently from other classification works in the literature (at the process level and) relying on different notations (as, for instance, Workflow Nets [19, 20] and π-calculus [21]), our study directly addresses BPMN collaboration models. By relying on a uniform formal framework, we properly study the relationships among the considered properties. Fig. 4 summarizes the obtained results. It shows 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.

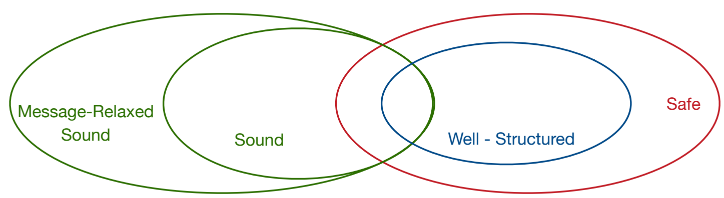


Figure 4: Classification of BPMN collaborations.

Item *(i)* states that well-structured collaborations represent a proper subclass of safe collaborations. We show that such an inclusion is valid at process level. On Workflow Nets, instead, a process model, to be safe, has to be well-structured and sound [20].

Item *(ii)* states that there are well-structured collaborations that are not sound. Well-structuredness, instead, implies soundness at the process level. This confirms the results provided on Workflow Nets, where well-structuredness implies soundness [22], and relaxes the one obtained in Petri Nets [19], where relaxed soundness and well-structuredness together imply soundness.

(i) and *(ii)* confirm the limits of well-structuredness. It turns out to be a very strict correctness criterion, as some safe and sound models that are not well-structured are taken a part.

Item *(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 [23]. 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 [24] [25]. Thus, a sound workflow is always safe. In BPMN instead there are unsafe processes that are sound.

Summing up, item *(i)* together with *(ii)* and *(iii)*, are novel results, also at process level. As clarified below, this is mainly due to the effects of the behaviour of the terminate event and sub-processes, that have an impact on the classification of the models, both at the process and collaboration level.

3.3. Advances in Classifying BPMN Models

Our formalisation focusses on the following BPMN features: different abstraction levels (i.e., sub-processes, processes and collaborations), asynchronous communication paradigm between pools, and different types of process/collaboration completion.

Our formalisation of collaboration models allows to observe both the execution of the processes involved in the collaboration, through the flow of tokens along sequence edges, and the exchange of messages between pools, through the flow of messages along message edges. 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 tokens both on message and sequence edges. Indeed, such distinction is not considered in the available mappings [4] [26], because a message is rendered as a (standard) token in a place. Hence, a safe BPMN collaboration, where

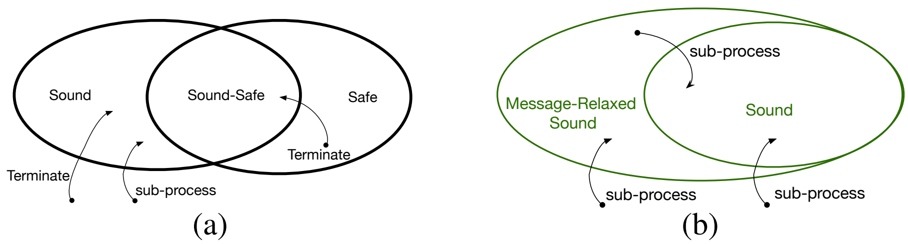


Figure 5: Reasoning at process level (a) and collaboration level (b).

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 directly applied as it is to BPMN collaboration models. Similarly, regarding to 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 reasonings are precluded using the current translations from BPMN to Petri Nets.

The study of BPMN models via the frameworks based on Petri Nets has another limitation concerning the management of the terminate event. Most of the available mappings, such as the ones in [26] and [27], 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 of the 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 [28], stating that modelling a vacuum cleaner, (i.e., a construct to remove all the tokens from a given fragment of a net) it is possible but results in a spaghetti-like model.

The ability of our formal framework to properly distinguish sequence flow tokens and message flow tokens, jointly with our management of the terminate event and sub-processes, without any of the above mentioned restrictions, allowed us to provide a more precise classification of the BPMN models as summarized in Fig. 5(a) and Fig. 5(b).

In particular, Fig. 5(a) underlines reasonings that can be done at process level on soundness (independently from safeness and well-structureness). It clearly emerges the impact of the terminate event on the soundness of models, as using a

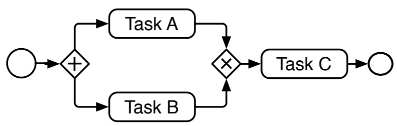


Figure 6: Unsound process.

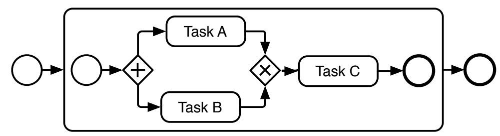


Figure 7: Sound process with an unsound sub-process.

terminate event in place of an end event might let sound an unsound model. For example, let us consider the process in Fig. 6; it is a simple process that first runs in parallel Task A and Task B, then performs 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 the model obatined by replacing the end event in Fig. 6 with a terminate event. The resulting model is sound and this is due to the behaviour of the terminate event that, when reached, removes all tokens in a process. It is worth noticing 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. 7 shows a simple process model where the unsound process in Fig. 6 is included in the sub-process. 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. 7 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 since, as we will show in Sec. 7, soundness is not compositional; namely, the composition of two sound processes not necessarily turns out to be sound.

Interesting situations also arise when focussing on the collaboration level, as highlighted in Fig. 5(b). Worth to notice is the possibility to transform, with a small change, an unsound collaboration into a sound one.

In Fig. 8, Fig. 9 and Fig. 10 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. 8 is neither sound nor message-relaxed sound, since on ORG A there is a configuration with two tokens on the end event and a pending message. Now let us consider another model obtained from that in Fig. 8 by introducing a sub-process. The resulting collaboration is as in Fig. 9 and turns out to be 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

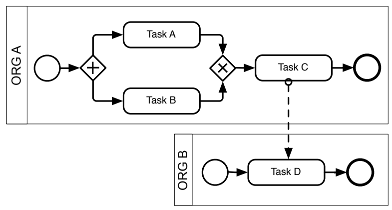


Figure 8: An example of unsound and message-relaxed unsound collaboration.

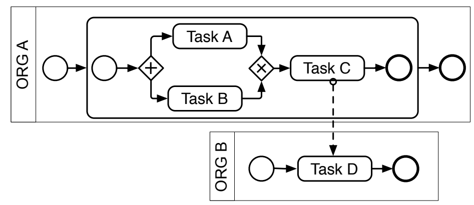


Figure 9: An example of message-relaxed sound and unsound collaboration.

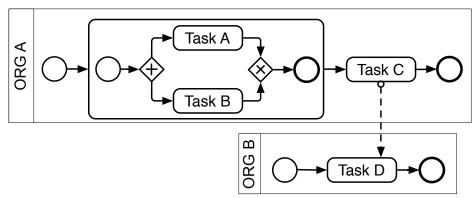


Figure 10: An example of message-relaxed sound and sound collaboration.

D. Differently, Fig. 10 shows that enclosing within a sub-process only the part of the model generating multiple tokens leads to 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 [29], but 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. 11). 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.

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

Figure 11: 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. Similarly, operator $\|$ at process level permits to compose process elements in order to render a process structure in terms of a collection of elements. Notably, in the considered formal framework it is not possible to distinguish the difference between communicative tasks and intermediate events (see Fig. 12).

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 one-to-one correspondence between the syntax used here to represent *Process/Collaboration Structures* and the graphical notation of BPMN has been already illustrated in Sec. 2. 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.

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 $edges(P)$ to refer the edges in the scope of P and $edgesEl(P)$ to indicate the edges in the scope of P without considering the spurious edges (the complete definitions can be found in Appendix A).

Running Example (2/9). The BPMN model in Fig. 1 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 and P_{TA} are expressed as follows (where for simplicity we identify the edges in progressive order e_i (with $i = 0 \ldots 20$):

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

$$\begin{aligned} P_{TA} \quad = \quad & \mathsf{start}(\mathsf{e}_{11}, \mathsf{e}_{12}) \parallel \mathsf{xorJoin}(\{\mathsf{e}_{12}, \mathsf{e}_{13}\}, \mathsf{e}_{14}) \parallel \mathsf{taskSnd}(\mathsf{e}_{14}, \mathsf{Offer}, \mathsf{e}_{15}) \parallel \\ & \mathsf{andSplit}(\mathsf{e}_{15}, \{\mathsf{e}_{13}, \mathsf{e}_{16}\}) \parallel \mathsf{interRcv}(\mathsf{e}_{16}, \mathsf{Travel}, \mathsf{e}_{17}) \parallel \\ & \mathsf{taskSnd}(\mathsf{e}_{17}, \mathsf{Confirmation}, \mathsf{e}_{18}) \parallel \mathsf{interRcv}(\mathsf{e}_{18}, \mathsf{Payment}, \mathsf{e}_{19}) \parallel \\ & \mathsf{taskSnd}(\mathsf{e}_{19}, \mathsf{Ticket}, \mathsf{e}_{20}) \parallel \mathsf{terminate}(\mathsf{e}_{20}) \end{aligned}$$

Moreover, considering functions we defined on the structure we have: $participant(\mathsf{pool}(\mathsf{Customer}, P_C) \parallel \mathsf{pool}(\mathsf{TravelAgency}, P_{TA})) = \{P_C, P_{TA}\}$, $start(P_C) = \{\mathsf{e}_0\}$, $start(P_{TA}) = \{\mathsf{e}_{11}\}$, and $end(P_C) = \{\mathsf{e}_{10}\}$, $end(P_{TA}) = \varnothing$. Finally, $edges(P_C) = \{\mathsf{e}_0, ..., \mathsf{e}_{10}\}$, $edges(P_{TA}) = \{\mathsf{e}_{11}, ..., \mathsf{e}_{20}\}$. □

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 $\delta : \mathbb{M} \rightarrow \mathbb{N}$ 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, then, the process configuration is initial if $isInit(P, \sigma)$ holds. Predicate $isInit(P, \sigma)$ holds, if $\sigma(start(P)) = 1$, and $\forall\, \mathsf{e} \in edges(P) \backslash start(P)\ .\ \sigma(\mathsf{e}) = 0$.*

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

Running Example (3/9). The initial configuration of the collaboration in Fig. 1 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 $i = 1 \dots 10$, and $\langle P_{TA}, \sigma \rangle$,

$\sigma(\mathsf{e}_{11}) = 1$ and $\sigma(\mathsf{e}_j) = 0 \;\; \forall \mathsf{e}_j$ with $j = 12 \ldots 20$. 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 $\langle \mathcal{P}, \mathcal{A}, \rightarrow \rangle$ 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 $\rightarrow \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$ is a *transition relation*. We will write $\langle P, \sigma \rangle \xrightarrow{\alpha} \langle P, \sigma' \rangle$ to indicate that $(\langle P, \sigma \rangle, \alpha, \langle P, \sigma' \rangle) \in \rightarrow$ 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.

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

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, while kill denotes the termination action.

The transition relation over process configurations formalises the execution of a process; it is defined by the rules in Fig. 12. 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]. The definition of the *completed* predicate relies on the function $marked(\sigma, E)$, used to refer to the set of edges in E with at least one token:

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

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

Now, the sub-process completion can be formalised as follows.

Definition 3. *Let P be a process included in a sub-process, the predicate $completed(P, \sigma)$ holds if the following condition is satisfied:*

$$\exists\, \mathsf{e} \in end(P) \,.\, \mathsf{e} \in marked(\sigma, end(P)) \;\wedge\; \forall \mathsf{e} \in edges(P) \backslash end(P) \,.\, \sigma(\mathsf{e}) = 0$$

Notably, the completion of a sub-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.

We now briefly comment on the operational rules in Fig. 12. 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 12: BPMN Semantics - Process Level.

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-$AndSplit$ 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-$XorSplit$ 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 increments the token in one of the outgoing edges, non-deterministically chosen. Rule P-$AndJoin$ 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-$XorJoin$ is activated every time there is a token in one of the incoming edges, which is then moved to the outgoing edge. Rule P-$Task$ 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-$TaskRcv$ 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-$TaskSnd$, 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 *(i)* 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, and *(ii)* all edges of the sub-process are unmarked. Then, the sub-process behaves according to the behaviour of the elements it contains according to the rule 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[3], 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 a terminate event. The sub-process stops its internal behaviours and passes the control to the upper layer: all the tokens in the sub-process are removed and a token is added to the outgoing edge of the sub-process element. Rule P-$Kill$ deal with the propagation of killing action in

[3]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.

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.

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 $\langle C, \sigma, \delta \rangle \xrightarrow{\alpha} \langle \sigma', \delta' \rangle$. The rules related to the collaboration level are defined in Fig. 13

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

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

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

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

Figure 13: 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 level. 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 level, 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. [6]. 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 basic 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 functions: $in(P)$ and $out(P)$ determine, respectively, the incoming and outgoing sequence edges of a process element P (the full definition is relegated to Appendix A). 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 *(written* $isWS(P)$*) if* P *has one of the following forms:*

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

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 ($\mathsf{e}_1 \rightarrow \mathsf{e}_4 \rightarrow \mathsf{e}_6 \rightarrow \mathsf{e}_2 \rightarrow \mathsf{e}_1$) 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. 14. 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 $isWS(C_2)$.*

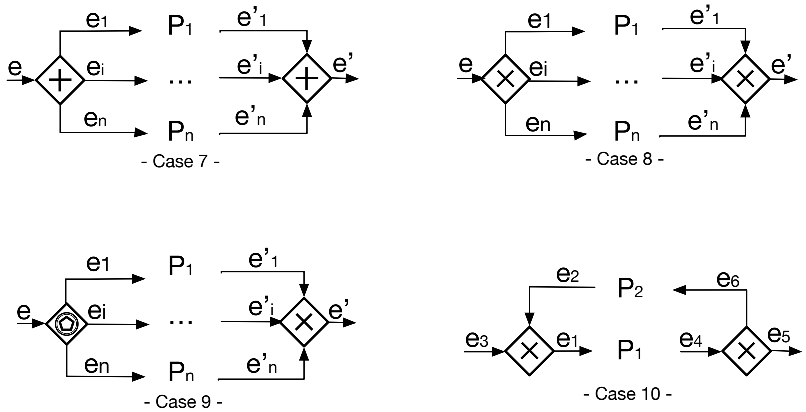


Figure 14: Well-structured nodes (cases 7-10).

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 safe BPMN processes and collaborations, we need to introduce 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* $\forall \mathsf{e} \in edgesEl(P) \,.\, \sigma(\mathsf{e}) \leqslant 1$.

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 $\langle P, \sigma \rangle \rightarrow^* \sigma'$ 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 σ' and δ' such that $\langle C, \sigma, \delta\rangle \rightarrow^* \langle \sigma', \delta'\rangle$ we have that $\forall\ P \in participant(C)$, $\langle P, \sigma'\rangle$ is cs-safe.*

Running Example (5/9). Let us consider again our running example depicted in Fig. 1. 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

Another relevant property for the business process domain is soundness. As usual, we define it both at the process and collaboration level. In a process it ensures that, once its execution starts with a token in the start event, it is always possible to reach one of these scenarios: *(i)* all marked end events are marked exactly by a single token and all sequence edges are unmarked; *(ii)* no token is observed in the configuration (meaning that a token has reached 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) $\forall\, \mathsf{e} \in marked(\sigma, end(P))\ .\ \sigma(\mathsf{e}) = 1\ \wedge\ \forall \mathsf{e} \in edges(P) \backslash end(P)\ .\ \sigma(\mathsf{e}) = 0.$

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

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 $\langle P, \sigma\rangle \rightarrow^* \sigma'$ we have that there exists σ'' such that $\langle P, \sigma'\rangle \rightarrow^* \sigma''$, 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 $\langle C, \sigma', \delta'\rangle \rightarrow^* \langle \sigma'', \delta''\rangle$, $\forall\ P \in participant(C)$ 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 $\forall\ P \in participant(C)$ 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 reverse 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. 11 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 ($\mathit{isInitEl}(P, \sigma)$, see Appendix A).

Definition 13 (Reachable processes). *A process configuration* $\langle P, \sigma \rangle$ *is* reachable *if there exists a configuration* $\langle P, \sigma' \rangle$ *such that* $\mathit{isInit}(P, \sigma')$ *and* $\langle P, \sigma' \rangle \rightarrow^{*} \sigma$.

Definition 14 (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* $\mathit{isInitEl}(P, \sigma')$ *and* $\langle P, \sigma' \rangle \rightarrow^{*} \sigma$.

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

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

Lemma 2. *Let* $\mathit{isWSCore}(P)$*, 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. 15 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.*

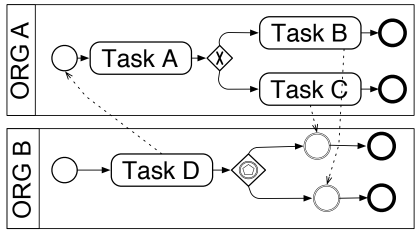


Figure 15: A safe BPMN collaboration not well-structured.

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 result 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 ($isCompleteEl(P, \sigma)$); we refer to Appendix A for the complete account of the definition.

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* $isWS(P)$*, 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. 16. 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.

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,* $isWS(C)$ *does not imply* C *is sound.*

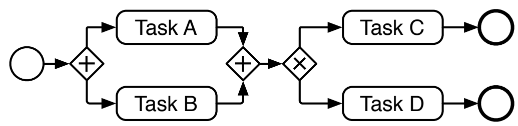


Figure 16: An example of sound process not Well-Structured.

Proof (sketch). We proceed by contradiction. □

Theorem 5. *Let C be a collaboration, $isWS(C)$ 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 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 interrelationships 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.*

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. 17. 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.* □

Example 2. *In Fig. 18 we have two unsafe processes, since each of them contains a loop capable of generating an unbounded number of tokens. However, if we*

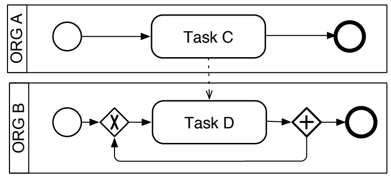


Figure 17: Safe collaboration with safe and unsafe processes.

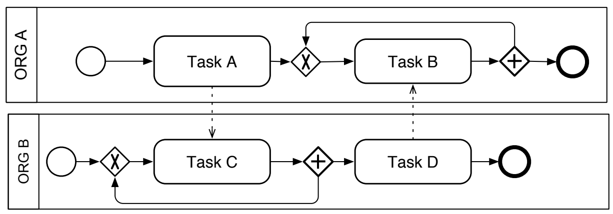


Figure 18: Safe collaboration with unsafe processes.

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. 19 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

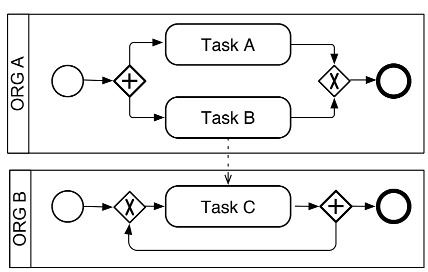


Figure 19: Unsafe collaboration with unsafe processes.

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. However, the resulting collaboration is not sound, since the message lists could not be empty. □

Example 4. *In Fig. 20 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.* □

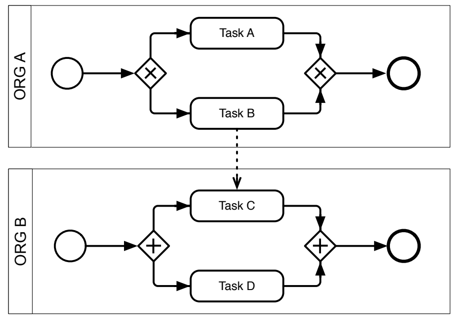


Figure 20: An example of unsound collaboration with sound processes.

Example 5. *Also the collaboration in Fig. 21 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.* □

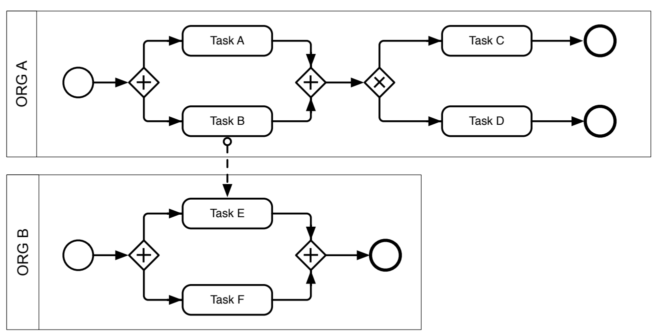


Figure 21: Sound collaboration with sound processes.

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). □

Remark 1. *We do not consider well-structuredness and message-relaxed soundness compositionality. In fact, the compositionaly of well-structuredness is implied by the property definition, while it is not possible to study the message-relaxed soundness compositionality since this property cannot be defined at the process level (where enqueuing of messages is not considered).*

8. Relevance into Practice the $\mathcal{S}^3$ tool

To get a clearer idea of the impact of well-structuredness, safeness, and soundness on the real-world modelling practice, we have analysed the BPMN 2.0 process and collaboration models available in a well-known, public, well-populated repository provided by the PROS Lab,[4] namely RePROSitory [15]. It includes 164 models[5] that has been retrieved by research papers accepted from the BPM conference, starting from 2011 that is the year when the BPMN standard has been released. Thus, the repository is particularly suitable to investigate real modelling practice, modelling styles and the relevance of modelling constructs.

From the technical point of view, well-structuredness, safeness and soundness have been checked using the $\mathcal{S}^3$ tool[6]. In this regards, an additional contribution we provide in the paper is the extension of the $\mathcal{S}^3$ Java stand-alone application with a new verification component for checking well-structuredness[7]. The application allows the user to load a *.bpmn* file to be checked, and hence to verify the considered properties. The graphical interface provides a text area reporting the verification results, and a button to visualise in a separate window the generated LTS.

Running a massive verification we obtained the results reported in Table 5. The models are grouped in classes depending on their size. Notably, given the number of models that are included in the classes with size 40-49, 50-59 and 60-69, we do not consider these classes in our reasoning below, even if also in these cases the theoretical results are confirmed by the empirical study.

Let us focus on the well-structuredness; 49% of models in the repository satisfy it. 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

[4]`https://pros.unicam.it/reprository`

[5]`https://pros.unicam.it:4200/guest/collection/a.m1903202001038`

[6]`http://pros.unicam.it/s3/`

[7]The updated stand-alone application of $\mathcal{S}^3$ is available at `http://pros.unicam.it:8080/S3Stand-alone/S3.zip`

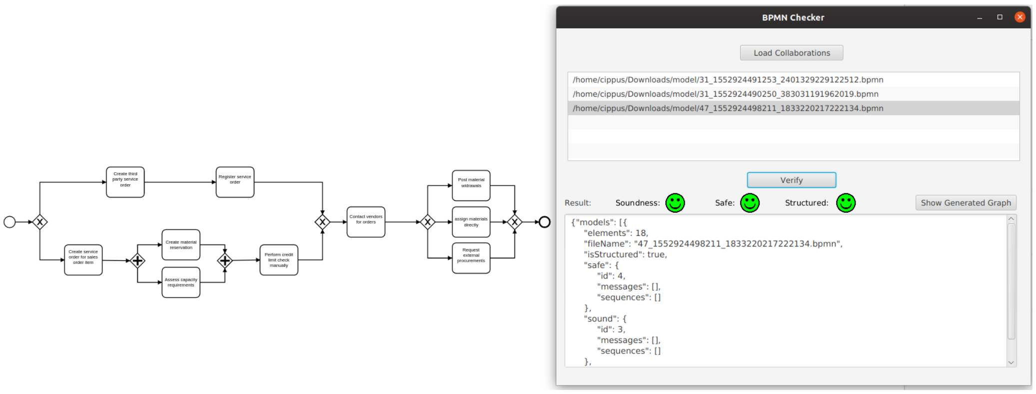


Figure 22: $\mathcal{S}^3$ Modelling Environment Interface.

Size	Dataset	WS	Non-WS	Safe	MR-Sound	Sound
0 - 9	59	34 (58%)	25 (48%)	57 (97%)	44 (75%)	44 (75%)
10 - 19	77	39 (51%)	38 (49%)	72 (94%)	58 (75%)	58 (75%)
20 - 29	20	6 (30%)	14 (70%)	18 (90%)	14 (70%)	14 (70%)
30 - 39	5	0 (0%)	5 (100%)	4 (80%)	3 (60%)	3 (60%)
40 - 49	1	0 (0%)	1 (100%)	1(100%)	0 (0%)	0 (0%)
50 - 59	1	1(100%)	0 (0%)	1(100%)	1(100%)	1(100%)
60 - 69	1	0 (0%)	1 (100%)	1(100%)	0(0%)	0(0%)
	164	80 (49%)	84 (51%)	154 (94%)	120 (73%)	120 (73%)

Table 5: Classification of the models in RePROSitory.

to become unstructured when their size grows. This means that, even if structuredness is a good property, it should be regarded as a general guideline; 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 154 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 0 - 9, 10 - 19, and 20-29 including 156 models, of which only 9 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 topology.

Concerning soundness, it results that there are 120 sound models. Modelling in a sound way is a common practice, recognising soundness as one of the most important correctness criteria. Moreover, the data shows that there are well-structured models that are not sound, which confirms the limitation of well-structuredness. Concerning message-relaxed soundness, it results that the number of models satisfying this property are the same of the the sound ones. This could be due to a limitation of the data-set for what concerns the presence of collaboration diagrams, as it only includes 13 diagrams of this type.

9. Related Work

In this paper we provide a formal characterisation of well-structuredness for BPMN models. To do that we have been inspired by the definition of well-structuredness given in [6]. Other attempts are also available in the literature. Van der Aalst et al. [30] state that a workflow net is well-structured if the split/join constructions are properly nested. El-Saber and Boronat [31] 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 [32] and COWS [33], do not consider safeness, even if it is recognised as an important characteristic [34].

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 [35], 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 [9]. 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 [36] 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 [18], who define interaction soundness, which in turn is based on lazy soundness [21]. 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, as also already discussed in Sec. 3.2 and Sec. 3.3, our investigation of properties at collaboration level provides novel insights with respect to the state-of-the-art of BPMN formal studies.

10. Concluding Remarks

Our study formally defines some important correctness properties, namely well-structuredness, safeness, and soundness, both at the process and collaboration level of BPMN models. We demonstrate the relationships between the studied properties, with the aim of classifying BPMN collaborations according to the properties they satisfy. Rather than converting the BPMN models to Petri or Workflow Nets and studying relevant properties on the models resulting from the mapping, we directly define such properties on BPMN, thus dealing with its complexity and specificities directly. Our approach is based on a uniform formal framework and is not limited to models with a specific topology, i.e., models do not need to be block-structured.

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. 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 can play a significant role in practice. We also show that there are well-structured collaborations that are neither sound nor message-relaxed sound. Finally, we demonstrate there are sound and message-relaxed sound collaborations that are not safe. The resulting classification also provides a novel contribution by extending the reasoning from processes to collaborations. Moreover, being close to the BPMN standard, it permits to catch the language peculiarities, as the asynchronous communication model and the completeness notion that distinguishes the effect of a terminate end event from that of a classic end event. Finally, the empirical investigation we did by means of the $\mathcal{S}^3$ tool confirms our theoretical study, and makes evident its importance into practice.

In the future, we plan to continue our programme to reason on the properties of BPMN collaboration models, by considering variants of the correctness properties and a larger set of BPMN elements. 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] Corradini, F., Fornari, F., Polini, A., Re, B., Tiezzi, F.: A formal approach to modeling and verification of business process collaborations. Sci. Comput. Program. **166** (2018) 35–70

[6] 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

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

[8] 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

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

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

[12] 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

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

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

[15] Corradini, F., Fornari, F., Polini, A., Re, B., Tiezzi, F.: Reprository: a repository platform for sharing business process models. In: BPM (PhD/Demos). Volume 2420 of CEUR Workshop Proceedings., CEUR-WS.org (2019) 149–153

[16] Corradini, F., Morichetta, A., Polini, A., Re, B., Rossi, L., Tiezzi, F.: Safeness and soundness checking in multi-layer bpmn 2.0 collaborations. The Journal of Systems & Software **In press** (2020)

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

[18] 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

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

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

[21] 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

[22] 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

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

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

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

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

[27] 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

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

[29] 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

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

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

[32] 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

[33] 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

[34] 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

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

[36] 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. Definitions

Here we reported the complete definitions of some auxiliary notions used in the paper.

We define function $edges(P)$ to refer the edges in the scope of P and $edgesEl(P)$ to indicate the edges in the scope of P without considering the spurious edges.

$$edges(P_1 \parallel P_2) = edges(P_1) \cup edges(P_2)$$
$$edges(\mathsf{start}(\mathsf{e},\mathsf{e}')) = \{\mathsf{e},\mathsf{e}'\}$$
$$edges(\mathsf{end}(\mathsf{e},\mathsf{e}')) = \{\mathsf{e},\mathsf{e}'\}$$
$$edges(\mathsf{startRcv}(\mathsf{e},\mathsf{m},\mathsf{e}')) = \{\mathsf{e},\mathsf{e}'\}$$
$$edges(\mathsf{endSnd}(\mathsf{e},\mathsf{m},\mathsf{e}')) = \{\mathsf{e},\mathsf{e}'\}$$
$$edges(\mathsf{terminate}(\mathsf{e})) = \{\mathsf{e}\}$$
$$edges(\mathsf{eventBased}(\mathsf{e},(\mathsf{m}_1,\mathsf{e}'_1),\ldots,(\mathsf{m}_h,\mathsf{e}'_h))) = \{\mathsf{e},\mathsf{e}'_1,\ldots,\mathsf{e}'_h\}$$
$$edges(\mathsf{andSplit}(\mathsf{e},\mathsf{e}'_1,\ldots,\mathsf{e}'_h)) = \{\mathsf{e},\mathsf{e}'_1,\ldots,\mathsf{e}'_h\}$$
$$edges(\mathsf{xorSplit}(\mathsf{e},\mathsf{e}'_1,\ldots,\mathsf{e}'_h)) = \{\mathsf{e},\mathsf{e}'_1,\ldots,\mathsf{e}'_h\}$$
$$edges(\mathsf{andJoin}(\mathsf{e}_1,\ldots,\mathsf{e}_h,\mathsf{e}')) = \{\mathsf{e}_1,\ldots,\mathsf{e}_h,\mathsf{e}'\}$$
$$edges(\mathsf{xorJoin}(\mathsf{e}_1,\ldots,\mathsf{e}_h,\mathsf{e}')) = \{\mathsf{e}_1,\ldots,\mathsf{e}_h,\mathsf{e}'\}$$
$$edges(\mathsf{task}(\mathsf{e},\mathsf{e}')) = \{\mathsf{e},\mathsf{e}'\}$$
$$edges(\mathsf{taskRcv}(\mathsf{e},\mathsf{m},\mathsf{e}')) = \{\mathsf{e},\mathsf{e}'\}$$
$$edges(\mathsf{taskSnd}(\mathsf{e},\mathsf{m},\mathsf{e}')) = \{\mathsf{e},\mathsf{e}'\}$$
$$edges(\mathsf{empty}(\mathsf{e},\mathsf{e}')) = \{\mathsf{e},\mathsf{e}'\}$$
$$edges(\mathsf{interRcv}(\mathsf{e},\mathsf{m},\mathsf{e}')) = \{\mathsf{e},\mathsf{e}'\}$$
$$edges(\mathsf{interSnd}(\mathsf{e},\mathsf{m},\mathsf{e}')) = \{\mathsf{e},\mathsf{e}'\}$$
$$edges(\mathsf{subProc}(\mathsf{e},P,\mathsf{e}')) = \{\mathsf{e},\mathsf{e}'\} \cup edges(P)$$

$$edgesEl(P_1 \parallel P_2) = edgesEl(P_1) \cup edgesEl(P_2)$$
$$edgesEl(\mathsf{start}(\mathsf{e}, \mathsf{e}')) = \{\mathsf{e}'\}$$
$$edgesEl(\mathsf{end}(\mathsf{e}, \mathsf{e}')) = \{\mathsf{e}\}$$
$$edgesEl(\mathsf{startRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$$
$$edgesEl(\mathsf{endSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$$
$$edgesEl(\mathsf{terminate}(\mathsf{e})) = \{\mathsf{e}\}$$
$$edgesEl(\mathsf{eventBased}(\mathsf{e}, (\mathsf{m}_1, \mathsf{e}'_1), \ldots, (\mathsf{m}_h, \mathsf{e}'_h))) = \{\mathsf{e}, \mathsf{e}'_1, \ldots, \mathsf{e}'_h\}$$
$$edgesEl(\mathsf{andSplit}(\mathsf{e}, \mathsf{e}'_1, \ldots, \mathsf{e}'_h)) = \{\mathsf{e}, \mathsf{e}'_1, \ldots, \mathsf{e}'_h\}$$
$$edgesEl(\mathsf{xorSplit}(\mathsf{e}, \mathsf{e}'_1, \ldots, \mathsf{e}'_h)) = \{\mathsf{e}, \mathsf{e}'_1, \ldots, \mathsf{e}'_h\}$$
$$edgesEl(\mathsf{andJoin}(\mathsf{e}_1, \ldots, \mathsf{e}_h, \mathsf{e}')) = \{\mathsf{e}_1, \ldots, \mathsf{e}_h, \mathsf{e}'\}$$
$$edgesEl(\mathsf{xorJoin}(\mathsf{e}_1, \ldots, \mathsf{e}_h, \mathsf{e}')) = \{\mathsf{e}_1, \ldots, \mathsf{e}_h, \mathsf{e}'\}$$
$$edgesEl(\mathsf{task}(\mathsf{e}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$$
$$edgesEl(\mathsf{taskRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$$
$$edgesEl(\mathsf{taskSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$$
$$edgesEl(\mathsf{empty}(\mathsf{e}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$$
$$edgesEl(\mathsf{interRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$$
$$edgesEl(\mathsf{interSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$$
$$edgesEl(\mathsf{subProc}(\mathsf{e}, P, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\} \cup edgesEl(P)$$

We inductively define functions $in(P)$ and $out(P)$, which determine the incoming and outgoing sequence edges of a process element P.

$$
\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 = \{\mathsf{e}\} & \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 \setminus (out(P_1) \cup out(P_2)) & \qquad \setminus (in(P_1) \cup in(P_2))
\end{array}
$$

Definition 15 (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 $\forall \mathsf{e}'' \in edges(P) \,.\, \sigma(\mathsf{e}'') = 0$
$isInitEl(P_1 \| P_2,\sigma)$ *if* $\forall \mathsf{e} \in in(P_1 \parallel P_2) \,:\, isInitEl(\mathsf{getInEl}(\mathsf{e},P_1 \parallel P_2))$
and $\forall \mathsf{e} \in (edges(P_1 \parallel P_2) \setminus in(P_1 \parallel P_2)) \,:\, \sigma(\mathsf{e}) = 0$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 $\forall\ \mathsf{e}' \in edges(P) \backslash start(P)\ .\ \sigma(\mathsf{e}') = 0$, i.e. only the start event has a marking and all the other edges are unmarked. Hence, we have that $\forall \mathsf{e} \in edgesEl(P)\ .\ \sigma(\mathsf{e}) \leqslant 1$, 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: since by hypothesis $isWSCore(P)$, it can only be either a task or an intermediate event.

- $P = \mathsf{task}(\mathsf{e}, \mathsf{e}')$. By hypothesis $\langle P, \sigma \rangle$ is cs-safe, then $edgesEl(P) = edgesEl(\mathsf{task}(\mathsf{e}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$ is such that $\sigma(\mathsf{e}) \leqslant 1$ and $\sigma(\mathsf{e}') \leqslant 1$. 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 $\sigma(\mathsf{e}) > 0$; 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, $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$. Hence, we have that $\forall \mathsf{e}''' \in edgesEl(P)\ .\ \sigma'(\mathsf{e}''') \leqslant 1$, which allows us to conclude.

- $P = \mathsf{taskRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$. By hypothesis $\langle P, \sigma \rangle$ is cs-safe, then $edgesEl(P) = edgesEl(\mathsf{taskRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$ is such that $\sigma(\mathsf{e}) \leqslant 1$ and $\sigma(\mathsf{e}') \leqslant 1$. The only rule that can be applied to infer the transition $\langle P, \sigma \rangle \xrightarrow{\alpha} \sigma'$ is $P\text{-}TaskRcv$. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$; 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, $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$. Hence, we have that $\forall \mathsf{e}''' \in edgesEl(P)\ .\ \sigma'(\mathsf{e}''') \leqslant 1$, which allows us to conclude.

- $P = \mathsf{taskSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$. By hypothesis $\langle P, \sigma \rangle$ is cs-safe, then $edgesEl(P) = edgesEl(\mathsf{taskSnd}(\mathsf{e}, \mathsf{m}, \mathsf{e}')) = \{\mathsf{e}, \mathsf{e}'\}$ is such that $\sigma(\mathsf{e}) \leqslant 1$ and $\sigma(\mathsf{e}') \leqslant 1$. The only rule that can be applied to infer the transition $\langle P, \sigma \rangle \xrightarrow{\alpha} \sigma'$ is $P\text{-}TaskSnd$. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$; 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, $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$. Hence, we have that $\forall \mathsf{e}''' \in edgesEl(P)\ .\ \sigma'(\mathsf{e}''') \leqslant 1$, which allows us to conclude.

- $P = \mathsf{interRcv}(\mathsf{e},\mathsf{m},\mathsf{e}')$. By hypothesis $\langle P,\sigma\rangle$ is cs-safe, then $edgesEl(P) = edgesEl(\mathsf{interRcv}(\mathsf{e},\mathsf{m},\mathsf{e}')) = \{\mathsf{e},\mathsf{e}'\}$ is such that $\sigma(\mathsf{e}) \leqslant 1$ and $\sigma(\mathsf{e}') \leqslant 1$. The only rule that can be applied to infer the transition $\langle P,\sigma\rangle \xrightarrow{\alpha} \sigma'$ is $P\text{-}InterRcv$. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$; 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, $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$. Hence, we have that $\forall \mathsf{e}''' \in edgesEl(P)\ .\ \sigma'(\mathsf{e}''') \leqslant 1$, which allows us to conclude.
- $P = \mathsf{interSnd}(\mathsf{e},\mathsf{m},\mathsf{e}')$. By hypothesis $\langle P,\sigma\rangle$ is cs-safe, then $edgesEl(P) = edgesEl(\mathsf{interSnd}(\mathsf{e},\mathsf{m},\mathsf{e}')) = \{\mathsf{e},\mathsf{e}'\}$ is such that $\sigma(\mathsf{e}) \leqslant 1$ and $\sigma(\mathsf{e}') \leqslant 1$. The only rule that can be applied to infer the transition $\langle P,\sigma\rangle \xrightarrow{\alpha} \sigma'$ is $P\text{-}InterSnd$. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$; 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, $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$. Hence, we have that $\forall \mathsf{e}''' \in edgesEl(P)\ .\ \sigma'(\mathsf{e}''') \leqslant 1$, which allows us to conclude.
- $P = \mathsf{empty}(\mathsf{e},\mathsf{e}')$. By hypothesis $\langle P,\sigma\rangle$ is cs-safe, then $edgesEl(P) = edgesEl(\mathsf{empty}(\mathsf{e},\mathsf{e}')) = \{\mathsf{e},\mathsf{e}'\}$ is such that $\sigma(\mathsf{e}) \leqslant 1$ and $\sigma(\mathsf{e}') \leqslant 1$. The only rule that can be applied to infer the transition $\langle P,\sigma\rangle \xrightarrow{\alpha} \sigma'$ is $P\text{-}Empty$. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$; 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, $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e}') = 1$. Hence, we have that $\forall \mathsf{e}''' \in edgesEl(P)\ .\ \sigma'(\mathsf{e}''') \leqslant 1$, which allows us to conclude.

Inductive cases:

- Let us consider $\langle \mathsf{andSplit}(\mathsf{e},E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{andJoin}(E',\mathsf{e}'),\sigma\rangle$, with $\forall j \in [1..n]\ \ isWSCore(P_j),\ \ in(P_j) \subseteq E,\ \ out(P_j) \subseteq E'$. 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, $\forall \mathsf{e}''' \in edgesEl(\mathsf{andSplit}(\mathsf{e},E))\ .\ \sigma(\mathsf{e}''') \leqslant 1$. 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 $\langle \mathsf{andSplit}(E,\mathsf{e}) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{andJoin}(E',\mathsf{e}'),\sigma'\rangle$ is cs-safe.

– 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, $\forall \mathsf{e}''' \in edgesEl(\mathsf{andJoin}(E', \mathsf{e}')) \,.\, \sigma(\mathsf{e}''') \leqslant 1$. 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 $\langle \mathsf{andSplit}(E, \mathsf{e}) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{andJoin}(E', \mathsf{e}'), \sigma' \rangle$ is cs-safe.

- Let us consider $\langle \mathsf{xorSplit}(\mathsf{e}, E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E', \mathsf{e}''), \sigma \rangle$, with $\forall j \in [1..n] \;\; isWSCore(P_j), \;\; in(P_j) \subseteq E, \;\; out(P_j) \subseteq E'$. There are the following possibilities:

  – $\langle \mathsf{xorSplit}(\mathsf{e}, E), \sigma \rangle$ evolves by means of rule *P-$XorSplit$*. We can exploit the fact that this is a core reachable well-structured configuration to prove that $\sigma(\mathsf{e}) = 1$ and $\forall \mathsf{e}'' \in E . \sigma(\mathsf{e}'') = 0$. Thus, $\mathsf{xorSplit}(\mathsf{e}, \{\mathsf{e}'\} \cup E), \sigma \rangle \xrightarrow{\epsilon} \sigma'$, with $\sigma' = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$. Hence, $\forall \mathsf{e}''' \in edgesEl(\mathsf{xorSplit}(\mathsf{e}, E)) \,.\, \sigma(\mathsf{e}''') \leqslant 1$. By hypothesis $\langle \mathsf{xorSplit}(\mathsf{e}, E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E', \mathsf{e}''), \sigma \rangle$ is cs-safe, i.e. if $\sigma'(\mathsf{e}') = 1$, that is there is a token on one of the outgoing edges of the XOR-Split in the state $\langle \mathsf{xorSplit}(\mathsf{e}, E), \sigma' \rangle$, then all the other edges are unmarked. This means that cs-safeness is not affected. Therefore, the overall term $\langle \mathsf{xorSplit}(\mathsf{e}, E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E', \mathsf{e}''), \sigma' \rangle$ 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.

  – $\langle \mathsf{xorJoin}(\{\mathsf{e}\} \cup E, \mathsf{e}'), \sigma \rangle$ evolves by means of rule *P-$XorJoin$*. We can exploit the fact that this is a core reachable well-structured configuration to prove that $\sigma(\mathsf{e}) = 1$, $\forall \mathsf{e}'' \in E' . \sigma(\mathsf{e}'') = 0$ and $\sigma(\mathsf{e}') = 0$. Thus $\langle \mathsf{xorJoin}(\{\mathsf{e}\} \cup E, \mathsf{e}'), \sigma \rangle \xrightarrow{\epsilon} \sigma'$, with $\sigma' = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$. Hence, $\forall \mathsf{e}''' \in edgesEl(\mathsf{xorJoin}(\{\mathsf{e}\} \cup E, \mathsf{e}')) \,.\, \sigma(\mathsf{e}''') \leqslant 1$. By hypothesis $\langle \mathsf{xorSplit}(\mathsf{e}, E) \mid \mid P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E', \mathsf{e}''), \sigma \rangle$ is cs-safe, i.e. if there is a token on the outgoing edge of the XOR-Join in the state $\langle \mathsf{xorJoin}(\{\mathsf{e}\} \cup E, \mathsf{e}'), \sigma' \rangle$

all the other edges do not have tokens. This means that cs-safeness is not affected. Therefore, the overall term $\langle \mathsf{xorSplit}(\mathsf{e}, E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E', \mathsf{e}''), \sigma' \rangle$ is cs-safe.

- Let us consider $\mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j, \mathsf{e}'_j) | j \in [1..n]\}) \parallel P_1 \parallel \ldots \mid P_n \parallel \mathsf{xorJoin}(E, \mathsf{e}'')$, with $\forall j \in [1..n] \;\; isWSCore(P_j), \;\; in(P_j) = \mathsf{e}'_j, \;\; out(P_j) \subseteq E$. There are the following possibilities:
  - $\langle \mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j, \mathsf{e}'_j) | j \in [1..n]\}), \sigma \rangle$ evolves by means of rule *P-$EventG$*. We can exploit the fact that this is a core reachable well-structured configuration to prove that $\sigma(\mathsf{e}) = 1$ and $\forall \mathsf{e}'_j | j \in [1..n]. \sigma(\mathsf{e}'_j) = 0$. Thus, $\langle \mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j, \mathsf{e}'_j) | j \in [1..n]\}), \sigma \rangle \xrightarrow{?\mathsf{m}_j} \sigma'$, with $\sigma' = inc(dec(\sigma, \mathsf{e}), \mathsf{e}'_j)$. Hence, $\forall \mathsf{e}''' \in edgesEl(\mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j, \mathsf{e}'_j) | j \in [1..n]\}))\ .\ \sigma(\mathsf{e}''') \leqslant 1$. By hypothesis $\langle \mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j, \mathsf{e}'_j) | j \in [1..n]\}), \sigma \rangle$ is cs-safe, i.e. if $\sigma'(\mathsf{e}'_j) = 1$, that is there is a token on one of the outgoing edges of the Event Based in the state $\langle \mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j, \mathsf{e}'_j) | j \in [1..n]\}), \sigma' \rangle$, then all the other edges are unmarked. This means that cs-safeness is not affected. Therefore, the overall term $\langle \mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j, \mathsf{e}'_j) | j \in [1..n]\}) \parallel P_1 \parallel \ldots \mid P_n \parallel \mathsf{xorJoin}(E, \mathsf{e}''), \sigma' \rangle$ 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.
  - $\langle \mathsf{xorJoin}(\{\mathsf{e}\} \cup E, \mathsf{e}'), \sigma \rangle$ evolves by means of rule *P-$XorJoin$*. We can exploit the fact that this is a core reachable well-structured configuration to prove that $\sigma(\mathsf{e}) = 1$, $\forall \mathsf{e}'' \in E' . \sigma(\mathsf{e}'') = 0$ and $\sigma(\mathsf{e}') = 0$. Thus $\langle \mathsf{xorJoin}(\{\mathsf{e}\} \cup E, \mathsf{e}'), \sigma \rangle \xrightarrow{\epsilon} \sigma'$, with $\sigma' = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$. Hence, $\forall \mathsf{e}''' \in edgesEl(\mathsf{xorJoin}(\{\mathsf{e}\} \cup E, \mathsf{e}'))\ .\ \sigma(\mathsf{e}''') \leqslant 1$. By hypothesis $\langle \mathsf{xorSplit}(\mathsf{e}, E) \mid \mid P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E', \mathsf{e}''), \sigma \rangle$ is cs-safe, i.e. if there is a token on the outgoing edge of the XOR-Join in the state $\langle \mathsf{xorJoin}(\{\mathsf{e}\} \cup E, \mathsf{e}'), \sigma' \rangle$ all the other edges do not have tokens. This means that cs-safeness is not affected. Therefore, the overall term $\langle \mathsf{xorSplit}(\mathsf{e}, E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E', \mathsf{e}''), \sigma' \rangle$ is cs-safe.
- Let us consider $\mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}') \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{\mathsf{e}^{\mathsf{v}}, \mathsf{e}^{\mathsf{vi}}\})$ with $in(P_1) = \{\mathsf{e}'\}$, $out(P_1) = \{\mathsf{e}^{\mathsf{iv}}\}$, $in(P_2) = \{\mathsf{e}^{\mathsf{vi}}\}$, $out(P_2) = \{\mathsf{e}''\}$. We have the following possibilities:
  - $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}'), \sigma \rangle$ 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 $\sigma(\mathsf{e}') = 0$ and either $\sigma(\mathsf{e}'') = 1$ or $\sigma(\mathsf{e}''') = 1$; let us assume the marking is $\sigma(\mathsf{e}''') = 1$ (since the other case is similar). Thus $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}'), \sigma\rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = inc(dec(\sigma, \mathsf{e}'''), \mathsf{e}')$. Hence, $edgesEl(\mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}')) = \{\mathsf{e}'', \mathsf{e}''', \mathsf{e}'\}$ and $\sigma'(\mathsf{e}') = 1$, $\sigma'(\mathsf{e}''') = 0$ and $\sigma'(\mathsf{e}'') = 0$, that is $\forall \mathsf{e} \in edgesEl(\mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}'))\ .\ \sigma'(\mathsf{e}) \leqslant 1$. By hypothesis $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}') \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}^{iv}, \{\mathsf{e}^{v}, \mathsf{e}^{vi}\}), \sigma\rangle$ is cs-safe, i.e. if there is a token on e' in the state $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}'), \sigma'\rangle$ all the other edges do not have token. This means that cs-safeness is not affected. Therefore, the overall term $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}') \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}^{iv}, \{\mathsf{e}^{v}, \mathsf{e}^{vi}\}), \sigma'\rangle$ 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 xor join and xor split gateways. In this case we can reason like in the first case, by relying on inductive hypothesis.

– $\langle \mathsf{xorSplit}(\mathsf{e}^{iv}, \{\mathsf{e}^{v}, \mathsf{e}^{vi}\}), \sigma\rangle$ 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 $\sigma(\mathsf{e}^{iv}) = 1$. Hence, it evolves in a cs-safe term; in fact let us assume that it evolves in this way $\langle \mathsf{xorSplit}(\mathsf{e}^{iv}, \{\mathsf{e}^{v}, \mathsf{e}^{vi}\}), \sigma\rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = inc(dec(\sigma, \mathsf{e}^{iv}), \mathsf{e}^{v})$. Hence, $edgesEl(\mathsf{xorSplit}(\mathsf{e}^{iv}, \{\mathsf{e}^{v}, \mathsf{e}^{vi}\})) = \{\mathsf{e}^{iv}, \mathsf{e}^{v}, \mathsf{e}^{vi}\}$ and $\sigma'(\mathsf{e}^{iv}) = 0, \sigma'(\mathsf{e}^{v}) = 1, \sigma'(\mathsf{e}^{vi}) = 0$, that is $\forall \mathsf{e} \in edgesEl(\mathsf{xorSplit}(\mathsf{e}^{iv}, \{\mathsf{e}^{v}, \mathsf{e}^{vi}\}))\ .\ \sigma'(\mathsf{e}) \leqslant 1$. By hypothesis $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}') \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}^{iv}, \{\mathsf{e}^{v}, \mathsf{e}^{vi}\}), \sigma\rangle$ is cs-safe, i.e. if there is a token on e^{v} in the state $\langle \mathsf{xorSplit}(\mathsf{e}^{iv}, \{\mathsf{e}^{v}, \mathsf{e}^{vi}\}), \sigma'\rangle$ all the other edges do not have token. This means that cs-safeness is not affected. Therefore, the overall term $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}') \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}^{iv}, \{\mathsf{e}^{v}, \mathsf{e}^{vi}\}), \sigma'\rangle$ is cs-safe.

- Let us consider $\mathsf{subProc}(\mathsf{e}, \mathsf{start}(\mathsf{e}', \mathsf{e}'') \parallel P'_1 \parallel \mathsf{end}(\mathsf{e}''', \mathsf{e}^{iv}), \mathsf{e}^{v})$. By hypothesis this is a cs-safe process configuration, then $edgesEl(\mathsf{subProc}(\mathsf{e}, \mathsf{start}(\mathsf{e}', \mathsf{e}'') \mid\mid P'_1 \parallel \mathsf{end}(\mathsf{e}''', \mathsf{e}^{iv}), \mathsf{e}^{v})) = \{\mathsf{e}, \mathsf{e}'', \mathsf{e}''', \mathsf{e}^{v}\} \cup edgesEl(P'_1)$ are such that $\forall \mathsf{e}^{vi} \in edgesEl(\mathsf{subProc}(\mathsf{e}, \mathsf{start}(\mathsf{e}', \mathsf{e}'') \parallel P'_1 \parallel \mathsf{end}(\mathsf{e}''', \mathsf{e}^{iv}), \mathsf{e}^{v}))\ .\ \sigma(\mathsf{e}^{vi}) \leqslant 1$. We have the following possibilities:

  – $\langle \mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}^{v}), \sigma\rangle$ evolves by means of rule *P-SubProcStart*. In order to apply the rule it should be $\sigma(\mathsf{e}) > 0$; hence, by cs-safeness, $0 < \sigma(\mathsf{e}) \leqslant 1$, i.e. $\sigma(\mathsf{e}) = 1$. We can exploit the fact that this is a reachable process configuration to prove that $\sigma(\mathsf{e}^{v}) = 0$ and $\sigma(edges(P_1)) = 0$. Thus, $\langle \mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}^{v}), \sigma\rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = inc(dec(\sigma, \mathsf{e}), start(P_1))$. Hence, $\forall \mathsf{e}^{vi} \in edgesEl(\mathsf{subProc}(\mathsf{e}, \mathsf{start}(\mathsf{e}', \mathsf{e}'') \parallel P'_1 \parallel \mathsf{end}(\mathsf{e}''', \mathsf{e}^{iv}), \mathsf{e}^{v}))\ .\ \sigma'(\mathsf{e}^{vi}) \leqslant 1$. By hypothesis $\langle \mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}^{v}), \sigma\rangle$ is cs-safe and reachable, i.e. if there is a token on $start(P_1)$ in the state $\langle \mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}^{\mathsf{v}}), \sigma' \rangle$, then all other edges are unmarked. This means that cs-safeness is not affected. Therefore, the overall term is cs-safe.

– P_1 evolves. Thus, $\langle \mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}^{\mathsf{v}}), \sigma \rangle$ can evolve by means of rules $P\text{-}SubProcEvolution$, $P\text{-}SubProcEnd$ or $P\text{-}SubProcKill$. In all the cases we can conclude by relying on the inductive hypothesis and on the fact that we consider core reachable configurations.

- 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 $edgesEl(\cdot)$ function we have that $edgesEl(P) = edgesEl(P_1) \cup edgesEl(P_2)$. By inductive hypothesis we have that $\forall \mathsf{e} \in edgesEl(P_1) \,.\, \sigma(\mathsf{e}) \leqslant 1$ 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.

□

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 $P\text{-}Start$. 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 $\forall \mathsf{e}^{\mathsf{v}} \in edgesEl(P') \,.\, \sigma(\mathsf{e}^{\mathsf{v}}) \leqslant 1$.

  Now $\forall \mathsf{e}^{\mathsf{v}} \in edgesEl(P') \,.\, \sigma(\mathsf{e}^{\mathsf{v}}) \leqslant 1$ and $\forall \mathsf{e}^{\mathsf{v}} \in edgesEl(P') \,.\, \sigma'(\mathsf{e}^{\mathsf{v}}) \leqslant 1$. Therefore $edgesEl(P) = \{\mathsf{e}', \mathsf{e}''\} \cup edgesEl(P')$ are such that $\sigma'(\mathsf{e}') = 1$, $\sigma'(in(P')) \leqslant 1$, $\sigma'(out(P')) \leqslant 1$, $\sigma'(\mathsf{e}'') \leqslant 1$. Thus, $\langle P, \sigma' \rangle$ is cs-safe.

- $\mathsf{end}(\mathsf{e}'', \mathsf{e}''')$ evolves by means of the rule $P\text{-}End$. 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

$\langle \mathsf{end}(\mathsf{e}'', \mathsf{e}'''), \sigma \rangle \xrightarrow{\epsilon} \sigma_1'$ with $\sigma_1' = inc(dec(\sigma, \mathsf{e}''), \mathsf{e}''')$. 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 $\forall \mathsf{e} \in edgesEl(P') \,.\, \sigma'(\mathsf{e}) \leqslant 1$. By hypothesis, P is cs-safe therefore $edgesEl(\mathsf{start}(\mathsf{e}, \mathsf{e}')) = \{\mathsf{e}'\}$ is such that $\sigma'(\mathsf{e}') \leqslant 1$ and $edgesEl(\mathsf{end}(\mathsf{e}'', \mathsf{e}''')) = \{\mathsf{e}''\}$ is such that $\sigma'(\mathsf{e}'') \leqslant 1$. 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*-$Terminate$.We can exploit the fact that this is a reachable well-structured configuration to prove that the term is marked as $\sigma(\mathsf{e}'') = 1$. By applying the rule we have $\langle \mathsf{terminate}(\mathsf{e}''), \sigma \rangle \xrightarrow{kill} \sigma_1'$ with $\sigma_1' = dec(\sigma, \mathsf{e}'')$. 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.

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

- The start event evolves: like the previous case.
- The end message event evolves: the only transition we can apply is *P*-$EndSnd$. 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 $\sigma(\mathsf{e}''') = 0$. By applying the rule we have $\langle \mathsf{endSnd}(\mathsf{e}'', \mathsf{m}, \mathsf{e}'''), \sigma \rangle \xrightarrow{!\mathsf{m}} \sigma_1'$ with $\sigma_1' = inc(dec(\sigma, \mathsf{e}''), \mathsf{e}''')$ 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: similar to the previous cases.

Now we consider the case P= $\mathsf{startRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'', \mathsf{e}''')$.

- $\mathsf{startRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$ evolves by means of the rule *P*-$StartRcv$. 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 $\langle \mathsf{startRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}'), \sigma \rangle \xrightarrow{?\mathsf{m}} \sigma_1'$ with

$\sigma'_1 = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$ where $\sigma'_1(\mathsf{e}) = 0$ and $\sigma'_1(\mathsf{e}') = 1$. Now, $\langle P, \sigma'_1 \rangle = \mathsf{startRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}') \parallel P' \parallel \mathsf{end}(\mathsf{e}'', \mathsf{e}'''), \sigma'_1 \rangle$ can evolve only through the application of *P-Int₁* 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 $\forall \mathsf{e}^{\mathsf{v}} \in edgesEl(P') \,.\, \sigma(\mathsf{e}^{\mathsf{v}}) \leqslant 1$.

Now $\forall \mathsf{e}^{\mathsf{v}} \in edgesEl(P') \,.\, \sigma(\mathsf{e}^{\mathsf{v}}) \leqslant 1$ and $\forall \mathsf{e}^{\mathsf{v}} \in edgesEl(P') \,.\, \sigma'(\mathsf{e}^{\mathsf{v}}) \leqslant 1$. Therefore $edgesEl(P) = \{\mathsf{e}', \mathsf{e}''\} \cup edgesEl(P')$ are such that $\sigma'(\mathsf{e}') = 1$, $\sigma'(in(P')) \leqslant 1$, $\sigma'(out(P')) \leqslant 1$, $\sigma'(\mathsf{e}'') \leqslant 1$. Thus, $\langle P, \sigma' \rangle$ is cs-safe.

- $\mathsf{end}(\mathsf{e}'', \mathsf{e}''')$ evolves by means of the rule *P-End*. It follows as in the first case.

- P' moves, that is $\langle P', \sigma \rangle \xrightarrow{\alpha} \sigma'$. By Lemma 2 $\langle P', \sigma' \rangle$ is safe, thus $\forall \mathsf{e} \in edgesEl(P') \,.\, \sigma'(\mathsf{e}) \leqslant 1$. By hypothesis, P is cs-safe therefore $edgesEl(\mathsf{startRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')) = \{\mathsf{e}'\}$ is such that $\sigma'(\mathsf{e}') \leqslant 1$ and $edgesEl(\mathsf{end}(\mathsf{e}'', \mathsf{e}''')) = \{\mathsf{e}''\}$ is such that $\sigma'(\mathsf{e}'') \leqslant 1$. We can conclude that $\langle P, \sigma' \rangle$ is safe.

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

- $\mathsf{startRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$ evolves by means of the rule *P-StartRcv*: like in the previous case.

- The end terminate event evolves: the only transition we can apply is *P-Terminate*: like in the case P= $\mathsf{start}(\mathsf{e}, \mathsf{e}') \parallel P' \parallel \mathsf{terminate}(\mathsf{e}'')$.

- P' moves, that is $\langle P', \sigma \rangle \xrightarrow{\alpha} \sigma'$. By Lemma 2 $\langle P', \sigma' \rangle$ is safe, thus $\forall \mathsf{e} \in edgesEl(P') \,.\, \sigma'(\mathsf{e}) \leqslant 1$. By hypothesis, P is cs-safe therefore $edgesEl(\mathsf{startRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')) = \{\mathsf{e}'\}$ is such that $\sigma'(\mathsf{e}') \leqslant 1$ and $edgesEl(\mathsf{terminate}(\mathsf{e}'')) = \{\mathsf{e}''\}$ is such that $\sigma'(\mathsf{e}'') \leqslant 1$. We can conclude that $\langle P, \sigma' \rangle$ is safe.

Now we consider the case P= $\mathsf{startRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}') \parallel P' \parallel \mathsf{endSnd}(\mathsf{e}'', \mathsf{m}, \mathsf{e}''')$.

- $\mathsf{startRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$ evolves by means of the rule *P-StartRcv*: like in the previous case.

- $\mathsf{endSnd}(\mathsf{e}'', \mathsf{m}, \mathsf{e}''')$ evolves by means of *P-EndSnd* : like in the case P= $\mathsf{start}(\mathsf{e}, \mathsf{e}') \parallel P' \parallel \mathsf{endSnd}(\mathsf{e}'', \mathsf{m}, \mathsf{e}''')$.

- P' moves, that is $\langle P', \sigma \rangle \xrightarrow{\alpha} \sigma'$. By Lemma 2 $\langle P', \sigma' \rangle$ is safe, thus $\forall \mathsf{e} \in edgesEl(P') \,.\, \sigma'(\mathsf{e}) \leqslant 1$. By hypothesis, P is cs-safe therefore $edgesEl(\mathsf{startRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')) = \{\mathsf{e}'\}$ is such that $\sigma'(\mathsf{e}') \leqslant 1$ and $edgesEl(\mathsf{endSnd}(\mathsf{e}'', \mathsf{m}, \mathsf{e}''')) = \{\mathsf{e}''\}$ is such that $\sigma'(\mathsf{e}'') \leqslant 1$. We can conclude that $\langle P, \sigma' \rangle$ is safe.

□

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, given σ and δ such that $isInit(C, \sigma, \delta)$ there exists a collaboration configuration $\langle C, \sigma', \delta'\rangle$ such that $\langle C, \sigma, \delta\rangle \rightarrow^* \langle C, \sigma', \delta'\rangle$ and $\exists P$ in C, $\langle P, \sigma'\rangle$ not cs-safe. From hypothesis $isInit(C, \sigma, \delta)$, we have $isInit(P, \sigma)$. Thus, also $\langle P, \sigma'\rangle$ is reachable. 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.

- $P = \mathsf{task}(\mathsf{e}, \mathsf{e}')$. 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.

- $P = \mathsf{taskRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}')$. The only rule we can apply is *P-$TaskRcv$*. 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 $\langle \mathsf{taskRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}'), \sigma\rangle \xrightarrow{?\mathsf{m}} \sigma'$ 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.

- P = $\mathsf{taskSnd(e, m, e')}$. The only rule we can apply is *P-*$\mathit{TaskSnd}$. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$. Since $\mathit{isWSCore}(P)$, $\langle P, \sigma\rangle$ 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 $\langle \mathsf{taskSnd(e, m, e')}, \sigma\rangle \xrightarrow{\mathsf{!m}} \sigma'$ with $\sigma' = \mathit{inc}(\mathit{dec}(\sigma, \mathsf{e}), \mathsf{e'})$. Thus, we have $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e'}) = 1$, which permits us to conclude.

- P = $\mathsf{interRcv(e, m, e')}$. The only rule we can apply is *P-*$\mathit{InterRcv}$. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$. Since $\mathit{isWSCore}(P)$, $\langle P, \sigma\rangle$ 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 $\langle \mathsf{interRcv(e, m, e')}, \sigma\rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = \mathit{inc}(\mathit{dec}(\sigma, \mathsf{e}), \mathsf{e'})$. Thus, we have $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e'}) = 1$, which permits us to conclude.

- P = $\mathsf{interSnd(e, m, e')}$,. The only rule we can apply is *P-*$\mathit{InterSnd}$. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$. Since $\mathit{isWSCore}(P)$, $\langle P, \sigma\rangle$ 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 $\langle \mathsf{interSnd(e, m, e')}, \sigma\rangle \xrightarrow{\mathsf{!m}} \sigma'$ with $\sigma' = \mathit{inc}(\mathit{dec}(\sigma, \mathsf{e}), \mathsf{e'})$. Thus, we have $\sigma'(\mathsf{e}) = 0$ and $\sigma'(\mathsf{e'}) = 1$, which permits us to conclude.

- $P = \mathsf{empty(e, e')}$,. The only rule we can apply is *P-*Empty. In order to apply the rule there must be $\sigma(\mathsf{e}) > 0$. Since $\mathit{isWSCore}(P)$, $\langle P, \sigma\rangle$ 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 $\langle \mathsf{empty(e, e')}, \sigma\rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = \mathit{inc}(\mathit{dec}(\sigma, \mathsf{e}), \mathsf{e'})$. 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 $P\text{-}\mathit{AndSplit}$. 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, $\langle \mathsf{andSplit}(\mathsf{e}, E), \sigma\rangle \xrightarrow{\epsilon} \sigma'_1$ with $\sigma'_1 = \mathit{inc}(\mathit{dec}(\sigma, \mathsf{e}), E)$. Now, P can evolve only through the application of $P\text{-}\mathit{Int}_1$ producing $\langle P, \sigma'_2\rangle$ with $\sigma'_2(\mathit{in}(P_1)) = \ldots = \sigma'_2(\mathit{in}(P_n)) = 1$. By inductive hypothesis there exists a state σ'_3 such that $\mathit{isCompleteEl}(P_1 \mid \mid \ldots \parallel P_n, \sigma'_3)$. Now, P can only evolve by applying rule $P\text{-}\mathit{Int}_1$, producing $\langle P, \sigma'_4\rangle$ where $\forall \mathsf{e'''} \in E'\, .\, \sigma'_4(\mathsf{e'''}) = 1$. Now, $\langle \mathsf{andJoin}(E', \mathsf{e'}), \sigma'_4\rangle$ can evolve by means of rule $P\text{-}\mathit{AndJoin}$. The application of the rule produces $\langle \mathsf{andJoin}(E', \mathsf{e'}), \sigma'_4\rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = \mathit{inc}(\mathit{dec}(\sigma'_4, E'), \mathsf{e'})$, i.e. $\sigma'(\mathsf{e'}) = 1$ and $\forall \mathsf{e'''} \in E'\, .\, \sigma'(\mathsf{e'''}) = 0$. 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.

- Let us consider $P = \langle \mathsf{xorSplit}(\mathsf{e}, E) \parallel P_1 \parallel \ldots \parallel P_n \parallel \mathsf{xorJoin}(E', \mathsf{e}''), \sigma \rangle$. There are the following possibilities:
  - $\langle \mathsf{xorSplit}(\mathsf{e}, E), \sigma \rangle$ evolves by means of rule $P\text{-}XorSplit$. 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, $\langle \mathsf{xorSplit}(\mathsf{e}, \{\mathsf{e}'\} \cup E), \sigma \rangle \xrightarrow{\epsilon} \sigma'_1$ with $\sigma'_1 = inc(dec(\sigma, \mathsf{e}), \mathsf{e}')$. Now, P can evolve only through the application of $P\text{-}Int_1$ producing $\langle P, \sigma'_2 \rangle$ with $\sigma'_2(in(P_1)) = \ldots = \sigma'_2(in(P_n)) = 1$. By inductive hypothesis there exists a state σ'_3 such that $isCompleteEl(P_1 \mid \mid \ldots \parallel P_n, \sigma'_3)$. Now, P can only evolve by applying rule $P\text{-}Int_1$, producing $\langle P, \sigma'_4 \rangle$ where $\exists \mathsf{e}''' \in E'\ .\ \sigma'_4(\mathsf{e}''') = 1$, let us say $\sigma'_4(\mathsf{e}^{\mathsf{iv}}) = 1$. Now, $\langle \mathsf{xorJoin}(\{\mathsf{e}^{\mathsf{iv}}\} \cup E', \mathsf{e}'), \sigma'_4 \rangle$ can evolve by means of rule $P\text{-}XorJoin$. The application of the rule produces $\langle \mathsf{xorJoin}(\{\mathsf{e}^{\mathsf{iv}}\} \cup E', \mathsf{e}'), \sigma'_4 \rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = inc(dec(\sigma, \mathsf{e}^{\mathsf{iv}}), \mathsf{e}')$, i.e. $\sigma'(\mathsf{e}') = 1$ and $\forall \mathsf{e}''' \in E'\ .\ \sigma'(\mathsf{e}''') = 0$. 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.

- Let us consider $P = \mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j, \mathsf{e}'_j) | j \in [1..n]\}) \parallel P_1 \parallel \ldots \mid P_n \parallel \mathsf{xorJoin}(E, \mathsf{e}'')$. There are the following possibilities:
  - $\langle \mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j, \mathsf{e}'_j) | j \in [1..n]\}), \sigma \rangle$ evolves by means of rule $P\text{-}EventG$. We can exploit the fact that this is a core reachable well-structured configuration to prove that $\sigma(\mathsf{e}) = 1$ and $\forall \mathsf{e}'_j | j \in [1..n].\sigma(\mathsf{e}'_j) = 0$. Thus, $\langle \mathsf{eventBased}(\mathsf{e}, \{(\mathsf{m}_j, \mathsf{e}'_j) | j \in [1..n]\}), \sigma \rangle \xrightarrow{?\mathsf{m}_j} \sigma'_1$ with $\sigma'_1 = inc(dec(\sigma, \mathsf{e}), \mathsf{e}'_j)$. Now, P can evolve only through the application of $P\text{-}Int_1$ producing $\langle P, \sigma'_2 \rangle$ with $\sigma'_2(in(P_1)) = \ldots = \sigma'_2(in(P_n)) = 1$. By inductive hypothesis there exists a state σ'_3 such that $isCompleteEl(P_1 \parallel \ldots \mid \mid P_n, \sigma'_3)$. Now, P can only evolve by applying rule $P\text{-}Int_1$, producing $\langle P, \sigma'_4 \rangle$ where $\exists \mathsf{e}''' \in E'\ .\ \sigma'_4(\mathsf{e}''') = 1$, let us say $\sigma'_4(\mathsf{e}^{\mathsf{iv}}) = 1$. Now, $\langle \mathsf{xorJoin}(\{\mathsf{e}^{\mathsf{iv}}\} \cup E, \mathsf{e}'), \sigma'_4 \rangle$ can evolve by means of rule $P\text{-}XorJoin$. The application of the rule produces $\langle \mathsf{xorJoin}(\{\mathsf{e}^{\mathsf{iv}}\} \cup E, \mathsf{e}'), \sigma'_4 \rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = inc(dec(\sigma, \mathsf{e}^{\mathsf{iv}}), \mathsf{e}')$, i.e. $\sigma'(\mathsf{e}') = 1$ and $\forall \mathsf{e}''' \in E\ .\ \sigma'(\mathsf{e}''') = 0$. 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.

- Let us consider $\mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}') \parallel P_1 \parallel P_2 \parallel \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{\mathsf{e}^{\mathsf{v}}, \mathsf{e}^{\mathsf{vi}}\})$ with $in(P_1) = \{\mathsf{e}'\}$, $out(P_1) = \{\mathsf{e}^{\mathsf{iv}}\}$, $in(P_2) = \{\mathsf{e}^{\mathsf{vi}}\}$, $out(P_2) = \{\mathsf{e}''\}$. There are the following possibilities:
  - $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}'), \sigma \rangle$ 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 $\sigma(\mathsf{e}') = 0$ and either $\sigma(\mathsf{e}'') = 1$ or $\sigma(\mathsf{e}''') = 1$; let us assume the marking is $\sigma(\mathsf{e}''') = 1$ (since the other case is similar). Thus $\langle \mathsf{xorJoin}(\{\mathsf{e}'', \mathsf{e}'''\}, \mathsf{e}'), \sigma \rangle \xrightarrow{\epsilon} \sigma_1'$ with $\sigma_1' = inc(dec(\sigma, \mathsf{e}'''), \mathsf{e}')$. Now, P can evolve only through the application of *P-Int₁* producing $\langle P, \sigma_2' \rangle$ with $\sigma_2'(in(P_1)) = \sigma_2'(in(P_2)) = 1$. By inductive hypothesis there exists a state σ_3' such that $isCompleteEl(P_1 \parallel P_2, \sigma_3')$. Now, P can only evolve by applying rule *P-Int₁*, producing $\langle P, \sigma_4' \rangle$ with, $\sigma_4'(\mathsf{e}^{\mathsf{iv}}) = 1$. Now, $\langle \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{\mathsf{e}^{\mathsf{v}}, \mathsf{e}^{\mathsf{vi}}\}), \sigma_4' \rangle$ can evolve by means of rule *P-XorSplit*. The application of the rule produces $\langle \mathsf{xorSplit}(\mathsf{e}^{\mathsf{iv}}, \{\mathsf{e}^{\mathsf{v}}, \mathsf{e}^{\mathsf{vi}}\}), \sigma' \rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = inc(dec(\sigma_4', \mathsf{e}^{\mathsf{iv}}), \mathsf{e}^{\mathsf{v}})$, i.e. $\sigma'(\mathsf{e}^{\mathsf{v}}) = 1$ and $\sigma'(\mathsf{e}^{\mathsf{iv}}) = \sigma'(\mathsf{e}^{\mathsf{vi}}) = 0$. This permits us to conclude.
  - $P_1 \parallel P_2$ evolves without affecting the split and join gateways. In this case we can easily conclude by inductive hypothesis.
  - $P_1 \parallel P_2$ evolves and affects the split and/or join gateways. In this case we can reason like in the first case.

- Let us consider $\mathsf{subProc}(\mathsf{e}, \mathsf{start}(\mathsf{e}', \mathsf{e}'') \parallel P_1' \parallel \mathsf{end}(\mathsf{e}''', \mathsf{e}^{\mathsf{iv}}), \mathsf{e}^{\mathsf{v}})$ with $isWSCore(P_1'), in(P_1') = \{\mathsf{e}''\}, out(P_1') = \{\mathsf{e}'''\}$. Let us call $P_1 = \mathsf{start}(\mathsf{e}', \mathsf{e}'') \parallel P_1' \parallel \mathsf{end}(\mathsf{e}''', \mathsf{e}^{\mathsf{iv}})$, thus the overall term becomes $\mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}^{\mathsf{v}})$ The we have:
  - $\mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}^{\mathsf{v}})$ evolves by means of rule *P-SubProcStart*. We can exploit the fact that this is a core reachable well-structured configuration to prove that $\sigma(\mathsf{e}) = 1$ and $\forall \mathsf{e}^{\mathsf{vi}} \in edgesEl(\mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}^{\mathsf{v}})) \backslash \{\mathsf{e}\} \,.\, \sigma(\mathsf{e}^{\mathsf{vi}}) = 0$. The application of the rule produces $\langle \mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}^{\mathsf{v}}), \sigma \rangle \xrightarrow{\epsilon} \sigma_1'$ with $\sigma_1' = inc(dec(\sigma, \mathsf{e}), start(P_1))$. Now, P_1 can evolve only through the application of *P-Int₁*. Thus, $\langle \mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}^{\mathsf{v}}), \sigma \rangle$ can evolve by means of rules *P-SubProcEvolution*, or *P-SubProcKill*. In all the cases, by relying on the inductive hypothesis there exists a state σ_3' such that $isCompleteEl(P_1, \sigma_3')$. This means that there is a token on the incoming edge of the end event

of process P_1 and all other edges are unmarked, that is $\sigma'_3(end(P_1)) = \sigma'_3(\mathsf{e}^{sfiv}) = 1$ and $\forall \mathsf{e} \in edges(P_1) \backslash end(P_1) \,.\, \sigma(\mathsf{e}) = 0$. Indeed, predicate $completed(P_1, \sigma'_3)$ holds. We can now apply rule *P-SubProcEnd* producing $\langle \mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}^{\mathsf{v}}), \sigma'_3 \rangle \xrightarrow{\epsilon} \sigma'$ with $\sigma' = inc(zero(\sigma, end(P_1)), \mathsf{e}^{\mathsf{v}})$ that permits us to conclude.

- Let us consider $\langle P, \sigma \rangle = \langle P_1 \parallel P_2, \sigma \rangle$, with $isWSCore(P_1), isWSCore(P_2), out(P_1) = in(P_2)$. The relevant case for cs-safeness is when P evolves by applying *P-Int₁*. We have that $\langle P_1 \parallel P_2, \sigma \rangle \xrightarrow{\alpha} \sigma'_1$ with $\langle P_1, \sigma \rangle \xrightarrow{\alpha} \sigma'_1$. By inductive hypothesis we have that there exists σ' such that $isCompleteEl(P_1, \sigma')$. By hypothesis $out(P_1) = in(P_2)$ thus, $isCompleteEl(\mathsf{getOutEl}(\mathsf{e}, P_1 \parallel P_2)) = isCompleteEl(\mathsf{getOutEl}(\mathsf{e}, P_1))$, that holds by inductive hypothesis. By hypothesis P_2 is well structured and core reachable, then we have that $edges(P_2) \backslash out(P_2)) : \sigma'(\mathsf{e}) = 0$ By definition of $isCompleteEl(P_1, \parallel P_2, \sigma')$ we can conclude.

□

Theorem 3. *Let $isWS(P)$, then P 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 $\forall\ \mathsf{e}^{\mathsf{iv}} \in edges(P) \backslash start(P)\ .\ \sigma(\mathsf{e}^{\mathsf{iv}}) = 0$. 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-Int₁* 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-Int₁*. 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 $\forall \mathsf{e}^{\mathsf{v}} \in edges(P')\ .\ \sigma'_3(\mathsf{e}^{\mathsf{v}}) = 0$. We can now apply rule *P-End* that decrements the token in e'' and produces a token in e''', which permits us to conclude.

□

Theorem 4. *Let C be a collaboration, $isWS(C)$ 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. □

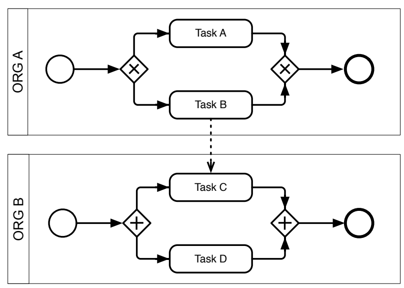


Figure B.23: An example of unsound collaboration with sound WS processes.

Theorem 5. *Let C be a collaboration, $isWS(C)$ 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 process, P is unsafe does not imply P is unsound.*

Proof. Let P be a unsafe process, 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. □

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

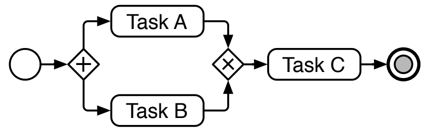


Figure B.24: An example of unsafe but sound process.

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. □

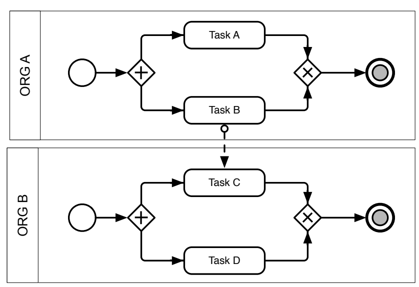


Figure B.25: An example of unsafe but sound collaboration.

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, $\exists \mathsf{e}''' \in edgesEl(P_1)\ .\ \sigma'_2(\mathsf{e}''') \geqslant 1$. By definition of function $edgesEl(\cdot)$, we have that $edgesEl(P) = edgesEl(\mathsf{subProc}(\mathsf{e}, P_1, \mathsf{e}')) \cup edgesEl(P_2)$. By safeness of P we have that 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 such that $\forall \mathsf{e} \in edgesEl(P) . \sigma'(\mathsf{e}) \leqslant 1$. Choosing $\sigma' = \sigma'_2$ we have that $\exists \mathsf{e}''' \in edgesEl(P)\ .\ \sigma'_2(\mathsf{e}''') \geqslant 1$. 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 $\forall\ P \in participant(C)$ 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'_2such that $\langle P_1, \sigma\rangle \rightarrow^* \sigma'_2$ we have that does not exist σ'_3such 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. □

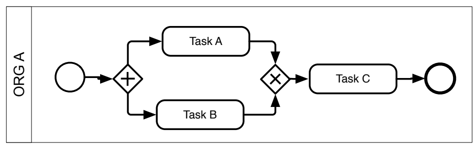


Figure B.26: An example of unsound process.

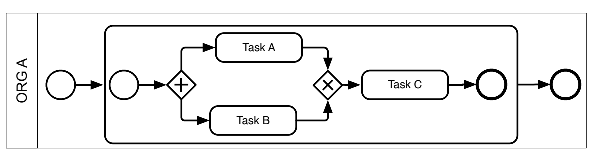


Figure B.27: An example of sound process with unsound sub-process.